

UNIVERSITÀ POLITECNICA DELLE MARCHE



FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA MAGISTRALE IN

INGEGNERIA INFORMATICA E DELL'AUTOMAZIONE

**Monitoraggio delle condizioni operative di macchine rotanti
tramite tecniche basate su misure di vibrazione**

Vibration based condition monitoring of rotating machines

Relatore:

Dott. Alessandro Freddi

Correlatore:

Ing. Matteo Pirro

Candidato:

Nicola Paolini

a.a. 2019/2020

Sommario

1.	Introduzione.....	1
2.	Motori Asincroni Monofase.....	3
2.1.	Diagnosi mediante analisi acustica.....	6
2.2.	Analisi della Corrente Statorica.....	10
2.3.	Analisi Vibrazionale.....	13
3.	Strumentazione e Metodi.....	15
3.1.	Acquisizione.....	17
3.2.	Features.....	22
3.2.1.	Features nel dominio del tempo.....	23
3.2.2.	Features nel dominio della frequenza.....	24
3.3.	Selezione delle Features.....	25
3.4.	Classificazione.....	25
4.	Implementazione.....	27
4.1.	Implementazione Hardware.....	27
4.2.	Simulazione.....	28
4.3.	Implementazione streaming dati.....	41
4.4.	Analisi inMatlab.....	43
4.4.1.	Diagnosi nel caso di cuscinetto ammaccato.....	47
4.4.2.	Risultati per tutte le altre categorie di guasto.....	63
5.	Conclusione.....	71
6.	Bibliografia.....	72

1. INTRODUZIONE

Il progetto di tesi sviluppato in collaborazione con l'azienda Meta srl tratta del monitoraggio delle condizioni operative di una serie di motori asincroni monofase impiegati in cappe aspiranti. Nello specifico, i motori sono sottoposti a tecniche di rivelamento dei guasti, descritte peculiarmente in [1] e [2], che possono comparire durante il loro funzionamento e tecniche di classificazione che permettono di capire il tipo di guasto che affligge il motore in un determinato intervallo temporale.

Poter rilevare i guasti (*fault*) e, di conseguenza, studiarne le contromisure prima che il motore vada in stato di rottura (*failure*), il quale richiederebbe del tempo per il fermo macchina e sostituzione del pezzo, è oggetto di studi molto attuali, specialmente nel campo della manutenzione predittiva. I benefici principali derivanti dalla possibilità di diagnosticare i guasti in breve tempo riguardano il risparmio di tempo e dei costi riguardanti un'eventuale manutenzione o sostituzione del motore.

L'obiettivo riportato precedentemente viene perseguito utilizzando un sistema di acquisizione dati caratterizzato dalla integrazione di diversi strumenti hardware e software, che verranno descritti successivamente. In questo documento è descritto il processo di analisi, di progettazione e implementazione del sistema di acquisizione e individuazione dei guasti. Le cause che possono portare il motore per cappe a guastarsi sono diverse e spaziano da cause esterne (e.g. filtro della cappa intasato e quindi il motore non riesce a lavorare alle condizioni nominali; la prolungata permanenza in questa condizione di lavoro può portare alla rottura del motore) o interne (e.g. cuscinetto attorno all'albero motore poco lubrificato che porta il motore a girare in maniera innaturale). Per poter apprezzare bene quali sono gli effetti che i guasti hanno sul motore si è scelto di analizzare i dati di natura vibrazionale. Il sistema di acquisizione raccoglie i dati vibrazionali dei motori in maniera continua e tale acquisizione è svolta utilizzando un accelerometro piezoelettrico posizionato direttamente sul motore lungo le direzioni assiale e radiale rispetto all'albero motore, sia in condizione di avviamento (transitorio) sia in condizione stazionaria. L'accelerometro è collegato ad un PLC BECKHOFF che permette di leggere le vibrazioni acquisite e, in ambiente TwinCAT 3, di collezionare i dati raccolti all'interno di strutture dati dalla lunghezza prefissata. Questa fase prende il nome di pre-elaborazione.

Successivamente alla pre-elaborazione dei dati sono state impostate due diverse modalità di elaborazione dei dati con conseguente estrazione delle features:

1. La prima modalità riguarda lo streaming di dati tra PLC e PC, tramite il software LabView, con conseguente estrazione delle features del segnale direttamente da PC. In questo caso l'analisi del segnale viene effettuata offline.
2. L'alternativa è di elaborare il segnale ed estrarre le feature direttamente in ambiente Twincat 3, senza necessità di instaurare una connessione con un PC. Il sistema di acquisizione è stato progettato a partire da una prima fase di pre-elaborazione di un segnale sinusoidale simulato all'interno dell'ambiente di sviluppo Twincat 3, sul quale sono stati codificati degli algoritmi in linguaggio strutturato che permettano di ricavare lo spettro in frequenza del segnale acquisito e le relative features. Una volta ottimizzato l'algoritmo si è passati ad una sua implementazione online.

In particolare, l'ultima soluzione è quella più interessante dato che implementa una diagnostica online in cui è sufficiente effettuare lo streaming diretto delle features in frequenza, anziché dell'intero pacchetto di dati acquisiti, lasciando al PC solo il compito di effettuare la classificazione del guasto. Così facendo si hanno vantaggi a livello di memoria del sistema di acquisizione e nello streaming, dato che il PC si troverà a lavorare con una mole di dati decisamente inferiore.

2. MOTORI ASINCRONI MONOFASE

Viene di seguito riportato il funzionamento del motore asincrono, una cui descrizione accurata è disponibile in [3] - [6]. Quando non è disponibile un'alimentazione trifase, è possibile utilizzare i motori asincroni monofase per macchine come seghe circolari, perforatrici, macchine d'aspirazione e ventilazione, elettrodomestici, macchine per l'ufficio.

I motori monofase ad induzione sono costruttivamente molto simili ai motori asincroni trifase (Figura 1). Normalmente il rotore è del tipo a gabbia di scoiattolo e lo statore presenta solo due avvolgimenti: uno principale (di marcia) e uno ausiliario (di avviamento), ed è proprio l'avvolgimento ausiliario che consente, mediante l'uso di opportuni accorgimenti, l'avviamento di questo tipo di motore: alimentando il solo avvolgimento principale, non si ottiene un campo rotante necessario per porre in rotazione il rotore, ma un campo magnetico alternato che è la risultante dalla sovrapposizione di due vettori rotanti in senso opposto dimezzato rispetto a quello principale. Il rotore in tali condizioni non può mettersi in rotazione, perché viene sollecitato nei due sensi con una coppia di uguale valore.

Però se con l'intervento di una forza esterna lo si lancia in una direzione o nell'altra, si viene a rompere l'equilibrio delle due coppie opposte a favore di quella che sollecita il rotore nel senso della forza esterna. In questo caso il motore proseguirà nella sua rotazione anche se viene a mancare la forza esterna che l'ha lanciato in rotazione. È chiaro che un motore che dovesse essere lanciato manualmente, o con altri sistemi, non sarebbe praticamente utilizzabile e non avrebbe potuto trovare quella diffusione che il motore monofase ad induzione ha avuto.

Dal punto di vista costruttivo, gli avvolgimenti di marcia e di avviamento vengono montati con uno sfasamento di 90° elettrici. Inoltre è necessario, affinché il motore possa autoavviarsi, che le correnti presenti nei due avvolgimenti siano sfasate il più possibile, in modo da creare un campo magnetico rotante in un senso ben definito così da permettere l'avviamento del motore. Lo sfasamento può essere ottenuto con un avvolgimento ausiliario resistivo, oppure ponendo in serie sempre all'avvolgimento ausiliario un condensatore di avviamento di capacità opportuna; quest'ultima soluzione è di solito la preferita, anche perché non sono necessari normalmente interruttori centrifughi che disinseriscano l'avvolgimento ausiliario.

L'avvolgimento ausiliario viene costruito con un filo di sezione abbastanza simile a quello di marcia, ed anche le cave occupate dall'avvolgimento ausiliario possono essere di poco inferiori a quelle del principale; in alcuni casi anzi i due avvolgimenti sono molto simili, sia per

numero di cave occupate sia per numero di spire. Il calcolo della capacità del condensatore di avviamento da adottare può essere ricavato matematicamente e che richiedono la conoscenza dei parametri costruttivi del motore. Ci si affida perciò generalmente a formule empiriche che forniscono ugualmente risultati soddisfacenti, eventualmente ritoccati in seguito alle prove pratiche.

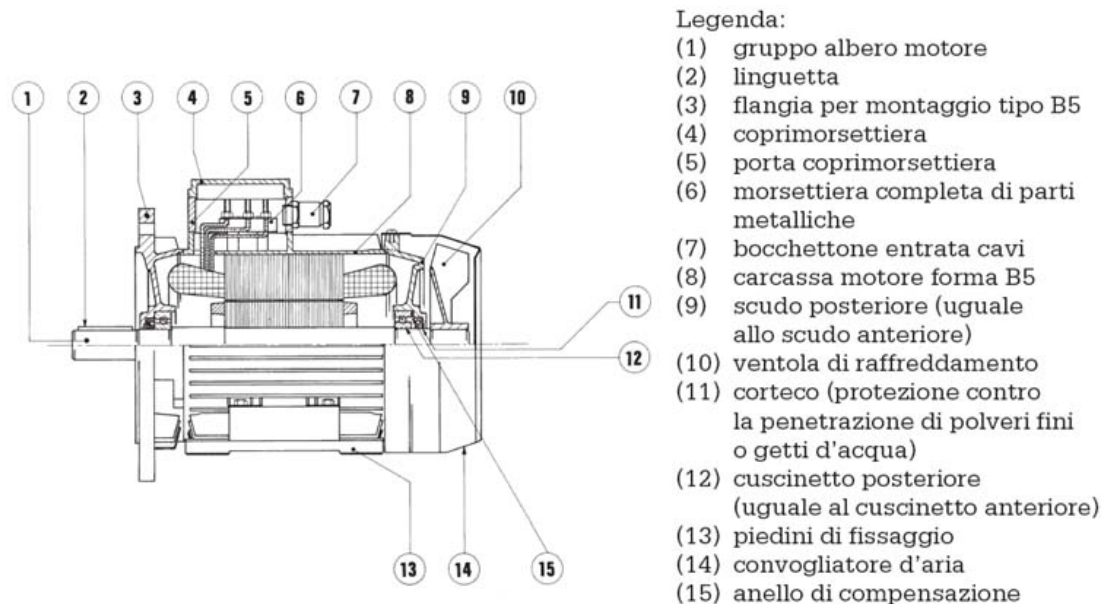


Figura 1: schema di un motore elettrico asincrono monofase

Il motore trifase è soggetto a guasti che possono colpire diverse grandezze elettriche del motore, portando a condizioni quali sovratensione, sovracorrente, sottotensione, sottocorrente, sovraccarico, sovratemperatura ecc. Quindi, per identificare tali guasti, i parametri misurabili più importanti sono tensione, corrente e temperatura. Inoltre, i guasti che possono affliggere il motore si dividono in elettrici, che includono cortocircuiti intermedi negli avvolgimenti dello statore, circuiti aperti negli avvolgimenti dello statore, barre del rotore rotte e anelli terminali rotti, e in guasti meccanici, che includono guasti ai cuscinetti (come ammaccatura o rottura), filo di rame che graffia sul rotore o calotta del motore allentata. Nel progetto di tesi ci si è concentrati sui guasti di natura meccanica appena menzionati.

In letteratura esistono molte tecniche che permettono di diagnosticare i guasti a seconda della grandezza elettrica analizzata:

1. **Tecniche diagnostiche basate sull'analisi delle firme del guasto nelle correnti elettriche.**

Queste tecniche hanno il vantaggio di essere molto efficienti nel riconoscimento del guasto. Questo grazie, anche, al fatto che segnale elettrico è facile da elaborare, rispetto ai segnali acustici (che risultano molto mescolati tra loro).

Sfortunatamente il segnale di corrente può essere utilizzato solo per stati difettosi limitati come avvolgimenti in corto, barre rotte, anello difettoso della gabbia di scoiattolo.

2. Tecniche basate su segnali di vibrazioni.

Le tecniche basate sull'analisi delle vibrazioni sono molto comuni. Analogamente all'analisi di correnti elettriche, tecniche basate sull'analisi delle vibrazioni hanno un'alta efficienza di riconoscimento. I vantaggi delle tecniche diagnostiche basate sulle vibrazioni sono: accelerometro economico, misurazione immediata del segnale di vibrazione, possibilità di analizzare componenti elettrici (avvolgimenti in cortocircuito, barre spezzate, anello difettoso della gabbia di scoiattolo) e guasti meccanici (cuscinetti, albero del rotore ecc.). Gli svantaggi delle tecniche di diagnosi dei guasti basate sulle vibrazioni sono che il set di accelerometri deve essere molto vicino al motore e deve essere lo stesso per effettuare le diverse misurazioni (misura negli assi X, Y, Z);

3. Tecniche diagnostiche di motori elettrici rotanti basate sull'analisi di immagini termiche.

Anche la misurazione delle immagini termiche è immediata e non invasiva. L'analisi delle immagini termiche è molto efficiente per il rilevamento dei guasti. Tuttavia, ci sono alcuni svantaggi quali il costo della termocamera, il fatto che il set di termocamere deve essere lo stesso (misurazione negli assi X, Y, Z), un elevato tempo necessario per riscaldare il motore e per elaborare le immagini termiche.

4. Tecniche diagnostiche di guasti basate sull'acustica.

I segnali acustici del motore sono miscelati con altri segnali acustici (segnali riflessi, segnali sovrapposti ecc.). Un altro svantaggio delle tecniche diagnostiche di guasto basate sull'acustica è la mancanza di cambiamenti nel segnale acustico per alcuni tipi di apparecchiature elettriche. Tuttavia, ci sono alcuni vantaggi come: facilità nell'accessibilità del segnale acustico, microfono economico, la possibilità di analizzare guasti elettrici e meccanici (avvolgimenti in cortocircuito, barre rotte, cuscinetti, albero del rotore ecc.). La misurazione dei segnali acustici è anche immediata e non invasiva.

Tra tutte le tecniche citate, quelle che restituiscono i risultati migliori in ambito di rilevamento, isolamento e diagnosi dei guasti sono le seguenti:

1. Analisi acustica

2. Analisi della corrente di statore
3. Analisi vibrazionale

2.1 Diagnosi mediante analisi acustica

La tecnica di analisi dei segnali acustici del motore di seguito riportata è tratta da [7] e presenta una descrizione dei metodi diagnostici di guasto dei cuscinetti, dello statore e del rotore di un motore a induzione monofase. I metodi presentati utilizzano segnali acustici, e in particolare vengono analizzati cinque stati del motore a induzione monofase:

- 1) *Motore sano;*
- 2) *Motore con bobine dell'avvolgimento ausiliario e avvolgimento principale in corto;*
- 3) *Motore con bobine dell'avvolgimento ausiliario in corto;*
- 4) *Motore con barre rotoriche rotte e anello della gabbia di scoiattolo difettoso;*
- 5) *Motore con cuscinetto difettoso.*

Una delle fasi cruciali del metodo di rilevamento guasti basata su segnali acustici è la selezione delle feature. A tale proposito viene proposto il metodo SMOFS-22-MULTIEXPANDED (*Shortened Method of Frequencies Selection Multiexpanded*). La fase di classificazione è stata eseguita utilizzando il classificatore NN (*Nearest - Neighbour*). Lo schema in Figura 2 sintetizza la metodologia diagnostica proposta:

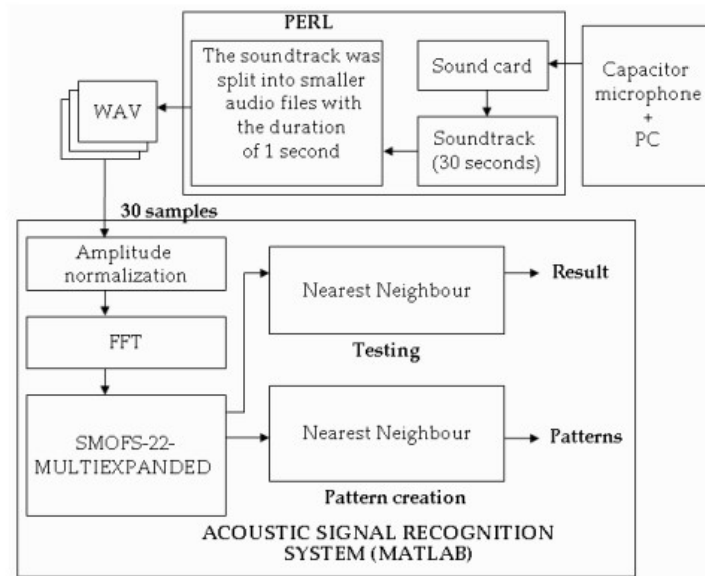


Figura 2: schema di classificazione proposto nel documento [7]

Per la misurazione del segnale acustico è stato utilizzato un microfono a bassa capacità e dal basso costo (nel dettaglio ZALMAN ZM-MIC1), all'interno di una stanza di 4x4 metri. È stato ottenuto il formato dei dati audio WAVE, canale mono e frequenza di campionamento pari a 44100 Hz.

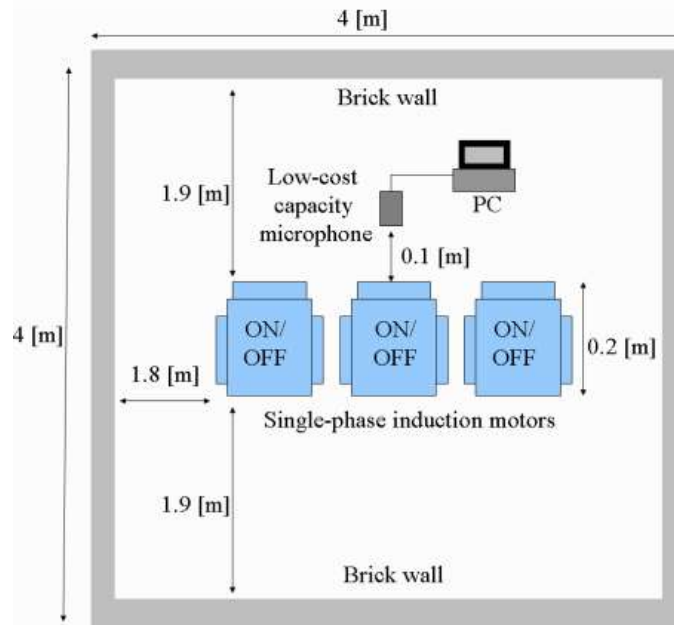


Figura 3: Metodologia e strumentazione adottate per effettuare le acquisizioni dei dati acustici

I dati acustici sono stati successivamente suddivisi in file di dati di 5 secondi. I file di dati sono stati elaborati in frequenza mediante una *Fast Fourier Transform*, che ha processato 16384 componenti in frequenza. Il metodo SMOFS-22- MULTIEXPANDED processa 22 vettori di features comuni. L'ultimo passo dell'approccio proposto è la classificazione usando il classificatore NN: nel processo di test un campione di prova sconosciuto (file audio) viene confrontato con campioni di addestramento cercando di minimizzare la distanza fra i due vettori (nel dettaglio, la distanza di Manhattan).

Il SMOFS-22-MULTIEXPANDED si basa sulle differenze tra gli spettri di frequenza dei segnali acustici, che sono differenti a seconda dello stato di funzionamento del motore:

1. Vengono costruiti cinque vettori (a,b,c,d,e), uno per ogni stato di funzionamento, da 16384 elementi l'uno;
2. Calcola i valori assoluti delle differenze di vettori precedentemente formati: $|a-b|$, $|a-c|$, $|a-d|$, $|a-e|$, $|b-c|$, $|b-d|$, $|b-e|$, $|c-d|$, $|c-e|$, $|d-e|$;
3. Viene utilizzata la formula seguente per selezionare le componenti in frequenza. In particolare, vengono selezionate le componenti in frequenza più grandi di una soglia:

$$\|FS_A - FS_B\| < ThrSel_x$$

dove a sinistra è data la norma della differenza fra due spettri in frequenza di due diversi stati di funzionamento (A e B);

4. La soglia è calcolata dalla formula seguente: $ThrSel_x = \frac{\sum_{N_0FC_x=1}^{N_0FC_x} \|FS_A - FS_B\|}{N_0FC_x}$, dove N_0FC_x è il numero delle componenti selezionate alla x -sima iterazione: se la variabile $N_0FC_x < 22$, la SMOFS viene interrotta, se invece $N_0FC_x > 2$ viene eseguita l'equazione del calcolo della soglia;
5. Bisogna impostare il parametro TCFC – MULTI (*Threshold of common frequency components*), ovvero il rapporto fra il numero delle componenti in frequenza richieste per il set di addestramento considerato e il numero delle differenze processate;
6. Trovare 22 componenti in frequenza richiesti;
7. Formare un vettore finale contenente 22 componenti trovati al passo precedente;

Di seguito lo schema che riassume i passi precedentemente riportati:

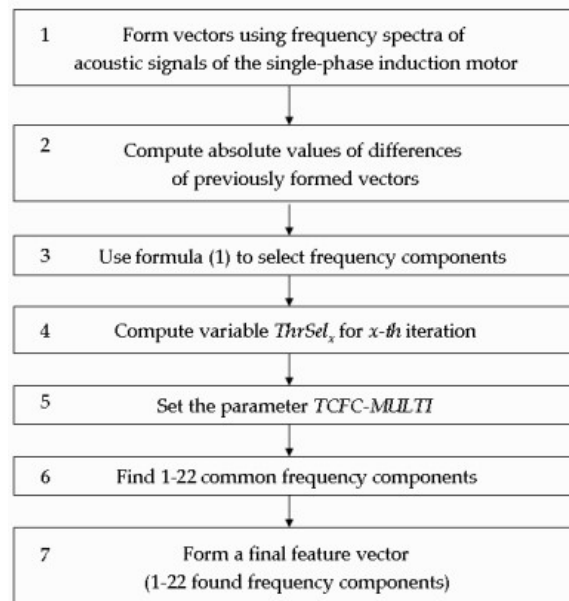


Figura 4:Diagramma rappresentativo dei passi da svolgere per portare a termine la classificazione

La tecnica di classificazione impiegata è il *NearestNeighbour*, che cerca i vettori di addestramento e vettori di test che più si somigliano. Successivamente assegna il vettore di test alla classe di guasto più vicina (classe). Il classificatore attraversa l'intero set di test calcolando d (distanza di somiglianza) tra il vettore di features di test e ciascun vettore delle features di

addestramento. Infine, il vettore di test viene assegnato alla classe con il vettore di addestramento più vicino.

