Dipartimento di Ingegneria Meccanica ed Aerospaziale

Tesi di Laurea Magistrale in Ingegneria Meccanica

Anno Accademico 2022/2023

Caratterizzazione di un banco prova per test HiL di powertrain ibridi ed elettrici per veicoli NRMM e sviluppo in test SiL di una GUI per acquisirne dati CAN-Bus

Relatori

Prof. Aurelio Somà

Dott. Francesco Mocera

Candidato

Giovanni Mantini

Matricola: 291446

Abstract

Fino ad alcuni anni fa tutti i veicoli terrestri utilizzavano motori a combustione interna dato l'ottimale rapporto peso/potenza in relazione al costo contenuto sia di fabbricazione sia di utilizzo. Tale soluzione ha però contribuito per anni ad aumentare i livelli di CO_2 in atmosfera e ad emettere in atmosfera gas inquinanti come gli NO_X e i prodotti di combustione incompleta, determinando un aumento delle temperature stagionali medie e contribuendo a causare i noti eventi catastrofici, dannosi sia per l'ambiente sia per gli insediamenti umani.

Questi avvenimenti hanno determinato l'introduzione di normative sempre più stringenti, volte alla riduzione delle emissioni anche dei veicoli. Per questi ultimi le norme fanno però riferimento solo alle emissioni misurate allo scarico: questo significa che un veicolo, ad esempio con alimentazione completamente elettrica, è definibile ad emissioni zero nonostante la produzione di elettricità che vi è a monte con i sistemi convenzionali comporti comunque l'immissione in atmosfera di gas nocivi.

In una prima fase, per soddisfare i requisiti necessari all'omologazione, per i veicoli terrestri è stato sufficiente adottare sistemi di post-trattamento dei gas di scarico. Con l'introduzione via via di vincoli sempre più stringenti, queste strategie però non sono più sufficienti.

Nel caso di veicoli dedicati all'agricoltura, o più in generale per i Non-Road Mobile Machineries (NRMM), vi è inoltre la problematica che i sistemi dedicati al trattamento dei gas uscenti dai motori a combustione occupano volumi non indifferenti, spazio che è quindi difficile da trovare a bordo del veicolo, specie nel caso di veicoli specializzati che devono sottostare a vincoli di ingombro per poter eseguire lavorazioni in particolari tipologie di coltivazioni.

Parallelamente al mondo dei veicoli per il trasporto di persone, quindi anche per i veicoli da lavoro ci si è orientati all'installazione di motori elettrici, con lo scopo di affiancarlo al motore termico (nel caso di veicoli ibridi) durante le condizioni di lavoro più critiche (e riducendo quindi i consumi di combustibile) evitando così che quest'ultimo vada a lavorare in punti a bassa efficienza, oppure di sostituirlo completamente all'unità termica in caso di veicoli full-electric.

Le macchine elettriche possono inoltre lavorare sia da motori (e quindi fornire coppia motrice) sia come generatori (assorbendo coppia e ritrasformandola in energia elettrica), consentendo quindi il recupero di quel plus di energia che verrebbe altrimenti inutilmente dissipata (si pensi ad esempio alle fasi di frenata oppure alle lavorazioni eseguite in condizioni di pendenza negativa). Questa energia recuperata può quindi essere stoccata all'interno di apposite batterie, caratterizzate però da una capacità di accumulare carica limitata e tendenzialmente proporzionale al loro volume.

Si comprende quindi da qui quella che è la problematica comune a tutti i veicoli ibridi ed elettrici: l'ingombro ed il peso eccessivo dovuti alla necessità di installare a bordo veicoli queste batterie. La sfida diventa quindi quella di dover trovare la giusta calibrazione tra la necessità di ridurre le emissioni e il non snaturare completamente le caratteristiche dei veicoli: bisogna quindi riuscire ad installare a bordo il motore della taglia di potenza corretta in relazione alle necessità che quel veicolo deve soddisfare nonché adottare delle strategie di controllo software specifiche all'ottimizzazione delle fasi di scarica e di carica delle batterie, evitando quindi di utilizzarne eccessivamente in numero o di dimensioni troppo elevate.

Nel caso dei veicoli NRMM questa sfida è ulteriormente gravosa, dato che tipicamente questi mezzi sono utilizzati in modo intensivo durante una giornata lavorativa (anche per più delle otto ore lavorative tipiche giornaliere): è necessario allora implementare strategie ancora più fini ed efficienti sia durante le fasi di utilizzo sia durante le fasi di ricarica del veicolo.

Per testare quindi la strategia di controllo nonché tutta la componentistica hardware (motori, azionamenti, batterie) riproducendo e simulando tutti i possibili scenari di lavoro che tali veicoli possono trovarsi a fronteggiare, è quindi necessario avere un banco di prova del powertrain, attraverso il quale evitare di poter testare il veicolo, attraverso i field test, solo quando quest'ultimo è stato effettivamente prodotto.

E' stato quindi messo a punto un banco prova in scala 1:1 attraverso il quale è possibile effettuare test Hardware in the Loop (HiL), ovvero testare da un lato il software della centralina (simulato su un PC in ambiente Simulink) e dall'altro rilevare come queste strategie software rispondano sull'hardware fisico, andando ad effettuare delle misure di coppia, velocità e temperatura sul motore nonché dello stato di carica (SOC) delle batterie, componenti che poi verranno montati sul veicolo una volta costruito.

Il banco è stato costruito secondo una logica di elevata modularità, che consente di testare diverse configurazioni di powertrain e fino ad una taglia di potenza di 80 kW, vale a dire la potenza di targa del freno-generatore. Il banco in questione si caratterizza inoltre come banco a ricircolo di potenza elettrica in quanto l'energia immessa nel sistema dal motore, a meno delle perdite elettro-meccaniche dell'intero sistema, viene rigettata tramite il freno-generatore all'interno di batterie che chiudono il loop di potenza.

L'utilizzo di un banco prova di questo tipo consente di ridurre notevolmente tempi e costi necessari allo sviluppo dei veicoli di interesse grazie alla versatilità delle simulazioni HiL che consentono di valutare l'effetto prodotto da ogni modifica, effettuate in questa fase semplicemente sul software di controllo.

Essendo stato utilizzato per la prima volta in concomitanza a questa attività di tesi, si è quindi reso necessario, prima di porre dei motori sotto test, andare a caratterizzare ed evidenziare le perdite meccaniche associate alla linea di trasmissione del banco stesso, con il fine di risalire la linea di potenza ed estrarre in maniera corretta i dati caratteristici dei motori montati, ovvero la coppia da questi erogati a vari step di velocità.

Sono quindi state eseguite delle acquisizioni sperimentali sul banco prova prima a vuoto (senza alcun motore montato) per verificare la potenza (o coppia) assorbita dalla linea di trasmissione differenti velocità e successivamente, una volta note le caratteristiche intrinseche del sistema, dei test sotto carico (ovvero con un motore in prova montato) attraverso i quali si è andati a:

1. Eseguire la BEMF e a ricostruire la curva caratteristica del motore in prova;
2. Verificare l'effettivo funzionamento del circuito di raffreddamento predisposto per l'azionamento ed il motore sotto test valutando le curve di Temperatura con l'avanzare del tempo di prova;
3. Verificare le perdite di efficienza dei sistemi a banco al variare della Temperatura e all'avanzare del tempo di prova.

Inoltre, dato che i vari sistemi che compongono il banco sono posti in comunicazione tra loro e con il PC (su cui viene eseguito il controllo) tramite un protocollo di comunicazione denominato CAN-BUS, standard utilizzato in tutto il settore automotive e non solo, si è provveduto alla realizzazione di una Interfaccia grafica (GUI) che consente di inviare gli input di comando al software che gestisce la centrina e di ricevere i messaggi CAN provenienti dai vari sottosistemi del banco, in modo tale da estrapolare e salvare i dati delle simulazioni svolte in maniera più efficiente.

Questa GUI per l'invio e la ricezione di messaggi CAN-BUS è stato testato, per questa attività di tesi, all'interno di un modello di simulazione realizzato in Matlab-Simulink che replica il comportamento del banco interamente su software: è stato quindi eseguito un test SiL (Software in the Loop) del software creato, prima di poterlo utilizzare in sicurezza direttamente collegato alla rete CAN-BUS del banco prova. Questo test è stato eseguito con alcune semplificazioni rispetto a quello che sarà poi il collegamento effettiva della GUI al banco prova fisico, consentendo comunque nel contempo di effettuare delle valutazioni sulle prestazioni dello stesso attraverso il suo modello virtuale.

Gli sviluppi futuri di questa attività saranno quindi quelli di eseguire le opportune modifiche al codice della GUI per renderlo interamente compatibile con la centralina e la linea CAN fisica del banco, in modo tale da rendere il controllo e l'acquisizione dati più ordinata ed efficace per i prossimi utilizzi del banco prova.

Indice

Indice delle Figure

Indice delle Tabelle

Capitolo 1 : Le emissioni inquinanti degli ICE e la progressiva elettrificazione di veicoli NRMM

Problematiche relative ai gas emessi dai motori a combustione interna (ICE) di veicoli

Le questioni relative alla qualità dell'aria, soprattutto nei luoghi con una elevata densità abitativa quali città e metropoli, costituisce una delle tematiche più delicate degli ultimi decenni. L'attenzione su tali aspetti risale in realtà già alle prime rivoluzioni industriali, anni in cui si iniziarono a notare gli effetti negativi che i gas derivanti dalla combustione di idrocarburi fossili potessero avere sulla salute dell'uomo: si era infatti riscontrato uno stretto legale tra l'aumento del tasso di mortalità in relazione all'esposizione ad aria inquinata. In particolare, quest'ultima può portare a ictus, problemi cardiaci e diverse malattie polmonari.
Oltre all'impatto anche mortale sull'uomo, l'inquinamento ha effetti malevoli anche sull'economia ed in particolare può provocare: perdita della capacità produttiva (causata sempre dai decessi), aumento dei costi sanitari ed effetti sulla agricoltura.

Si definiscono inquinanti primari quelli generati direttamente dall'attività umana, mentre si definiscono secondari quelli prodotti dalla reazione tra gli inquinanti primari, la luce del sole ed altri componenti atmosferici. Tra gli inquinanti secondari più significati vi sono l'ozono troposferico, che deriva da processi fotochimici che coinvolgono gli NO_x ed i composti organici volatili, lo smog fotochimico (in cui l'ozono è uno dei principali componenti) e le piogge acide.

Nella Figura 1.1 di seguito vengono riportati i più importanti inquinanti prodotti dalle attività umane e il loro impatto sull'atmosfera terrestre.

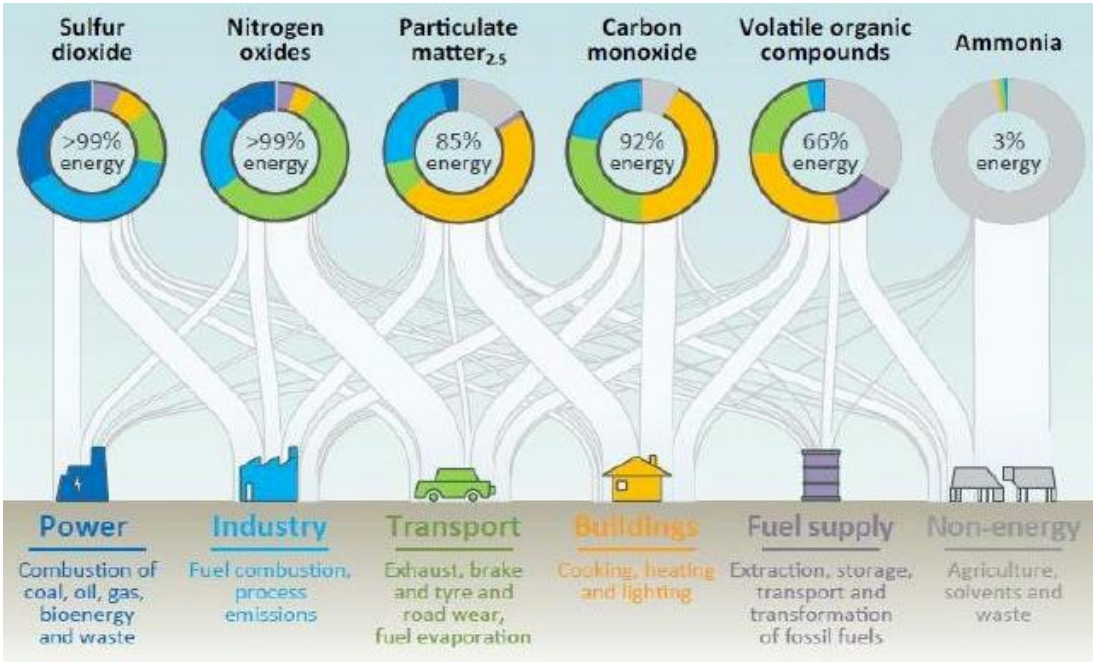


Figura 1.1: ***Effetto delle principali attività umane sull'inquinamento dell'atmosfera terrestre***

Inizialmente, i primi tentativi per salvaguardare la salute delle persone si limitarono allo spostamento di fabbriche e industrie nelle lontananze dei grandi centri urbani.

Tuttavia, con l'avvento e la diffusione dei veicoli a combustione a interna, si è ripresentata la problematica dell'esposizione dell'uomo a fumi e gas potenzialmente tossici. E' quindi progressivamente nata la necessità di imporre normative sempre più rigide sulle emissioni inquinanti: a livello globale, il tema dell'inquinamento dell'aria è quindi finito sotto i riflettori a partire dagli anni '70, con la stesura negli anni successivi di accordi internazionali (Protocollo di Kyoto) con i quali le nazioni partecipanti si vincolarono a contenere il loro impatto ambientale per quanto riguarda le emissioni in atmosfera.

Entrando nello specifico invece delle normative riguardanti le emissioni dei veicoli, queste hanno portato in primis all'adozione dei sistemi di post-trattamento dei gas di scarico per ridurne la nocività, ed in seguito (ed in particolar modo negli ultimi anni) all'adozione di nuove tecnologie, come motori elettrici e pacchi batterie, con il fine di soddisfare i diversi requisiti sulle emissioni. Per poter affrontare in maniera esaustiva il tema dell'elettrificazione dei veicoli, ed in particolar modo in questa trattazione l'elettrificazione dei veicoli NRMM, è anche importante comprendere quali sono i principali inquinanti prodotti dai motori a combustione interna.
Secondo la reazione di combustione ideale, infatti, gli unici prodotti derivanti dalla combustione di un generico idrocarburo sono acqua ed anidride carbonica, entrambi elementi che non comportano alcun rischio per la salute dell'essere umano:

$$C_xH_y + O_2 = H_2O + CO_2 \tag{1.1}$$

In un processo di combustione reale però, come quello che avviene all'interno delle camere di combustione dei motori tradizionali, la reazione di combustione libera numerosi inquinanti che possono o essere direttamente dannosi per la salute dell'uomo o essere altamente reattivi con l'atmosfera, generando altri composti tossici.
Si riportano quindi in Figura 1.2 ed in Tabella 1.1 in disamina quelli più comuni e presenti in maggiore quantità nella combustione all'interno dei motori termici:

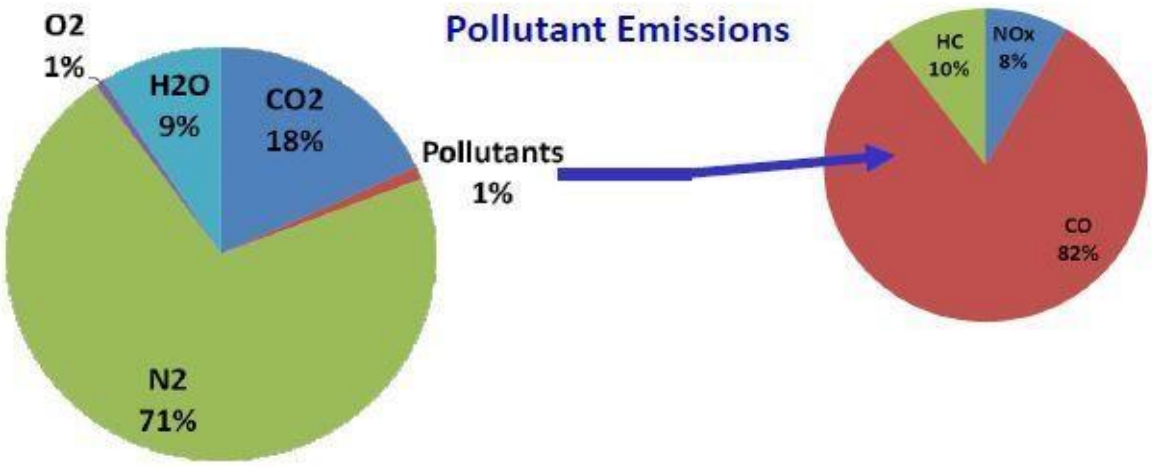


Figura 1.2: ***Composizione percentuale dei gas emessi dallo scarico in atmosfera***

Sostanza Inquinante	Descrizione del fenomeno	Effetti su atmosfera e salute
Idrocarburi incombusti HC	Durante la combustione, non tutto il combustibile riesce tipicamente a prendere parte alla reazione per diverse cause, che possono essere: presenza di interstizi, di olio lubrificante che assorbe combustibile vaporizzato, di fenomeni di spegnimento della fiamma, di undermixing.	Gli HC hanno un effetto negativo sull'ambiente poiché possono dare origine all'effetto noto come *smog fotochimico.* Reagendo con l'atmosfera portano alla formazione di ulteriori composti cancerogeni (come la formaldeide) e molto pericolosi per la salute dell'uomo
Monossido di carbonio CO	La CO è emessa principalmente dai motori ad accensione comandata, dal momento che è causata dall'incompleta combustione in miscele ricche. La CO può anche derivare dalla dissociazione di CO_2 che può avvenire ad alte temperature (T > 1900 K).	Il monossido di carbonio è una emotossina, ovvero se inalato non consente all'ossigeno di legarsi all'emoglobina. Pertanto risulta pericoloso per la salute umana.
Particolato PM	Il particolato è sostanzialmente un aerosol costituito da una serie di particelle solide, come le particelle carboniose o le ceneri, da idrocarburi condensati o assorbiti e infine da solfati. Le particelle possono avere dimensioni e composizione differenti.	Il PM ha un impatto negativo sia sul clima che sulla salute umana. In particolar modo le particelle molto fini (nanoparticelle, dimensioni dell'ordine delle decine di nm) risultano molto pericolose.
Ossidi di azoto NO_x	L'azoto normalmente non prende parte a reazioni di combustione perché inerte. Le alte temperature in camera di combustione (T > 2000 K) possono decomporre le molecole di ossigeno e azoto in O ed N, le quali si combinano dando luogo a molecole di NO e NO_2.	Gli NO_x hanno impatto negativo sulla salute degli esseri umani (il gas è causa di irritazioni respiratorie ed ha capacità di reagire con l'atmosfera generando ozono, altra fonte di patologie respiratorie) e per l'ambiente (gas di questo tipo emessi ad alta quota reagiscono con l'ozono formando ossigeno e riducendo lo strato ozono-protettivo dai raggi UV.
Anidride carbonica CO_2	A differenza degli altri composti sopra citati, l'anidride carbonica non è propriamente un gas inquinante in quanto non è né dannoso per la salute né un prodotto indesiderato della reazione.	La CO_2 è responsabile diretta dell'effetto serra, ovvero crea una specie di velo nell'atmosfera che non consente ai raggi UV di disperdersi causando un innalzamento delle temperature medie con conseguenti effetti catastrofici sul clima e, di conseguenza, sull'uomo.