Il NN classifica utilizzando la *distanza di somiglianza* come: **Manhattan, Euclidean, Minkowski, Jaccard, Chebyshev**. Viene selezionata la distanza di Manhattan (i risultati ottenuti con le altre distanze sono più o meno simili) ed è stata presentata come:

$$d(a, b) = \sum_{i=1}^l (a_i - b_i)$$

in cui a è vettore di test e b il vettore di addestramento, consistente delle componenti in frequenza.

Sono stati utilizzati 20 file audio di un secondo per la creazione del modello e 200 file audio di un secondo per il testing (fase di previsione). Sono stati analizzati campioni di test e di addestramento dei segnali acustici del motore. È stata espressa l'efficienza del riconoscimento del segnale acustico nel seguente modo:

$$E_{RAS} = \frac{N_{PTSAS}}{N_{ATSAS}} 100\%$$

dove E_{RAS} è l'efficienza del riconoscimento del segnale acustico della classe selezionata, N_{ptsas} è il numero dei file audio della classe selezionata testata, N_{atsas} è il numero di tutti i file audio della classe selezionata).

L'efficienza totale del riconoscimento è stata definita come segue:

$$TE_{RAS} = (E_{RAS1} + E_{RAS2} + E_{RAS3} + E_{RAS4} + E_{RAS5})/5$$

dove E_{RAS1} è l'efficienza di riconoscimento in condizione di motore funzionante, E_{RAS2} è l'efficienza di riconoscimento del motore con bobine di avvolgimento ausiliario e avvolgimento principale in corto, E_{RAS3} è l'efficienza di riconoscimento del motore con bobine di avvolgimento ausiliario in corto, E_{RAS4} è l'efficienza di riconoscimento del motore con le barre rotoriche rotte e gli anelli della gabbia di scoiattolo guasti, E_{RAS5} è l'efficienza di riconoscimento del motore con i cuscinetti guasti.

Sono state analizzate cinque diverse condizioni di funzionamento di un motore monofase, in particolare si sono analizzati guasti riguardanti statore, rotore e cuscinetti. L'analisi condotta ha fornito ottimi risultati (TE_{RAS} era compreso tra il 94% e il 97%). Il SMOFS-22-MULTIEXPANDED ha calcolato 4-14 componenti di frequenza comuni.

Le informazioni ottenute dai segnali acustici ci permettono di pianificare revisioni e riparazioni in funzione dei guasti in maniera immediata e non invasiva (le tecniche di misura acustica non sono invasive).

Il metodo proposto può trovare applicazione per i guasti meccanici ed elettrici di macchine rotanti. Lo svantaggio è che le onde sonore possono mescolarsi tra loro (*Beating*) e il setup sperimentale di misura è soggetto a numerosi disturbi, quindi poco adatto a misure in ambienti non controllati. La tecnica, similmente ad altre presenti in letteratura, utilizza features in frequenza per l'estrazione dell'informazioni e una classificazione per il riconoscimento guasti.

2.2 Analisi della Corrente Statorica

La tecnica di seguito descritta è riportata in [8] e in questa modalità di diagnosi vengono studiati i dati sulla corrente di statore, ottenuti in laboratorio da motori ad induzione alimentati da inverter e vengono esaminate le firme del motore, sia quando è sano sia quando è difettoso, con l'obiettivo di sviluppare un metodo di rilevamento guasti basato sull'analisi di features note per eseguire il rilevamento online di problemi di guasto al motore (come barra di rotore rotta e guasti dei cuscinetti).

I dati sulle correnti dello statore raccolti dai motori a induzione sono stati analizzati sfruttando la *Fast Fourier Transform (FFT)* e i risultati ottenuti da FFT sono stati ulteriormente analizzati dal *Independent Component Analysis (ICA)* per ottenere componenti indipendenti e features ai quali si farà riferimento come *FFT-ICA features* delle correnti di statore. Le funzionalità FFT-ICA contengono informazioni dettagliate sulle firme dei motori sia funzionanti che difettosi, che sono ulteriormente analizzati per creare un database di features note (*Knowledge features*) che sarà usato per il rilevamento dei guasti online.

Infine, viene applicato il principio di similarità per identificare il guasto confrontando le features della corrente di statore rilevata online con quelle del database noto.

I dati riguardanti la corrente di statore sono raccolti sia dal motore funzionante che da quello guasto (in particolare cuscinetti rotti e barre rotoriche rotte). I motori esaminati, in entrambi i casi di funzionamento, sono alimentati dalla stessa tensione mediante un inverter. Le correnti di statore sono misurate da un oscilloscopio digitale a 4 canali e l'esperimento viene condotto incrementando la resistenza di carico da zero al carico massimo in 5 diversi passi. Il motore è collegato ad un alimentatore AC trifase attraverso un inverter, il quale provvede a selezionare frequenze differenti, comprese fra 0 e 55Hz. Le forme d'onda della corrente di

statore sono collezionate da ogni motore a ciascuna frequenza con una frequenza di campionamento di 3.84kHz.

Le firme nel dominio in frequenza possono essere ottenute da una forma d'onda nel dominio del tempo sulla quale si calcola la FFT. Le firme nel dominio in frequenza che risultano dall'applicazione di una FFT includono l'informazione sull'ampiezza di ogni singola componente in frequenza:

$$f(t) \rightarrow (f(t_1)f(t_2)..f(t_L))$$

$$F(w) = FFT(f(t))$$

$$F(w) = (a(w_1)a(w_2) ... a(w_N))$$

dove $f(t_i)$ rappresenta l'ampiezza al tempo t_i ed L è la lunghezza dei campioni registrati. Mentre $a(w_i)$ è l'ampiezza della componente in frequenza w_i . La variazione $\Delta w = w_{i+1} - w_i$ è la risoluzione selezionata, mentre N è il numero di componenti in frequenza.

$F(w)$ rappresenta le firme nel dominio in frequenza del motore sano, mentre nel caso delle due condizioni di malfunzionamento (barre rotte e cuscinetto rotto) si ha:

$$F_{br}(w) = (a_{br}(w_1)a_{br}(w_2) ... a_{br}(w_N))$$

$$F_{bc}(w) = (a_{bc}(w_1)a_{bc}(w_2) ... a_{bc}(w_N))$$

Tutte e tre le equazioni possono essere riunite nella seguente matrice:

$$F_{signals} = \begin{pmatrix} A_b & 0 & 0 \\ A_b & A_{br} & 0 \\ A_b & 0 & A_{bc} \end{pmatrix}$$

Da notare che le caratteristiche in frequenza dei motori sani e guasti sono rappresentati dalle ampiezza delle componenti in frequenza piuttosto che dalle componenti in frequenza stesse. Questo indica che le componenti in frequenza sono costanti e di conseguenza solo le ampiezze possono essere utilizzate per il rilevamento dei guasti. Dall'analisi delle componenti indipendenti si ha:

$$F_{signals} = A \cdot IC_s$$

dove IC_s è il set di vettori che rappresenta le componenti indipendenti e A è la matrice costante che vogliamo ricavare. La matrice IC_s è:

$$IC_x = \begin{pmatrix} IC_1 \\ IC_2 \\ IC_3 \end{pmatrix} = \begin{pmatrix} ic_{11} & ic_{12} & \dots & ic_{1N} \\ ic_{21} & ic_{22} & \dots & ic_{2N} \\ ic_{31} & ic_{32} & \dots & ic_{3N} \end{pmatrix}$$

La matrice appena menzionata e la matrice $F_{signals}$ contribuiscono a ricavare le *FFT-ICA features*. In altre parole, tutta l'informazione sull'ampiezza delle componenti in frequenza nelle firme in frequenza $F_{signals}$ contribuiscono a ricavare le *FFT-ICA features*.

Gli elementi della matrice $F_{signals}$ possono essere convertiti in una feature di dimensione M dalla matrice delle componenti indipendenti:

$$(F_Feature_{i1} \quad F_Feature_{i2} \quad \dots \quad F_Feature_{iM}) = F_Signal_i \cdot \begin{pmatrix} ic_{11} & ic_{12} & \dots & ic_{1N} \\ ic_{21} & ic_{22} & \dots & ic_{2N} \\ \dots & \dots & \dots & \dots \\ ic_{M1} & ic_{M2} & \dots & ic_{MN} \end{pmatrix}^T$$

Ogni elemento della matrice è una *FFT-ICA feature*. In realtà si è ridotta la dimensionalità del problema, passando da una matrice di dimensione N ad una dimensione M.

Ognuna di queste componenti M-dimensionali del set di training proveniente dai motori sani e guasti viene utilizzata per determinare le features che andranno a popolare il database.

Il database ottenuto è il seguente:

$$F_signals = \begin{pmatrix} A_b(1,1) & 0 & 0 \\ A_{br}(1,1) & A_{br}(1,1) & 0 \\ A_{bc}(1,1) & 0 & A_{bc}(1,1) \\ \vdots & \vdots & \vdots \\ A_b(1,2) & 0 & 0 \\ A_{br}(1,2) & A_{br}(1,2) & 0 \\ A_{bc}(1,2) & 0 & A_{bc}(1,2) \\ \vdots & \vdots & \vdots \\ A_b(1,L) & 0 & 0 \\ A_{br}(1,L) & A_{br}(1,L) & 0 \\ A_{bc}(1,L) & 0 & A_{bc}(1,L) \\ \vdots & \vdots & \vdots \\ A_b(I,L) & 0 & 0 \\ A_{br}(I,L) & A_{br}(I,L) & 0 \\ A_{bc}(I,L) & 0 & A_{bc}(I,L) \end{pmatrix}$$

dove gli elementi A sono indicati da:

1. b=Healthy;
2. be= bearing faults;
3. br= broken bar;
4. una coppia di indici (i,j) in cui i indica le diverse frequenza dell'inverter e j indica i diversi livello di carico;

I risultati della simulazione si sono dimostrati altamente efficienti portando a una diagnosi del motore a induzione.

Ci sono alcuni problemi importanti che non sono stati studiati. Innanzitutto, in questa trattazione sono stati considerati solo due tipi di guasti che possono affliggere il motore. In realtà esistono altri tipi di errore che dovrebbero essere studiati e non si è approfondita l'analisi dell'entità del guasto (il suo rischio). Come già notato in precedenza, la tecnica richiede l'estrazione delle features in frequenza. Da un punto di vista applicativo, però, non sempre è possibile disporre delle correnti di statore e, pertanto, metodi più versatili sono preferibili.

2.3 Analisi Vibrazionale

L'analisi vibrazionale permette di rilevare le anomalie ai macchinari e pianificare interventi di manutenzione primache si verifichi il guasto, facendo affidamento su misure prese esternamente alla macchina rotante.

Le metodologie di analisi delle vibrazioni sono molteplici e possono essere eseguite in continuo o ad intervalli programmati. Nel primo caso si provvede all'installazione permanente di un apparato di misurazione direttamente alla macchina, i cui dati possono essere raccolti online (soluzione scelta per monitorare le vibrazioni dei motori per il progetto sostenuto) o da remoto. Nel caso di misurazioni programmate viene utilizzata strumentazione portatile.

Al termine dell'analisi vibrazionale vengono prodotti rapporti diagnostici dettagliati; la classificazione della gravità dei guasti su quattro livelli consente di assegnare priorità agli interventi di manutenzione riducendo tempo e costi.

Stando a quanto riportato in [9], l'analisi vibrazionale necessita di una vasta conoscenza riguardante la tipologia di vibrazione, la sorgente e la causa della vibrazione stessa. Per determinare quali possano essere i punti deboli del motore, che causano tali problematiche, si utilizzando degli accelerometri che provvederanno a restituire informazioni necessarie all'analisi del segnale vibrazionale restituito dal motore. Le vibrazioni sono caratterizzate da frequenza, che mostra la velocità dell'oscillazione, e ampiezza, che rappresenta la forza della vibrazione. Queste due grandezze sono quelle che conferiscono le informazioni per identificare la radice della vibrazione. La misurazione del segnale vibrazionale viene svolta posizionando opportunamente l'accelerometro sul motore (nel caso specifico in direzione assiale e radiale rispetto all'albero motore) per un intervallo di tempo considerevole (indicativamente 15 minuti). Per avere

un'analisi completa delle vibrazioni è necessario acquisire i dati ed elaborarli sia nel dominio del tempo che in quello della frequenza.

Seguendo la metodologia appena citata, nei capitoli successivi verranno approfondite le modalità online e offline dell'implementazione di tale modalità di diagnosi.

3. STRUMENTAZIONE E METODI

Ai fini della progettazione del sistema di acquisizione e diagnosi sono stati utilizzati molteplici strumenti hardware e software, utili per garantire diverse soluzioni di progettazione. Facendo una distinzione tra strumenti software e hardware, si ottiene la seguente lista:

- **PLC BECKHOFF:** Il PLC, acronimo di Programmable Logic Controller che tradotto in italiano significa Controllore a Logica Programmabile, è l'elemento base del sistema di controllo di macchine e processi industriali. Lo si può paragonare ad un computer dotato di circuiti, ovvero interfacce ingresso/uscita, capaci di dialogare con dispositivi che possono essere pulsanti, sensori, azionamenti e apparecchiature elettroniche di qualsiasi tipo.



Figura 5: PLC Beckhoff

- **TWINCAT 3:** È l'ambiente di sviluppo legato al marchio Beckhoff che permette di interfacciarsi con il PLC a livello software. I componenti di ingegneria di TwinCAT 3 consentono la configurazione, la programmazione e il debug delle applicazioni. Il runtime di TwinCAT 3 è costituito da ulteriori componenti: componenti e funzioni di base. I componenti di base possono essere ampliati con funzioni. Il software di automazione TwinCAT 3 su piattaforma Visual Studio integra il controllo Real Time con funzioni PLC, NC e CNC, Robotics, HMI, Measurement Technology, analytics, safety, Connectivity, Vision in un singolo pacchetto. TwinCAT 3 supporta oltre al linguaggio IEC61131-3 anche C/C++ MATLAB/Simulink, Safety C/FDB.

- **Accelerometri:** è stato utilizzato un accelerometro piezoelettrico in modo che riesca a rilevare esclusivamente le vibrazioni (variazione del segnale) del motore, restituendo zero nel caso in cui il valore del segnale rimanga costante. Il modelli utilizzati per la misurazione sono DYTRAN 3055D2 e un PCB.

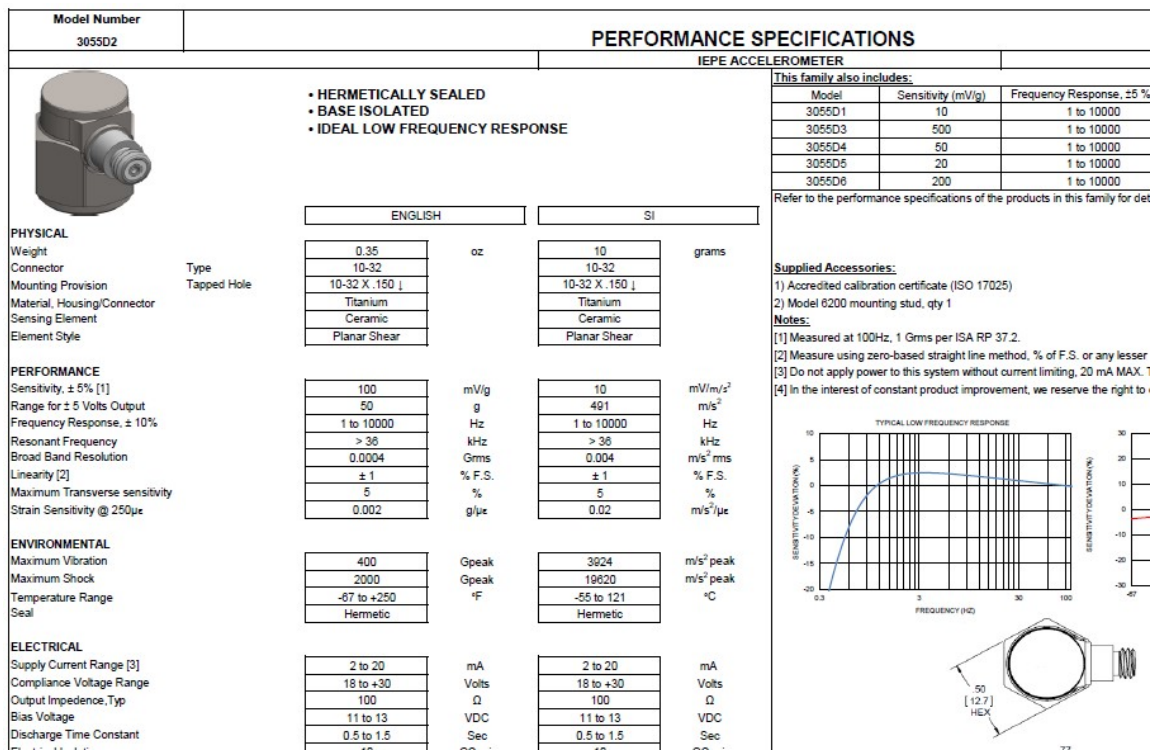


Figura 6:scheda tecnica dell'accelerometro

- **LabView:** LabVIEW (Laboratory Virtual Instrumentation Engineering Workbench) è l'ambiente di sviluppo integrato per il linguaggio di programmazione visuale di National Instruments. La definizione di strutture dati ed algoritmi avviene con icone e altri oggetti grafici, ognuno dei quali incapsula funzioni diverse, uniti da linee di collegamento (wire), in modo da formare una sorta di diagramma di flusso. Un programma creato in LabView prende il nome di VI ed è costituito da:
 - Il pannello frontale;
 - Lo schema a blocchi;
 - Il riquadro connettori;

- **MATLAB:** MATLAB (Matrix Laboratory) è un ambiente per il calcolo numerico e l'analisi statistica scritto in C, che comprende anche l'omonimo linguaggio di programmazione creato dalla *MathWorks*. MATLAB consente di manipolare matrici, visualizzare funzioni e dati, implementare algoritmi, creare interfacce utente, e interfacciarsi con altri programmi. Nel caso specifico di studio viene utilizzato un tool di Matlab, il *Diagnostic Feature Designer*, che permette di analizzare il segnale acquisito e operare una classificazione delle features estratte. Tale tool verrà esplorato nel dettaglio nei capitoli successivi;

L'uso degli strumenti appena riportati permette, dunque, di applicare l'analisi di vibrazione condotta mediante la determinazione di un set di *features* calcolate dal segnale sia nel dominio del tempo sia in quello delle frequenze così eliminare le componenti senza contenuto informativo. Il tutto viene svolto in relazione alla possibilità di poter ridurre la complessità di calcolo del processo di estrazione delle *features* di diagnosi, garantendo un'alta affidabilità da parte del sistema proposto.

3.1 Acquisizione

L'acquisizione dei dati vibrazionali avviene in maniera continuativa tramite l'impiego dell'accelerometro, posizionato direttamente sul motore del quale ci interessa osservare le condizioni operative. Per imbastire il banco di acquisizione si è fatto riferimento al funzionamento delle *Queued State Machines*, impostando la comunicazione come fosse una struttura *Producer – Consumer* con buffer.

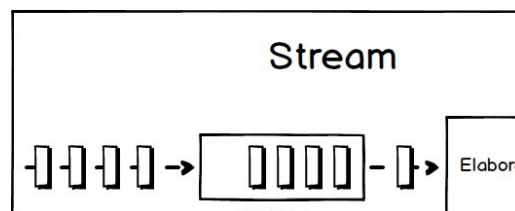


Figura 7: Schema rappresentativo del concetto di Queued States Machines

La struttura delle QSM permette di passare dati da una macchina ad un'altra (pattern produttore – consumatore). Il passaggio dei dati avviene attraverso l'utilizzo di un buffer, che può essere visto come un contenitore in cui vengono salvati i dati generati dal produttore e, successivamente, sempre dal buffer, possono essere prelevati quegli stessi dati dal consumatore, in modo che quest'ultimo possa processarli. La caratteristica fondamentale della struttura QSM è

che ogni macchina svolge il proprio compito indipendentemente dalle altre, garantendo un flusso di trasmissione dei dati continuo e che non comporti perdita di informazioni:

- Una macchina (O VI nel caso di Labview) si occupa di generare i dati e immagazzinarli in un buffer;
- L'altra macchina preleverà (uno alla volta o a gruppi) i dati posti sul buffer dal produttore e li processerà;

La clausola da rispettare è che il consumatore dovrà prelevare i dati ad una velocità (almeno) doppia rispetto a quella con la quale il produttore li posiziona sul buffer. Questo perché, generalmente, il consumatore deve operare diverse funzioni sui dati raccolti, e questo richiede alcuni ms.

Per concretizzare il concetto di QSM sono state sviluppate delle VI (Virtual Instruments, programmi in ambiente LabView) in LabView che rappresentano la trasmissione dati che si intende, poi, progettare in ambito diagnostico. Il programma sviluppato su Labview è rappresentato da 3 VI:

1. La VI produttore che, appunto, produce un segnale (e.g. sinusoidale) e ogni secondo inserisce nel buffer 1000 campioni della rampa. Generata a scopo di test, in questa prima versione non si considera la differenza di velocità tra PLC e PC (sarà approfondita in seguito) ed è sostituita nella realtà dalla macchina

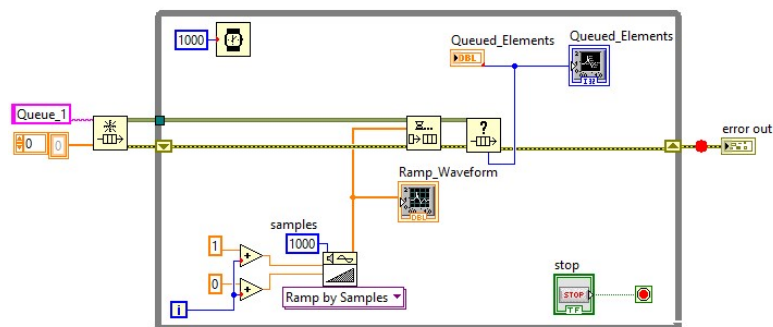


Figura 8:VI Produttore di una rampa puntuale crescente

2. La prima VI consumatore estrae gli elementi dal buffer_1 per poterli processare. In particolare è stato reso possibile sia estrarre un elemento alla volta sia estrarre gli elementi ogni K iterazioni. Le funzioni che vengono svolte sui dati raccolti sono il calcolo del massimo, media e minimo dei campioni raccolti e vengono inseriti in un secondo buffer, ovvero buffer_2, diverso dal primo.