Tabella 1.1: ***Descrizione degli inquinanti primari emessi allo scarico da motori a combustione interna***

Per ridurre gli effetti sulla salute dell'uomo è stato quindi necessario un massiccio intervento legislativo per migliorare la qualità dell'aria: per quanto riguarda i veicoli stradali e non stradali (NRMM), seppur attraverso delle legislazioni differenti, i vari Stati si sono allineati sui valori limite ammessi per i vari inquinanti. Nel paragrafo a seguire si esaminerà la casistica dei veicoli NRMM.

Normative inerenti le emissioni inquinanti di veicoli NRMM (Non Road Mobile Machineries) e sistemi EAS (Emission Aftertreatment System) di motori Diesel

Viene definita Macchina Mobile Non Stradale una qualsiasi macchina mobile, apparecchiatura trasportabile o veicolo, dotata o meno di carrozzeria e/o ruote, non destinata al trasporto di passeggeri o merci su strada, ivi comprese le macchine installate su telai di veicoli in realtà destinati al trasporto passeggeri o merci su strada. Data la definizione piuttosto generale, questa comprende quindi una vasta gamma di veicoli: macchine da giardinaggio, macchine da cantiere, macchine agricole, locomotive, automotrici, navi e così via.

A causa della diversa destinazione d'uso degli NRMM, è stato necessario introdurre una normativa specifica per tale settore: per i veicoli tradizionali gli standard omologativi prendono il nome di "Euro" seguito da un numero crescente al variare della normativa, mentre per gli NRMM si utilizza lo standard omologativo "Stage" seguito da un numero romano crescente al variare della normativa. La principale differenza tra i due standard normativi sopra citati sta nel fatto che per la "Stage" è previsto il testing solo del gruppo propulsivo e non dell'intero veicolo, questo perché uno stesso propulsore può equipaggiare diversi macchinari, che svolgono attività anche completamente differenti tra loro.
La nuova regolamentazione "Stage V", entrata in vigore a partire dal 1° Gennaio 2019, prevede la definizione di 10 categorie di motori, suddivisi per tipologia di applicazione delle macchine e per potenza, includendo anche ICE con potenze inferiori ai 19 kW e superiori ai 560 kW:

- **NRSh**: Motori portatili ad accensione comandata con una potenza di riferimento inferiore a 19 kW, destinati esclusivamente a essere utilizzati in macchine portatili.
- **NRS**: Motori ad accensione comandata con una potenza di riferimento inferiore a 56 kW e non inclusi nella categoria NRSh.
- **NRE**: Motori per macchine mobili non stradali. Sono compresi in questa categoria anche i motori con una potenza di riferimento inferiore a 560 kW utilizzati al posto di quelli rispondenti alla fase V delle categorie IWP, IWA, RLL o RLR.
- **NRG**: Motori con una potenza di riferimento superiore a 560 kW, destinati esclusivamente a essere utilizzati in gruppi elettrogeni; i motori per gruppi elettrogeni diversi da quelli che presentano tali caratteristiche sono inclusi nelle categorie NRE o NRS a seconda delle relative caratteristiche.
- **RLL**: Motori da utilizzare esclusivamente nelle locomotive, per la loro propulsione o destinati alla loro propulsione.
- **RLR**: Motori da utilizzare esclusivamente nelle automotrici, per la loro propulsione o destinati alla loro propulsione; motori utilizzati al posto dei motori stage V della categoria RLL.
- **IWP**: Motori destinati esclusivamente ad essere utilizzati su navi per la navigazione interna, per la propulsione diretta o indiretta, con potenza di riferimento pari o superiore a 19 kW.
- **IWA**: Motori ausiliari da utilizzare esclusivamente su navi della navigazione interna e con una potenza di riferimento pari o superiore a 19 kW.
- **SMB**: Motori ad accensione comandata destinati esclusivamente a essere utilizzati in motoslitte; i motori per motoslitte diversi dai motori ad accensione comandata sono inclusi nella categoria NRE.

- **ATS**: Motori ad accensione comandata destinati esclusivamente a essere utilizzati in ATV e SbS; i motori per ATV e SbS diversi dai motori ad accensione comandata sono inclusi nella categoria NRE.

La normativa NRMM stabilisce i limiti di emissione per tutte le classi di motori, stabilendo anche le procedure che i costruttori di motori devono seguire al fine di ottenerne l'omologazione.
Le modalità di determinazione delle emissioni di inquinanti gassosi e particolato inquinante prodotti dal motore sottoposto a prova prevedono l'applicazione di due cicli di test:

- **Ciclo NRSC (ciclo stazionario non stradale)**: il motore a caldo viene fatto lavorare a diverse combinazioni di velocità angolare e carico, le emissioni vengono misurate nelle varie condizioni stazionarie. Per ogni inquinante la combinazione dei valori misurati e di fattori di peso per ciascun punto di lavoro deve soddisfare i limiti imposti dalla normativa.
- **Ciclo NRTC (ciclo transitorio non stradale)**: il motore viene sottoposto ad un profilo di carico dinamico. Il test viene eseguito due volte, una prima volta con il motore a freddo e la seconda con il motore a caldo, le emissioni misurate nei due casi vengono combinate a dei fattori di peso (10% a freddo, 90% a caldo) e devono soddisfare i limiti imposti dalla normativa.

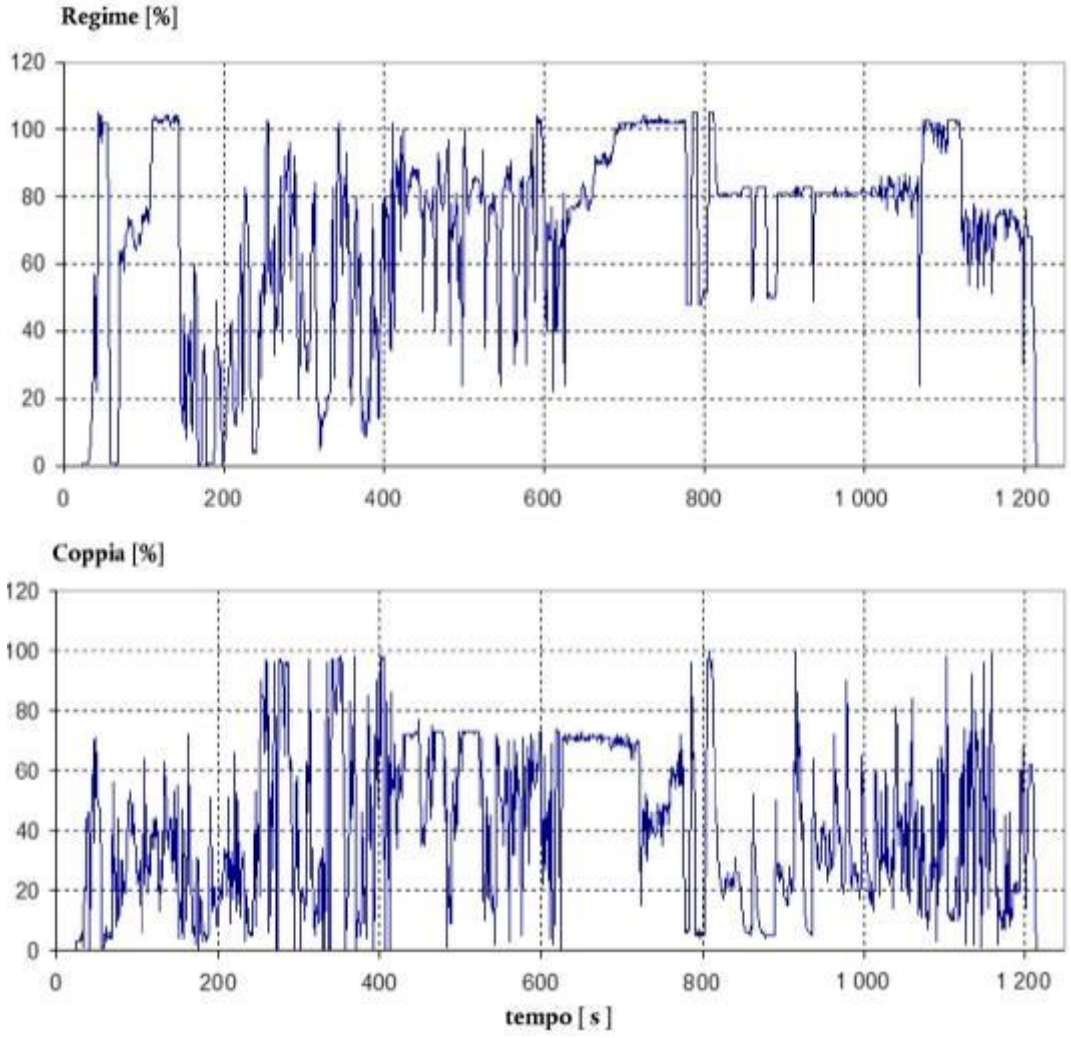


Figura 1.3: ***Il ciclo NRTC***

In questa trattazione ci si concentra su veicoli agricoli capaci di erogare al massimo una potenza di 100 kW, e pertanto si farà riferimento esclusivamente alla categoria sopra citata NRE. Nella tabella a seguire sono quindi riportati i limiti di emissione previsti dalla "Stage V" per i motori appartenenti alla categoria sopra citata:

Fase di emissioni	Sottocategoria di motori	Intervallo di potenza	Tipo di accensione	CO	HC	NO_x	Massa del particolato	PN	A
		kW		g/kWh	g/kWh	g/kWh	g/kWh	#/kWh	
Fase V	NRE-v-1 NRE-c-1	0 < P < 8	accensione spontanea	8,00	($HC + NO_x \leq 7,50$)		0,40 (1)	—	1,10
Fase V	NRE-v-2 NRE-c-2	8 ≤ P < 19	accensione spontanea	6,60	($HC + NO_x \leq 7,50$)		0,40	—	1,10
Fase V	NRE-v-3 NRE-c-3	19 ≤ P < 37	accensione spontanea	5,00	($HC + NO_x \leq 4,70$)		0,015	1×10^{12}	1,10
Fase V	NRE-v-4 NRE-c-4	37 ≤ P < 56	accensione spontanea	5,00	($HC + NO_x \leq 4,70$)		0,015	1×10^{12}	1,10
Fase V	NRE-v-5 NRE-c-5	56 ≤ P < 130	tutti	5,00	0,19	0,40	0,015	1×10^{12}	1,10
Fase V	NRE-v-6 NRE-c-6	130 ≤ P < 560	tutti	3,50	0,19	0,40	0,015	1×10^{12}	1,10
Fase V	NRE-v-7 NRE-c-7	P > 560	tutti	3,50	0,19	3,50	0,045	—	6,00

(1) 0,6 per motori a iniezione diretta, raffreddati ad aria, con avviamento a mano.

Tabella 1.2: ***Limiti per i motori Stage V per la categoria NRE***

Si evidenzia come i limiti sulle emissioni siano particolarmente stringenti per motori con potenza superiore ai 56 kW, in quanto i motori di taglie inferiori sono generalmente montati su veicoli il cui costo diventerebbe troppo elevato se fosse necessario implementare sistemi di post-trattamento più spinti. Appare infatti evidente come i limiti di emissione per gli HC e gli NO_x, nel caso di veicoli con motori termici di taglia ridotta, siano molto meno vincolanti.

Per poter rimanere nei vincoli sempre più stretti posti dalla normativa, si è dovuto provvedere a trattare i gas di scarico per ridurre gli inquinanti prodotti. Per far ciò sono stati sviluppati dispositivi in grado di trasformare i gas indesiderati in prodotti il più possibile ideali (quelli che si avrebbero nella reazione ideale di combustione ed ossidazione di idrocarburi). Dato che la quasi totalità dei veicoli operanti in campo agricolo utilizzano ICE ad accensione per compressione (Diesel), si fornirà allora una panoramica esclusivamente sui sistemi di after-treatment sviluppati per questo tipo di motori.

A differenza dei motori ad accensione comandata (che lavorano con miscele pressoché stechiometriche), i motori Diesel lavorano con miscele globalmente magre, il che impedisce l'utilizzo di catalizzatori a 3 vie che lavorano con alti rendimenti esclusivamente con gas di scarico prodotti dalla combustione di miscele quasi stechiometriche (scostamento massimo del ±1%). Il trattamento delle emissioni allo scarico nei motori Diesel allora avviene grazie ai seguenti sistemi:

1) Catalizzatore ossidante (DOC)

Lo scopo è quello di facilitare le reazioni di ossidazione del CO e degli HC utilizzando metalli catalizzanti come platino Pt e/o palladio Pd. Un uso combinato di tali metalli (catalizzatori bimetallici) porta ad ottenere dei risultati migliori rispetto al loro singolo utilizzo.

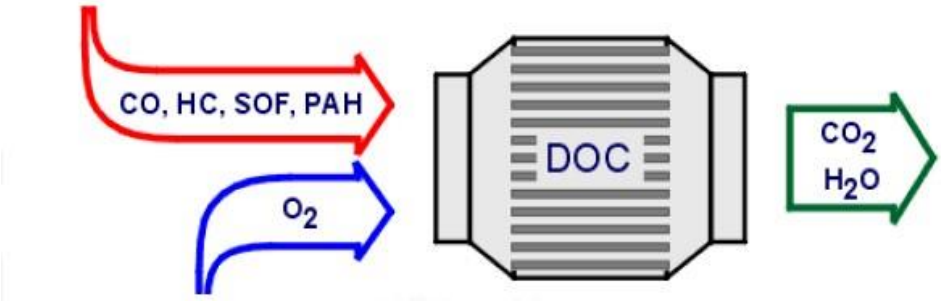


Figura 1.4: ***Catalizzatore ossidante (DOC)***

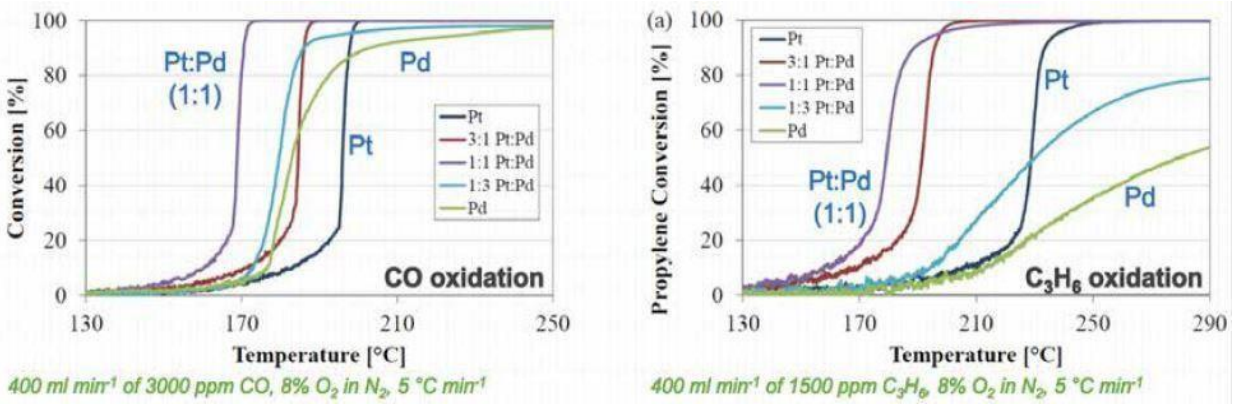
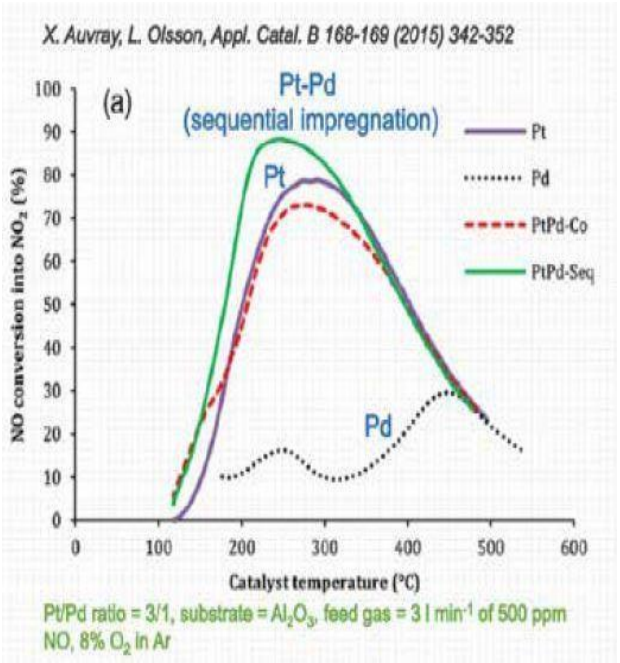


Figura 1.5: ***Efficienza di ossidazione del DOC in diverse casistiche***



Oltre all'ossidazione dei due composti sopra citati, il DOC promuove anche la reazione di ossidazione del NO in NO_2. Tale effetto è desiderato in quanto i dispositivi di trattamento montati a valle (DPF e SCR) raggiungono efficienze maggiori (rispettivamente di rigenerazione e di riduzione) se la reazione di ossidazione è tendenzialmente completa. Altre reazioni promosse ma indesiderate sono:

- Ossidazione del biossido di zolfo (SO_2) in triossido di zolfo (SO_3);

- Formazione di acido solforico (H_2SO_3) dato dalla reazione del triossido di zolfo (SO_3) con il vapor acqueo (H_2O), responsabile dell'aumento di particolato PM prodotto.

Figura 1.6: ***Efficienza nel DOC per la formazione di NO_2***

Si può notare come per entrambe le reazioni la temperatura rivesta un ruolo fondamentale in quanto le reazioni per avvenire necessitano di temperature superiori ai 200°C.

2) Filtro antiparticolato DPF (Diesel Particulate Filters)

A differenza del DOC, il DPF non promuove nessuna reazione ma ha lo scopo di filtrare i gas di scarico per poter intrappolare il particolato prodotto. L'operazione di filtraggio avviene mediante passaggio dei gas attraverso una barriera ceramica porosa che sfrutta due meccanismi: il filtraggio a letto profondo (diametro medio dei pori maggiore del diametro medio delle particelle bloccate) ed il filtraggio a separazione superficiale (filtraggio meccanico).

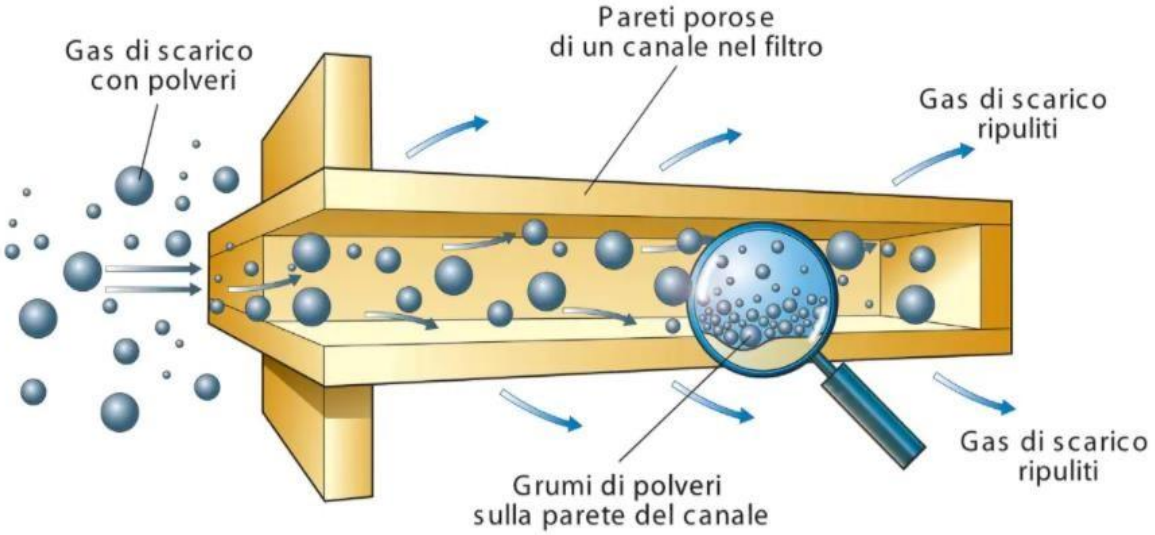


Figura 1.7: ***Funzionamento del DPF***

Di conseguenza il filtro necessita di una operazione di rimozione delle particelle di particolato raccolte nota come rigenerazione del filtro antiparticolato: tale processo avviene in automatico ed in maniera del tutto invisibile all'utente e consiste nel bruciare le catene carboniose all'interno del DPF tramite del diesel introdotto nel condotto di scarico che, reagendo con l'ossigeno, promuove la combustione. Il diossido di azoto presente durante questa fase agisce da catalizzatore, diminuendo la temperatura minima necessaria per la reazione permettendo quindi di diminuire la quantità di diesel introdotta.

3) Catalizzatore riducente ($DeNO_x$,LNT,SCR)

Questi ultimi devono invece favorire la reazione di riduzione della NO, decomposta in N_2 e O_2. Vi sono diverse tipologie di sistemi che possono realizzare ciò:

- $DeNO_x$: sfrutta il carburante come riducente e necessita della presenza di catalizzatori quali platino in combinazione all'allumina (Pt/Al_2O_3) o al rame e zeolite (Cu/ZSM5). La problematica di tale sistema è di non essere sufficientemente resistente all'invecchiamento e di avere rendimenti massimi molto bassi, raggiungibili in un ristretto campo di temperature. Pertanto non è più utilizzato nei motori.

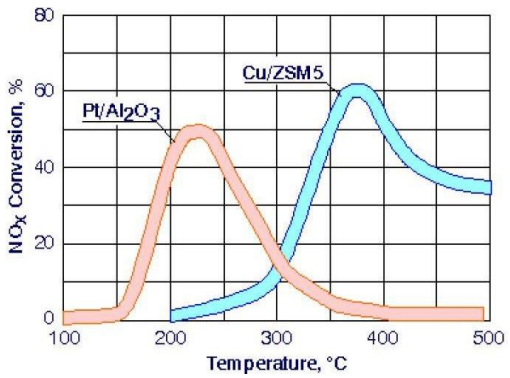


Figura 1.8: ***Rendimenti dei $DeNO_x$***

- LNT (Trappole per NO_x): anche queste usano il Diesel come riducente ma sfruttano le proprietà degli ossidi di bario (BaO) per trattenere ossidi dell'azoto (NO_2 ed NO_3) per poi liberarli in rigenerazione, ovvero quando è disponibile combustibile per ridurre tali gas in presenza però del Rodio (Rh), che funge da catalizzatore.

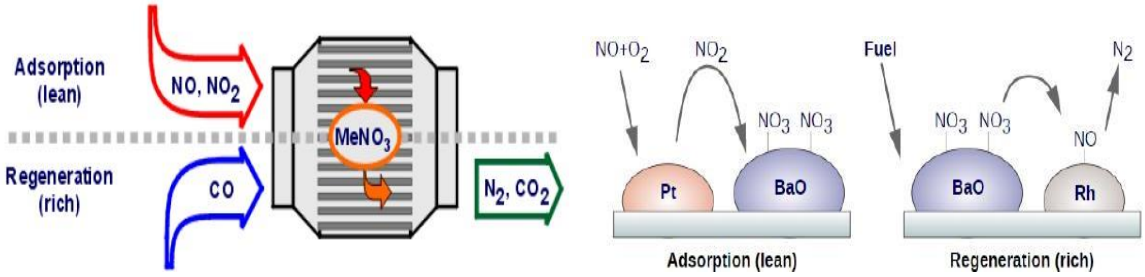
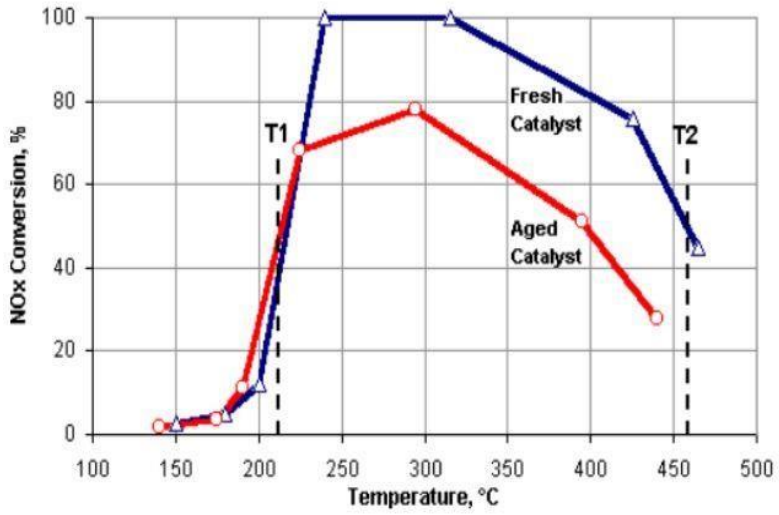


Figura 1.9: ***Schema e principio di funzionamento delle trappole LNT***



Tale sistema garantisce rendimenti di conversione molto elevati per un ampio range termico.

Dato che all'interno del catalizzatore sono dispersi metalli nobili, il costo aumenta notevolmente all'aumentare della cilindrata del motore, così come anche l'uso di combustibile come riducente causa aumento dei costi.

Figura 1.10: ***Rendimento delle trappole LNT***

- SCR (Selective Catalytic Reduction): costituisce il sistema più recente e basa il suo funzionamento sull'utilizzo dell'ammoniaca, che reagisce selettivamente con gli NO_x, non necessitando quindi di metalli nobili (che si traduce in un minor aumento di costi con l'aumentare della cilindrata del motore).

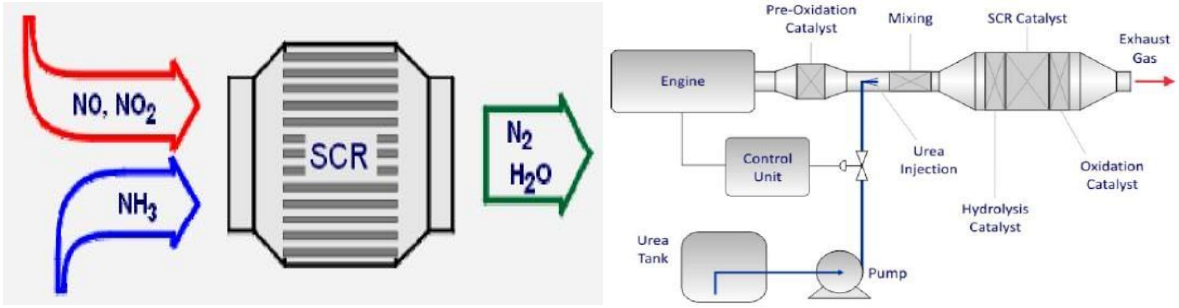
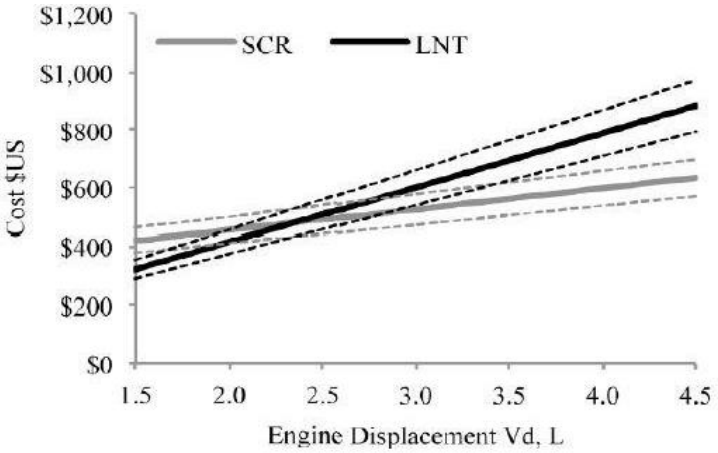


Figura 1.11: ***Schema di un SRC e layout di un sistema propulsivo in cui è installato***

E' importante introdurre nell'SCR le proporzioni corrette di NO_2 ed ammoniaca, in quanto degli eccessi o dei difetti possono generare prodotti indesiderati come l'N_2O. Dalla Figura 1.11 si può inoltre notare come questo sistema abbia l'inconveniente di essere piuttosto ingombrante, data la richiesta di un apposito impianto per la fornitura dell'NH_3.

Al posto dell'ammoniaca è possibile utilizzare soluzioni di urea ed acqua (l'urea reagisce con l'apporto di calore diventando ammoniaca), dal nome commerciale AdBlue. In ognuno dei due casi, comunque, la presenza di un ulteriore liquido (oltre il combustibile) richiede dei costi di investimento iniziali maggiori. Il sistema, tuttavia, è caratterizzato da rendimenti prossimi al 100% in un range termico molto esteso.



Nonostante la tendenza sia oramai quella di montare esclusivamente questo sistema a bordo dei moderni veicoli, il fattore discriminante nella scelta tra SCR ed LNT è nella cilindrata del motore: per basse cubature è più economico l'uso dell'LNT, mentre con un aumento di cubatura diventa più economico l'SCR.

Figura 1.12: ***Comparazione costi tra sistema SCR ed LNT***

Le emissioni di CO2: verso l'ibridizzazione e l'elettrificazione dei veicoli. Il caso particolare dei veicoli NRMM da lavoro

I sistemi EAS descritti nel paragrafo precedente costituiscono degli strumenti estremamente efficaci per portare al minimo le emissioni dovuti a combustione non ideale, consentendo quindi di rientrare nei limiti posti dalle normative, seppur questi ultimi siano sempre più stringenti.

Tuttavia, come già riportato in Tabella 1.1, permane la problematica relativa all'Anidride Carbonica, che si è detto non essere un prodotto indesiderato essendo un prodotto anche della combustione ideale. Anche in condizioni ideali quindi non si riesce a impedirne la formazione. La problematica relativa alla formazione di CO_2 non è quindi molto legata a ragioni di salute dell'essere umano, quanto a ragioni ambientali: la CO_2 è infatti la principale causa dell'effetto serra; il suo accumulo in atmosfera determina infatti la nascita di uno schermo monodirezionale in quanto lascia passare verso la Terra tutte le radiazioni proprie della luce solare ma al tempo stesso riflette le radiazioni emesse dal suolo. Questo fenomeno ha come effetto l'aumento generalizzato della temperatura media atmosferica, a sua volta causa di altre problematiche ambientali come, ad esempio, lo scioglimento dei ghiacciai.

Dal punto di vista veicolistico la legislazione non pone alcuna soglia massima all'emissione di anidride carbonica ai fini dell'ottenimento dell'omologazione dei mezzi, a differenza dei gas inquinanti. Esiste però un limite, stabilito dalla Commissione Europea, sulla quantità massima di CO_2 che l'intera flotta di veicoli commercializzata da ogni singola casa produttrice può immettere ogni anno in atmosfera. Conseguentemente l'attenzione delle aziende automobilistiche si sta spostando dall'emissione di gas inquinanti (che ha oramai raggiunto livelli soddisfacenti) al trovare nuove soluzioni energetiche con le quali incrementare i rendimenti dei veicoli: un aumento di rendimento (inteso come rapporto tra energia utile ed energia immessa all'interno del sistema) determina infatti un minor consumo di combustibile (energia chimica immessa) a fronte della stessa quantità di energia utile ottenuta. Da qui nasce quindi l'idea di dotare i veicoli di una o più macchine elettriche, da affiancare o sostituire al motore termico convenzionale per ridurre o abbattere completamente le emissioni di CO_2 misurate allo scarico. L'introduzione dei motori elettrici nei veicoli apre infatti ad un ventaglio molto ampio di possibili soluzioni di powertrain: questi possono costituire un aiuto al motore termico, facendo lavorare quest'ultimo in punti della mappa consumi a maggior rendimento attraverso la realizzazione di un down-sizing della macchina termica (questa strategia può anche essere realizzata anche senza installare alcun motore elettrico, ma i benefici in termini sia di down-sizing sia di riduzione di emissioni di CO_2 sarebbero limitati), oppure sostituirsi completamente alla propulsione termica. Nascono così le definizioni di veicoli ibridi (termico+elettrico, accoppiabili secondo numerose e differenti strategie che si analizzeranno nel capitolo a seguire) e di veicoli completamente elettrici (propulsione interamente proveniente da motore/i elettrici). Inoltre per alcune particolari architetture di powertrain ibridi è possibile disabilitare interamente la propulsione termica, muovendo quindi anche in questo caso il veicolo in modalità full-electric ed ottenendo così anche in questo caso emissioni nulle allo scarico, con la ricarica delle batterie effettuate in tal caso attraverso la frenata rigenerativa: la reversibilità dei motori elettrici infatti consente a questi ultimi di lavorare come generatori nelle fasi di decelerazioni non brusche del veicolo. I generatori, quindi, eserciteranno una coppia resistente che frena il veicolo (senza far intervenire il sistema di frenatura tradizionale tipicamente basato su un circuito idraulico + delle pastiglie freno che convertono l'energia di frenata in calore), e nel contempo si andranno a ricaricare i pacchi batterie per aumentare l'autonomia in questa modalità di erogazione di potenza.

Nel caso particolare di veicolo NRMM da lavoro, queste soluzioni risultano inoltre applicabili non solo alla trazione del veicolo, ma è possibile ibridizzare o elettrificare anche la presa di potenza (PTO, Power Take Off), attraverso la quale si vanno ad azionare attrezzature o macchinari supplementari (come trinciatrici, erpici rotanti o gli atomizzatori e così via) riducendo il carico che deve essere fornite dall'eventuale propulsore termico.

Figura 1.13: ***Esempio di PTO meccanica per l'azionamento di attrezzature***

Sul mercato ci sono già alcuni veicoli appartenenti alla categoria NRMM che fanno uso della propulsione ibrida, come ad esempio la pala caricatrice ibrida John Deere 644K o il sollevatore telescopico ibrido Merlo TF 40.7.

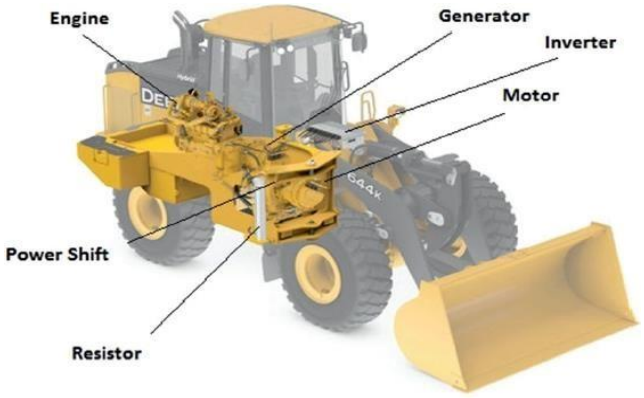


Figura 1.14: ***Pala caricatrice ibrida John Deere 644K***

Figura 1.15: ***Sollevatore telescopico ibrido Merlo TF 40.7***

Per quanto riguarda invece le trattrici agricole, in Italia alcune aziende si stanno muovendo nell'ottica di ibridizzare ed elettrificare queste tipologie di macchine: è il caso ad esempio di Antonio Carraro S.p.a. , che in collaborazione con il Politecnico di Torino tramite la Startup Ecothea S.r.l, ha realizzato sia un prototipo di trattore da vigneto-frutteto Mild hybrid denominato SRX Hybrid sia una trattrice full electric denominata e-SP, già presentati rispettivamente in occasione di EIMA 2021 ed EIMA 2022.