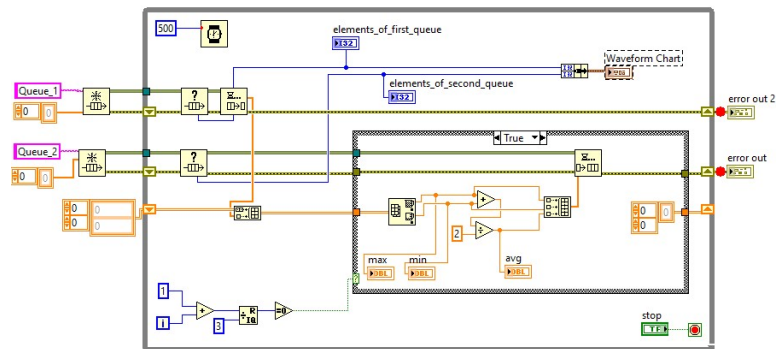


Figura 9:VI Consumatore che prende i punti della rampa caricati nel buffer_1, ne calcola massimo, minimo e media e inserisce i risultati nel buffer_2

3. La seconda VI consumatore prende il risultato del processamento della VI precedente dal buffer_2 e lo stampa su file di testo.

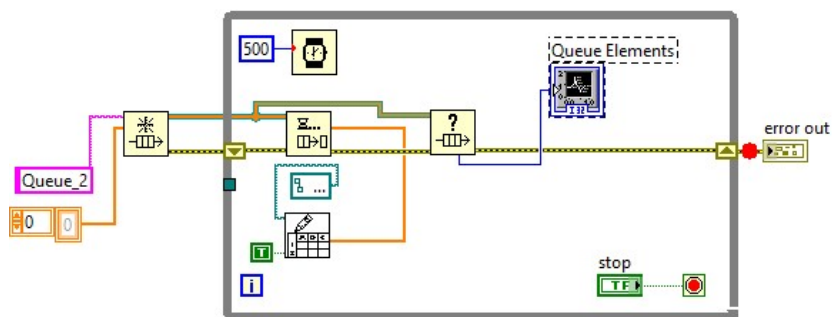


Figura 10: VI Consumatore che stampa il risultato su file di testo

Si riporta a titolo esemplificativo con un segnale sinusoidale (sia puro che viziato da *white noise*) sul quale è stata successivamente eseguita una FFT:

- Viene generata dalla prima VI una sinusoide a 60Hz con frequenza di campionamento di 1Khz

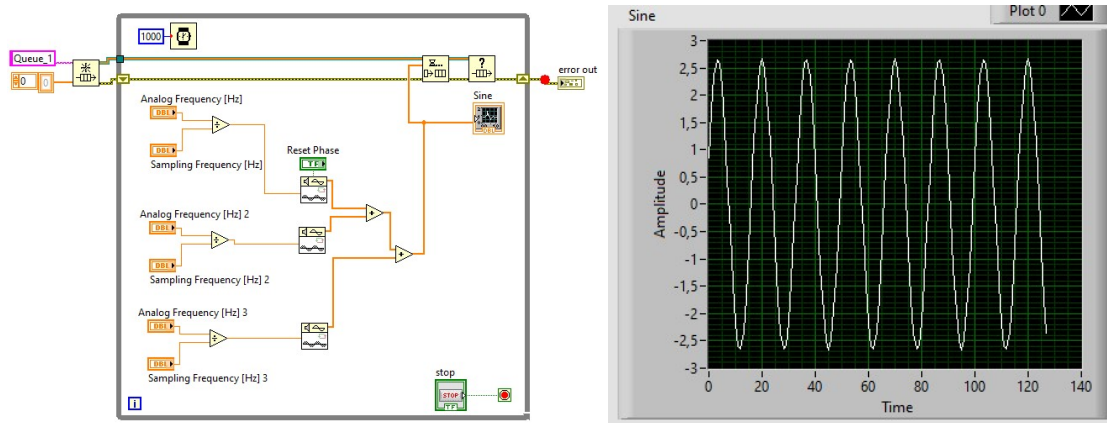


Figura 11: VI Produttore di una sinusoide

- FFT con finestra di campionamento scelta di 10000 campioni (10 secondi di campionamento)

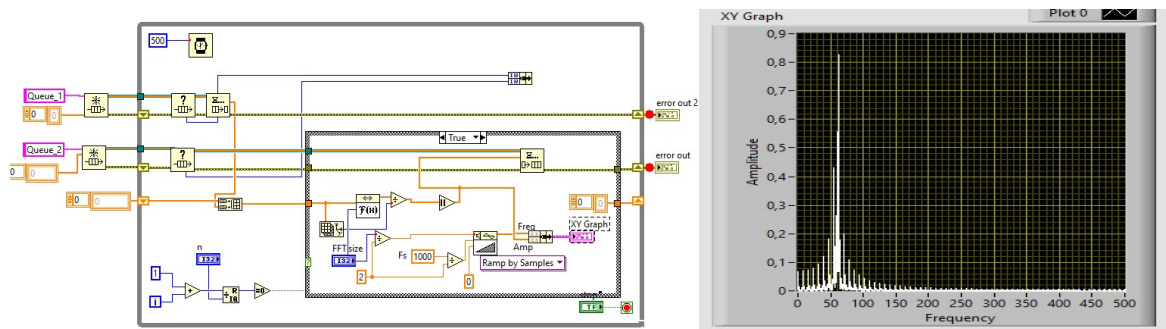


Figura 12: VI Consumatore che calcola la FFT della sinusoide

E' fondamentale riuscire a stabilire una comunicazione coerente tra PLC e PC. Il PC non opera perfettamente real-time (rispetto al tempo di acquisizione del PLC), quindi per non perdere pacchetti di dati fondamentali si utilizza un buffer che possa contenere i dati acquisiti dal PLC in modo da gestire meglio i tempi di acquisizione. Una soluzione può essere quella di accumulare dati sul buffer e poi inviarli, e nel contempo il PLC continua ad acquisire i dati e il PC continua ad elaborarli.

La velocità con quale avviene l'elaborazione PC deve essere incrementata (generalmente 3-4 secondi più veloce dell'acquisizione). Sono state realizzate delle VI in Labview per simulare quanto accade in un reale sistema PLC-PC:

- La prima VI è la generazione del segnale, non cambia nulla rispetto a quanto già visto nel caso precedente. In questo caso viene generata una rampa di punti (1-2-3...) e non viene mai fermata l'elaborazione. I punti vengono inseriti in un buffer che verrà utilizzato per comunicare con il secondo VI.

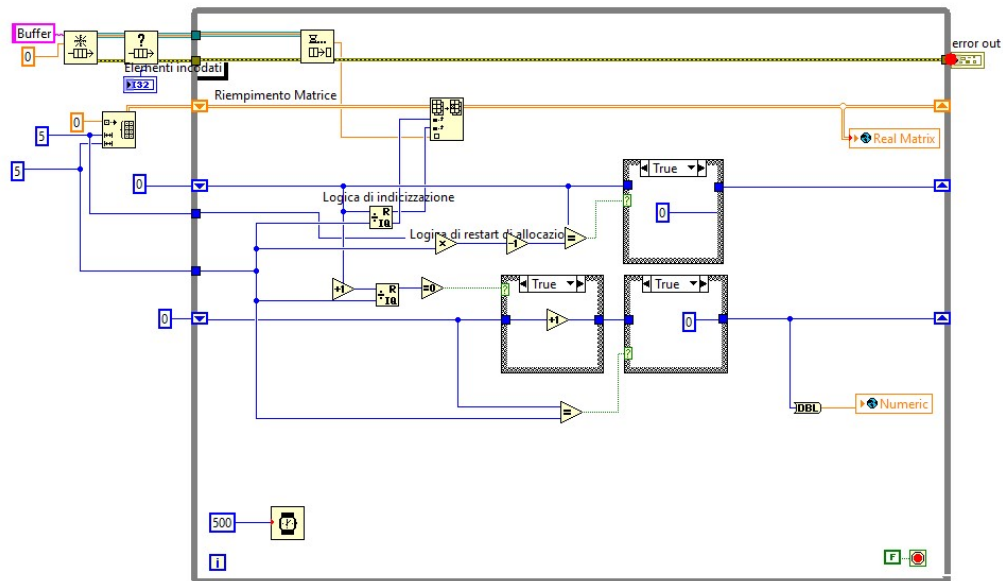


Figura 13:VI di generazione del segnale

- Il secondo VI serve a simulare la memoria del PLC. Nel caso in questione è stata scelta una frequenza di campionamento di 1kHz e sono stati scelti un numero di buffer interni pari a 5 e un numero pari a 5000 di campioni per ogni buffer. Dato che ogni campione è dato da 1000 elementi, quella che andiamo a progettare è a tutti gli effetti una matrice 5x5. In questa matrice verranno salvati i dati inviati sul buffer e un contatore andrà ad essere incrementato ogni volta che questo accade. Il contatore verrà letto dal PC per prelevare i dati: ogni volta che il clock ha un valore diverso dal precedente allora il PC preleva il dato e lo elabora. Questa lettura del clock avviene tramite un sistema di variabili globali proprio di Labview, e stesso discorso vale per la matrice generata: entrambe le grandezze non vengono mandate tramite buffer ma tramite comunicazione globale tra le VI.

delle features è ripetuta ad intervalli estremamente brevi in cui il processo può essere considerato stazionario, l'applicazione della FFT è stata ritenuta più che sufficiente per l'obiettivo prefissato.

Le features da estrarre, sia nel dominio del tempo che in quello della frequenza, sono individuati attraverso il *Diagnostic Feature Designer*[10]. Tale strumento è impiegato in fase di analisi, ovvero restituisce le feature migliori da estrarre relativamente ad una base di dati disponibile. Una volta individuate le features migliori, esse saranno poi calcolate e dati in ingresso alla fase di diagnosi basata sui classificatori.

3.2.1 Features nel dominio del tempo

Le features estratte dal segnale di vibrazione nel dominio del tempo sono indicatori statistiche che forniscono informazioni di varia natura sulla distribuzione di dati considerata. Questi indicatori sono ampiamente sfruttati in diversi ambiti applicativi per il processamento del segnale finalizzato poi ad una classificazione, in ambito diagnostico offrono però una limitata capacità d'identificazione del guasto; sono pertanto stati utilizzate delle ulteriori features calcolate nel dominio delle frequenze.

Mean	$T_m = 1/n \sum_{i=1}^n x_i$	Shape Factor	$T_{sf} = \frac{T_{rms}}{\bar{x}}$
Root Mean Square	$T_{rms} = \sqrt{1/n \sum_{i=1}^n x_i^2}$	Crest Factor	$T_{cf} = \frac{x_{max}}{x_{rms}}$
Root	$T_r = 1/n \sum_{i=1}^n \sqrt{ x_i }^2$	Impulse Factor	$T_{if} = \frac{T_{max}}{\bar{x}}$
Standard Deviation	$T_m = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2}$	Clearance Factor	$T_{clf} = \frac{x_{max}}{x_r}$
Skewness	$T_{sk} = \frac{\sum_{i=1}^n (x_i - \bar{x})^3}{(n-1)T_{sd}^3}$	Skewness factor	$T_{skf} = \frac{T_{sk}}{T_{rms}^3}$
Kurtois	$T_{ku} = \frac{\sum_{i=1}^n (x_i - \bar{x})^4}{(n-1)T_{sd}^4}$	Kurtois Factor	$T_{kuf} = \frac{T_{ku}}{T_{rms}^4}$
THD (Total Harmonic Distorsion)	$THD = 10 \log \left(\frac{V_h^2}{V_f^2} \right)$	SNR (Signal – to – noise – ratio)	$SNR = 10 \log \left(\frac{V_f^2}{V_n^2} \right)$

SINAD (Signal – to -noise and distorsion ratio)	$THD = 10 \log \left(\frac{\overline{V_f^2}}{\overline{V_n^2 + V_h^2}} \right)$		
---	--	--	--

In tabella sono presenti i seguenti valori:

- V_f : potenza totale nella componente di frequenza fondamentale del segnale testato;
- V_h : potenza totale dell'armonica (generalmente seconda e decima) della componente in frequenza del segnale testato;
- V_n : potenza totale nelle componenti in frequenza non fondamentale e non relativa ad un'armonica in frequenza;
- x_i : è il segnale in serie temporale per $i=1,2,\dots,n$;
- n : è il numero di data-points;
- x_{max} : è il massimo di x_i ;
- $\bar{x} = \frac{1}{n} \sum_{i=1}^n |x_i|$: è il valore medio assoluto di x_i ;

3.2.2 Features nel dominio della frequenza

Il calcolo di grandezze caratteristiche del segnale nel dominio delle frequenze permettedi andare ad individuare la componente della macchina che ha subito un guasto, cioè quell' elemento per cui la frequenza caratteristica ricade all'interno del range di frequenze considerate e per cui vengono rilevate delle anomalie nel valore e/o nella dispersione dei dati acquisiti.

Lo strumento basilare utilizzato è la trasformata veloce di Fourier, algoritmo ottimizzato per il calcolo della trasformata discreta di Fourier. L'applicazione di questo operatore ad un segnale reale discreto costituito da N campioni nel tempo fornisce come output N valori complessi, i coefficienti della trasformata, ognuno dei quali rappresenta il contributo del segnale in un range di frequenze d'ampiezza pari a $\frac{fs}{N}$ in cui fs rappresenta la frequenza a cui il segnale è stato campionato. I coefficienti della trasformata forniscono informazioni relative al range di frequenze $[0Hz; \frac{fs}{2}Hz]$, per un segnale composto da N campioni si hanno dunque N+1 componenti.

Le feature nel dominio della frequenza scelte tramite il *Diagnostic Feature Designer* sono:

- *Band Power*;
- *Peak Frequency*;
- *Peak Magnitude*;

3.3 Selezione delle features

Anche in questo è stato sfruttato il *Diagnostic Feature Designer* per scegliere quali features sono più significative di altre per, poi, effettuare la classificazione. Le metriche proposte dal tool di classificazione dei guasti sono le seguenti:

- **T – Test**: calcolo del valore assoluto fra due campioni con stima della varianza aggregata;
- **Entropy**: classifica le features secondo l'entropia relativa utilizzando un criterio di valutazione indipendente;
- **Bhattacharyya**: minimizza l'errore di classificazione;
- **ROC**: classifica le feature in base al valore dell'area tra la caratteristica di funzionamento del ricevitore (Receiver Operating Characteristic) e la pendenza del classificatore;
- **Wilcoxon**: classifica le features calcolando il valore assoluto della statistica-U standardizzata di un un test di Wilcoxon;

Le feature vengono sottoposte ad algoritmi di *ranking*, che posizionano le feature in ordine decrescente, partendo da quella più efficace nella separazione dei dati in funzione delle condizioni di funzionamento. E' possibile scegliere tra due categorie di metodi di ranking:

1. **Metodi a due classi**: utilizzati nel caso in cui la variabile di condizionamento ha solo due etichette (healthy, faulty);
2. **Metodi Multiclasse**: se la variabile di condizionamento ha più di due etichette(healthy, faulty_1, faulty_2);

Dato che sono stati utilizzati solamente dati relativi a due condizioni di funzionamento (motore sano o guasto), è sufficiente adottare le metriche basate su metodi Two-Classes.

Tra le metriche proposte, il T-Test è quello che restituisce i risultati più affidabili senza necessitare di un elevato sforzo computazionale. In questo modo viene ottenuto un set di features di dimensione ridotta rispetto a quello iniziale, migliorando le prestazioni dei predittori, rendendoli più veloci e meno costosi.

3.4 Classificazione

Questo strumento è stato implementato principalmente con il doppio scopo di comprendere quanto effettivamente le features selezionate fornissero informazione utile per discriminare tra diverse condizioni di funzionamento e di provare ad implementare un modulo software per la

diagnosi che non richiedesse elevate capacità di calcolo e di memoria e che prevedesse un processo d'apprendimento veloce.

Nello specifico, la classificazione viene svolta tramite il tool di Matlab già precedentemente menzionato. Quest'ultimo, a partire dalle features estratte, permette di effettuare una loro classificazione in funzione della qualità con la quale queste riescono a discriminare le diverse condizioni di funzionamento.

Il classificatore messo a disposizione del tool di Matlab cerca di trovare quali sono i **predittori** che meglio separano le classi tramite la rappresentazione dei predittori su un sistema di assi cartesiani (X: primo predittore, Y: secondo predittore). I predittori vengono scelti tra le features selezionate precedentemente.

Sempre all'interno del tool di predizione è possibile svolgere una PCA (Principal Component Analysis) che riduce ulteriormente la dimensionalità dello spazio delle features. E' consigliabile scegliere questa strada per effettuare la classificazione perché riducendo lo spazio delle features si previene il problema dell'overfitting, spesso causato dalla ridondanza dei dati all'interno del set di apprendimento.

4 IMPLEMENTAZIONE

Dopo aver illustrato, a livello teorico, come si è deciso di impostare lo studio del segnale vibrazionale di un motore asincrono monofase, avendo evidenziato nel precedente capitolo i passi specifici che bisogna affrontare per classificare i guasti riscontrati sul motore, bisogna concretizzare il sistema di diagnosi mediante l'utilizzo degli strumenti menzionati precedentemente.

Nel presente capitolo si vedrà come è stato effettivamente costruito il sistema di diagnosi, a partire da come viene impostato il collegamento tra i componenti hardware fino a mostrare come lavorano in sincronia i software utilizzati per la diagnosi dei guasti: in prima istanza si vedrà come è stato codificato l'algoritmo che permette di estrarre le feature direttamente in ambiente Beckhoff su un segnale simulato, successivamente si discuterà su come è stata implementata l'analisi di dati offline.

4.1 Implementazione Hardware

Per realizzare il sistema diagnostico è necessario l'impiego dei seguenti strumenti:

- Motore asincrono monofase;
- Accelerometro;
- Modulo di acquisizione per PLC;
- PC;

Per il progetto sono stati resi disponibili dei motori per cappe aspiranti, tre dei quali erano afflitti da guasti di diversa natura che compromettevano la corretta rotazione del motore, introducendo una differenza sostanziale nel segnale vibrazionale rilevato per ognuno di essi.

L'accelerometro è collegato al PLC tramite un modulo di acquisizione compatibile, nel dettaglio il CX-3B30B2, che permette la lettura dei dati analogici trasformandoli in digitale per il PLC. Per la raccolta dei dati sono state scelte due modalità di acquisizione: acquisizione in direzione radiale rispetto all'albero motore e una in direzione assiale, lungo la direzione dell'albero motore.

Infine i dati vengono elaborati tramite Twincat 3, facendo però attenzione alla forma con la quale questi dati vengono mostrati. Nel caso in esame la misura viene mostrata in **punti**. A questo punto bisogna scalare la misura, svolgendo due semplici passaggi matematici, per ottenere la misura in m/s^2 :

- Inizialmente bisogna scalare la misura secondo la risoluzione del sensore utilizzato. Nel nostro caso la risoluzione è pari a $640 \frac{\mu V}{\text{punti}}$. Bisogna moltiplicare il valore mostrato su Twincat per il valore della risoluzione, così da ottenere la misura in V;

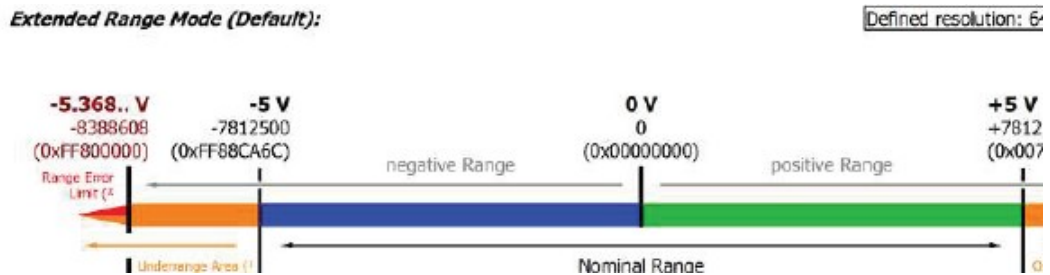


Figura 15: Range di risoluzione del modulo di acquisizione collegato al PLC

- Bisogna moltiplicare la misura precedentemente ottenuta per l'inverso della sensibilità dell'accelerometro, così da ottenere, come misura, l'accelerazione.

4.2 Simulazione

L'obiettivo del progetto è quello di riuscire ad implementare un sistema diagnostico che usufruisca il meno possibile dell'utilizzo della potenza computazionale del PC, lasciando a quest'ultimo esclusivamente il compito di dover classificare i guasti in base alle feature estratte. E' necessario sviluppare gli algoritmi in Twincat 3 dato che il PLC, sul quale è installato il modulo di acquisizione connesso all'accelerometro, funge da mini PC, sostituendo, in funzionalità quali l'elaborazione dei dati vibrazionali e la loro acquisizione, il PC esterno impiegato solitamente per effettuare una *fault diagnosis*. A tale proposito bisogna elaborare il segnale ed estrarre le feature online direttamente in ambiente Twincat 3 e passare, tramite streaming dati continuo, al PC direttamente un buffer contenente le feature di interesse.

Per realizzare quanto appena dichiarato si parte da quanto descritto nelle VI presentate nel capitolo precedente, in cui è stato generato un segnale sinusoidale. Nello specifico si vuole replicare quanto implementato in LabView in Twincat3, dato che è l'ambiente di sviluppo specifico per il PLC. Quindi viene generato un segnale sinusoidale simulato tramite un algoritmo appositamente realizzato in ambiente Twincat 3 e in cui è stato codificato l'algoritmo che permette la pre-elaborazione del segnale e, successivamente, la sua elaborazione. In particolare si è deciso di calcolare la FFT del segnale simulato, in modo da determinare solo le componenti

fondamentali in frequenza dello spettro del segnale, e solo una volta ottenuta la FFT ottimale si è passati al calcolo delle features in frequenza (RMS, Peak, Peak to Peak).

Prima di procedere nel dettaglio di come è stato costruito l'algoritmo per l'estrazione delle feature del segnale simulato si mostrano gli obiettivi da perseguire in questa elaborazione:

- Generare un segnale sinusoidale con frequenza di campionamento pari a 1KHz;
- Costruire una matrice rappresentativa della totalità dei canali di acquisizione a disposizione del PLC, in cui verranno raccolti i dati;
- Creare un buffer che raccolga i dati immagazzinati nella matrice e che venga utilizzato come "contenitore" dal quale attingere i dati per le successive elaborazioni;
- Costruire le funzioni per il calcolo delle features e della Fast Fourier Transform;

In Twincat, per definire il codice, vengono definite le POUe quali tipologie si dividono in programmi, blocchi di funzione e funzioni. Nel caso specifico sono stati utilizzati solamente i programmi e come linguaggio di programmazione è stato scelto in linguaggio strutturato.

Prima di analizzare le singole POUe è necessario mostrare le variabili definite ed utilizzate nei differenti algoritmi. Le variabili che vengono utilizzate dal MAIN e i suoi sottoprogrammi prendono il nome di variabili globali e sono definite all'interno di un'apposita sezione. Di seguito le variabili globali utilizzate:

VAR_GLOBAL CONSTANT

```
N      : INT:= 128;
                                     // Corrisponde alla dimensione di un singolo buffer

M      : INT:= 64;
                                     // Corrisponde al numero di buffer

L: INT:= 16;                          // Numero di array concatenati per generare il vettore di
                                     preelaborazione

cOversamples : UDINT := 1;           // numero di campioni

cFFTLenght   : UDINT:= 2048;// lunghezza FFT

cBufferLength: UDINT:=cFFTLenght;   // lunghezza buffer di calcolo FFT

cFFTResult: UDINT := cFFTLenght/2+1;// risultato dello spettro in frequenza
```

END_VAR

VAR_GLOBAL

```
//Variabili generali

Count_Row      : INT:=0;    // contatore del numero di righe riempite

funct          : LREAL;    //valore della funzione seno ad ogni istante di tempo

Amplitude      : LREAL:= 1;    //Ampiezza della sinusoide

Data           : LREAL:= 0;    //dati generati

//Variabili necessarie per la pre-elaborazione

SwitchAT %Q*: ARRAY[0..0] OF LREAL;           //switch per spostarsi da un buffer di
elaborazione all'altro. Viene indicato come variabile di output da portare su PC per
l'elaborazione

AcqTable_sin   : ARRAY[0..M-1] OF ARRAY [0..N-1] OF LREAL; //Matrice di
acquisizione (rappresenta il PLC di dimensioni M canali di lunghezza N)

Elaboration_Buffer_1      AT %Q*: ARRAY[0..N*L-1] OF LREAL; //primo buffer di
elaborazione. Viene indicato come variabile di output da portare su PC per
l'elaborazione

Elaboration_Buffer_2      AT %Q*: ARRAY[0..N*L-1] OF LREAL;; //secondo buffer di
elaborazione. Viene indicato come variabile di output da portare su PC per
l'elaborazione

Buffer_locale      :ARRAY[0..N*L-1] OF LREAL; //Buffer utilizzato per il calcolo delle
features

ActRow            : INT:= 0; //Rappresenta l'indice della riga attualmente completata

//Variabili utilizzate per l'analisi in frequenza

rms               : LREAL;    // Variabile rms

re               : INT;      // esprime la parte reale del FFT

fSampleTaskCycleTime: LREAL;    // velocità di esecuzione del task in secondi

fbGetTaskIdx      : GETCURTASKINDEX;
```

END_VAR

Analizzando più nel dettaglio le singole POU, si hanno:

1. **MAIN:** Il *MAIN program* è direttamente legato al task di esecuzione del PLC e grazie al suo avvio viene avviato qualunque altro programma legato al medesimo task. Per far sì che i vari sottoprogrammi possano essere eseguiti una volta avviato il MAIN è necessario richiamare questi ultimi all'interno del MAIN stesso.

All'interno del MAIN è stata definita anche l'operazione di generazione di dati puntuali che incrementano ad ogni ciclo del task. Sarà utile per la generazione del segnale sinusoidale.

```
PROGRAM MAIN
```

```
Data := Data+1;  
FUN_GEN();  
ACQ();  
PRE_ELAB();  
PRE_ELAB_1();  
FFT();  
FREQ_FEATURES();
```

2. **FUN_GEN:** in questo program viene realizzato il segnale sinusoidale, di frequenza pari a 100 HZ con campionamento di frequenza pari a 1 KHz.

```
PROGRAM FUN_GEN
```

```
VAR CONSTANT
```

```
    pi : REAL := 3.1459265358;
```

```
END_VAR
```

```
VAR
```

```
    f : LREAL := 100;
```

```
    fs : LREAL := 1000;
```

```
END_VAR
```

```
funct := Amplitude*SIN(2*pi*(f/fs)*Data);
```

La variabile Data definita nel MAIN serve a rendere incrementale la generazione della funzione sinusoidale (funge da iteratore). Il segnale generato, in un intervallo di 10 secondi, si presenta come nella figura seguente:

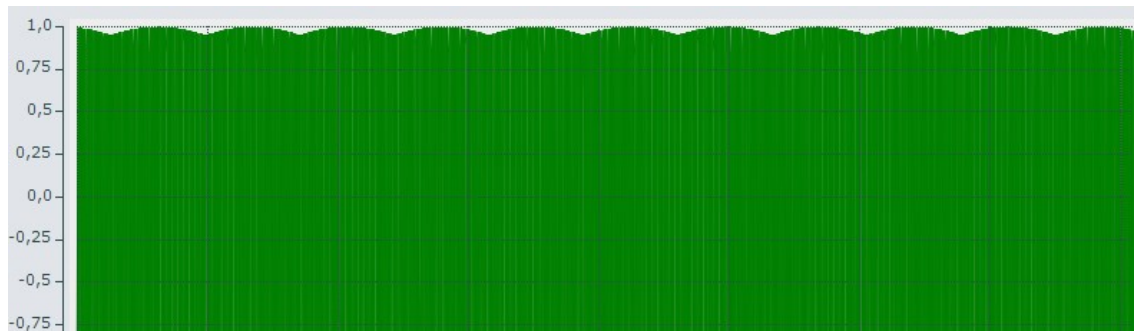


Figura 16: Sinusoide simulata su Twincat 3

- 3. ACQ:** qui si simula l'acquisizione del flusso continuo di dati in ingresso. Il tutto viene gestito in modo da garantire la compatibilità dimensionale tra le grandezze in gioco.

PROGRAM ACQ

VAR

Temp : ARRAY [0..N-1] OF LREAL;

i : INT:=0;

k: INT:=0;

END_VAR

Temp[k]:=funct;

k:=k+1;

IF k = N THEN

AcqTable_sin[i] := Temp;

k := 0;

i :=i+1;

ActRow := i;

Count_Row:=Count_Row+1;

END_IF

IF i = M THEN

i := 0;

END_IF

Viene inizializzato un contatore K pari a zero, il quale indica la posizione all'interno del vettore *temp* ausiliario in cui viene inserito il rispettivo elemento *funct*. L'incremento di Data e K è simultaneo, essendo che entrambi iterano alla

stessa velocità definita dal task, così da avere una corrispondenza uno a uno tra posto nel vettore ed elemento inserito.

L'inserimento dei dati nel vettore continua finché K non risulta pari alla dimensione della riga della matrice di acquisizione. Successivamente il contatore K viene impostato pari a zero. Prima che K venga resettato, il vettore viene assegnato alla prima riga della matrice e, successivamente viene incrementato il contatore i delle righe. Ad ogni iterazione viene aggiornato il vettore con degli elementi nuovi e viene fatto scattare il conteggio della riga della matrice di uno step, quindi si passa alla riga i+1, i+2, ecc...fino a che la matrice viene riempita completamente. Una volta che la matrice è piena viene resettato il contatore delle righe e il procedimento riprende da capo andando a sovrascrivere i dati memorizzati dal vecchio ciclo.

- 4. PRE_ELAB:** Per la pre elaborazione del segnale acquisito viene svolta in una POU dedicata. Questo procedimento è necessario per spostare i dati presenti nella matrice di acquisizione all'interno del vettore di pre elaborazione, il quale verrà successivamente utilizzato per passare i dati al suo interno dal PLC al PC.

```
PROGRAM PRE_ELAB
```

```
VAR
```

```
counter                :INT:=0; //Contatore per l'incremento degli elementi
                        del vettore di elaborazione
i                       :INT:=0; //Indice riga della tabella di acquisizione
j                       :INT:=0; //Indice della colonna della tabella di
                        acquisizione
a                       :INT:=0; //indice del primo array di elaborazione
b                       :INT:=0; //indice del secondo array di elaborazione
c                       :INT:=0;
```

```
END_VAR
```

```
IF Count_Row > 1 THEN
```

```
    counter:=counter+1;
```

```
        IF switch[c] = 0 THEN
```

```
            IF counter = N-1 THEN
```

```
                FOR j := 0 TO N-1 BY 1 DO
```

```

Elaboration_Buffer_1[a] := AcqTable_sin[i][j];
a:=a+1;
END_FOR
END_IF
END_IF
IF switch[c] = 1 THEN
    IF counter = N-1 THEN
        FORj := 0 TO N-1 BY 1 DO
            Elaboration_Buffer_2[b] := AcqTable_sin[i][j];
            b:=b+1;
        END_FOR
    END_IF
END_IF
i:=i+1;
END_IF
IF a = L*N THEN
    a:=0;
    switch[c] := 1;
END_IF
IF b = L*N THEN
    b:=0;
    switch[c] := 0;
END_IF
IF i = M THEN
    i:=0;
END_IF
IF counter=N-1 THEN
    counter:=0;
END_IF

```

Analizzando l’algoritmo, sono stati creati due vettori di elaborazione, *Elaboration_buffer_1* ed *Elaboration_buffer_2*, necessari per gestire il “ping-pong” tra i due vettori: viene riempito il primo vettore con l’implementazione di un ciclo for in cui, ad ogni iterazione, viene copiata una riga della matrice di acquisizione all’interno del vettore di pre elaborazione. Non appena è pieno, viene

modificato il valore di switch da 0 a 1. Appena lo switch viene cambiato viene fatto riempire il secondo vettore e il programma di streaming su PC acquisisce il primo vettore pieno. Quando anche il secondo è pieno allora lo switch viene riportato al valore 0, il programma di streaming preleva i dati dal vettore e viene sovrascritto l'*Elaboration_buffer_1e* il procedimento si ripete per tutto il periodo di acquisizione.

Si è deciso di gestire la trasmissione dei dati in questa modalità così da evitare che si perdessero dei dati, dato che la velocità con la quale lavorano il PLC e il PC è sensibilmente differente (il PC va circa il doppio più veloce rispetto al PLC). Se fosse stato utilizzato un solo vettore ci sarebbe stato il rischio per cui il PC avrebbe acquisito più volte gli stessi dati.

5. **PRE_ELAB_1:** Questo programma ha lo stesso principio di funzionamento del precedente, la differenza sta nel fatto che in questo caso viene gestito un solo vettore di elaborazione (*Buffer_locale*) invece di due. Questo vettore sarà utilizzato successivamente per il calcolo della FFT del segnale e per l'estrazione delle features.

```
PROGRAM PRE_ELAB_1
```

```
VAR
```

```
counter                :INT:=0; //Contatore per l'incremento degli elementi
                        del vettore di elaborazione
```

```
i                      :INT:=0; //Indice riga della tabella di acquisizione
```

```
j                      :INT:=0; //Indice della colonna della tabella di
                        acquisizione
```

```
a                      :INT:=0; //indice del primo array di elaborazione
```

```
END_VAR
```

```
IF Count_Row > 1 THEN
```

```
    counter:=counter+1;
```

```
    IF counter = N-1 THEN
```

```
        FOR j := 0 TO N-1 BY 1 DO
```

```
            Buffer_locale[a] := AcqTable_sin[i][j];
```

```
            a:=a+1;
```

```
        END_FOR
```

```
        i:=i+1;
```

```
END_IF
END_IF
```

```
IF a = L*N THEN
a:=0;
END_IF
```

6. FFT: Il calcolo della FFT è stato realizzato con la funzione codificata che si può trovare tra le librerie di Twincat. Il concetto con la quale è stata realizzata la funzione di creazione del FFT è basato sulla trasmissione dei dati internamente alla funzione stessa. Vengono utilizzate due funzioni, *sink* e *source*:

- Source: è la funzione che prende in ingresso il vettore di pre elaborazione e verrà inviato alla funzione di sink;
- Sink: restituisce in uscita il vettore risultante della trasformata, chiamato *SpectrumResult*;

Ottenuto il vettore dei risultati, viene calcolato il valore assoluto in relazione alla lunghezza della finestra scelta per il calcolo del FFT, ovvero la variabile *FFTLenght*. Il calcolo del valore assoluto dello spettro serve ad eliminare le frequenza a parte immaginaria negativa, ottenendo una FFT con solo parte reale e immaginaria positiva.

```
PROGRAM FFT
VAR CONSTANT
//Condizioni iniziali del FB_Source
  cInitSource : ST_MA_MultiArray_InitPars :=(eTypeCode:=
  eMA_TypeCode_LREAL, nDims := 1, aDimSizes := [cBufferLength]);

  cInitFFT: ST_CM_RealFFT_InitPars :=(nFFT_Length := cFFTLenght, bForward :=
  TRUE);
END_VAR
VAR
fbRealFFT: FB_CMA_RealFFT :=(stInitPars := cInitFFT, nOwnID := eID_FFT,
aDestIDs := [eID_Sink] );
nSample: UDINT;
SpectrumAbs : ARRAY[1..cFFTResult] OF LREAL;
```



```

(* Source buffers *)
fbSource: FB_CMA_Source :=(stInitPars := clnitSource,nOwnID := eID_Source,
aDestIDs := [eID_FFT]);          // Initialize source buffers
(* Result buffers *)
aSpectrumResult: ARRAY[1..cFFTResult] OF LCOMPLEX;
// localresultvariable<- from sink
fbSink:FB_CMA_Sink := (nOwnID := eID_Sink);// Initializesink buffers
(* post-processing of fft *)
nCountResults          :ULINT;// increments for everyfftcalculation
bCalculate              :BOOL;      // trueiffftcalculation ready
                          // magnitudespectrumvariable

(* max amplitudecalculatation *)
fMaxAmpl               : LREAL;
nIdxOfMaxAmpl          : UDINT;// index of max amplitude in spectrumresult
                          array

(* frequency calculation *)
fSampleRate            : LREAL;      // Hz   ( sample rate =
oversamplingfactor / cycle time )
fResolution            : LREAL;     // Hz/Idx
fFrequency             : LREAL;     // Hz
END_VAR

```

```

fbRealFFT.Call();
fbGetTaskIdx();
fSampleTaskCycleTime                                     :=
UDINT_TO_LREAL(TwinCAT_SystemInfoVarList._TaskInfo[fbGetTaskIdx.index].Cyc
leTime) / (10.0 * 1000.0 * 1000.0);
(* Call source to collect data *)
fbSource.Input1D(pDataIn          := ADR(Buffer_locale),
                 nDataInSize      := SIZEOF(Buffer_locale),
                 eElementType :=eMA_TypeCode_LREAL,
                 nWorkDim         := 0,
                 pStartIndex      := 0,
                 nOptionPars      :=cCMA_Option_MarkInterruption );

(* Pushresults to sink *)
fbSink.Output1D      (pDataOut          := ADR(aSpectrumResult),

```

```

        nDataOutSize := SIZEOF(aSpectrumResult),
        eElementType :=eMA_TypeCode_LCOMPLEX,
            nWorkDim      := 0,
            nElements     := 0,
            pStartIndex   := 0,
            nOptionPars   := 0,
            bNewResult    =>bCalculate );
(* post-processfftresult *)
IFbCalculateTHEN
    nCountResults := fbSink.nCntResults;
    // CalculateabsolutevalueasaSpectrumResultis of complextype
    FORnSample := 1 TOcFFTResultDO
        SpectrumAbs[nSample]:=SQRT(aSpectrumResult[nSample].re*
        aSpectrumResult[nSample].re+aSpectrumResult[nSample].im*
        SpectrumResult[nSample].im);
    // scale spectrum
        IFnSample = 1 THEN
            SpectrumAbs[nSample]:=SpectrumAbs[nSample]/cFFTLen
            gth; // scale DC value
        ELSE
            SpectrumAbs[nSample]:=
            2*SpectrumAbs[nSample]/cFFTLlength; // scale
            frequencies notequal to 0
        END_IF
    END_FOR;
    IF NOT Irealisnan(SpectrumAbs[1]) THEN
    // Calculate max amplitude in spectrum
    fMaxAmpl := 0;
        FORnSample:=1 TOcFFTResultDO
            IFSpectrumAbs[nSample] >fMaxAmplTHEN
                fMaxAmpl := SpectrumAbs[nSample];
                nIdxOfMaxAmpl := nSample;
            END_IF
        END_FOR
    // Calculate frequency for max amplitude[ frequency = index * (sample
    rate / FFT length) ]

```

```

fSampleRate := cOversamples / fSampleTaskCycleTime;
fResolution := fSampleRate / cFFTLenght;
fFrequency := (nIdxOfMaxAmpl-1) * fResolution;    // -1 to consider DC =
0*fResolution
END_IF

```

END_IF

Oltre alla trasformata di Fourier vengono anche calcolati l'ampiezza massima del segnale e la frequenza alla quale si trova, la frequenza di campionamento e la risoluzione. Di seguito è mostrato il risultato ottenuto dopo il calcolo della trasformata:

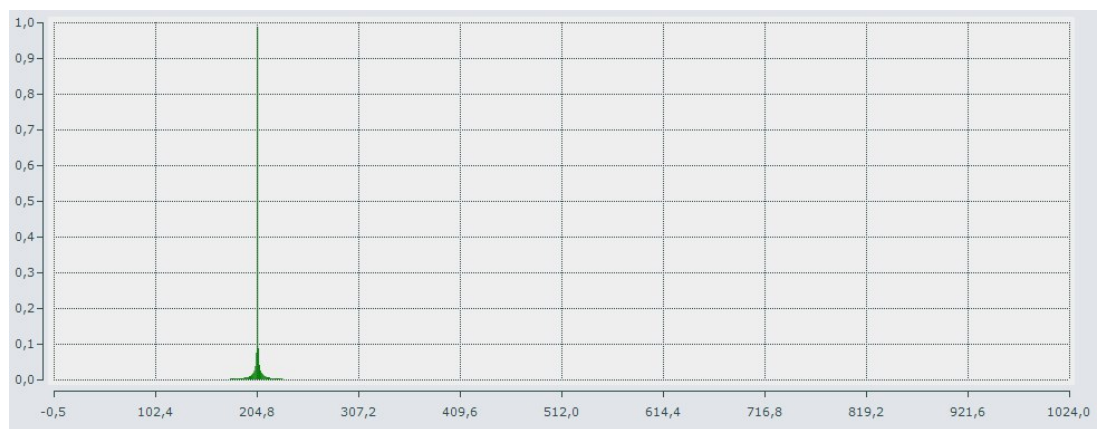


Figura 17:FFT risultate dall'utilizzo dell'algoritmo implementato su Twincat 3

La frequenza fondamentale alla quale si ha il massimo dell'ampiezza in frequenza nel testo con sinusoidale è pari a 204,8 Hz. Per verificare se il risultato è corretto è sufficiente calcolare una semplice proporzione: $1000:2048 = 100 : X$, dove 1000 è la frequenza di campionamento, 2048 è la lunghezza della finestra in cui calcolare la FFT e 100 è la frequenza dell'onda generata.

7. **FREQ_FEATURES:** le features che vengono calcolate sono tre: RMS, picco e picco-picco. Sono feature estratte nel dominio temporale, quindi direttamente dal segnale acquisito senza passare per il calcolo della trasformata. Si estraggono le features dal *Buffer_locale*

```
PROGRAM FREQ_FEATURES
```

```
VAR CONSTANT
```

```

C :INT:=2047;    //indica il numero di campioni scelti nell'intervallo (int per via
della funzione MOD che non prende in ingresso valori reali)

```

```
O:LREAL:=2047; //indica il numero di campioni scelti nell'intervallo (viene
usatosolo per il calcolo del RMS)
```

```
END_VAR
```

```
VAR
```

```
i :INT:=0;//indice del vettore di elaborazione
```

```
sum :LREAL:=0; //contatore somma
```

```
p2p :LREAL; //variabile contenente il risultato Picco-Picco
```

```
peak :LREAL; //variabile contenente il valore di Picco
```

```
END_VAR
```

```
IFCount_Row>0 THEN
```

```
sum:=sum+ EXPT(Buffer_locale[i],2);
```

```
i:=i+1;
```

```
END_IF
```

```
IF i = C THEN
```

```
rms:=SQRT (sum/O);
```

```
sum:=0;
```

```
i:=0;
```

Terminata la panoramica sui programmi implementati in Twincat bisogna passare alla parte relativa alla trasmissione dei dati. A tal proposito si aprono due strade:

- **Elaborazione del segnale offline:** il segnale vibrazionale viene acquisito tramite l'utilizzo di un accelerometro, collegato al PLC grazie ad un modulo di acquisizione installato su di esso. I dati vibrazionali acquisiti e riordinati all'interno di buffer codificati su Twincat (tramite le POU's analizzate nei paragrafi precedenti) vengono trasmessi al PC per mezzo della VI di LabView, necessaria affinché i dati vibrazionali possano essere elaborati su PC, in particolare su Matlab. In Matlab vengono rielaborati i dati in *ensambles* e vengono caricati *Diagnostic Feature Designer*, in cui avviene l'effettiva estrazione delle features e successiva classificazione;
- **Elaborazione del segnale online:** il segnale vibrazionale viene acquisito tramite l'utilizzo di un accelerometro, collegato al PLC grazie ad un modulo di acquisizione installato su di esso. I dati vibrazionali acquisiti e riordinati all'interno di buffer codificati su Twincat vengono elaborati e sottoposti al calcolo della FFT,

così da identificare la frequenza fondamentale del segnale. La differenza rispetto al caso di elaborazione offline è che l'estrazione delle features viene fatta direttamente su Twincat grazie alle POU's viste nel capitolo precedente, invece di utilizzare il *Diagnostic Feature Designer*. Una volta estratte, le features vengono trasmesse al PC e viene fatta direttamente la classificazione.