Figura 1.16: ***Prototipo SRX Hybrid Antonio Carraro***

Figura 1.17: ***Prototipo e-SP full electric Antonio Carraro***

La ricerca delle aziende sopracitate sul tema comunque continua, grazie anche al contributo dell'Unione Europea tramite l'aggiudicazione del bando di progetto LifeAtena, con il quale si svilupperanno e valideranno nei prossimi 3 anni (2023-2026) due ulteriori trattori: un nuovo modello full hybrid ed un ulteriore modello full electric.

Capitolo 2 : Tipologie di powertrain ibridi ed elettrici per veicoli NRMM da lavoro

Panoramica sulle principali architetture ibride ed elettriche per veicoli

In generale, quando si parla di veicoli ibridi ci si riferisce a dei sistemi propulsivi che presentano due diverse sorgenti di potenza di trazione (termica ed elettrica nel caso di veicoli) che possono essere organizzate in diversi modi, ottenendo caratteristiche differenti a seconda del campo di applicazione del veicolo stesso.

Un powertrain puramente termico opera per la maggior parte della vita operativa in punti a basso carico e a basso rendimento. Aggiungendo una ulteriore fonte energetica, attraverso l'uso congiunto delle due sorgenti si riesce invece a lavorare in punti a maggior rendimento; inoltre attraverso sistemi ibridi termici-elettrici è possibile sfruttare la frenata rigenerativa, rimuovere il funzionamento a regime minimo e realizzare il down-sizing dell'unità termica oppure di aggiungere ulteriori funzionalità al sistema.

In un powertrain puramente elettrico invece si elimina completamente la sorgente termica andando ad utilizzare dei motori elettrici di potenza più elevata, accompagnata da una maggiorazione delle dimensioni anche dei sistemi di accumulo di energia (pacchi batterie) per garantire una autonomia del veicolo più elevata.

I macrogruppi in cui si possono suddividere i propulsori ibridi-elettrici sono sostanzialmente 4:

- Gli ibridi Serie (SHEV)
- Gli ibridi Paralleli (PHEV)
- Gli ibridi Complessi (Power Split)
- I full-electric (BEV)

Ognuno dei gruppi comprende a sua volta delle sottovarianti di sistema, le quali offrono soluzioni tra loro simili ma ottimizzate per tipologia di applicazione. Per distinguere queste sottovarianti in modo univoco viene definito un parametro chiamato Hybridization Ratio R_h, il cui valore può variare dal valore numerico 0 fino al valore numerico 1. Per tutte le tipologie di ibrido il valore 0 indica univocamente una trazione puramente elettrica (veicolo full electric a batteria), mentre a seconda del tipo di ibrido varia l'interpretazione che assume il valore 1:

- Per gli ibridi serie un $R_h = 1$ rappresenta i veicoli con trasmissione elettrica
- Per gli ibridi paralleli un $R_h = 1$ corrisponde ad un veicolo tradizionale a propulsione termica

L'Hybridization Ratio è quindi calcolato rapportando la potenza del motore termico con la potenza elettrica (collegate tramite una apposita trasmissione) secondo delle equazioni differenti, una valida per gli ibridi serie ed una valida per gli ibridi paralleli.

$$R_{h,serie} = \frac{P_{ICE}}{P_{EL}} \tag{2.1}$$

$$R_{h,parallelo} = \frac{P_{ICE}}{P_{ICE}+P_{EL}} \quad (2.2)$$

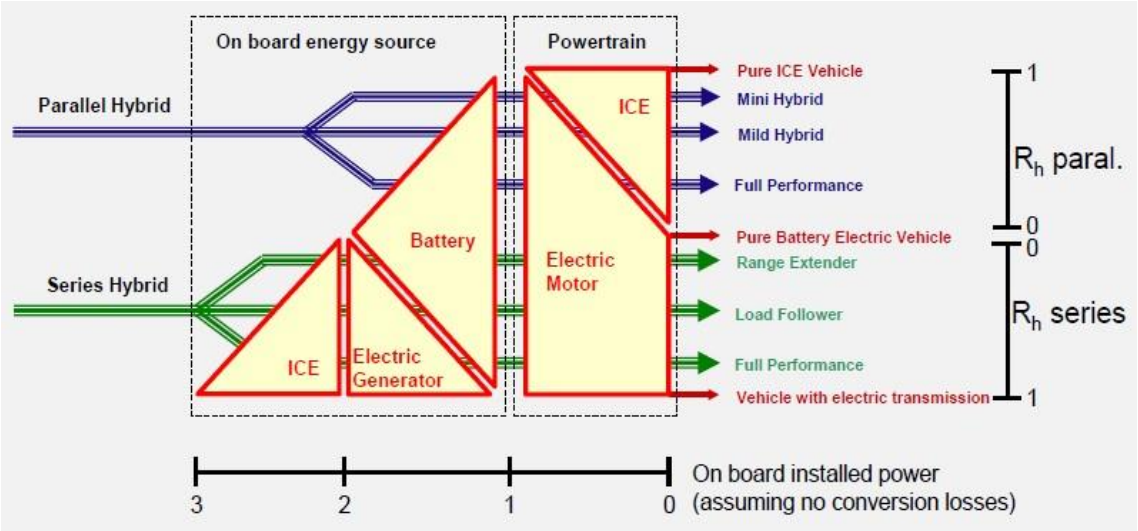


Figura 2.1: ***Tipologie di Propulsioni Ibride - elettriche in relazione all'Hybridization Ratio***

Come già detto ed evidenziato dalle equazioni (2.1) e (2.2), le relazioni assumono valore 0 quando $P_{ICE} = 0$, corrispondente al caso di ibridi serie e/o paralleli in cui la parte propulsiva termica è momentaneamente inattiva.
Da tale definizione se ne deriva però anche la definizione di full-electric vehicle, caratterizzati da $P_{ICE} = 0$ in quanto la sorgente termica è completamente assente nell'architettura del powertrain. Negli schemi successivamente proposti delle varie architetture, per ottenere un powertrain full electric è sufficiente tagliare ed eliminare la sorgente di potenza termica, adottando un opportuno sistema di accumulo.

Per quanto riguarda la classe degli ibridi Paralleli, oltre all' Hybridization Ratio R_h è vantaggioso anche definire il suo complementare, vale a dire l'Hybridization Factor HF così definito:

$$HF = \frac{P_{EL}}{P_{ICE}+P_{EL}} \quad (2.3)$$

Secondo questa definizione, in caso di powertrain convenzionale puramente termico l'HF sarà pari a 0, mentre in caso di veicolo puramente elettrico l'HF sarà pari a 1. Tra i due estremi è possibile definire una grande varietà di sistemi ibridi, che saranno analizzati successivamente.

Una precisazione da effettuare è come, a differenza di quanto accade per veicoli stradali, nei veicoli NRMM da lavoro il powertrain deve anche attivare l'attrezzatura portata: nel caso delle trattrici quindi l'intero sistema dovrà anche azionare una presa di potenza (PTO) oltre a garantire movimentazione del veicolo ed eventuale rimorchio. Nonostante non ci sia una classificazione ben definita dei powertrain ibridi per macchine da lavoro, un possibile approccio è quello di definire un apposito fattore di ibridizzazione HF_{wv}. Tale fattore è dato dalla combinazione di due rapporti, il primo legato alla potenza di trazione e il secondo alla potenza di azionamento delle attrezzature.

$$HF_{wv,drive} = \frac{P_{EL,drive}}{P_{ICE}+P_{EL,drive}} \qquad HF_{wv,load} = \frac{P_{EL,load}}{P_{ICE}+P_{EL,load}} \quad (2.3) \text{ e } (2.4)$$

$$HF_{wv} = \frac{1}{2}(HF_{wv,drive} + HF_{wv,load}) \quad (2.5)$$

Per una comprensione maggiore sulle tipologie di sistemi, si vanno quindi a descrivere più nel dettaglio le varie soluzioni.

1) Gli Ibridi Serie (SHEV)

Dal punto di vista del controllo del sistema propulsivo, l'ibrido serie costituisce sicuramente l'architettura più semplice.

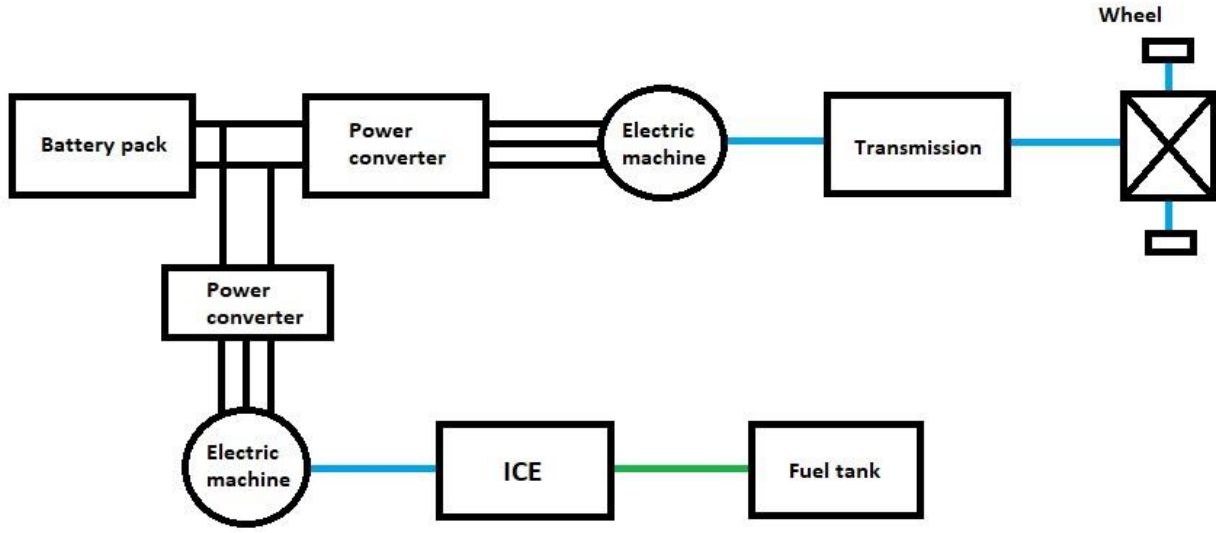


Figura 2.2: ***Schema di un generico powertrain ibrido Serie***

Come è visibile nello schema a blocchi riportato in Figura 2.2, in questa tipologia di sistema propulsivo si ha la conversione di energia chimica (linea verde) in energia meccanica (linea blu) attraverso un motore termico, ed una successiva conversione di tale energia meccanica in energia elettrica (linea nera) attraverso una macchina elettrica che lavora da generatore. L'elettricità generata, attraverso dei convertitori elettronici di potenza (necessari per trasformare una corrente alternata trifase in corrente continua per e dalle batterie), può essere utilizzata per ricaricare il pacco batterie (se presente) del veicolo oppure essere inviata ad una ulteriore macchina elettrica che lavora come motore e che provvede al moto del veicolo. Di conseguenza non è presente in questa architettura alcun collegamento meccanico tra l'unità termica e il sistema di trazione.

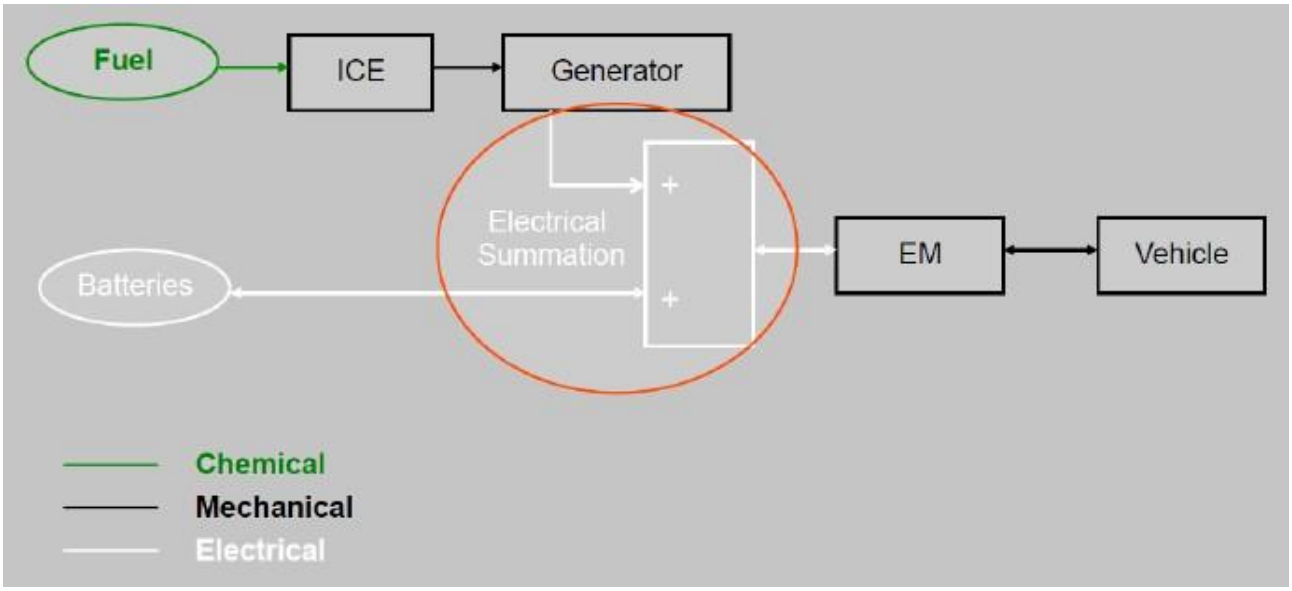


Figura 2.3: ***Schema di conversione e flusso di potenza in un ibrido Serie***

Questo costituisce il vantaggio preponderante di questa tipologia di powertrain: avendo svincolato la velocità del veicolo dalla velocità di rotazione dell'ICE, è allora possibile far lavorare quest'ultimo sempre nel punto di funzionamento a maggior rendimento, e conseguentemente a minor consumo di combustibile a parità di potenza meccanica generata. Di conseguenza le prestazioni in termini di velocità ed accelerazione del veicolo dipenderanno esclusivamente da quelle che sono le caratteristiche del motore elettrico. La potenza proveniente dal generatore e dalle batterie si sommano a monte del motore elettrico. Generalmente sia la macchina elettrica che lavora da generatore sia quella che lavora da motore di trazione sono reversibili: di conseguenza quest'ultima consente anche un flusso di potenza inverso dalle ruote verso il pacco batterie, permettendo così di realizzare la frenata rigenerativa in caso di decelerazioni non brusche.

All'interno di questa categoria di sistemi propulsivi, a seconda delle dimensioni relative tra motore termico e generatore e delle dimensioni della batteria è possibile distinguere tra:

- Range Extender

  Utilizza un ICE di dimensioni molto contenute, in grado quindi solo di ricaricare le batterie e lavorando sempre a punto fisso nominale (di massimo rendimento). Questa soluzione, come dal nome stesso, consente di aumentare notevolmente la distanza percorribile dal veicolo seppur diminuendo le dimensioni delle batterie (e quindi del peso complessivo del veicolo).

- Load Follower

  In questa tipologia di ibrido serie il motore termico eroga più potenza e non lavora più in un unico punto fisso nominale dato che deve poter garantire l'erogazione della potenza massima in condizioni di regime. Le batterie sono in questo caso di dimensioni estremamente ridotte in quanto devono solamente fornire la potenza mancante nelle fasi transitorie.

- Full Performance

  A differenza dei primi due sistemi ibridi serie, quest'ultimo non è dotato di alcuna batteria e di conseguenza sia l'ICE che il generatore devono essere dimensionati per poter garantire tutta la potenza richiesta anche nelle fasi transitorie di accelerazione. Non è di conseguenza consentita la frenata rigenerativa, non avendo a bordo veicolo alcun sistema di accumulo di energia.

  In generale la soluzione costruttiva di ibridi serie non è quella ottimale per i moderni veicoli ibrido-elettrici stradali per via dei diversi rendimenti di conversione in veicoli stradali: il maggior numero di macchine elettriche e di convertitori di potenza (seppur questi presi singolarmente hanno rendimenti superiori al 95%) fa sì che la moltiplicazione dei singoli rendimenti della catena faccia ottenere efficienze più basse, mascherando quindi in parte il vantaggio ottenuto dal far lavorare l'ICE a punto fisso.

  Al contrario, nei veicoli da lavoro si predilige questa tipologia di propulsione ibrida in quanto il layout è quello meno complesso possibile (che si traduce in un minor costo di produzione e valle di vendita) e si elimina il problema di accoppiamento dei motori con le ruote (le velocità ottimali dei diversi sistemi infatti differiscono tra loro), potendo quindi destinare la potenza in modo più ottimale alla PTO per eseguire le diverse lavorazioni.

2) Gli Ibridi Paralleli (PHEV)

Nei powertrain ibridi paralleli sia il motore termico sia il motore elettrico possono provvedere a fornire energia meccanica alle ruote.

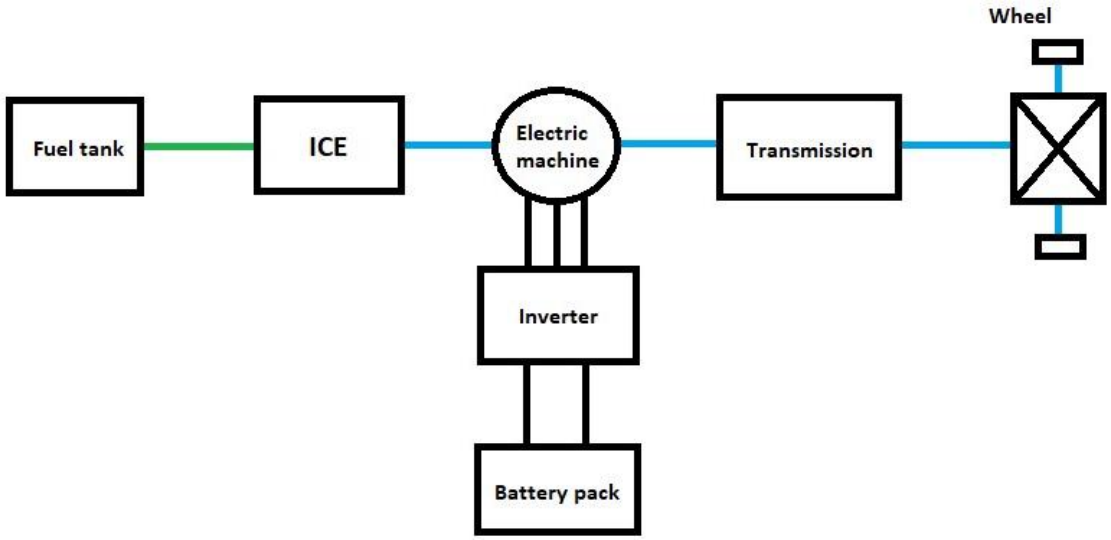


Figura 2.4: ***Schema di un generico powertrain ibrido Parallelo***

Come è visibile nello schema a blocchi riportato in Figura 2.4, in questa tipologia di sistema propulsivo si ha la conversione di energia chimica (linea verde) in energia meccanica (linea blu) attraverso un motore termico collegato meccanicamente ad una macchine elettrica reversibile che può agire sia da motore (se riceve corrente dalle batterie) sia da generatore (se fornisce corrente alle batterie), entrambi meccanicamente collegati alle ruote del veicolo attraverso un opportuno differenziale. Con tale soluzione si riduce allora il numero di convertitori di potenza e di macchine elettriche, ottenendo quindi un rendimento totale maggiore ma a scapito di una maggior complessità nell'accoppiamento delle varie sorgenti di potenza. Di conseguenza anche le logiche di controllo dei motori saranno più complesse. Anche con tale soluzione è possibile attuare la frenata rigenerativa, se la macchina elettrica è reversibile.

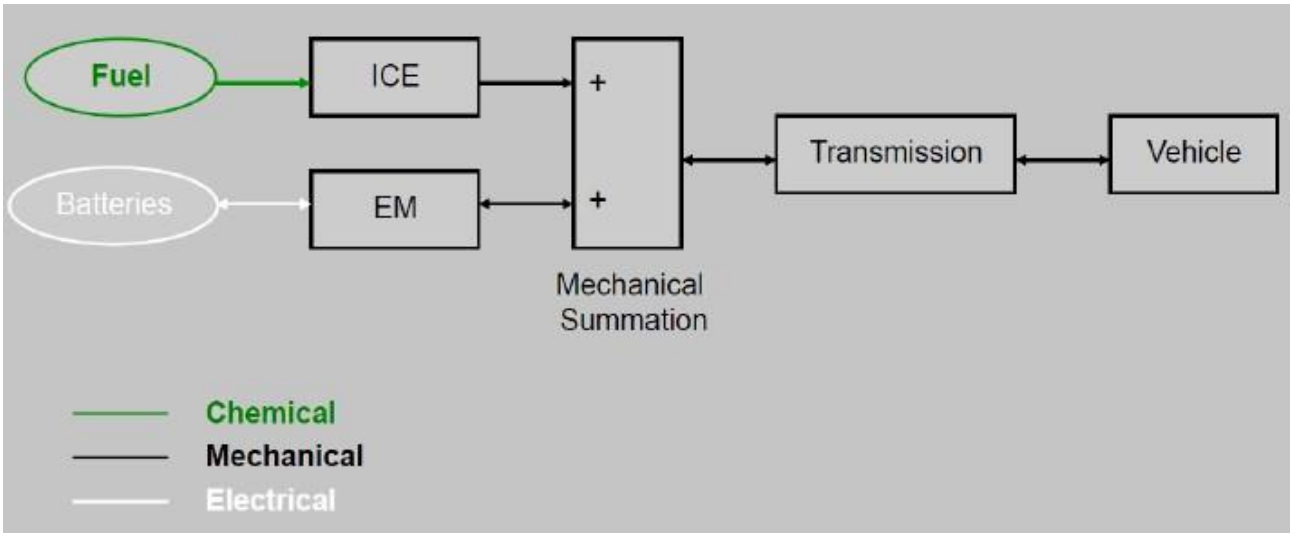


Figura 2.5: ***Schema di conversione e flusso di potenza in un ibrido Parallelo***

Quindi seppur più flessibile, il motore termico risulta vincolato cinematicamente alle ruote del veicolo (non può dunque lavorare a punto fisso) ma può comunque ricaricare tramite la macchina elettrica in modalità generatore il pacco batterie nelle fasi in cui la potenza erogata dalla sorgente termica è maggiore di quella effettivamente richiesta.

Come detto in precedenza, per gli ibridi paralleli è possibile effettuare una classificazione in base all' Hybridization Factor HF, secondo cui si hanno powertrain ibridi paralleli:

- Minimal Hybrid ($0 < HF < 0,1$): l'unità elettrica è un semplice elemento ausiliario all'unità principale termica in ben specifiche operazioni (come le start&stop operations)
- Mild Hybrid ($0,25 < HF < 0,5$): l'unità elettrica non è ancora in grado di compiere un ciclo di guida autonomamente, ma è ancora un ausilio al motore termico (seppur per operazioni più gravose)
- Full Hybrid ($0,5 < HF < 0,7$): l'unità elettrica è in questo caso in grado di movimentare il veicolo nei suoi principali cicli di guida standard

Delle ulteriori modalità di classificazione permettono invece di distinguere gli ibridi in base alla posizione relativa tra ICE e motore elettrico, oppure in base a come sono accoppiati. Si identificano allora:

- Classificazione Beretta, suddivisa in:
  1) Double Drive: i propulsori sono collegati mediante l'asfalto (esempio i veicoli AWD nei quali le ruote posteriori sono mosse esclusivamente da motori elettrici);
  2) Double Shaft: i propulsori sono collegati mediante frizioni e quindi possono lavorare in maniera indipendente l'uno con l'altro;
  3) Single Shaft: i propulsori sono in collegamento diretto e quindi devono girare alla stessa velocità
- Classificazione basata sulla posizione del motore elettrico: il motore elettrico può essere collegato direttamente al motore termico nel lato frontale/posteriore, collegato a monte della frizione oppure collegato a valle del cambio.

3) Gli Ibridi Complessi

Per ottenere un ibrido complesso si agisce aumentando il numero di unità termiche e/o il numero di motori elettrici (si agisce solitamente con la seconda opzione), aumentando le sorgenti di potenza oppure realizzando un powertrain in grado di accoppiare tra loro le architetture serie e parallelo. In questo modo si consente all'ICE di lavorare in condizioni di massimo rendimento in un ampio range di potenza richiesta.

Nel caso dei veicoli NRMM è inoltre possibile adottare queste tipologie di strategie per disaccoppiare la PTO dall'unità termica (aggiungendo uno o più motori elettrici ad esso dedicata), in modo che quest'ultima lavori sempre nei pressi del punto di minimo consumo di carburante.

Un esempio di ibrido complesso è quello definito convenzionalmente Power Split, adottato dalla Toyota sui veicoli stradali, che basa il suo funzionamento sull'uso di due macchine elettriche e una unità termica, un CVT (Continuously Variable Trasmission) controllato da una delle due macchine elettriche e da un rotismo epicicloidale sul quale sono contemporaneamente collegati l'ICE ed entrambe le macchine elettriche. Il compito del rotismo è di consentire lo svincolamento

del regime di velocità delle ruote da quello del motore termico, mentre il CVT (o una silent chain) va sostanzialmente a ripartire in un modo distribuito la fornitura di potenza dai motori di trazione.

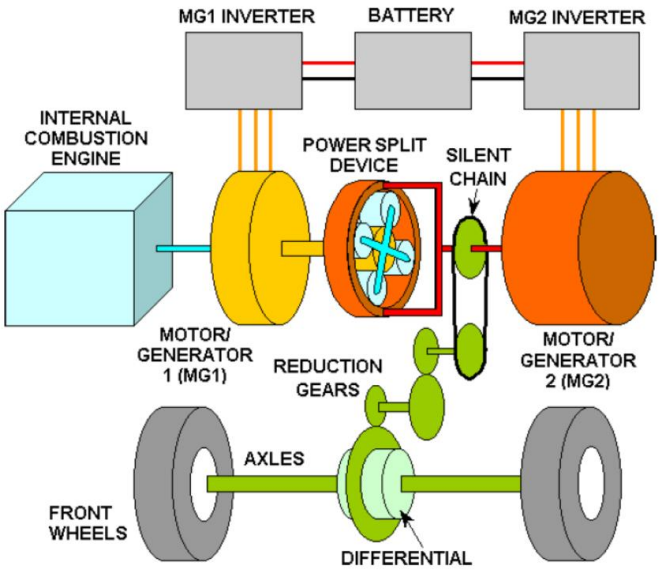


Figura 2.6: ***Schema di un powertrain ibrido complesso tipo Power Split***

Con questa soluzione, l'unità termica gira ad un regime variabile ma comunque differente da quello delle ruote di trazione. Nello schema proposto in Figura 2.6 l'altro motore di trazione è costituito dall'MG2, la cui potenza si somma (o si sottrae) all'occorrenza a quella dell'ICE. L'MG1 invece fornisce solo la funzione di start&stop per l'unità termica e per convertire l'energia elettrica dal motore termico all'MG2.

Si possono definire diverse modalità di funzionamento per tale sistema:

- Fase di partenza e di bassa velocità: l'ICE è spento e tutta la potenza di trazione viene derivata dalla batteria per mezzo dell'MG2.
- Fase di normale trazione: l'ICE, tramite il Power Split, fornirà potenza meccanica sia alle ruote sia all'MG1 che lavorando da generatore fornirà alimentazione all'MG2 che a sua volta tramite il CVT (o silent chain) fornirà potenza addizionale alle ruote (sommandosi al contributo del termico).
- Fasi di accelerazione: l'aumento di potenza richiesto fa intervenire la batteria, che andrà ad alimentare ulteriormente l'MG2 per fornire più coppia al veicolo.
- Fasi di decelerazione: l'MG2 lavora in questo caso come generatore andando a ricarica la batteria (frenata rigenerativa). La presenza del Power Split consente all'ICE di non risentire del cambio di verso di rotazione dell'albero collegato all'MG2.
- Veicolo fermo: a motore termico acceso, quest'ultimo andrà a ricaricare la batteria utilizzando l'MG1 come generatore.

Nonostante i benefici offerti da tale soluzione, il problema principale è costituito dal rotismo epicicloidale, componente molto costose e difficilmente gestibile in caso di guasto: il veicolo non potrebbe muoversi in alcun modo nel caso in cui il rotismo smetta di funzionare.

Per veicoli della categoria NRMM sono ancora in corso studi in merito all'applicabilità di tali sistemi.

4) Veicoli Full Electric (BEV)

Con questa categoria di sistema propulsivo si abbandona il concetto di veicolo elettrificato per entrare nel contesto dei veicoli 100% elettrici, i cui powertrain possono essere composti da uno, due o anche tre motori (a seconda dei modelli di veicoli).

Lo schema meccanico di un powertrain full electric è estremamente semplice: ci sono uno o due motori elettrici, di potenza solitamente superiore alla media dei veicoli ibridi, alimentati da un grande pacco batteria. Lo schema inoltre può essere ricavato da quello di un ibrido Serie depurato della linea di potenza proveniente dall'ICE. Anche in questo caso può essere effettuata la frenata rigenerativa nelle fasi decelerazioni non brusche.

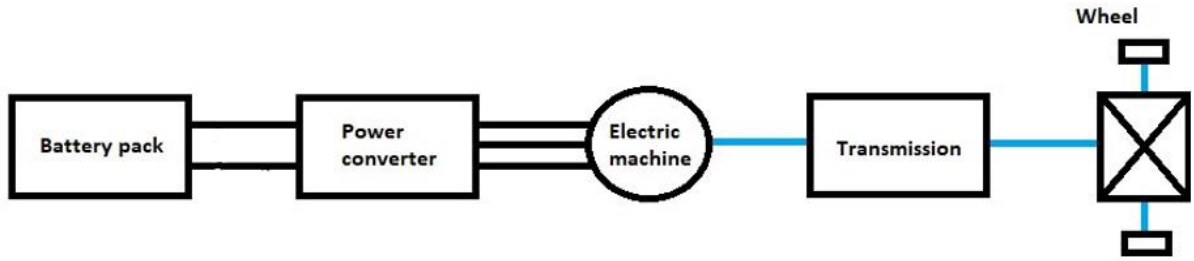


Figura 2.7: ***Schema di un generico powertrain full-electric a singolo motore***

Nel caso di veicolo NRMM da lavoro, a valle del motore elettrico di trazione (o direttamente calettato sul suo rotore) saranno anche presenti delle derivazioni di potenza meccanica verso la pompa per l'alimentazione del circuito idrostatico e verso la PTO meccanica, con opportuna frizione di accoppiamento/disaccoppiamento.

Esempi di powertrain ibridi in veicoli NRMM commercializzati

L'introduzione di powertrain elettrificati nel caso dei veicoli NRMM è iniziata con il settore delle costruzioni (escavatori e pale caricatrici) per passare poi ai sollevatori telescopici ed infine al settore delle macchine agricole. Come già accennato, l'architettura comunemente impiegata per questa tipologia di veicoli è quella Ibrida Serie, anche se è necessario distinguere in che modo viene utilizzato il powertrain tra le funzioni di guida e di lavoro.

Si riportano e si descrivono brevemente alcuni modelli realizzati per i veicoli NRMM dei vari settori.

1. **BullDozer**

Il primo bulldozer ibrido ad essere immesso nel mercato fu costruito da Caterpillar, il Caterpillar D7E, rimpiazzando il precedente modello D7R.

Dalle dichiarazioni della casa produttrice, il modello ibrido garantirebbe un aumento della produttività ed una diminuzione del consumo di carburante del 24%.

L'architettura utilizzata per la propulsione è quella di un ibrido serie con sistema di accumulo (batteria o condensatori) dell'energia elettrica. Il sistema idraulico, invece, è tradizionale in quanto la pompa di alimentazione è azionata direttamente dal motore termico.

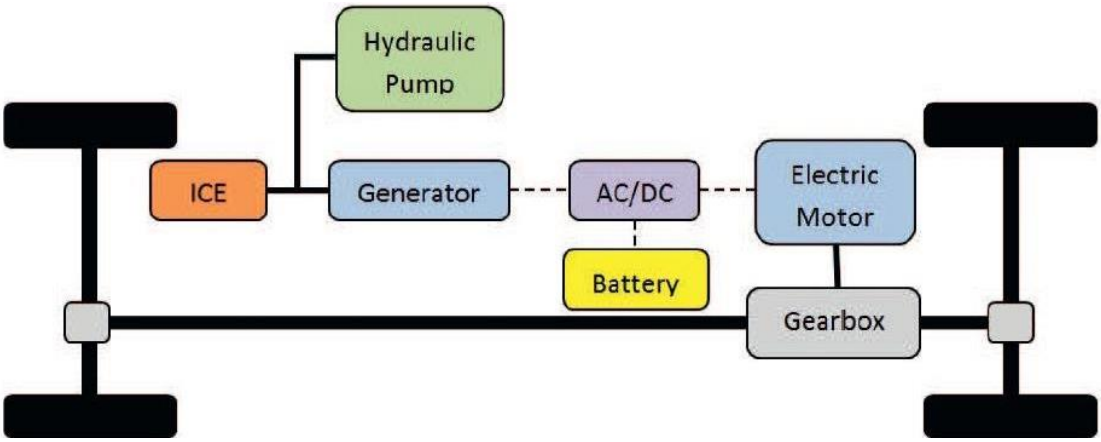


Figura 2.8: ***Schema del powertrain ibrido serie del dozer Caterpillar D7E***

2. Escavatori

Questa tipologia di macchinari è principalmente utilizzata per la rimozione di terreno durante le operazioni di scavo. Alcuni tra i maggiori produttori mondiali di questa categoria di macchine hanno aggiunto al loro parco veicoli degli escavatori con powertrain ibridi: si citano Kobelco, Hitachi, Komatsu e Doosan. Una delle soluzioni più promettenti prevede un powertrain serie-parallelo, dove l'ICE alimenta direttamente un generatore, che a sua volta alimenta secondo una configurazione serie delle pompe idrauliche, mentre il motore di rotazione (swing motor) è alimentato in configurazione parallelo dal generatore e dall'ESS. Il powertrain è dotato di un sofisticato sistema di recupero dell'energia durante la rotazione dello stesso per poi immagazzinarla in ultracondensatori (supercapacitor), e non nelle classiche batterie.
Sono tuttavia stati realizzati anche degli escavatori serie e degli escavatori parallelo.

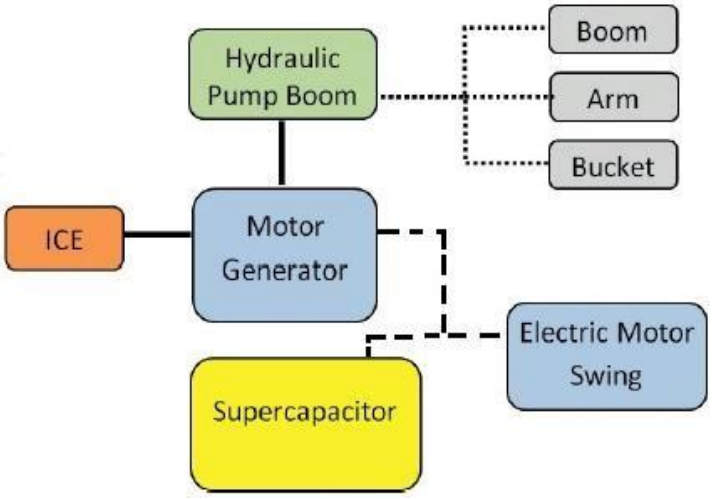


Figura 2.9: ***Schema del powertrain di un escavatore ibrido serie-parallelo***

3. Pale caricatrici

Le pale caricatrici sono veicoli da lavoro utilizzati per la movimentazione di materiale sciolto, al fine di spostarlo e caricarlo su altri veicoli che provvederanno a portarlo ad una opportuna destinazione. Queste macchine possono anche svolgere dei piccoli lavori di scavo. Tra i principali produttori di pale caricatrici, Volvo, John Deere e Hitachi hanno sviluppato delle alternative ibride. Volvo ha realizzato un veicolo ibrido parallelo, denominato Volvo L220F. L'idea è quella di fornire potenza elettrica addizionale quando richiesta, rigenerando la macchina durante le normali operazioni. A partire da tale veicolo, Volvo ha realizzato anche i veicoli LX1 e LX2. Anche l'azienda John Deere ha sviluppato una pala caricatrice con powertrain ibrido, il nome del modello in questione è John Deere 944K. Tale architettura è simile a quella del Volvo LX1, per cui l'ICE è collegato al generatore, il quale è elettricamente collegato a 4 motori elettrici posti a livello delle ruote. La differenza principale è che nel John Deere 944K non vi è un sistema di stoccaggio dell'energia elettrica a bordo veicolo.