Il primo metodo verrà discusso nel prossimo capitolo, mentre il secondo metodo si è rivelato inapplicabile a causa dell'incompatibilità tra la l'architettura della CPU montata sul PLC e quella richiesta per applicare la *function* della FFT.

Per testare l'effettiva efficacia di questa tecnica è necessario utilizzare un PLC con un'architettura compatibile con quella richiesta dalla funzione oppure trovare una funzione alternativa che calcoli una FFT. Il problema, riscontrato con l'hardware a disposizione, può trovare soluzione con l'impiego di un modello che implementi una CPU differente.

4.3 Implementazione dello streaming di dati

Nei precedenti capitoli è stata descritta la possibilità di instaurare un collegamento tra il PLC e il PC e di garantire uno streaming di dati continuo tra le due parti. Per realizzare tale infrastruttura è stato realizzato, in LabView, un programma di connessione tra il software e il PLC. Per poter utilizzare un oggetto *TwinCAT.Ads* in un programma LabView è necessario inserire un nodo di costruzione dalla tavolozza funzioni .NET nel diagramma a blocchi. Il nodo di costruzione fornisce un riferimento all'oggetto. In TwinCAT, l'architettura del sistema permette ai singoli moduli del software (ad esempio TwinCAT PLC, TwinCAT NC, ...) di essere trattati come dispositivi indipendenti: per ogni attività c'è un modulo software ("Server" o "Client" "). I messaggi tra questi oggetti vengono scambiati tramite un'interfaccia ADS coerente con il protocollo del router. In questo modo vengono gestiti e distribuiti tutti i messaggi nel sistema e sulle connessioni TCP/IP. I router di messaggi TwinCAT sono presenti su ogni PC TwinCAT e su ogni controller bus BeckhoffBCxxxx. Ciò consente a tutti i programmi server e client TwinCAT di scambiare comandi e dati.

Di seguito è riportato uno schema implementativo della comunicazione PLC-PC preso direttamente dal sito Beckhoff [3]:

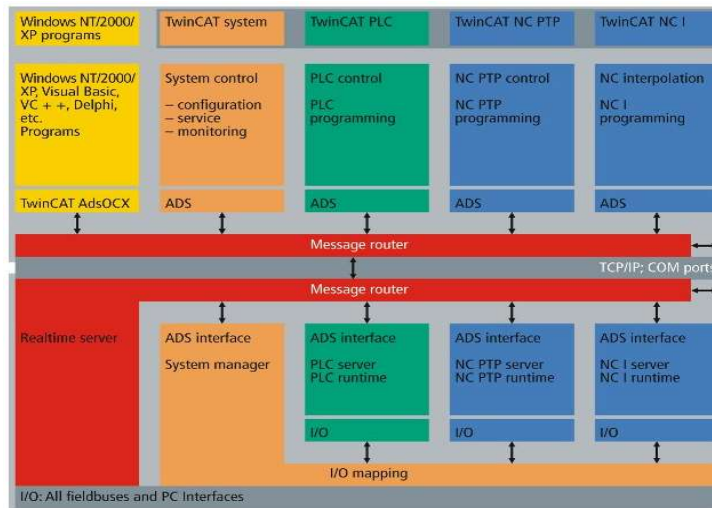


Figura 18: Schema delle modalità implementative del modulo ADS di Bekchoff per la comunicazione PLC-PC

Le VI sono state sviluppate da Meta sistemi e sono state riproposte per il progetto di tesi. Sono state apportate delle modifiche nella VI che si occupa dell'acquisizione vera e propria. Sono stati aumentati il numero di canali di acquisizione, passando a tre canali acquisiti in rapida successione, partendo da quello collegato alla variabile switch definita in Twincat. Viene interrogata la variabile switch, se questa è pari a 0 allora viene acquisito il canale collegato a Elaboration_Buffer_1, se è pari ad 1 viene acquisito l'Elaboration_Buffer_2

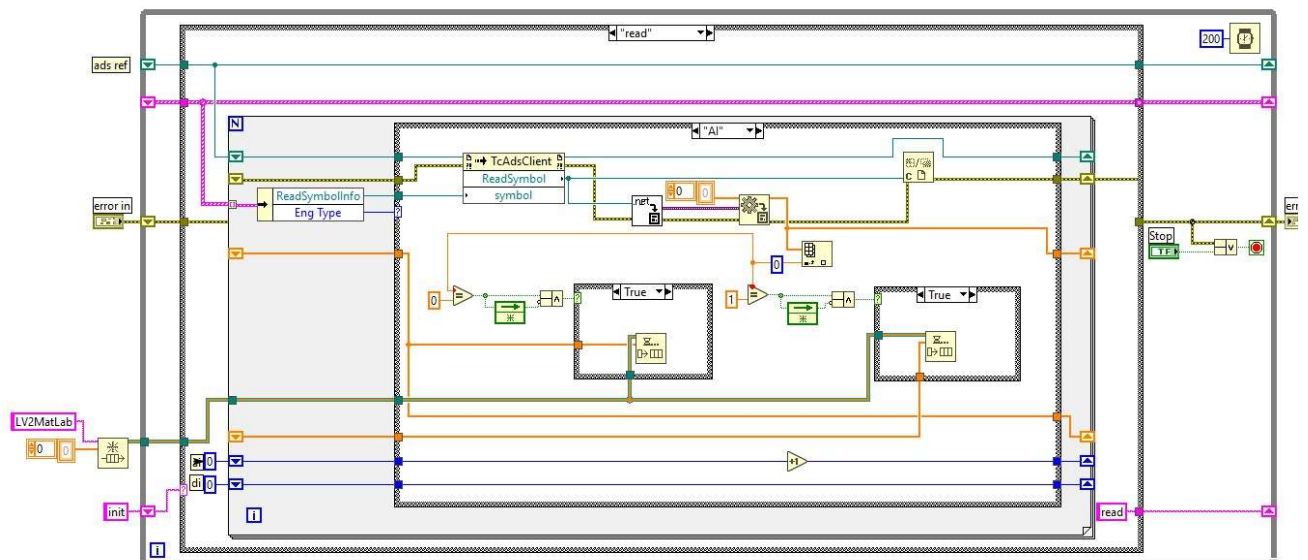


Figura 19: VI di gestione dello streaming dei dati tra LabView e il PLC

Come è possibile notare dall'intestazione riportata nel loop box della VI, nell'immagine è stata presentata esclusivamente l'operazione di lettura dei dati da PC. Tali dati vengono

immagazzinati all'interno di un buffer che verrà poi richiamato all'interno della VI che si occupa della stampa dei valori su file .tdms. Il buffer viene riempito prima con gli elementi propri di Elaboration_buffer_1 e poi con quelli di Elaboration_buffer_2, ed il passaggio da un vettore all'altro è gestito tramite la lettura della variabile switch. In questo modo si evita la dispersione di dati fondamentali o la ridondanza dei dati letti. Questo genere di problemi insorge per via della netta differenza di velocità di elaborazione che c'è tra PLC e PC. Infatti, come riportato già precedentemente, la VI definita in Labview elabora molto più velocemente i dati rispetto a quanto velocemente acquisisce il PLC.

Successivamente è stata sviluppata una VI che permette di riportare i dati vibrazionali trasmessi dal PLC al PC, tramite la VI precedentemente mostrata su un file .tdms leggibile da Matlab, così da poter utilizzare il toolbox per l'estrazione delle features e della classificazione dei guasti. Di seguito è riportato lo schema a blocchi.

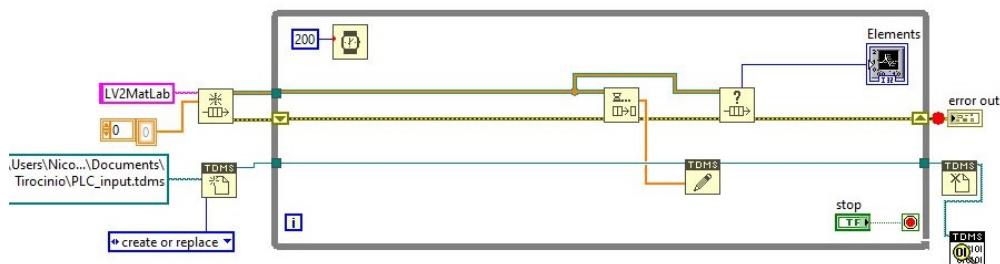


Figura 20: VI che permette di scrivere i dati acquisiti su file .tdms

4.4 Analisi in Matlab

L'ultimo passo dell'elaborazione offline dei dati vibrazionali consiste nell'estrarre le feature dal dataset di interesse e classificarne i guasti. Questo procedimento viene svolto tramite il *Diagnostic Feature Designer* di Matlab, già menzionato precedentemente. A tal proposito viene mostrata una panoramica su come è stato implementato il tool. Diagnostic Feature Designer è un'app che consente di sviluppare le feature e valutare potenziali indicatori di condizione utilizzando un'interfaccia grafica multifunzione.

Di seguito è riportata una tabella in cui sono indicate le tipologie di misure effettuate:

Famiglia Motore	ID misura	Tipo di guasto	Prima acquisizione	Seconda acquisizione
R50	1	Nessuno	dytran incollato su statore in direzione radiale	pcb incollato su statore in direzione assiale
R50	2	Cuscinetto rumoroso	dytran incollato su statore in direzione radiale	pcb incollato su statore in direzione assiale

R50	3	Filo tocca rotore	dytran incollato su statore in direzione radiale	pcb incollato su statore in direzione assiale
R50	4	Cuscinetto ammaccato	dytran incollato su statore in direzione radiale	pcb incollato su statore in direzione assiale
R50	5	Nessuno	dytran incollato su statore in direzione radiale	pcb incollato su statore in direzione assiale
R50	6	Cuscinetto ammaccato	dytran incollato su statore in direzione radiale	pcb incollato su statore in direzione assiale
R50	7	Cuscinetto ammaccato	dytran incollato su statore in direzione radiale	pcb incollato su statore in direzione assiale
R50	8	Cuscinetto ammaccato	dytran incollato su statore in direzione radiale	pcb incollato su statore in direzione assiale
R50	9	Calotta lenta	dytran incollato su statore in direzione radiale	pcb incollato su statore in direzione assiale
R50	10	Calotta lenta	dytran incollato su statore in direzione radiale	pcb incollato su statore in direzione assiale
R50	11	Nessuno	dytran incollato su statore in direzione radiale	pcb incollato su statore in direzione assiale
R50	12	Nessuno	dytran incollato su statore in direzione radiale	pcb incollato su statore in direzione assiale
R50	13	Nessuno	dytran incollato su statore in direzione radiale	pcb incollato su statore in direzione assiale
R50	14	Nessuno	dytran incollato su statore in direzione radiale	pcb incollato su statore in direzione assiale
R50	15	Nessuno	dytran incollato su statore in direzione radiale	pcb incollato su statore in direzione assiale
R50	16	Nessuno	dytran incollato su statore in direzione radiale	pcb incollato su statore in direzione assiale
R50	17	Nessuno	dytran incollato su statore in direzione radiale	pcb incollato su statore in direzione assiale
R50	18	Nessuno	dytran incollato su statore in direzione radiale	pcb incollato su statore in direzione assiale
R50	19	Nessuno	dytran incollato su statore in direzione radiale	pcb incollato su statore in direzione assiale
R50	20	Nessuno	dytran incollato su statore in direzione radiale	pcb incollato su statore in direzione assiale
R50	21	Nessuno	dytran incollato su statore in direzione radiale	pcb incollato su statore in direzione assiale
R50	22	Nessuno	dytran incollato su statore in direzione radiale	pcb incollato su statore in direzione assiale

L'app funziona su gruppi di dati, chiamati *Ensembles*. Un insieme è una raccolta di set di dati, creati misurando o simulando un sistema in condizioni variabili. Un singolo set di dati che rappresenta un sistema in una serie di condizioni è un membro. *Diagnostic Feature Designer* elabora tutti i membri dell'ensemble durante l'esecuzione di un'operazione.

Il primo passo è quello di riordinare i dati ottenuti in ensemble. Tramite lo streaming dati impostato con Labview si ottengono una serie di file .tdms, ognuno facente riferimento ad una misurazione effettuata. Questo tipo di file non è leggibile direttamente dal *Diagnostic Feature Designer*, quindi è stato necessario installare le librerie che permettono la lettura dei dati .tdms in Matlab e, in fine, raggruppare i dati in ensemble, ognuno relativo al tipo di guasto e alla modalità con la quale è stata svolta la misura. Per creare gli insiemi di dati sono stati definiti diversi vettori identificati come FaultCode1, FaultCode2, FaultCode3, FaultCode 4, di dimensione pari al numero di dataset interni ad un singolo insieme. In questi vettori sono riportati valori pari a 0,1,2,3 e 4, i quali indicano le condizioni di funzionamento del motore:

- 0: funzionamento ottimale;
- 1: cuscinetto ammaccato;
- 2: cuscinetto rumoroso;
- 3: filo di rame che tocca il rotore;
- 4: calotta lenta;

I vettori FaultCode sono:

- FaultCode1: vettore delle condizioni del motore rispetto al cuscinetto ammaccato (guasto 1);
- FaultCode2: vettore delle condizioni del motore rispetto al cuscinetto rumoroso (guasto 2);
- FaultCode3: vettore delle condizioni del motore rispetto al filo in contatto con il rotore (guasto 3);
- FaultCode4: vettore delle condizioni del motore rispetto alla calotta lenta (guasto 4);

Vengono costruite 16 tabelle, ognuna caratterizzata da due campi, ottenute combinando le condizioni di misura, ovvero misura assiale, misura radiale, sia stazionarie che nel transitorio, per ognuno dei quattro tipi di guasto analizzati:

	1	2
	Vibr_Assiale_Ss1	FaultCode1
1	51200x1 double	0
2	51200x1 double	0
3	51200x1 double	1
4	51200x1 double	1
5	51200x1 double	0
6	51200x1 double	0
7	51200x1 double	1
8	51200x1 double	1
9	51200x1 double	1
10	51200x1 double	1
11	51200x1 double	1
12	51200x1 double	1
13	51200x1 double	0
14	51200x1 double	0
15	51200x1 double	0
16	51200x1 double	0
17	51200x1 double	0
18	51200x1 double	0
19	51200x1 double	0
20	51200x1 double	0
21	51200x1 double	0
22	51200x1 double	0
23	51200x1 double	0
24	51200x1 double	0

Figura 21: Ensemble di dataset costruito su Matlab in formato data-table

Come mostrato in figura 21, il valore 0 corrisponde al dataset di misure ottenute in condizioni di lavoro ottimali, invece il valore 1 corrisponde al dataset corrispondente alla condizione di guasto (cuscinetto ammaccato).

A questo punto bisogna caricare le tabelle ottenute, una per volta, ed elaborare i dati:

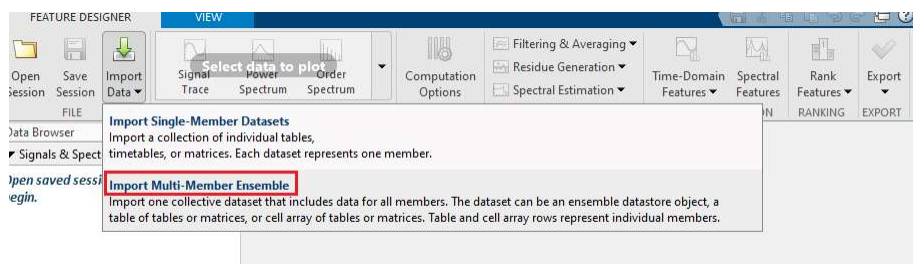


Figura 22: Si evidenzia dove bisogna cliccare per caricare l'intera tabella costruita in formato data-table

Viene richiesto che tipo di ensemble si sta caricando, e nel caso in esame è un **dataTable** in cui il campo FaultCode è un *condition variable* e Vibration/time è un *signal*.

Una volta confermate le impostazioni dell'insieme caricato è possibile tracciarne il grafico dell'andamento nel tempo cliccando sul pulsante *Signal Trace*.

4.4.1 Diagnosi nel caso di cuscinetto ammaccato

I grafici faranno riferimento al primo caso di studio, ovvero la diagnosi del cuscinetto ammaccato con acquisizione lungo la direzione assiale in regime stazionario.

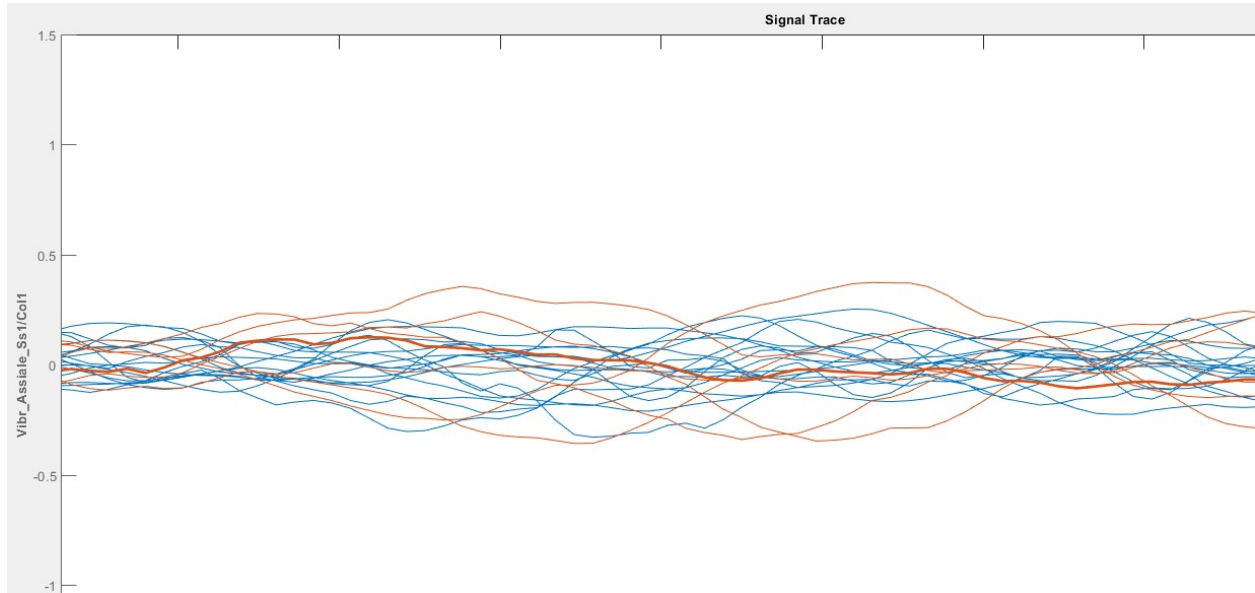


Figura 23: Segnale illustrato all'interno del toolbox Diagnostic Features Designer

Per analizzare i guasti viene mostrato quali membri hanno corrispondenti valori di `faultCode` utilizzando la codifica a colori cliccando su *Ensemble ViewPreferences > Group by "faultCode"*. L'area di disegno mostra un diagramma di traccia del segnale di tutti i membri dell'ensemble caricato. Quando si sposta il cursore sui dati, un indicatore nell'angolo in basso a destra identifica il membro su cui si trova il cursore. La traccia del segnale risultante mostra che tutti i picchi di vibrazione più elevati sono associati a dati provenienti dalle condizioni di guasto. Tuttavia, non è detto che tutti dataset difettosi presentino picchi più elevati.

Diagnostic Feature Designer consente di progettare funzionalità che forniscono queste funzioni di diagnostica.

- Si possono generare le features direttamente utilizzando i segnali importati.
- Altre volte è necessario eseguire un'ulteriore elaborazione del segnale, come il filtraggio e la media, per ottenere risultati significativi.

Oltre a mostrare il segnale nel dominio del tempo si può ricavare anche lo spettro in frequenza per elaborarne le features relative. Per generare uno spettro di potenza, selezionare *SpectralEstimation > Power Spectrum*

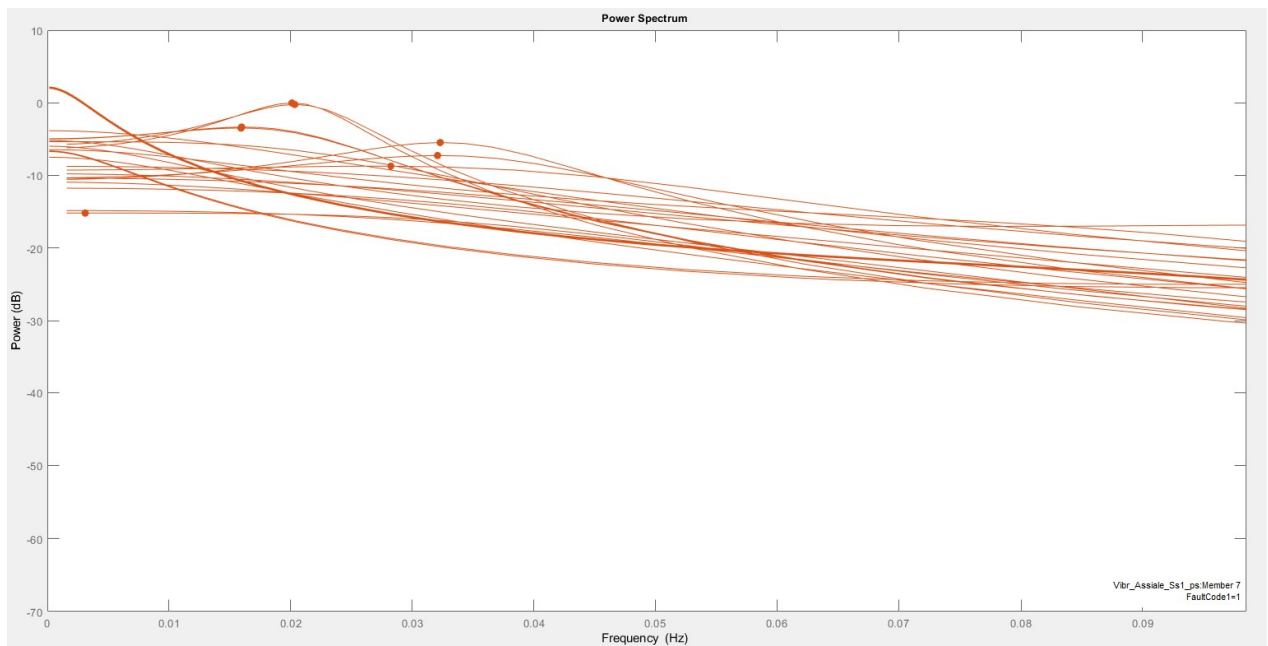


Figura 24:segnale spettrale generato a partire dal segnale importato nel toolbox

Per estrarre le features conviene considerare solamente l'intervallo di frequenze che comprendono i picchi di potenza visibili in figura.

Il passo successivo è la generazione delle features, sia nel dominio del tempo che in quello della frequenza. Partendo dalla generazione delle features nel tempo, è sufficiente selezionare *Time Domain Features > Signal Features* dalla barra delle funzionalità e si aprirà una finestra in cui si potranno selezionare le features che si intende estrarre. Di seguito è riportata la finestra appena descritta:

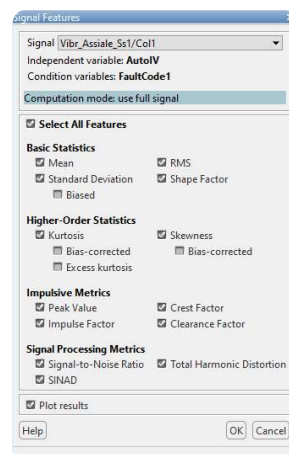


Figura 25:finestra di calcolo delle features nel dominio del tempo

Si preferisce selezionare tutte le features possibili, così da avere una panoramica completa sull'efficacia dell'elaborazione. Per ogni feature selezionata, l'app calcola un valore per ciascun

membro dell'ensemble e visualizza i risultati in un istogramma il quale visualizza la capacità di ogni features selezionata di separare i dati sani da quelli non sani.



Figura 26: Istogrammi rappresentativi della efficienza relativa ad ogni singola features nel tempo calcolata

La scheda Istogramma visualizza i parametri che determinano il contenuto e la risoluzione degli istogrammi. L'istogramma raggruppa, secondo dei codici colore, i dati in base alla variabile di condizione faultCode. I dati blu sono relativi ai motori sani e i dati arancioni a quelli difettosi, come indicato dalla legenda (la codifica a colori potrebbe apparire diversa). Per i valori delle caratteristiche in cui le etichette sane e difettose si sovrappongono, il colore appare marrone a causa della sovrapposizione tra blu e arancione. Si può avere un'idea approssimativa di quali funzioni sono efficaci valutando quali separano chiaramente i dati blu dai dati arancioni. Nel caso considerato, features quali *Sinad*, *SNR*, *shapefactor* e, in parte, *STD* ed *RMS* sono quelle che meglio riescono a distinguere i colori degli istogrammi, nonostante la presenza di piccole porzioni di sovrapposizione. Ci sono solo piccole aree di sovrapposizione. Al contrario, *Skewness* e *Mean* hanno grandi quantità di sovrapposizione. Si conclude che queste due features sembrano inefficaci per questi dati e questa variabile di condizione. Di default, l'app traccia gli

istogrammi per tutte le features ma è possibile visualizzare solo un sottoinsieme di istogrammi più importanti.

Una volta calcolate le features, queste vengono sottoposte ad un riordinamento in base alla loro importanza, già leggibile dagli istogrammi. Cliccando su *rank features* si ottiene una classifica delle features estratte, come mostrato in figura:

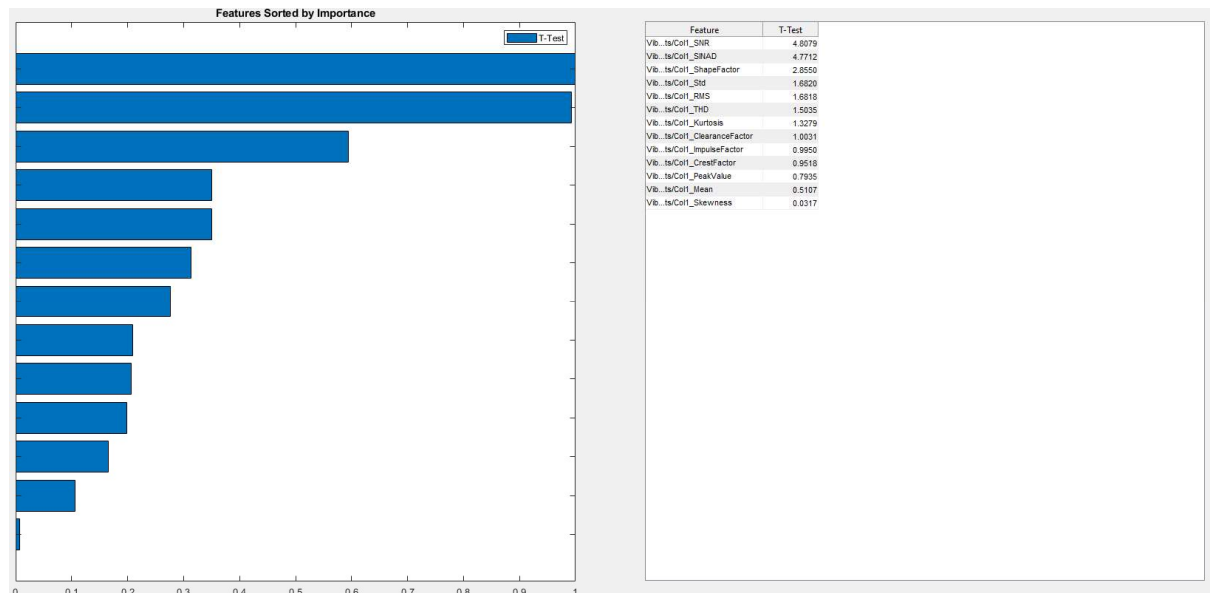


Figura 27: Classifica delle features nel tempo in funzione dell'efficienza

Per calcolare la classifica viene utilizzata la metrica del t-test. Concordamente a quanto riportato dai grafici, le features che hanno un valore di efficienza più elevato sono *Sinad*, *SNR*, *shapefactor* e, in parte, *STD* ed *RMS*. Questo grafico è importante sotto il punto di vista della scelta delle features per effettuare la classificazione del modello. Infatti non è utile effettuare una classificazione considerando tutte le caratteristiche del segnale, ma è sufficiente valutare solo quelle più significative.

Per effettuare la classificazione è sufficiente cliccare su *Export to Classifier Learner* e selezionare solamente le features scelte. Durante la selezione delle features è disponibile la possibilità di applicare la tecnica di validazione incrociata, utile per evitare eventuali problemi di *overfitting*. La convalida incrociata consiste nella suddivisione dell'insieme di dati totale in k parti di uguale numerosità e, a ogni passo, la k^a parte dell'insieme di dati viene a essere quella di convalida, mentre la restante parte costituisce sempre l'insieme di addestramento. Così si allena il modello per ognuna delle k parti. Il tool mette a disposizione diversi livelli di validazione incrociata, e per avere una più ampia gamma di risultati sono stati scelti quattro diversi livelli: *No- Validation*, *5 folds – Validation*, *10 folds – Validation*, *Maxfolds – Validation*

(massimo numero di gruppi in cui può essere diviso il dataset, ovvero pari al numero di dataset che fanno parte di un singolo ensemble)

Quello che si ottiene è un grafico cartesiano in cui sono riportati, su asse X e asse Y, due delle features selezionate. Il grafico mostra i cluster di dati divisi per classi (colore blu per motore sano, colore arancio per motore guasto) che vengono sottoposti ad addestramento.

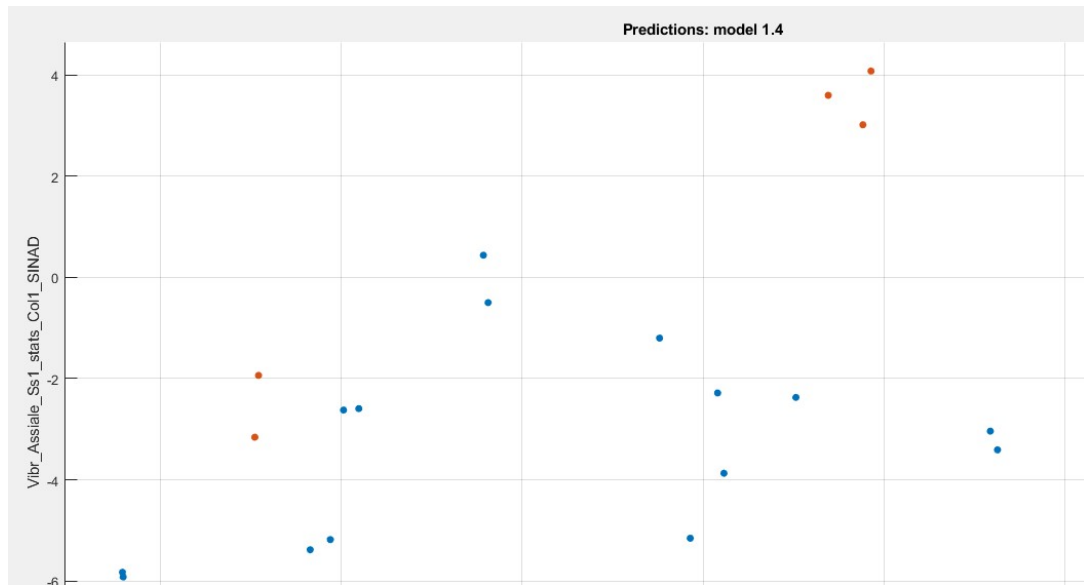


Figura 28: Diagramma cartesiano in cui viene eseguito il “plot” dei predittori, evidenziando le classi di operatività del motore

Per addestrare il modello, il tool mette a disposizione diversi algoritmi di classificazione, quali varianti del *K-Nearest-Neighbours* e del *DecisionalTree*:

- **K Nearest- Neighbours:** è un algoritmo basato sul calcolo delle distanze tra data points (punti sul diagramma cartesiano), in particolare sul concetto di somiglianza delle caratteristiche (più un’istanza del modello è simile al dataset analizzato più l’algoritmo li considera simili). Il parametro K identifica il numero di data points più vicini. L’algoritmo valuta le k minime distanze così ottenute. La classe che ottiene il maggior numero di queste distanze minime è scelta come previsione.
- **DecisionalTree:** albero di decisione viene utilizzato per classificare le istanze di grandi quantità di dati. Un albero di decisione descrive una struttura ad albero dove i nodi foglia rappresentano le classificazioni e le ramificazioni l’insieme delle proprietà che portano a quelle classificazioni. Di conseguenza ogni nodo interno

risulta essere una macro-classe costituita dall'unione delle classi associate ai suoi nodi figli.

Una volta scelto l'algoritmo, viene riportata la percentuale di precisione con la quale avviene la predizione del modello: più è alta e più le classi di dati sono ben distinte tra loro.

Dividendo fra i casi in cui viene applicata o meno la validazione incrociata, sono stati ottenuti i seguenti risultati, valutati in funzione dell'*accuracy*, ovvero il rapporto tra il numero di decisioni corrette rispetto al numero totale di decisioni. Una misura è tanto più accurata quanto più la media delle misure si approssima al valore vero della grandezza:

- 1. No Cross-Validation:** Il primo caso esclude l'impiego della validazione incrociata, lasciando che l'intero dataset sia sfruttato per l'addestramento e, di conseguenza, per il testing. I risultati sono divisi in PCA ON e PCA OFF, la quale serve a ridurre la dimensione del dataset andando a considerare solo un set ristretto di dati, evitando la ridondanza delle misure.

1.1 ☆ Tree Last change: Fine Tree	Accuracy: 91.7% 5/5 features
1.2 ☆ Tree Last change: Medium Tree	Accuracy: 91.7% 5/5 features
1.3 ☆ Tree Last change: Coarse Tree	Accuracy: 91.7% 5/5 features
1.4 ☆ KNN Last change: Fine KNN	Accuracy: 100.0% 5/5 features
1.5 ☆ KNN Last change: Medium KNN	Accuracy: 91.7% 5/5 features
1.6 ☆ KNN Last change: Coarse KNN	Accuracy: 66.7% 5/5 features
1.7 ☆ KNN Last change: Cosine KNN	Accuracy: 91.7% 5/5 features
1.8 ☆ KNN Last change: Cubic KNN	Accuracy: 91.7% 5/5 features
1.9 ☆ KNN Last change: Weighted KNN	Accuracy: 100.0% 5/5 features

Figura 29: Risultati ottenuti classificando tramite le features nel tempo senza validazione

Il risultato è ottimo, con tutte le tecniche di classificazione impiegate si raggiunge una percentuale sopra il 90% e in due casi specifici si raggiunge il 100% (*Fine KNN* e *Weighted KNN*). Quindi, in fase di classificazione online è possibile impiegare due delle tecniche che restituiscono una *accuracy* del 100% per avere un modello ben addestrato.

2.1 ☆ Tree Last change: Fine Tree	Accuracy: 91.7% 1/5 features (PCA on)
2.2 ☆ Tree Last change: Medium Tree	Accuracy: 91.7% 1/5 features (PCA on)
2.3 ☆ Tree Last change: Coarse Tree	Accuracy: 91.7% 1/5 features (PCA on)
2.4 ☆ KNN Last change: Fine KNN	Accuracy: 100.0% 1/5 features (PCA on)
2.5 ☆ KNN Last change: Medium KNN	Accuracy: 91.7% 1/5 features (PCA on)
2.6 ☆ KNN Last change: Coarse KNN	Accuracy: 66.7% 1/5 features (PCA on)
2.7 ☆ KNN Last change: Cosine KNN	Accuracy: 75.0% 1/5 features (PCA on)
2.8 ☆ KNN Last change: Cubic KNN	Accuracy: 91.7% 1/5 features (PCA on)
2.9 ☆ KNN Last change: Weighted KNN	Accuracy: 100.0% 1/5 features (PCA on)

Figura 30: Risultati ottenuti classificando tramite le features nel tempo con PCA attiva senza validazione

Con la PCA i risultati rimangono simili, eccetto per il *Cosine KNN* in cui si ha un abbassamento di *accuracy*, dal 91,7% al 75%.

2. 5 folds Cross-Validation: Il dataset viene diviso in 5 parti.

1.1 ☆ Tree Last change: Fine Tree	Accuracy: 79.2% 5/5 features
1.2 ☆ Tree Last change: Medium Tree	Accuracy: 79.2% 5/5 features
1.3 ☆ Tree Last change: Coarse Tree	Accuracy: 79.2% 5/5 features
1.4 ☆ KNN Last change: Fine KNN	Accuracy: 100.0% 5/5 features
1.5 ☆ KNN Last change: Medium KNN	Accuracy: 66.7% 5/5 features
1.6 ☆ KNN Last change: Coarse KNN	Accuracy: 66.7% 5/5 features
1.7 ☆ KNN Last change: Cosine KNN	Accuracy: 66.7% 5/5 features
1.8 ☆ KNN Last change: Cubic KNN	Accuracy: 70.8% 5/5 features
1.9 ☆ KNN Last change: Weighted KNN	Accuracy: 91.7% 5/5 features

Figura 31: Risultati ottenuti classificando tramite le features nel tempo con 5 folds validation

Osservando i risultati si può notare come, in generale, questo modello risponde abbastanza bene all'addestramento effettuato tramite tutti gli algoritmi, anche se i risultati sono peggiorati rispetto al caso senza validazione incrociata. Infatti, la classificazione presenta una *accuracy* che oscilla tra il 66,7% e il 91,7%, raggiungendo nel caso di classificazione tramite *Fine KNN* una percentuale del 100%. Si può anche effettuare una **PCA** (*Principal Component Analysis*) prima di classificare il modello. I risultati sono diversi:

2.1 ☆ Tree Last change: Fine Tree	Accuracy: 91.7% 1/5 features (PCA on)
2.2 ☆ Tree Last change: Medium Tree	Accuracy: 91.7% 1/5 features (PCA on)
2.3 ☆ Tree Last change: Coarse Tree	Accuracy: 91.7% 1/5 features (PCA on)
2.4 ☆ KNN Last change: Fine KNN	Accuracy: 91.7% 1/5 features (PCA on)
2.5 ☆ KNN Last change: Medium KNN	Accuracy: 91.7% 1/5 features (PCA on)
2.6 ☆ KNN Last change: Coarse KNN	Accuracy: 66.7% 1/5 features (PCA on)
2.7 ☆ KNN Last change: Cosine KNN	Accuracy: 70.8% 1/5 features (PCA on)
2.8 ☆ KNN Last change: Cubic KNN	Accuracy: 91.7% 1/5 features (PCA on)
2.9 ☆ KNN Last change: Weighted KNN	Accuracy: 91.7% 1/5 features (PCA on)

Figura 32: Risultati ottenuti classificando tramite le features nel tempo con PCA attiva con 5 folds validation

Come si può notare, le percentuali sono aumentate in quasi tutti i casi, ma non si raggiunge il massimo nella precisione per nessuno degli algoritmi di classificazione. Inoltre, rispetto al precedente caso i risultati sono decisamente peggiori, dato che non ci sono algoritmo di classificazione che, sotto PCA, restituiscono la percentuale massima.

3. 10 folds Cross-Validation: Il dataset viene diviso in 10 parti.

1.1 ☆ Tree Last change: Fine Tree	Accuracy: 87.5% 5/5 features
1.2 ☆ Tree Last change: Medium Tree	Accuracy: 87.5% 5/5 features
1.3 ☆ Tree Last change: Coarse Tree	Accuracy: 87.5% 5/5 features
1.4 ☆ KNN Last change: Fine KNN	Accuracy: 100.0% 5/5 features
1.5 ☆ KNN Last change: Medium KNN	Accuracy: 79.2% 5/5 features
1.6 ☆ KNN Last change: Coarse KNN	Accuracy: 66.7% 5/5 features
1.7 ☆ KNN Last change: Cosine KNN	Accuracy: 83.3% 5/5 features
1.8 ☆ KNN Last change: Cubic KNN	Accuracy: 83.3% 5/5 features
1.9 ☆ KNN Last change: Weighted KNN	Accuracy: 91.7% 5/5 features

Figura 33: Risultati ottenuti classificando tramite le features nel tempo con 10 folds validation

Rispetto al caso precedente c'è un leggero miglioramento nella maggior parte dei risultati, mantenendo la percentuale del 100% per il *Fine KNN*.

3.1 ☆ Tree Last change: Fine Tree	Accuracy: 91.7% 1/5 features (PCA on)
3.2 ☆ Tree Last change: Medium Tree	Accuracy: 91.7% 1/5 features (PCA on)
3.3 ☆ Tree Last change: Coarse Tree	Accuracy: 91.7% 1/5 features (PCA on)
3.4 ☆ KNN Last change: Fine KNN	Accuracy: 79.2% 1/5 features (PCA on)
3.5 ☆ KNN Last change: Medium KNN	Accuracy: 91.7% 1/5 features (PCA on)
3.6 ☆ KNN Last change: Coarse KNN	Accuracy: 66.7% 1/5 features (PCA on)
3.7 ☆ KNN Last change: Cosine KNN	Accuracy: 79.2% 1/5 features (PCA on)
3.8 ☆ KNN Last change: Cubic KNN	Accuracy: 91.7% 1/5 features (PCA on)
3.9 ☆ KNN Last change: Weighted KNN	Accuracy: 79.2% 1/5 features (PCA on)

Figura 34: Risultati ottenuti classificando tramite le features nel tempo con PCA attiva con 10 foldsvalidation

Con la PCA attiva i risultati si mantengono sulla stessa qualità del caso precedente: rispetto al corrispondente risultato senza PCA c'è un leggero peggioramento, data l'assenza di una percentuale massima che ci garantisca una classificazione estremamente efficace.

4. Maxfolds Cross-Validation: Il dataset viene diviso in 24 parti.

1.1 ☆ Tree Last change: Fine Tree	Accuracy: 83.3% 5/5 features
1.2 ☆ Tree Last change: Medium Tree	Accuracy: 83.3% 5/5 features
1.3 ☆ Tree Last change: Coarse Tree	Accuracy: 83.3% 5/5 features
1.4 ☆ KNN Last change: Fine KNN	Accuracy: 100.0% 5/5 features
1.5 ☆ KNN Last change: Medium KNN	Accuracy: 91.7% 5/5 features
1.6 ☆ KNN Last change: Coarse KNN	Accuracy: 66.7% 5/5 features
1.7 ☆ KNN Last change: Cosine KNN	Accuracy: 91.7% 5/5 features
1.8 ☆ KNN Last change: Cubic KNN	Accuracy: 91.7% 5/5 features
1.9 ☆ KNN Last change: Weighted KNN	Accuracy: 91.7% 5/5 features

Figura 35: Risultati ottenuti classificando tramite le features nel tempo con PCA attiva con max foldsvalidation

In questa situazione, gli algoritmi di albero decisionale restituiscono risultati leggermente migliori rispetto alla divisione in 10 parti, ma peggiore rispetto agli altri due casi, a differenza degli algoritmi di KNN che invece costituiscono il secondo miglior risultato, dopo il caso senza validazione.

2.1	☆ Tree	Accuracy: 91.7%
	Last change: Fine Tree	1/5 features (PCA on)
2.2	☆ Tree	Accuracy: 91.7%
	Last change: Medium Tree	1/5 features (PCA on)
2.3	☆ Tree	Accuracy: 91.7%
	Last change: Coarse Tree	1/5 features (PCA on)
2.4	☆ KNN	Accuracy: 79.2%
	Last change: Fine KNN	1/5 features (PCA on)
2.5	☆ KNN	Accuracy: 91.7%
	Last change: Medium KNN	1/5 features (PCA on)
2.6	☆ KNN	Accuracy: 66.7%
	Last change: Coarse KNN	1/5 features (PCA on)
2.7	☆ KNN	Accuracy: 75.0%
	Last change: Cosine KNN	1/5 features (PCA on)
2.8	☆ KNN	Accuracy: 91.7%
	Last change: Cubic KNN	1/5 features (PCA on)
2.9	☆ KNN	Accuracy: 79.2%
	Last change: Weighted KNN	1/5 features (PCA on)

Figura 36: Risultati ottenuti classificando tramite le features nel tempo con PCA attiva con max foldvalidation

Purtroppo, attivando la PCA, si ottengono risultati pressochè identici ai casi precedenti in cui è stata effettuata la PCA, ovvero risultati peggiori rispetto al caso senza PCA.