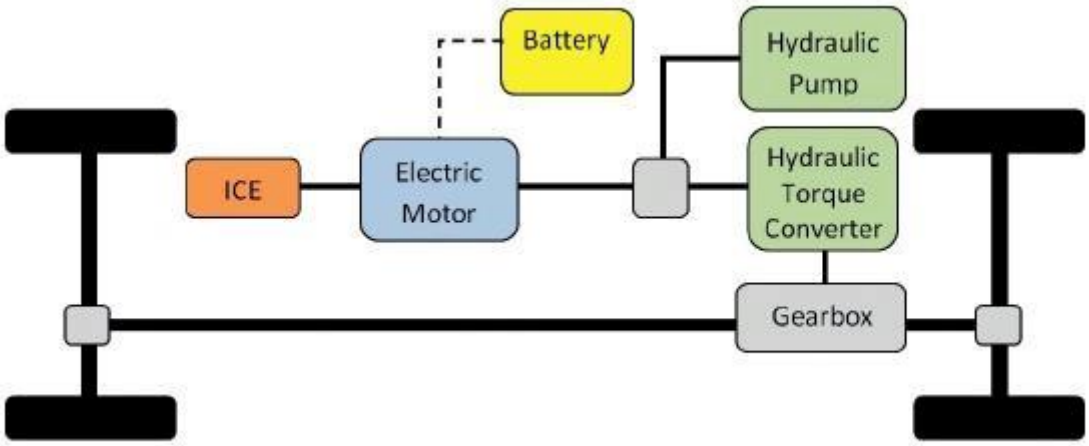


Figura 2.10: ***Schema del powertrain ibrido parallelo (PHEV) di Volvo L220F***

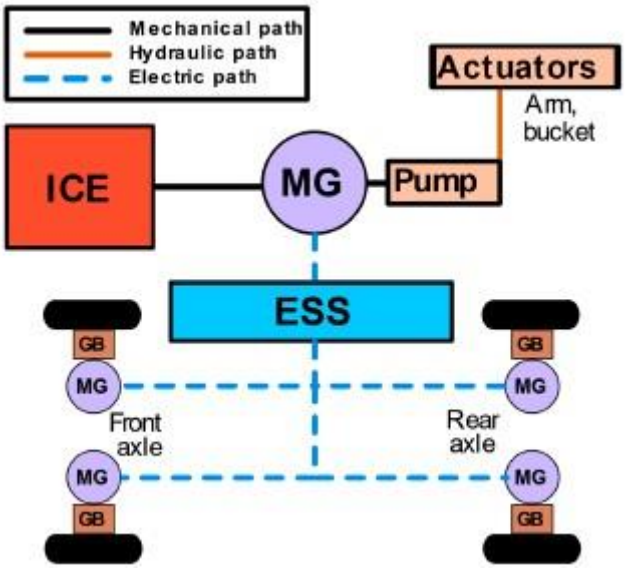


Figura 2.11: ***Schema del powertrain ibrido dalla pala caricatrice Volvo LX1***

4. Sollevatori telescopici

Questa tipologia di veicoli viene usata per sollevare carichi mediante un elemento telescopico, il quale può essere dotato, per esempio, di forche regolabili. In questo campo architetture ibride sono state sviluppate da Merlo, Liebherr e Manitou. In figura 2.24 è rappresentato lo schema del powertrain serie-parallelo presentato dall'azienda Merlo. Tale configurazione si presenta come un ibrido serie per quanto concerne la trazione, mentre si comporta come un ibrido parallelo per l'azionamento della pompa idraulica.

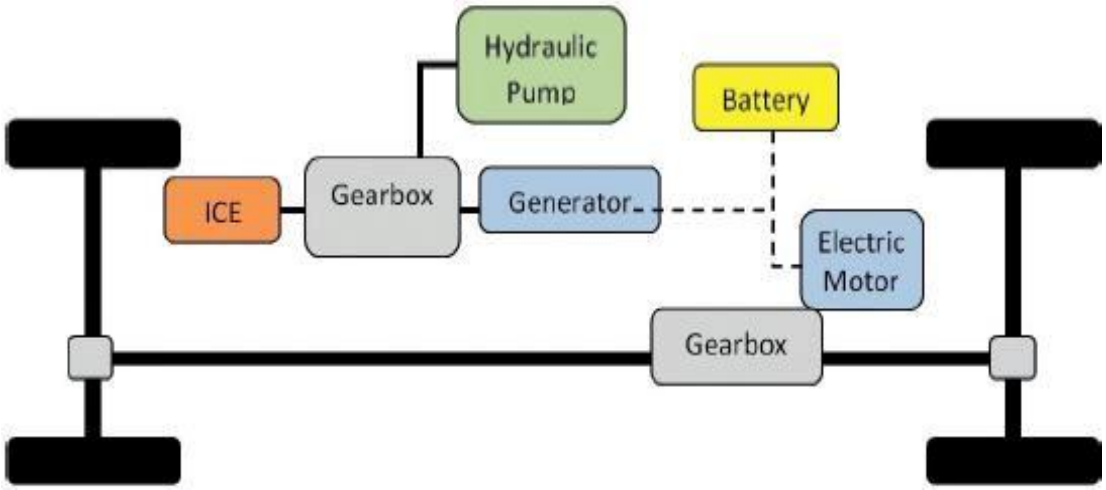


Figura 2.12: ***Schema del powertrain ibrido del sollevatore telescopico Merlo TF 40.7***

5. Trattori agricoli

I trattori agricoli vengono utilizzati per svolgere operazioni nei campi, quali l'erpicatura o l'irrorazione dei campi con particolari sostanze, oppure per il trasporto di materiali. Uno dei primi prototipi di trattori ibridi fu il Belarus/RuselProm 3022e, presentato nel 2009. Tale veicolo non è provvisto di un ESS e si identifica in un ibrido serie. Nel 2011 fu poi presentato dalla compagnia RigiTrac il modello EWD 120, il cui powertrain consiste in una architettura Serie dotato di 4 motori elettrici, uno per ogni ruota. Tale veicolo è inoltre dotato di una interfaccia elettrica per fornire potenza ad eventuali attrezzi attaccati, oltre che della normale PTO (meccanica) collegata direttamente al motore. Nel 2015 Claas ha sviluppato il modello Arion 650 Hybrid, dove l'unità elettrica è montata in parallelo all'ICE e nella trasmissione è presente un CVT. Nel 2016 John Deere ha presentato un trattore completamente elettrico ed un anno dopo anche l'azienda Fendt ha presentato un modello full electric. Come già evidenziato nel Capitolo 1, anche per i trattori specializzati vigneto/frutteto sono in corso di sviluppo i primi prototipi ibridi ed elettrici.

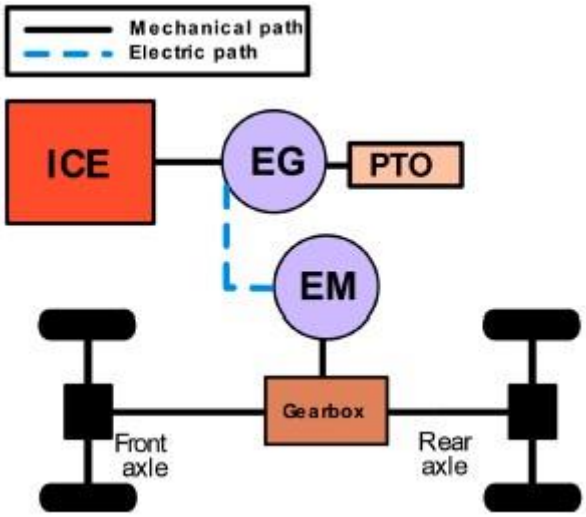


*Figura 2.13:**Schema del powertrain ibrido del trattore Belarus 3022e***

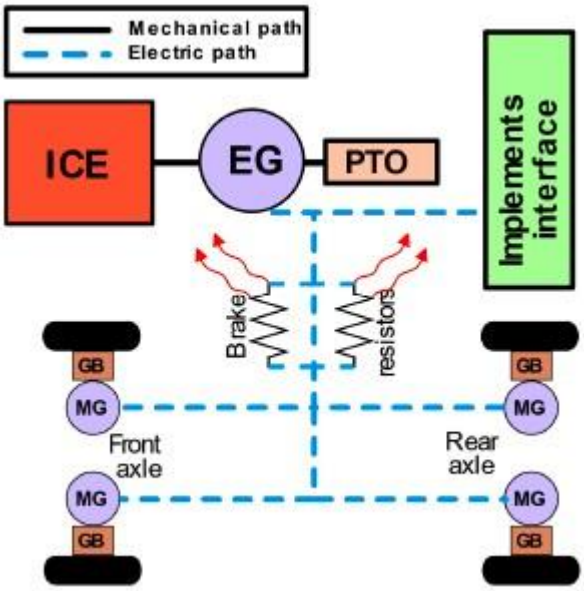


*Figura 2.14: **Schema del powertrain ibrido del trattore RigiTrac EWD120***

Capitolo 3 : Banco Prova in scala 1:1 per il test di motori elettrici per powertrain ibridi ed elettrici

Panoramica sulle tipologie di banchi prova per trasmissioni di potenza

Per poter verificare l'effettiva applicabilità di un certo tipo di sistema di trasmissione di potenza all'interno di un veicolo, è necessario però in qualche modo testare il sistema già in fase di prototipazione e di design del mezzo. In tal senso, i banchi prova costituiscono uno strumento estremamente importante nell'ambito della transmission systems engineering, con il loro impiego che a livello temporale solitamente precede le attività di test su strada o in campo (field test) in caso di veicoli NRMM. Questi sistemi sono infatti pensati per poter riprodurre in maniera più fedele possibile le condizioni di lavoro a cui saranno poi sottoposti i prodotti che si vogliono testare: grazie alle prove a banco si è in grado, infatti, di osservare come le variabili in gioco interagisco e come vanno ad influenzare i parametri sotto la lente di ingrandimento, senza la necessità ad esempio di formulare complessi modelli matematici che riescano a tener conto di tutte le variabili.
Nel caso particolare di moderni veicoli ibridi ed elettrici, inoltre, c'è la necessità anche di testare e di disporre di una rete di alimentazione elettrica di bordo necessaria all'alimentazione dei motori elettrici presenti. Per questo motivo, la funzionalità dei banchi di prova per sistemi di propulsione può essere estesa, creando una sezione di test anche del sistema di alimentazione elettrica e degli azionamenti dei motori, valutando in questo modo anche le interazioni tra sistema puramente meccanico e sistema elettrico. Attraverso quindi la presenza di dispositivi di accumulo energia (in questo caso i pacchi batterie) la sorgente di tensione agisce anche da consumatore di energia, simulando da un lato l'azione dei sistemi sotto test e dall'altro lato quelli di ricarica delle batterie, a meno dei rendimenti (e quindi delle dissipazioni meccaniche, elettriche e termofluide) dei vari componenti montati sul banco stesso: nasce così il concetto di Banco prova a ricircolo di potenza.

In generale, infatti, indipendentemente da quella che è la finalità di utilizzo, un banco prova può appartenere a due grandi classi:

- **Banchi prova a dissipazione di potenza**;
- **Banchi prova a ricircolo di potenza**.

I banchi prova a dissipazione di potenza sono composti da:

-un motore elettrico, che immette energia all'interno del sistema sotto forma di coppia all'albero motore. Nel caso in cui si stia testando proprio un componente che produce energia, come un motore endotermico, un intero veicolo o il motore elettrico stesso, quest'ultimo verrà utilizzato come produttore di energia.

- una trasmissione meccanica, che collega l'albero motore con il sistema dissipatore. Tra i vari componenti presenti all'interno di questa catena cinematica vi sarà anche quello da testare, nel caso in cui non sia presente negli altri due sottogruppi.

- un sistema dissipativo che misuri al tempo stesso l'energia in uscita, solitamente un freno magnetico, tramite cui viene calcolata la potenza dispersa dalla trasmissione o la potenza prodotta all'ingresso, in base a dove sia collocato l'elemento da testare.

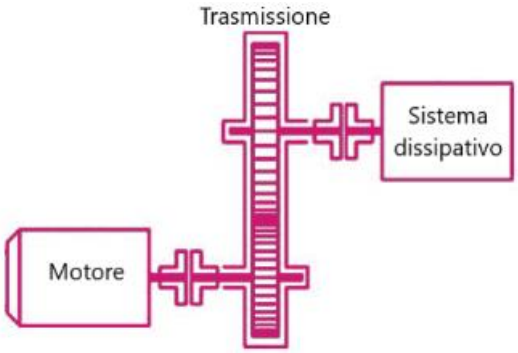


Figura 3.1: ***Schema di base di un banco prova a dissipazione di potenza***

I banchi prova a ricircolo di potenza, invece, sono ulteriormente suddivisibili in due gruppi a seconda della modalità di ricircolo: a ricircolo di potenza meccanica e a ricircolo di potenza elettrica.

I banchi a ricircolo di potenza meccanica sono molto diffusi in ambito industriale per applicazioni finalizzate al test di trasmissioni ad ingranaggi, prove su alberi, giunti e cuscinetti. Questi sistemi sono costituiti da:

- un motore elettrico, per fornire il moto all'ingresso;
- due riduttori disposti frontalmente per creare l'anello di ricircolo della potenza.

Questa configurazione assicura un'elevata variabilità della velocità del banco e permette l'utilizzo di un motore elettrico di piccola taglia, in quanto gli è richiesto solamente di fornire quel minimo di coppia necessaria a mettere in rotazione il cinematismo, sia in corrente continua che in corrente alternata, in abbinamento ad un inverter.
La potenza che viene introdotta dall'esterno, chiaramente in termini di assorbimento, rappresenta così soltanto il 4-5% della potenza complessiva circolante. Lo svantaggio principale di questa tipologia di banco prova è la necessità di realizzare l'anello di potenza con i due riduttori, il che implica una certa rigidezza a livello di applicabilità del sistema.

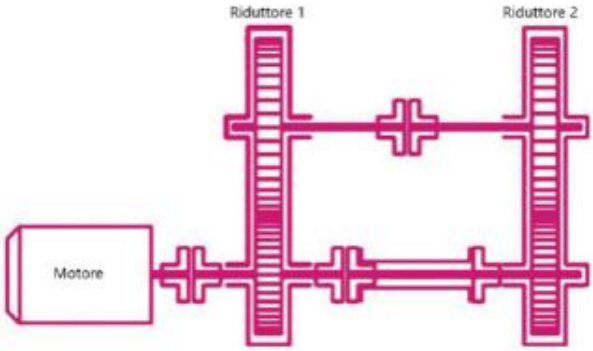


Figura 3.2: ***Schema di base di un banco prova a ricircolo di potenza meccanica***

I banchi a ricircolo di potenza elettrica sono realizzati seguendo lo schema già visto per i banchi dissipativi, ovvero con all'ingresso un motore elettrico collegato ad una trasmissione meccanica. A valle di questi due elementi è però installato un generatore di corrente elettrica, il quale ritrasforma la coppia meccanica posseduta dall'albero di uscita della trasmissione in corrente elettrica per alimentare parte del fabbisogno del motore posto all'ingresso.
Il vantaggio di questa tipologia di banco risiede nel limitato consumo di energia elettrica richiesta dal motore. Per contro la logica di controllo di questi sistemi è decisamente più complessa. Questi banchi prova sono l'ideale per testare motori elettrici.

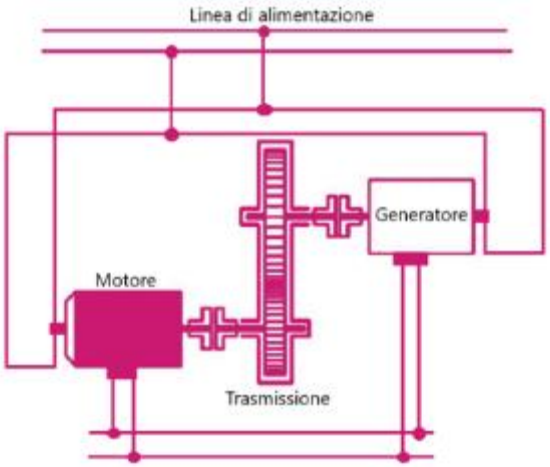


Figura 3.3: ***Schema di base di un banco prova a ricircolo di potenza elettrica***

Ultimo, ma non meno importante, punto di forza tipico dei banchi di prova è la loro versatilità/modularità: questi, infatti, possono e vengono in genere progettati o facilmente riadattati per testare intere classi di componenti o di macchine. Ci sarà dunque una parte del banco che rimane fissa, mentre un'altra porzione dello stesso estraibile e facilmente modificabile.

Test Hardware in the Loop a banco

Si è detto nel paragrafo precedente che i banchi prova sono pensati per poter riprodurre in maniera più fedele possibile le condizioni di lavoro a cui saranno poi sottoposti i prodotti che si vogliono testare. Questa tipologia di test rientra nella di Test in Hardware in the Loop.

L'Hardware in the Loop (HIL) è una metodologia di test e di sviluppo di unità e sistemi di controllo elettronico (come ad esempio le centraline delle autovetture) che coinvolge l'uso di componenti hardware reali (testati a banco) inseriti all'interno di un ambiente di simulazione virtuale (su PC): nell'ambiente HIL quindi si crea una connessione tra il sistema di controllo reale ed un modello virtuale /simulatore che va a rappresentare il comportamento del sistema circostante, potendo includere in quest'ultimo aspetti relativi a componenti meccanici, elettrici ed elettronici con cui il sistema di controllo deve interagire.
I componenti hardware reali, montati a banco, vengono collegati all'ambiente di simulazione attraverso delle interfacce elettroniche e software, che permettono al sistema di controllo reale di comunicare con il modello virtuale e di ricevere dati simulati come risposta alle sue azioni.

Lo scopo delle prove HIL è quindi quello di utilizzare i banchi prova per anticipare le verifiche su componenti, sottosistemi o interi sistemi già nelle fasi di progettazione e prototipazione, senza la necessità di attendere il prodotto finito in quanto i componenti reali che vengono testati rispondono ai segnali simulati come se stessero operando in ambiente reale, dato che non sono in grado di distinguere segnali provenienti da un ambiente software piuttosto che da un ambiente fisico. Il questo modo il metodo HIL permette di riprodurre con i banchi prova le più disparate condizioni operative, osservando come reagisce l'intero sistema e i singoli elementi che lo compongono. Considerando inoltre che il ciclo di sviluppo di nuovi componenti prevede anche la progettazione e la realizzazione dei relativi modelli software, questi ultimi possono essere verificati già in questo stadio di sviluppo dopo averlo installato su apposite centraline riprogrammabili, prima ancora di avere un sistema fisico che lo implementi anche solo in forma prototipale.

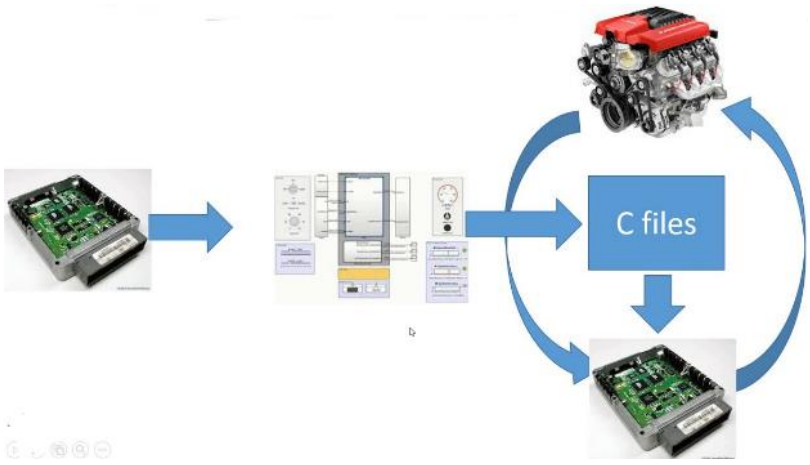


Figura 3.4: ***Esempio di test HIL su un motore termico***

Nel caso di banchi prova per trasmissioni di potenza le tecniche di simulazione HIL consentono quindi di testare il motore insieme al relativo gruppo di azionamento, andando a simulare su di essi quelle che potrebbero essere le reali condizioni di carico in cui si troverebbero ad operare in un ambiente fisico reale.

Il banco prova scala 1:1 realizzato: disposizione e caratteristiche delle parti che lo compongono

Per poter effettuare delle simulazioni il più possibile veritiere, è stato progettato e realizzato un banco di prova di dimensioni reali (in scala 1:1) in modo tale da poter riprodurre in maniera più fedele possibile le situazioni tipiche in cui potrebbero trovarsi, durante la vita operativa effettiva, i powertrain ibridi ed elettrici dei veicoli in esame, in questo caso gli NRMM, eseguendo delle prove HiL con cui si testa l'hardware fisico effettivo (che verrà poi montato sul veicolo effettivo) all'interno di un loop di simulazione del sistema di controllo elettronico realizzato via software.

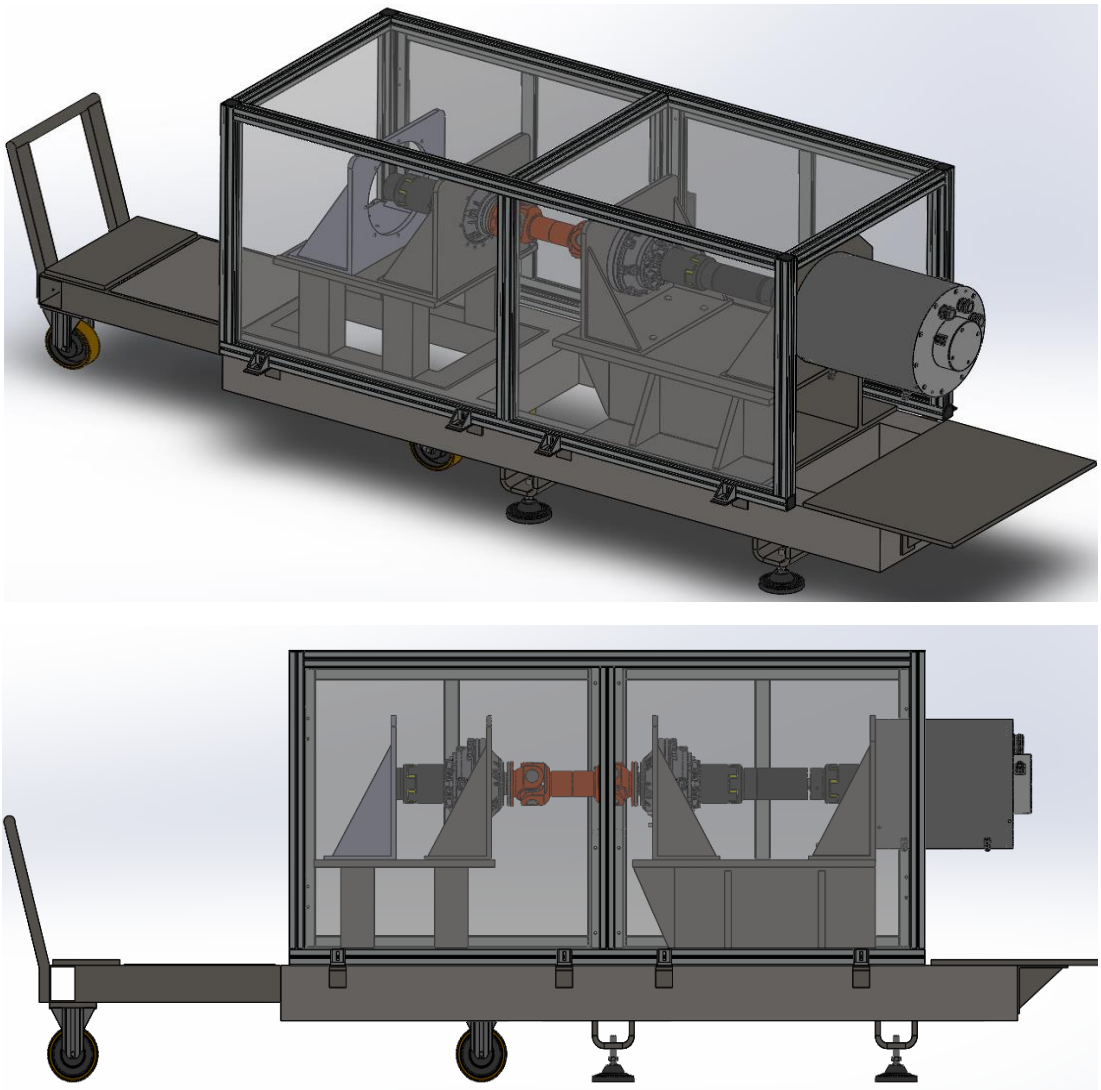


Figura 3.5: ***Vista isometrica e frontale del modello CAD del banco prova realizzato***

Come si può vedere dalla Figura 3.5, il banco è stato concepito per poter essere il più modulare possibile. Si distinguono infatti due aree fondamentali:

- Una parte fissa, collegata direttamente a terra tramite dei tasselli, la cui configurazione è costante nel tempo;
- Un carrello mobile, sul quale possono essere montati componenti differenti.

In questo modo il banco diventa idoneo a testare la parte elettrica di qualunque tipologia di powertrain: il carrello mobile può essere infatti sostituito da un altro carrello, oppure modificandone semplicemente la squadra di ancoraggio. La realizzazione del banco attraverso questa flessibilità nelle parti permette dunque di poter testare la parte elettrica di qualunque architettura ibrida, sia serie sia parallela, così come architetture full-electric: è possibile ricostruire in questo modo le varie architetture di trasmissione di potenza senza dover prevedere la riprogettazione di nuovi supporti ed in modo relativamente semplice in quanto sarà di fatto sufficiente cambiare la sola parte mobile del banco prova.

Entrando nel dettaglio della descrizione della componentistica, sulla parte fissa del banco, in ordine da destra verso sinistra, sono montati:

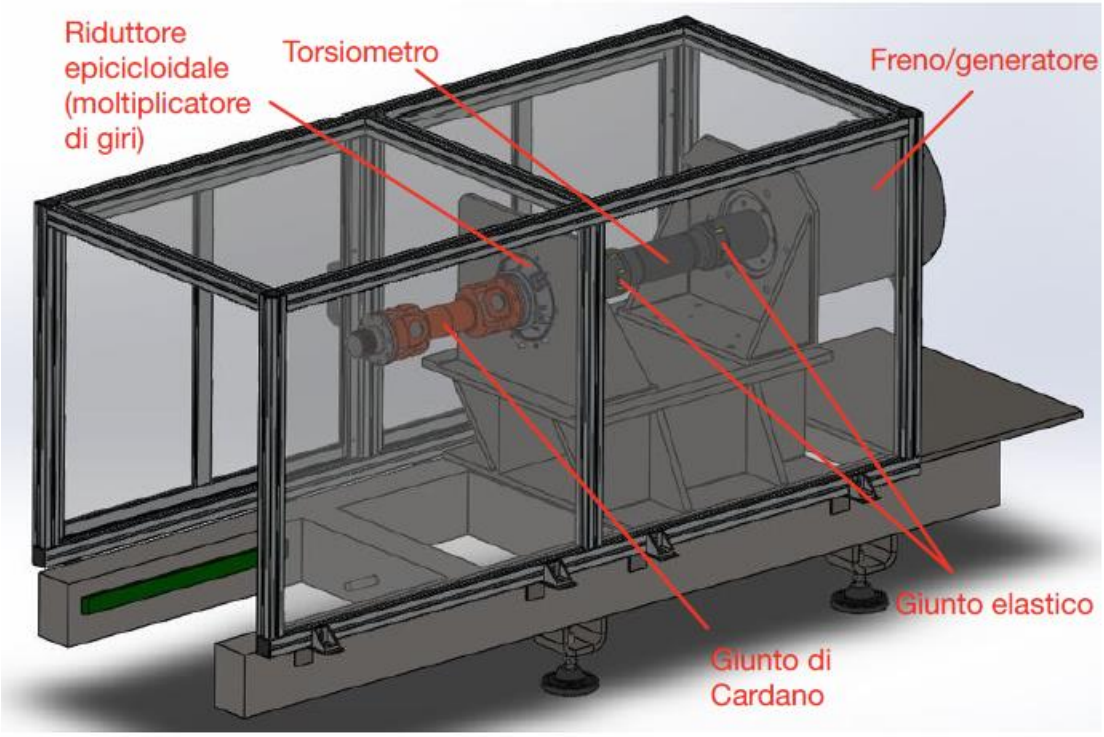


Figura 3.6: ***Modello CAD della parte fissa del banco***

- una macchina elettrica reversibile sovradimensionata, che agirà prevalentemente da freno/generatore. Quest'ultimo è dotato di un apposito azionamento e di un circuito termo-refrigerante dedicato (composto a sua volta da una pompa, da uno scambiatore di calore e dal fluido termovettore di raffreddamento), non rappresentate sul modello CAD per questioni di semplicità realizzativa dell'assieme;
- un giunto elastico di collegamento tra la macchina elettrica e la parte terminale del torsiometro;
- un torsiometro, per misurare la coppia e la velocità angolare sulla linea di trasmissione (la misura di velocità angolare può essere effettuata anche attraverso il resolver montato sul rotore del freno elettrico);
- un ulteriore giunto elastico, per il collegamento tra ingresso del torsiometro e riduttore epicicloidale;
- un riduttore epicicloidale, utilizzato in questo caso in realtà come moltiplicatore di giri per riportare la velocità angolare al valore nominale a cui viene condotto il test, dopo che questa è stata ridotta per poter installare un giunto cardanico;

- un giunto di Cardano, elemento di collegamento tra la parte fissa e la parte mobile del banco. Si preferisce utilizzare un giunto omocinetico di questo tipo in quanto tra la parte fissa del banco (ancorata a terra) ed il carrello mobile possono nascere dei disallineamenti (imputabili principalmente alle irregolarità della pavimentazione sulla quale il banco è posizionato ed alle tolleranze di lavorazione dell'intera struttura del banco). La necessità di installare questa tipologia di giunto giustifica anche la scelta di montare due riduttori in opposizione ai due lati del giunto stesso: infatti su questo banco prova possono essere testati motori che lavorano ad elevati numeri di giri, mentre il cardano ammette velocità di rotazione basse per limitare possibili centrifugazioni dello stesso. Per effettuare allora la compensazione dei disallineamenti, si installa un riduttore epicicloidale nella parte terminale del carrello mobile in ingresso al giunto (che abbassa la velocità di rotazione della linea) ed un moltiplicatore di giri a valle del cardano in ingresso alla parte fissa del banco, così da riportare la velocità angolare al valore nominale in ingresso prima al torsiometro e poi all'albero del freno/generatore.

Sulla parte mobile del banco, a prescindere dalla configurazione che si intende testare di cui si discuterà più in dettaglio nel seguito, come già appena detto vi sarà sicuramente nella sua parte terminale un riduttore epicicloidale per poter entrare nel cardano con una velocità di rotazione idonea, tematica discussa precedentemente.

Figura 3.7: ***Modello CAD della configurazione base di un carrello mobile del banco prova***

La modularità del banco ammette inoltre la possibilità di cambiare la coppia di riduttori del banco, installandone altri a diverso rapporto di trasmissione, a seconda delle specifiche necessità, oppure, più semplicemente, è possibile installare un ulteriore riduttore in ingresso alla trasmissione del banco qualora si volesse ulteriormente ridurre la velocità, compatibilmente con le coppie ammissibili dalla trasmissione stessa. Si riporta un possibile schema di quest'ultima soluzione costruttiva, in cui è quindi stato aggiunto un ulteriore riduttore epicicloidale alla linea meccanica, necessario per testare motori elettrici che lavorano a regimi di velocità più elevati di quelli ammissibili dalla configurazione base del banco prova, andando a effettuare una modifica sulla squadra di ancoraggio.

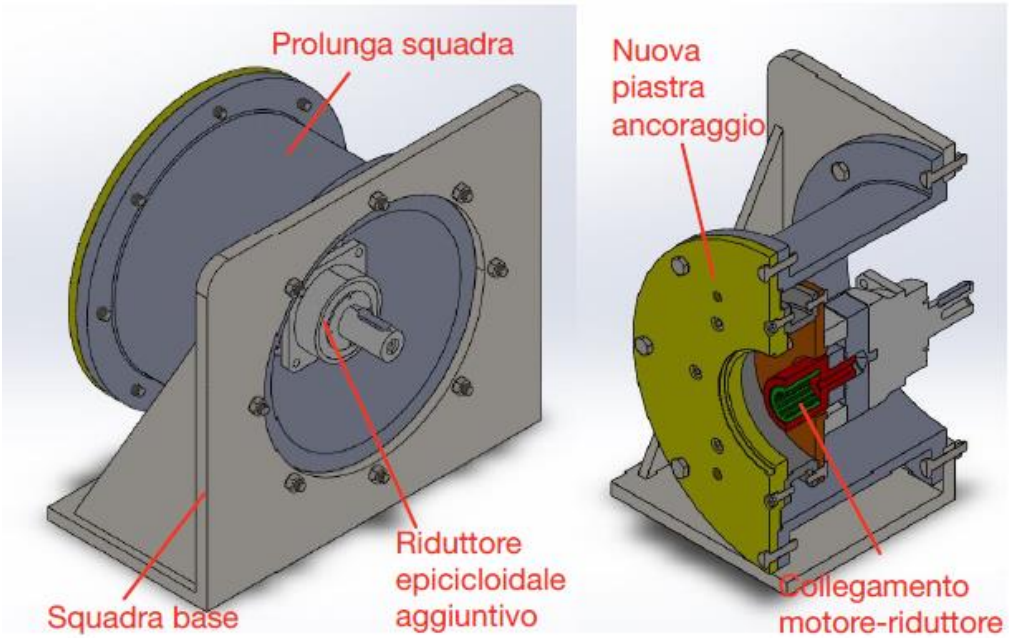


Figura 3.8: ***Modifica della trasmissione meccanica del banco prova con aggiunta di ulteriore riduttore epicicloidale***

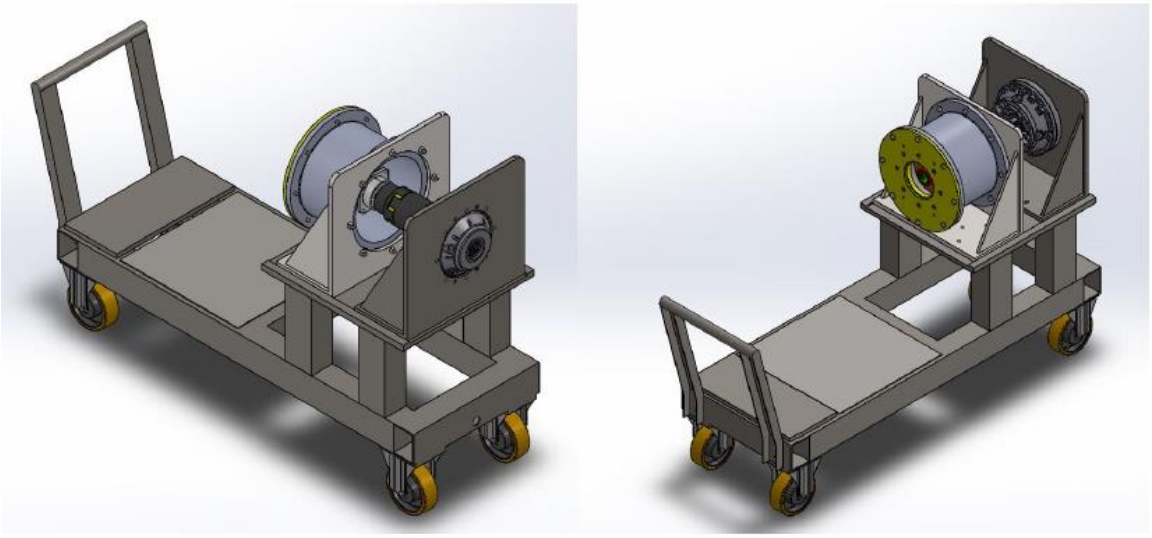


Figura 3.9: ***Viste assonometriche del carrello mobile del banco prova modificato per testare motori elettrici ad elevate velocità***

In entrambi i casi, comunque, la possibilità di customizzare il sistema in più modi consente di poter testare motori che lavorano in un ampio range di velocità, mantenendo invece la velocità dal lato del freno/generatore sempre all'interno del suo range nominale, eseguendo così le prove in completa sicurezza.