Una volta ottenuto il modello è possibile esportarlo in Matlab ed effettuare l'addestramento di un nuovo set di dati a partire dal modello appena ricavato. Questa funzionalità non è oggetto di studio per questa trattazione.

Lo stesso procedimento viene ripetuto per il calcolo delle features in frequenza:

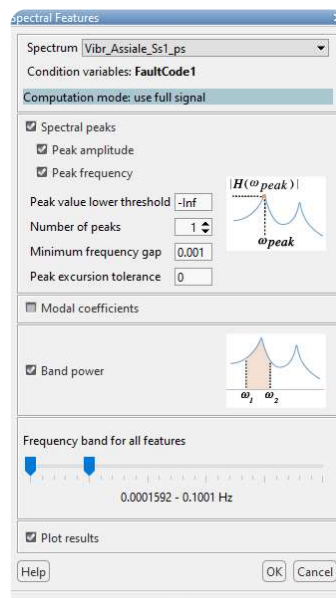


Figura 37: Finestra per la selezione e calcolo delle features in frequenza

In questo caso, oltre alla scelta delle features, è possibile modificare la banda in frequenza in cui calcolarle. Si sceglie la banda in cui compaiono più picchi di potenza nello spettro, dato che due features su tre vengono valutate in funzione dei picchi (ovvero *peak frequency* e

peakamplitude). Anche in questo caso vengono tracciati degli istogrammi che hanno le medesime caratteristiche di quelli visti per le features nel dominio del tempo:

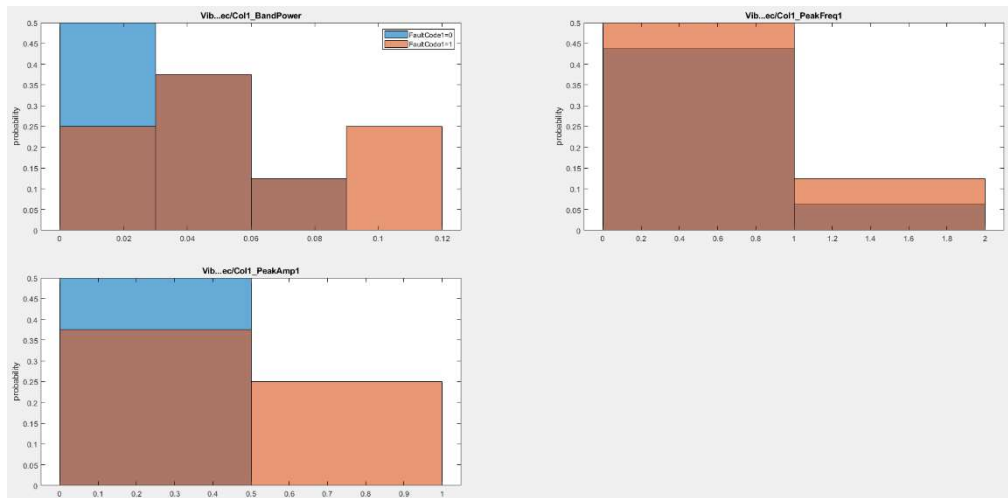


Figura 38: Istogrammi rappresentativi della efficienza relativa ad ogni singola features in frequenza calcolata

Essendo solo tre le features calcolate, conviene esportarle tutte nel *Classifier Learner*, così da avere una classificazione più accurata. Calcolando la classifica delle features, si ottiene il seguente risultato:

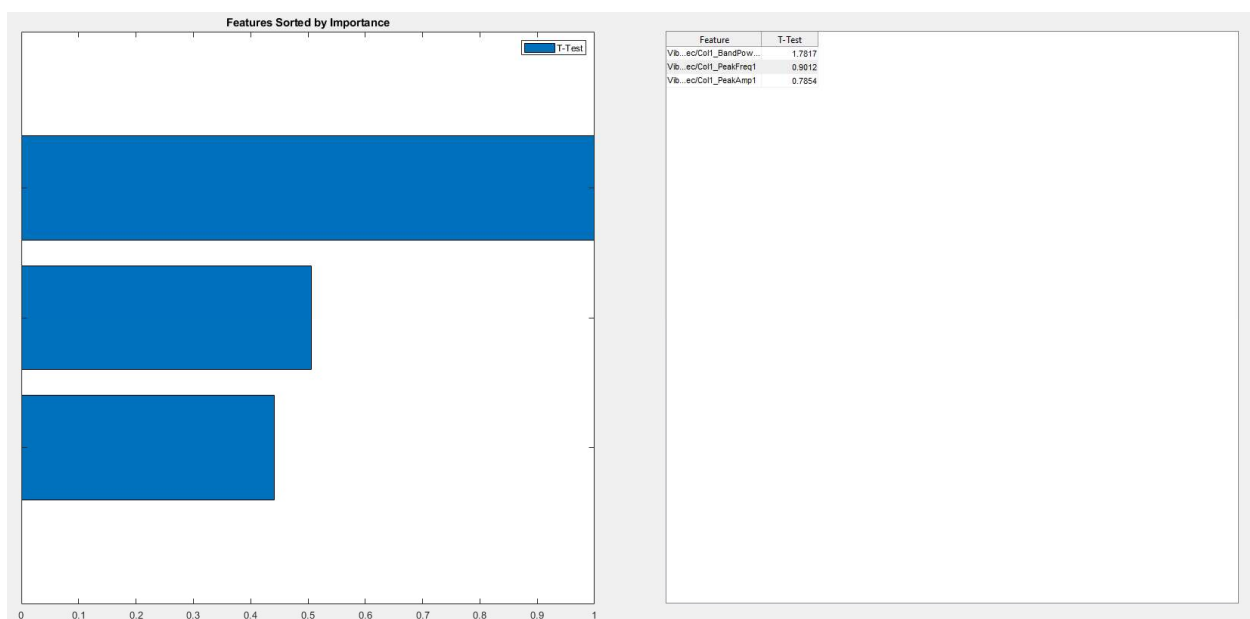


Figura 39: Classifica delle features in frequenza in funzione dell'efficienza

In questo caso è inutile fare una selezione delle features dato che tra loro non vi è un'elevata differenza in efficienza. Passando al *ClassificationLearner*, si ottiene il grafico come segue:

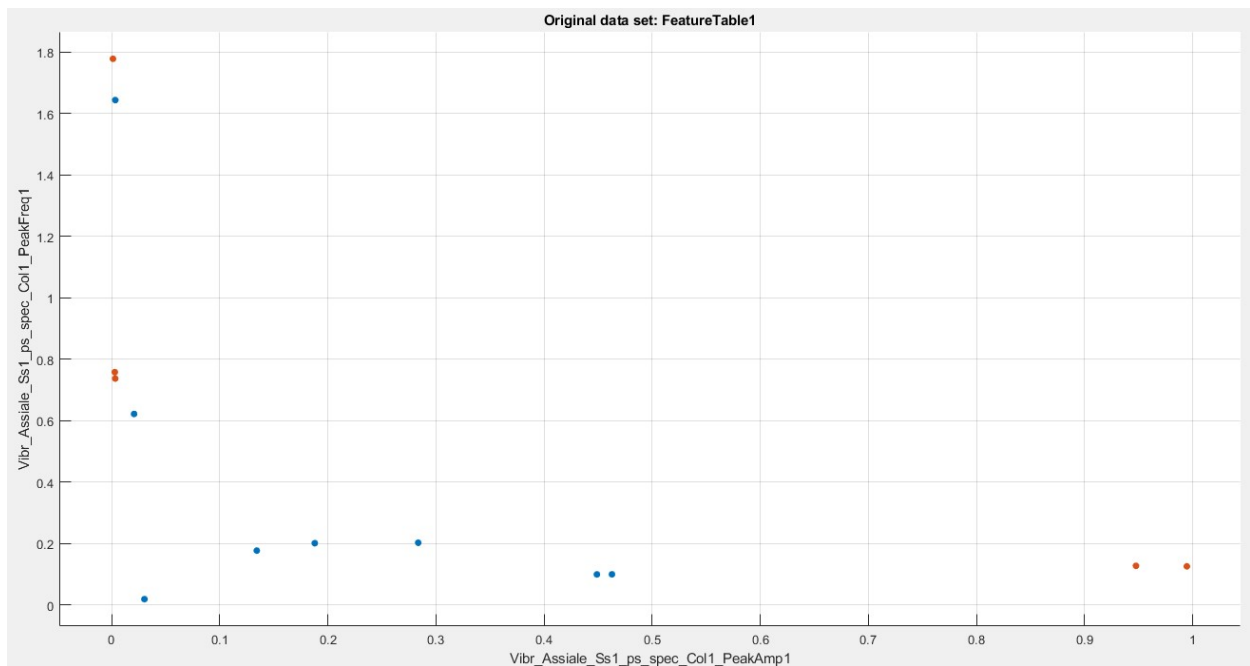


Figura 40:Diagramma cartesiano in cui viene eseguito il “plot” dei predittori, evidenziando le classi di operatività del motore

Ripercorrendo gli stessi passi fatti nel caso del dominio del tempo si ottengono i risultati riportati in basso:

1. No Cross-Validation:

1.1 ☆ Tree Last change: Fine Tree	Accuracy: 91.7% 3/3 features
1.2 ☆ Tree Last change: Medium Tree	Accuracy: 91.7% 3/3 features
1.3 ☆ Tree Last change: Coarse Tree	Accuracy: 91.7% 3/3 features
1.4 ☆ KNN Last change: Fine KNN	Accuracy: 87.5% 3/3 features
1.5 ☆ KNN Last change: Medium KNN	Accuracy: 66.7% 3/3 features
1.6 ☆ KNN Last change: Coarse KNN	Accuracy: 66.7% 3/3 features
1.7 ☆ KNN Last change: Cosine KNN	Accuracy: 66.7% 3/3 features
1.8 ☆ KNN Last change: Cubic KNN	Accuracy: 66.7% 3/3 features
1.9 ☆ KNN Last change: Weighted KNN	Accuracy: 87.5% 3/3 features

Figura 41:Risultati ottenuti classificando tramite le features in frequenza senza validazione

2.1 ☆ Tree Last change: Fine Tree	Accuracy: 83.3% 2/3 features (PCA on)
2.2 ☆ Tree Last change: Medium Tree	Accuracy: 83.3% 2/3 features (PCA on)
2.3 ☆ Tree Last change: Coarse Tree	Accuracy: 83.3% 2/3 features (PCA on)
2.4 ☆ KNN Last change: Fine KNN	Accuracy: 87.5% 2/3 features (PCA on)
2.5 ☆ KNN Last change: Medium KNN	Accuracy: 66.7% 2/3 features (PCA on)
2.6 ☆ KNN Last change: Coarse KNN	Accuracy: 66.7% 2/3 features (PCA on)
2.7 ☆ KNN Last change: Cosine KNN	Accuracy: 66.7% 2/3 features (PCA on)
2.8 ☆ KNN Last change: Cubic KNN	Accuracy: 66.7% 2/3 features (PCA on)
2.9 ☆ KNN Last change: Weighted KNN	Accuracy: 87.5% 2/3 features (PCA on)

Figura 42: Risultati ottenuti classificando tramite le features in frequenza con PCA attiva senza validazione

2. 5 folds Cross-Validation:

1.1 ☆ Tree Last change: Fine Tree	Accuracy: 62.5% 3/3 features
1.2 ☆ Tree Last change: Medium Tree	Accuracy: 62.5% 3/3 features
1.3 ☆ Tree Last change: Coarse Tree	Accuracy: 62.5% 3/3 features
1.4 ☆ KNN Last change: Fine KNN	Accuracy: 75.0% 3/3 features
1.5 ☆ KNN Last change: Medium KNN	Accuracy: 66.7% 3/3 features
1.6 ☆ KNN Last change: Coarse KNN	Accuracy: 66.7% 3/3 features
1.7 ☆ KNN Last change: Cosine KNN	Accuracy: 66.7% 3/3 features
1.8 ☆ KNN Last change: Cubic KNN	Accuracy: 66.7% 3/3 features
1.9 ☆ KNN Last change: Weighted KNN	Accuracy: 79.2% 3/3 features

Figura 43: Risultati ottenuti classificando tramite le features in frequenza con 5 folds validation

2.1 ☆ Tree Last change: Fine Tree	Accuracy: 75.0% 2/3 features (PCA on)
2.2 ☆ Tree Last change: Medium Tree	Accuracy: 75.0% 2/3 features (PCA on)
2.3 ☆ Tree Last change: Coarse Tree	Accuracy: 75.0% 2/3 features (PCA on)
2.4 ☆ KNN Last change: Fine KNN	Accuracy: 75.0% 2/3 features (PCA on)
2.5 ☆ KNN Last change: Medium KNN	Accuracy: 66.7% 2/3 features (PCA on)
2.6 ☆ KNN Last change: Coarse KNN	Accuracy: 66.7% 2/3 features (PCA on)
2.7 ☆ KNN Last change: Cosine KNN	Accuracy: 66.7% 2/3 features (PCA on)
2.8 ☆ KNN Last change: Cubic KNN	Accuracy: 66.7% 2/3 features (PCA on)
2.9 ☆ KNN Last change: Weighted KNN	Accuracy: 75.0% 2/3 features (PCA on)

Figura 44: Risultati ottenuti classificando tramite le features in frequenza con PCA attiva con 5 fold validation

3. 10 folds Cross-Validation:

1.1 ☆ Tree Last change: Fine Tree	Accuracy: 70.8% 3/3 features
1.2 ☆ Tree Last change: Medium Tree	Accuracy: 70.8% 3/3 features
1.3 ☆ Tree Last change: Coarse Tree	Accuracy: 70.8% 3/3 features
1.4 ☆ KNN Last change: Fine KNN	Accuracy: 75.0% 3/3 features
1.5 ☆ KNN Last change: Medium KNN	Accuracy: 66.7% 3/3 features
1.6 ☆ KNN Last change: Coarse KNN	Accuracy: 66.7% 3/3 features
1.7 ☆ KNN Last change: Cosine KNN	Accuracy: 66.7% 3/3 features
1.8 ☆ KNN Last change: Cubic KNN	Accuracy: 66.7% 3/3 features
1.9 ☆ KNN Last change: Weighted KNN	Accuracy: 79.2% 3/3 features

Figura 45: Risultati ottenuti classificando tramite le features in frequenza con 10 fold validation

3.1 ☆ Tree Last change: Fine Tree	Accuracy: 54.2% 2/3 features (PCA on)
3.2 ☆ Tree Last change: Medium Tree	Accuracy: 54.2% 2/3 features (PCA on)
3.3 ☆ Tree Last change: Coarse Tree	Accuracy: 54.2% 2/3 features (PCA on)
3.4 ☆ KNN Last change: Fine KNN	Accuracy: 75.0% 2/3 features (PCA on)
3.5 ☆ KNN Last change: Medium KNN	Accuracy: 66.7% 2/3 features (PCA on)
3.6 ☆ KNN Last change: Coarse KNN	Accuracy: 66.7% 2/3 features (PCA on)
3.7 ☆ KNN Last change: Cosine KNN	Accuracy: 62.5% 2/3 features (PCA on)
3.8 ☆ KNN Last change: Cubic KNN	Accuracy: 66.7% 2/3 features (PCA on)
3.9 ☆ KNN Last change: Weighted KNN	Accuracy: 75.0% 2/3 features (PCA on)

Figura 46: Risultati ottenuti classificando tramite le features in frequenza con PCA attiva con 10 fold validation

4. Maxfolds Cross-Validation:

1.1 ☆ Tree Last change: Fine Tree	Accuracy: 70.8% 3/3 features
1.2 ☆ Tree Last change: Medium Tree	Accuracy: 70.8% 3/3 features
1.3 ☆ Tree Last change: Coarse Tree	Accuracy: 70.8% 3/3 features
1.4 ☆ KNN Last change: Fine KNN	Accuracy: 75.0% 3/3 features
1.5 ☆ KNN Last change: Medium KNN	Accuracy: 66.7% 3/3 features
1.6 ☆ KNN Last change: Coarse KNN	Accuracy: 66.7% 3/3 features
1.7 ☆ KNN Last change: Cosine KNN	Accuracy: 66.7% 3/3 features
1.8 ☆ KNN Last change: Cubic KNN	Accuracy: 66.7% 3/3 features
1.9 ☆ KNN Last change: Weighted KNN	Accuracy: 79.2% 3/3 features

Figura 47: Risultati ottenuti classificando tramite le features in frequenza con max fold validation

2.1 ☆ Tree Last change: Fine Tree	Accuracy: 75.0% 2/3 features (PCA on)
2.2 ☆ Tree Last change: Medium Tree	Accuracy: 75.0% 2/3 features (PCA on)
2.3 ☆ Tree Last change: Coarse Tree	Accuracy: 75.0% 2/3 features (PCA on)
2.4 ☆ KNN Last change: Fine KNN	Accuracy: 75.0% 2/3 features (PCA on)
2.5 ☆ KNN Last change: Medium KNN	Accuracy: 66.7% 2/3 features (PCA on)
2.6 ☆ KNN Last change: Coarse KNN	Accuracy: 66.7% 2/3 features (PCA on)
2.7 ☆ KNN Last change: Cosine KNN	Accuracy: 66.7% 2/3 features (PCA on)
2.8 ☆ KNN Last change: Cubic KNN	Accuracy: 66.7% 2/3 features (PCA on)
2.9 ☆ KNN Last change: Weighted KNN	Accuracy: 75.0% 2/3 features (PCA on)

Figura 48: Risultati ottenuti classificando tramite le features in frequenza con PCA attiva con max fold validation

I risultati migliori, nel caso di classificazione tramite features in frequenza, si ottengono in assenza di una validazione incrociata e della PCA, come anche nel caso di features nel dominio del tempo. Confrontando i risultati con quelli ottenuti nel caso precedente, sono, in generale, peggiori. Infatti la migliore percentuale, nel caso migliore, è pari al 91,7% di *accuracy*, a differenza del caso del dominio del tempo in cui figurava un algoritmo con il 100% di *accuracy*. Anche tra risultati senza PCA e risultati con PCA non vi è una differenza significativa. In conclusione, per il caso di studio specifico, la soluzione migliore è di classificare il modello a partire dal calcolo delle features nel dominio temporale e, nel dettaglio, non è necessario effettuare una PCA prima della classificazione.

Il Diagnostic Features Designer non è altro che uno strumento di ausilio alla classificazione: in questo tool si possono addestrare i modelli di dati a disposizione per avere un'idea di quale possa essere la migliore tecnica di classificazione da applicare nel caso di diagnosi real-time del dataset. Una volta ottenuti i risultati in questo tool, l'obiettivo è di impostare un sistema di classificazione tramite strumenti differenti (nel caso specifico Labview tramite il tool Labview Analytics and Machine Learning Toolkit) il quale permetta di classificare i modelli ottenuti applicando uno degli algoritmi visti nel Diagnostic Features Design. Nello specifico, Labview Analytics and Machine Learning Toolkit richiede che il dataset sia caricato in formato .csv.

Non è stato possibile proseguire lungo questa strada per due motivi:

- Impossibilità nell'implementare lo streaming delle features tra il PLC e il PC, perché le funzioni utilizzate in Twincat richiedono una cpu integrata nel PLC con architettura x64, x86, mentre il PLC disponibile monta un ARM Cortex™-A8 CPU (Target Platform ARMV7 su Twincat).
- Mancanza di tempo per trovare una soluzione alternativa.

4.4.2 Risultati per tutte le altre categorie di guasto

Ripetendo gli stessi passi svolti nel capitolo precedente per effettuare la classificazione del modello relativo al cuscinetto ammaccato rilevato con misurazione in direzione assiale in regime stazionario, vengono riportati i risultati relativi agli altri casi di studio. Verranno riportati solo i migliori risultati tra classificazione tramite features nel dominio del tempo e in quello della frequenza, e tra i casi in cui viene effettuata la validazione incrociata (ai vari livelli) o no:

1. Cuscinetto rumoroso, direzione assiale, regime stazionario:

I risultati, qualunque siano le features o i livelli di validazione incrociata, sono tutti ottimi. La classificazione che fornisce più alternative di ugual qualità è quella adoperata con features nel tempo e senza alcuna validazione:

1.1 ☆ Tree Last change: Fine Tree	Accuracy: 100.0% 5/5 features
1.2 ☆ Tree Last change: Medium Tree	Accuracy: 100.0% 5/5 features
1.3 ☆ Tree Last change: Coarse Tree	Accuracy: 100.0% 5/5 features
1.4 ☆ KNN Last change: Fine KNN	Accuracy: 100.0% 5/5 features
1.5 ☆ KNN Last change: Medium KNN	Accuracy: 88.9% 5/5 features
1.6 ☆ KNN Last change: Coarse KNN	Accuracy: 88.9% 5/5 features
1.7 ☆ KNN Last change: Cosine KNN	Accuracy: 88.9% 5/5 features
1.8 ☆ KNN Last change: Cubic KNN	Accuracy: 88.9% 5/5 features
1.9 ☆ KNN Last change: Weighted KNN	Accuracy: 100.0% 5/5 features

Figura 49: Risultati migliori ottenuti per cuscinetto rumoroso, direzione assiale, regime stazionario

2. Filo che tocca il rotore, direzione assiale, regime stazionario:

I risultati, qualunque siano le features o i livelli di validazione incrociata, sono tutti ottimi. La classificazione che fornisce più alternative di ugual qualità è quella adoperata con features nel tempo e senza alcuna validazione:

1.1 ☆ Tree	Accuracy: 100.0%
Last change: Fine Tree	8/8 features
1.2 ☆ Tree	Accuracy: 100.0%
Last change: Medium Tree	8/8 features
1.3 ☆ Tree	Accuracy: 100.0%
Last change: Coarse Tree	8/8 features
1.4 ☆ KNN	Accuracy: 100.0%
Last change: Fine KNN	8/8 features
1.5 ☆ KNN	Accuracy: 88.9%
Last change: Medium KNN	8/8 features
1.6 ☆ KNN	Accuracy: 88.9%
Last change: Coarse KNN	8/8 features
1.7 ☆ KNN	Accuracy: 88.9%
Last change: Cosine KNN	8/8 features
1.8 ☆ KNN	Accuracy: 88.9%
Last change: Cubic KNN	8/8 features
1.9 ☆ KNN	Accuracy: 100.0%
Last change: Weighted KNN	8/8 features

Figura 50: Risultati migliori ottenuti per filo su rotore, direzione assiale, regime. Stazionario

3. Calotta lenta, direzione assiale, regime stazionario:

In questo caso si ottengono i medesimi risultati sia se si utilizzano le features nel tempo che quelle in frequenza. Stesso discorso vale per i livelli di validazione incrociata. L'unica debolezza sta nella PCA, che porta a risultati peggiori del caso senza:

1.1 ☆ Tree	Accuracy: 100.0%
Last change: Fine Tree	5/5 features
1.2 ☆ Tree	Accuracy: 100.0%
Last change: Medium Tree	5/5 features
1.3 ☆ Tree	Accuracy: 100.0%
Last change: Coarse Tree	5/5 features
1.4 ☆ KNN	Accuracy: 100.0%
Last change: Fine KNN	5/5 features
1.5 ☆ KNN	Accuracy: 80.0%
Last change: Medium KNN	5/5 features
1.6 ☆ KNN	Accuracy: 80.0%
Last change: Coarse KNN	5/5 features
1.7 ☆ KNN	Accuracy: 80.0%
Last change: Cosine KNN	5/5 features
1.8 ☆ KNN	Accuracy: 80.0%
Last change: Cubic KNN	5/5 features
1.9 ☆ KNN	Accuracy: 100.0%
Last change: Weighted KNN	5/5 features

Figura 51: Risultati migliori ottenuti per calotta lenta, direzione assiale, regime stazionario

4. Cuscinetto ammaccato, direzione radiale, regime stazionario

In tal caso la migliore modalità con cui fare classificazione porta al 100% di precisione utilizzando come predittorile features nel tempo senza fare validazione incrociata:

1.1 ☆ Tree Last change: Fine Tree	Accuracy: 83.3% 3/3 features
1.2 ☆ Tree Last change: Medium Tree	Accuracy: 83.3% 3/3 features
1.3 ☆ Tree Last change: Coarse Tree	Accuracy: 83.3% 3/3 features
1.4 ☆ KNN Last change: Fine KNN	Accuracy: 100.0% 3/3 features
1.5 ☆ KNN Last change: Medium KNN	Accuracy: 70.8% 3/3 features
1.6 ☆ KNN Last change: Coarse KNN	Accuracy: 66.7% 3/3 features
1.7 ☆ KNN Last change: Cosine KNN	Accuracy: 75.0% 3/3 features
1.8 ☆ KNN Last change: Cubic KNN	Accuracy: 70.8% 3/3 features
1.9 ☆ KNN Last change: Weighted KNN	Accuracy: 100.0% 3/3 features

Figura 52: Risultati migliori ottenuti per cuscinetto ammaccato, direzione radiale, regime stazionario

5. Cuscinetto rumoroso, direzione radiale, regime stazionario:

I risultati migliori si hanno utilizzando le features nel tempo, e tra i vari livelli di validazione incrociata non vi è alcuna differenza nei risultati ottenuti. E' notevole il fatto che sfruttando le features in frequenza si ottenga, per ogni tecnica disponibile sul tool, una *accuracy* pari al 88,9%, il che la porta ad essere considerata come una valida alternativa qualora non fosse possibile estrarre le features nel dominio temporale in maniera opportuna:

1.1 ☆ Tree Last change: Fine Tree	Accuracy: 94.4% 5/5 features
1.2 ☆ Tree Last change: Medium Tree	Accuracy: 94.4% 5/5 features
1.3 ☆ Tree Last change: Coarse Tree	Accuracy: 94.4% 5/5 features
1.4 ☆ KNN Last change: Fine KNN	Accuracy: 100.0% 5/5 features
1.5 ☆ KNN Last change: Medium KNN	Accuracy: 88.9% 5/5 features
1.6 ☆ KNN Last change: Coarse KNN	Accuracy: 88.9% 5/5 features
1.7 ☆ KNN Last change: Cosine KNN	Accuracy: 88.9% 5/5 features
1.8 ☆ KNN Last change: Cubic KNN	Accuracy: 88.9% 5/5 features
1.9 ☆ KNN Last change: Weighted KNN	Accuracy: 100.0% 5/5 features

Figura 53: Risultati migliori ottenuti per cuscinetto rumoroso, direzione radiale, regime stazionario

6. Filo che tocca il rotore, direzione radiale, regime stazionario:

Si ottengono i risultati migliori e con la medesima percentuale di precisione sia con lo sfruttamento di features nel tempo sia in frequenza (sia con che senza PCA), a patto che

non vi sia alcuna validazione incrociata, grazie alla quale i risultati genericamente peggiorano:

1.1 ☆ Tree Last change: Fine Tree	Accuracy: 88.9% 5/5 features
1.2 ☆ Tree Last change: Medium Tree	Accuracy: 88.9% 5/5 features
1.3 ☆ Tree Last change: Coarse Tree	Accuracy: 88.9% 5/5 features
1.4 ☆ KNN Last change: Fine KNN	Accuracy: 100.0% 5/5 features
1.5 ☆ KNN Last change: Medium KNN	Accuracy: 88.9% 5/5 features
1.6 ☆ KNN Last change: Coarse KNN	Accuracy: 88.9% 5/5 features
1.7 ☆ KNN Last change: Cosine KNN	Accuracy: 88.9% 5/5 features
1.8 ☆ KNN Last change: Cubic KNN	Accuracy: 88.9% 5/5 features
1.9 ☆ KNN Last change: Weighted KNN	Accuracy: 100.0% 5/5 features

Figura 54: Risultati migliori ottenuti per filo su rotore, direzione radiale, regime stazionario

7. Calotta lenta, direzione radiale, regime stazionario:

Una percentuale del 100% si raggiunge nel caso di features nel tempo senza PCA e senza validazione incrociata. All'aumentare del livello di validazione i risultati tendono a peggiorare, eccezione fatta per le features in frequenza, le quali mantengono una percentuale del 80% a tutti i livelli, sia con che senza PCA applicata:

1.1 ☆ Tree Last change: Fine Tree	Accuracy: 90.0% 5/5 features
1.2 ☆ Tree Last change: Medium Tree	Accuracy: 90.0% 5/5 features
1.3 ☆ Tree Last change: Coarse Tree	Accuracy: 90.0% 5/5 features
1.4 ☆ KNN Last change: Fine KNN	Accuracy: 100.0% 5/5 features
1.5 ☆ KNN Last change: Medium KNN	Accuracy: 80.0% 5/5 features
1.6 ☆ KNN Last change: Coarse KNN	Accuracy: 80.0% 5/5 features
1.7 ☆ KNN Last change: Cosine KNN	Accuracy: 80.0% 5/5 features
1.8 ☆ KNN Last change: Cubic KNN	Accuracy: 80.0% 5/5 features
1.9 ☆ KNN Last change: Weighted KNN	Accuracy: 100.0% 5/5 features

Figura 55: Risultati migliori ottenuti per calotta lenta, direzione radiale, regime stazionario

8. Cuscinetto ammaccato, direzione assiale, transitorio

I risultati migliori si ottengono sfruttando le features nel dominio del tempo, sia con che senza applicare la PCA. Si raggiunge il 100% di precisione nella classificazione senza

effettuare alcuna validazione incrociata. Applicandola, i risultati vanno peggiorando all'aumentare degli insiemi in cui è diviso il dataset:

1.1 ☆ Tree Last change: Fine Tree	Accuracy: 87.5% 6/6 features
1.2 ☆ Tree Last change: Medium Tree	Accuracy: 87.5% 6/6 features
1.3 ☆ Tree Last change: Coarse Tree	Accuracy: 87.5% 6/6 features
1.4 ☆ KNN Last change: Fine KNN	Accuracy: 100.0% 6/6 features
1.5 ☆ KNN Last change: Medium KNN	Accuracy: 66.7% 6/6 features
1.6 ☆ KNN Last change: Coarse KNN	Accuracy: 66.7% 6/6 features
1.7 ☆ KNN Last change: Cosine KNN	Accuracy: 66.7% 6/6 features
1.8 ☆ KNN Last change: Cubic KNN	Accuracy: 66.7% 6/6 features
1.9 ☆ KNN Last change: Weighted KNN	Accuracy: 100.0% 6/6 features

Figura 56: Risultati migliori ottenuti per cuscinetto ammaccato, direzione assiale, regime transitorio

9. Cuscinetto rumoroso, direzione assiale, transitorio: Si ottengono risultati del 100% sfruttando le feature nel tempo, sia con la PCA attiva sia disattiva, in ogni livello di validazione incrociata, eccetto che per il caso in cui si hanno 5 insiemi di validazione, in cui i risultati sono peggiori. Sfruttando le features in frequenza si ottiene un 88,9% di *accuracy* per ogni sotto caso di studio:

1.1 ☆ Tree Last change: Fine Tree	Accuracy: 100.0% 5/5 features
1.2 ☆ Tree Last change: Medium Tree	Accuracy: 100.0% 5/5 features
1.3 ☆ Tree Last change: Coarse Tree	Accuracy: 100.0% 5/5 features
1.4 ☆ KNN Last change: Fine KNN	Accuracy: 100.0% 5/5 features
1.5 ☆ KNN Last change: Medium KNN	Accuracy: 88.9% 5/5 features
1.6 ☆ KNN Last change: Coarse KNN	Accuracy: 88.9% 5/5 features
1.7 ☆ KNN Last change: Cosine KNN	Accuracy: 88.9% 5/5 features
1.8 ☆ KNN Last change: Cubic KNN	Accuracy: 88.9% 5/5 features
1.9 ☆ KNN Last change: Weighted KNN	Accuracy: 100.0% 5/5 features

Figura 57: Risultati migliori ottenuti per cuscinetto rumoroso, direzione assiale, regime transitorio

10. Filo che tocca il rotore, direzione assiale, transitorio:

Il caso migliore si ha con le features nel tempo senza PCA, sia con validazione (ad ogni livello) che senza:

1.1 ☆ Tree Last change: Fine Tree	Accuracy: 88.9% 5/5 features
1.2 ☆ Tree Last change: Medium Tree	Accuracy: 88.9% 5/5 features
1.3 ☆ Tree Last change: Coarse Tree	Accuracy: 88.9% 5/5 features
1.4 ☆ KNN Last change: Fine KNN	Accuracy: 100.0% 5/5 features
1.5 ☆ KNN Last change: Medium KNN	Accuracy: 88.9% 5/5 features
1.6 ☆ KNN Last change: Coarse KNN	Accuracy: 88.9% 5/5 features
1.7 ☆ KNN Last change: Cosine KNN	Accuracy: 88.9% 5/5 features
1.8 ☆ KNN Last change: Cubic KNN	Accuracy: 88.9% 5/5 features
1.9 ☆ KNN Last change: Weighted KNN	Accuracy: 100.0% 5/5 features

Figura 58: Risultati migliori ottenuti per filo su rotore, direzione assiale, regime transitorio

11. Calotta lenta, direzione assiale, transitorio:

Eccetto il caso senza validazione incrociata con features nel tempo, in cui si raggiunge il 100% di *accuracy*, se si decide di adottare la validazione incrociata allora conviene optare per l'utilizzo delle features in frequenza. Comunque il caso ottimo del 100% viene raggiunto solo nel primo caso:

1.1 ☆ Tree Last change: Fine Tree	Accuracy: 90.0% 5/5 features
1.2 ☆ Tree Last change: Medium Tree	Accuracy: 90.0% 5/5 features
1.3 ☆ Tree Last change: Coarse Tree	Accuracy: 90.0% 5/5 features
1.4 ☆ KNN Last change: Fine KNN	Accuracy: 100.0% 5/5 features
1.5 ☆ KNN Last change: Medium KNN	Accuracy: 80.0% 5/5 features
1.6 ☆ KNN Last change: Coarse KNN	Accuracy: 80.0% 5/5 features
1.7 ☆ KNN Last change: Cosine KNN	Accuracy: 80.0% 5/5 features
1.8 ☆ KNN Last change: Cubic KNN	Accuracy: 80.0% 5/5 features
1.9 ☆ KNN Last change: Weighted KNN	Accuracy: 100.0% 5/5 features

Figura 59: Risultati migliori ottenuti per calotta lenta, direzione. assiale, regime transitorio

12. Cuscinetto ammaccato, direzione radiale, transitorio:

Anche ora il risultato migliore è raggiunto nel caso di features temporali e senza validazione incrociata:

1.1 ☆ Tree Last change: Fine Tree	Accuracy: 91.7% 3/3 features
1.2 ☆ Tree Last change: Medium Tree	Accuracy: 91.7% 3/3 features
1.3 ☆ Tree Last change: Coarse Tree	Accuracy: 91.7% 3/3 features
1.4 ☆ KNN Last change: Fine KNN	Accuracy: 100.0% 3/3 features
1.5 ☆ KNN Last change: Medium KNN	Accuracy: 66.7% 3/3 features
1.6 ☆ KNN Last change: Coarse KNN	Accuracy: 66.7% 3/3 features
1.7 ☆ KNN Last change: Cosine KNN	Accuracy: 87.5% 3/3 features
1.8 ☆ KNN Last change: Cubic KNN	Accuracy: 75.0% 3/3 features
1.9 ☆ KNN Last change: Weighted KNN	Accuracy: 100.0% 3/3 features

Figura 60: Risultati migliori ottenuti per cuscinetto ammaccato, direzione radiale, regime transitorio

13. Cuscinetto rumoroso, direzione radiale, transitorio:

Nel caso senza validazione incrociata, si ottiene un risultato pari al 100% di *accuracy* se si utilizzano features temporali. In tutti gli altri casi i risultati migliori sono relativi all'impiego delle features in frequenza. Aumentando il livello di validazione da 5 a massimo il risultato rimane lo stesso:

1.1 ☆ Tree Last change: Fine Tree	Accuracy: 94.4% 5/5 features
1.2 ☆ Tree Last change: Medium Tree	Accuracy: 94.4% 5/5 features
1.3 ☆ Tree Last change: Coarse Tree	Accuracy: 94.4% 5/5 features
1.4 ☆ KNN Last change: Fine KNN	Accuracy: 100.0% 5/5 features
1.5 ☆ KNN Last change: Medium KNN	Accuracy: 88.9% 5/5 features
1.6 ☆ KNN Last change: Coarse KNN	Accuracy: 88.9% 5/5 features
1.7 ☆ KNN Last change: Cosine KNN	Accuracy: 88.9% 5/5 features
1.8 ☆ KNN Last change: Cubic KNN	Accuracy: 88.9% 5/5 features
1.9 ☆ KNN Last change: Weighted KNN	Accuracy: 100.0% 5/5 features

Figura 61: Risultati migliori ottenuti per cuscinetto rumoroso, direzione radiale, regime transitorio

14. Filo che tocca il rotore, direzione radiale, transitorio:

Si ottiene il 100% di *accuracy*, sia se si sfruttano le features nel tempo che quelle in frequenza, solo nel caso senza validazione. Nei casi successivi si ottiene il 100% solo se si sfruttano le features nel tempo senza PCA applicata:

1.1 ☆ Tree Last change: Fine Tree	Accuracy: 100.0% 5/5 features
1.2 ☆ Tree Last change: Medium Tree	Accuracy: 100.0% 5/5 features
1.3 ☆ Tree Last change: Coarse Tree	Accuracy: 100.0% 5/5 features
1.4 ☆ KNN Last change: Fine KNN	Accuracy: 100.0% 5/5 features
1.5 ☆ KNN Last change: Medium KNN	Accuracy: 88.9% 5/5 features
1.6 ☆ KNN Last change: Coarse KNN	Accuracy: 88.9% 5/5 features
1.7 ☆ KNN Last change: Cosine KNN	Accuracy: 88.9% 5/5 features
1.8 ☆ KNN Last change: Cubic KNN	Accuracy: 88.9% 5/5 features
1.9 ☆ KNN Last change: Weighted KNN	Accuracy: 100.0% 5/5 features

Figura 62: Risultati migliori ottenuti per filo su rotore, direzione radiale, regime transitorio

15. Calotta lenta, direzione radiale, transitorio:

Il 100% si raggiunge solo nel caso senza validazione incrociata con features nel dominio del tempo e senza PCA applicata. Aumentando il livello di validazione i risultati scendono del 20% circa in ogni tecnica applicata per la classificazione:

1.1 ☆ Tree Last change: Fine Tree	Accuracy: 90.0% 4/4 features
1.2 ☆ Tree Last change: Medium Tree	Accuracy: 90.0% 4/4 features
1.3 ☆ Tree Last change: Coarse Tree	Accuracy: 90.0% 4/4 features
1.4 ☆ KNN Last change: Fine KNN	Accuracy: 100.0% 4/4 features
1.5 ☆ KNN Last change: Medium KNN	Accuracy: 80.0% 4/4 features
1.6 ☆ KNN Last change: Coarse KNN	Accuracy: 80.0% 4/4 features
1.7 ☆ KNN Last change: Cosine KNN	Accuracy: 80.0% 4/4 features
1.8 ☆ KNN Last change: Cubic KNN	Accuracy: 80.0% 4/4 features
1.9 ☆ KNN Last change: Weighted KNN	Accuracy: 100.0% 4/4 features

Figura 63: Risultati migliori ottenuti per calotta lenta, direzione radiale, regime transitorio

In conclusione a quanto riportato nei diversi casi, risulta conveniente estrarre le feature del segnale nel dominio del tempo e la validazione incrociata peggiora il risultato nel singolo dataset. In alcuni casi, l'analisi delle features in frequenza e una successiva classificazione basata su tali features come predittori può portare a risultati buoni ma che non raggiungono mai l'eccellenza. Essendo fondamentale riuscire ad effettuare una classificazione ottimale, è consigliato utilizzare algoritmo che riportino il massimo di accuratezza: nei diversi casi gli algoritmo che più hanno rispettato tale esigenza sono il KNN e sue varianti.

5 CONCLUSIONE

Alla luce di quanto discusso nei precedenti paragrafi si è arrivati a delineare un percorso di classificazione del modello ben definito, per quanto riguarda la diagnosi offline dei guasti: si parte dall'acquisizione dei dati vibrazionali tramite accelerometro piezoelettrico, passando per un sistema di streaming tra PLC e PC, fino ad arrivare all'elaborazione del segnale su Matlab tramite il tool Diagnostic Feature Designer. Il tool è uno strumento di ausilio per chi si occupa di diagnosticare i guasti che possono comparire sul motore, infatti permette di estrarre, selezionare ed elaborare le features del segnale in modo da effettuare una classificazione del modello considerato. Tale classificazione restituisce, in valori percentuali, l'accuratezza con la quale l'algoritmo di classificazione riesce a discriminare le classi di funzionamento del modello. Una volta analizzati i risultati, è possibile replicare la classificazione anche online, utilizzando le stesse features impiegate nel Diagnostic Feature Designer e l'algoritmo di classificazione che restituisce la percentuale più alta.

Purtroppo quest'ultima parte è stata impossibile da trattare nella presente tesi, a causa dei problemi discussi all'interno della trattazione. Nel dettaglio, è stato ritenuto non necessario approfondire gli algoritmi di calcolo delle features in Twincat3, alcuni dei quali sono stati codificati a scopo di test su un segnale simulato (POU di calcolo delle features mostrata nel capitolo 4.2), data l'impossibilità di poterne verificare l'effettivo funzionamento su un dataset di dati vibrazionali reali, a causa dell'incompatibilità dell'hardware riscontrata durante lo svolgimento della tesi. Per potenziare il sistema diagnostico online sarebbe necessario utilizzare, quindi, un hardware compatibile con le funzionalità richieste dall'ambiente di sviluppo Twincat, così da poter elaborare le features online. Inoltre si potrebbe testare la bontà della classificazione su Matlab implementando un sistema di classificazione online su LabView, come già suggerito nei capitoli precedenti.

Concludendo, il lavoro svolto ha riportato ottimi risultati in termini di diagnostica offline, per la quale è comunque necessario approfondire la bontà dei risultati utilizzando il modello ricavato dalla classificazione su Matlab, come ulteriore di modello delle classificazioni successive (aspetto non trattato nel progetto). Di contro, per la classificazione online è stato possibile solamente elaborare una simulazione del reale funzionamento dell'intero processo, portando comunque alla codifica di algoritmi che possano estrarre features di interesse e, soprattutto, calcolare la Fast Fourier Transform del segnale in esame. A causa di problemi di tempistiche nella fornitura dell'hardware non è stato possibile approfondire il discorso citato precedentemente.

6 BIBLIOGRAFIA

- 1) **R.Iserman**, “*Fault-Diagnosis Systems: An Introduction from Fault Detection to Fault tolerance*”, Springer - Verlag Berlin, 2006
- 2) **L.H.Chiong,R.D.Broatz,E.L.Russel**, “*Fault Detection and Diagnosis in Industrial Systems*”, Springer – Verlag London, 2000
- 3) **A.Bellini**, “*Elettronica Industriale 2 – Azionamenti con motore in corrente alternata*”,Aracne Editrici, Roma 2006
- 4) **A.E.Fitzgerald, C.Kingsley Jr.,A.Kusko**, “*Macchine Elettriche*”, Franco Angeli Editore, 2006
- 5) **G.Bertoni,M.E.Penati,S.Simonini**, “*I Componenti dell’Automazione*”, Esculapio 2001
- 6) **Elektro.it**, <http://www.elektro.it/>, ultimo accesso Ottobre 2020
- 7) **Adam Glowacz** (2019) - *Fault diagnosis of single-phase induction motor based on acoustic signals, Mechanical Systems and Signal Processing* vol 117, pp. 65-80, 2019
- 8) **Ting Yang, Haibo Pen,Zhaoxia Wang, Che Sau Chang** - “*Feature Knowledge Based Fault Detection of Induction Motors Through the Analysis of Stator Current Data*”, *IEEE Transactions on Instrumentation and Measurement*, vol.65, no.3, pp. 549-558, 2016
- 9) **Katalin Agoston**, “*Fault Detection of the Electrical Motors based on Vibration analysis*”, Conference Interdisciplinarity in Engineering, Romania, 2014
- 10) **Beckhoff Information System**,<https://infosys.beckhoff.com/>, ultimo accesso Ottobre 2020