Dai modelli CAD presentati in precedenza non si evidenzia, oltre ai sistemi di azionamento e ai circuiti di raffreddamento predisposti e dedicati a motore e freno/generatore, un altro elemento fondamentale che caratterizza questo tipo di banco: il sistema di accumulo di energia elettrica, costituito da dei moduli posti in serie di batterie agli ioni di litio, di seguito rappresentati.

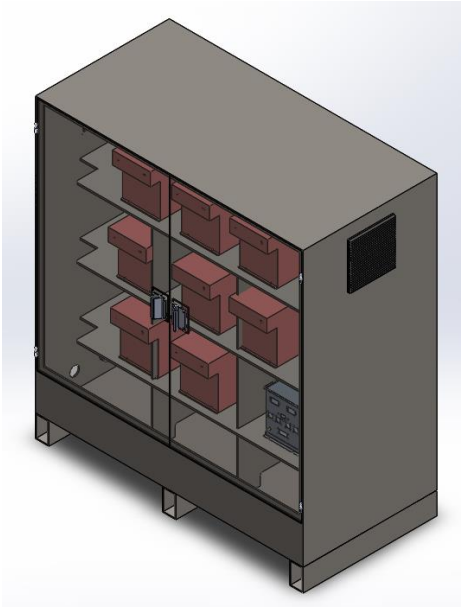


Figura 3.10: ***Modello CAD delle batterie predisposte per il banco prova***

Il banco realizzato si configura infatti come banco a ricircolo di potenza elettrica: il freno elettrico, montato sulla parte fissa del banco, non dissipa la potenza ma agisce come generatore di corrente, rigettando quest'ultima all'interno delle batterie. Queste ultime a loro volta forniranno l'energia elettrica necessaria all'alimentazione della parte elettrificata del powertrain sotto test. Si crea quindi un loop di energia, con le batterie che forniranno al netto solamente la quota di energia che viene dissipata durante le prove a banco, e schematizzabili come riportato in Figura 3.11:

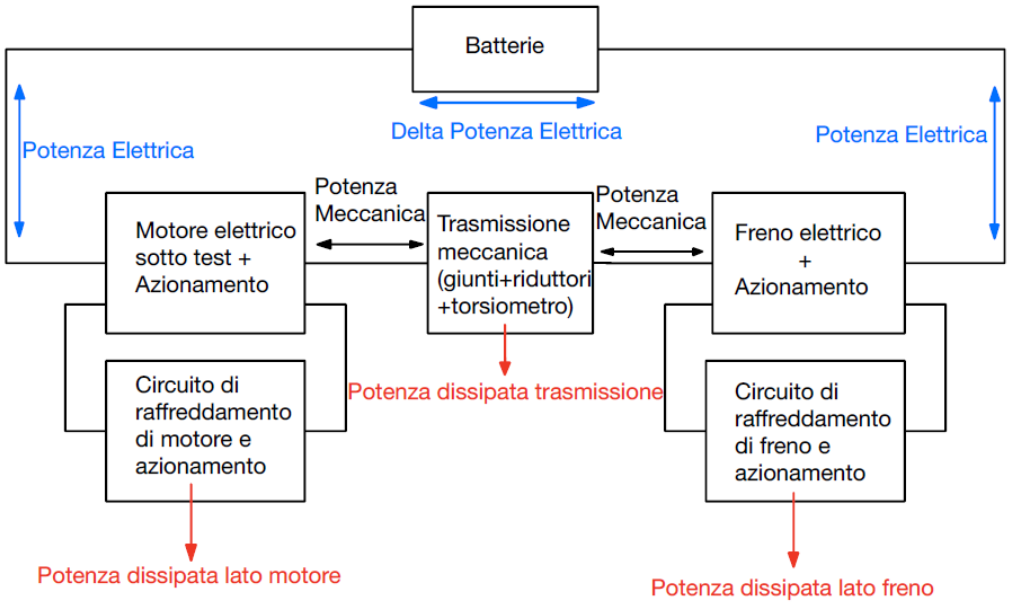


Figura 3.11: ***Schema di ricircolo di potenza del banco prova realizzato***

E' inoltre possibile il funzionamento inverso essendo le macchine elettriche reversibili.

Il banco sarà quindi in grado di funzionare fintanto che l'energia accumulata nei pacchi batteria è sufficiente a vincere tutte le fonti di dissipazione presenti sul banco stesso. Per poter valutare quindi l'autonomia del banco, è necessario quantificare le diverse forme di perdita di energia, andando a caratterizzare i vari elementi, sia attraverso campagne di acquisizioni sperimentali sia utilizzando i dati forniti da catalogo per i vari componenti scelti, anche nell'ottica di ricreare un modello virtuale il più veritiero possibile rispetto alle prestazioni del banco reale.

Si riportano quindi nella tabella che segue alcuni dati relativi alle componenti finora descritti per la costruzione del banco:

Dati di Targa Freno/Generatore	
Coppia Nominale	220 [Nm]
Velocità Nominale	3500 [rpm]
Potenza Nominale	80 [kW]
Corrente Nominale	100 [Arms]
Dati di Targa Modulo Batteria	
Numero di Moduli	8
Tensione Nominale	80 [V]
Capacità	25 [Ah]
Dati di Targa Riduttori epicicloidali RR310 FS r4 BOC	
Rapporto di Riduzione	4
Coppia Nominale Albero di Uscita	2600 [Nm]
Velocità Massima Albero di Ingresso	3500 [rpm]

Tabella 3.1: ***Dati di targa dei componenti del banco prova***

Caratteristiche del motore elettrico utilizzato durante le prove a banco e progettazione dei dispositivi di accoppiamento necessari per il suo ancoraggio

Tornando alla parte mobile del banco, si è detto come l'unico elemento certo nella parte terminale di qualunque carrello venga montato sia il riduttore epicicloidale collegato al giunto cardanico di accoppiamento alla parte fissa (vedi Figura 3.5).

Nello specifico però dei test effettuati per questa attività di tesi, il banco prova è stato utilizzato per testare un motore destinato ad un veicolo dotato di powertrain full-electric, nel quale quindi la potenza è interamente fornita dal motore elettrico in prova. Per testare questa tipologia di motore è stato possibile utilizzare il carrello estraibile riportato in Figura 3.7 nella sua configurazione di base: questo motore infatti lavora in un range di velocità perfettamente compatibile con quelle ammissibili dal banco, per come è stato costruito, non necessitando quindi di ulteriori riduzioni di velocità come mostrato in Figura 3.8 e 3.9, se non di modifiche necessarie a garantire i corretti accoppiamenti meccanici sia alla squadra di ancoraggio sia alla trasmissione precedentemente descritta.

Il motore elettrico in questione è un motore PMSM (Motore Sincrono a Magneti Permanenti), un motore sincrono a corrente alternata la cui eccitazione di campo viene fornita da magneti permanenti. Si riportano nella Figura 3.12 il modello CAD del motore in questione ed in Tabella 3.2 le relative specifiche tecniche:

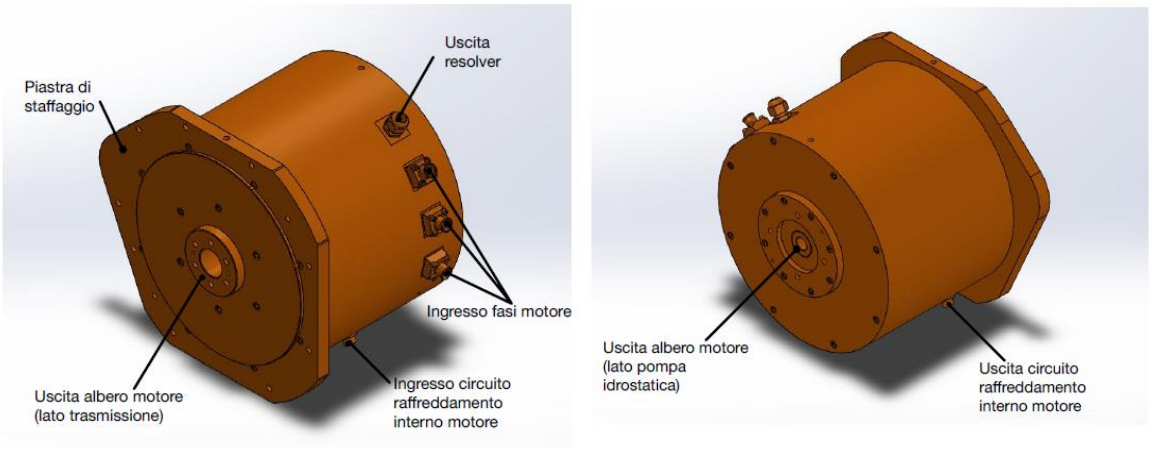


Figura 3.12: ***Modello CAD del motore elettrico PMSM testato a banco***

Dati di Targa Motore PMSM in test	
Coppia Nominale	117 [Nm]
Velocità Nominale	2880 [rpm]
Potenza Nominale	35 [kW]
Corrente Nominale	70 [Arms]
Temperatura Limite di Funzionamento dei Magneti Permanenti	120 [°C]

Tabella 3.2: ***Dati di targa motore in test***

Come anticipato, il motore in questione è stato progettato per poter essere montato su un veicolo NRMM full-electric: il motore deve allora soddisfare i requisiti di potenza dell'intera macchina, sia meccanica sia idraulica. Per questo motivo sull'albero del rotore, come da figura, sono predisposte due uscite: una per l'ingresso in trasmissione meccanica ed una per alimentare la pompa idraulica per l'appunto.

Data la geometria sia della piastra di staffaggio del motore, caratterizzata da una forma poco convenzionale e da fori disposti in posizioni particolari, sia dell'albero motore, per poter allora testare questo motore a banco è stato necessario agire in due direzioni:

- Riuscire a collegare un pattern circolare di fori della squadra di ancoraggio del carrello estraibile del banco con il pattern invece non circolare della piastra motore, distanziando in maniera sufficiente la squadra dalla piastra motore per non avere problematiche di ingombro, data la particolare geometria di quest'ultima;
- Contestualmente predisporre un dispositivo di trasmissione di potenza dall'albero del motore elettrico all'ingresso del banco prova.

Per risolvere contemporaneamente le due problematiche sono stati quindi studiati e realizzati degli adattatori, riportati nella Figura 3.13 a seguire:

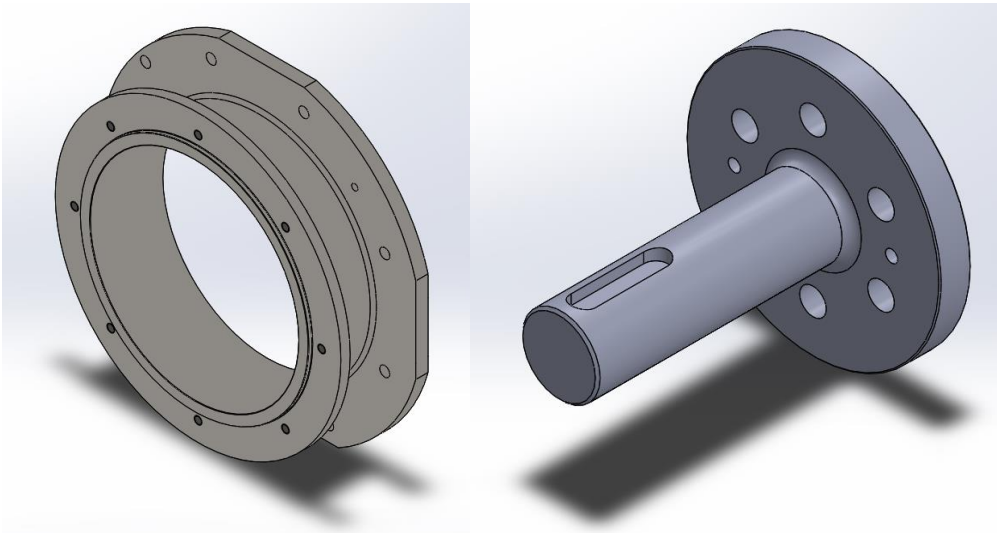


Figura 3.13: ***Modelli CAD dei dispositivi di adattamento del motore al banco prova***

La campana di collegamento riportata a sinistra della Figura 3.13 consente di risolvere la prima problematica, disponendo infatti di un pattern di fori che replica quelli predisposti sulla squadra di ancoraggio del banco e dalla parte opposta da un pattern di fori che replica quelli previsti dalla piastra del motore.

La prolunga di trasmissione riportata invece a destra della Figura 3.13 consente di risolvere la seconda problematica di trasmissione del moto: è stato adottato un collegamento per attrito, garantito dall'utilizzo di brugole serrate con chiave dinamometrica, tra l'albero del motore elettrico e la prolunga ed un collegamento tramite linguetta tra la prolunga ed il giunto elastico di ingresso al banco prova.

La lunghezza assiale dei due dispositivi soprariportati è inoltre strettamente correlata: la campana deve essere sufficientemente lunga per consentire un agevole montaggio delle viti di fissaggio ed allontanando la piastra motore dalla squadra del carrello, ma al tempo stesso non deve essere

eccessivamente lunga al fine di evitare una lunghezza eccessiva della prolunga (la posizione del giunto elastico è infatti vincolata e non può quindi essere variata).

Si riporta quindi nella Figura 3.14 a seguire il modello CAD dell'assemblaggio del motore con la squadra di ancoraggio del banco prova predisposta sul carrello estraibile.

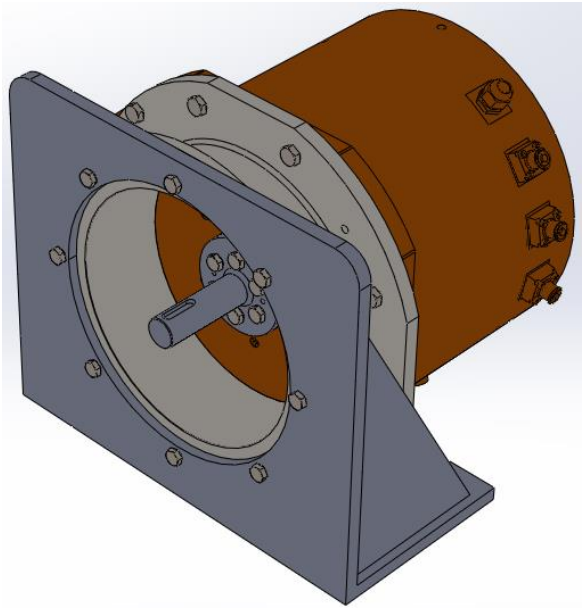


Figura 3.14: ***Modello CAD del motore montato sulla squadra di ancoraggio attraverso i dispositivi di accoppiamento progettati***

In Figura 3.14 si riporta invece il modello CAD dell'intero banco prova a seguito del montaggio anche del motore sotto test.

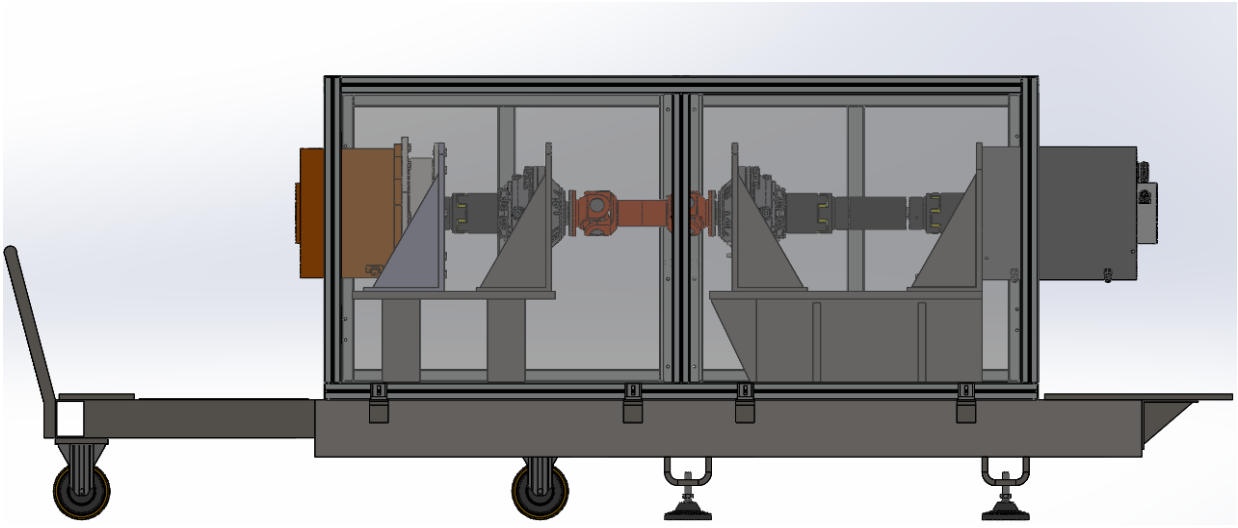


Figura 3.15: ***Modello CAD del banco prova completo di motore in test***

Una volta terminata la progettazione delle parti sopra descritte e definite le opportune tolleranze geometriche e dimensionali per garantire tutti gli accoppiamenti voluti, si è provveduto a far realizzare i componenti fisici ed una volta effettuati tutti gli accoppiamenti il risultato ottenuto è quello riportato in Figura 3.16:

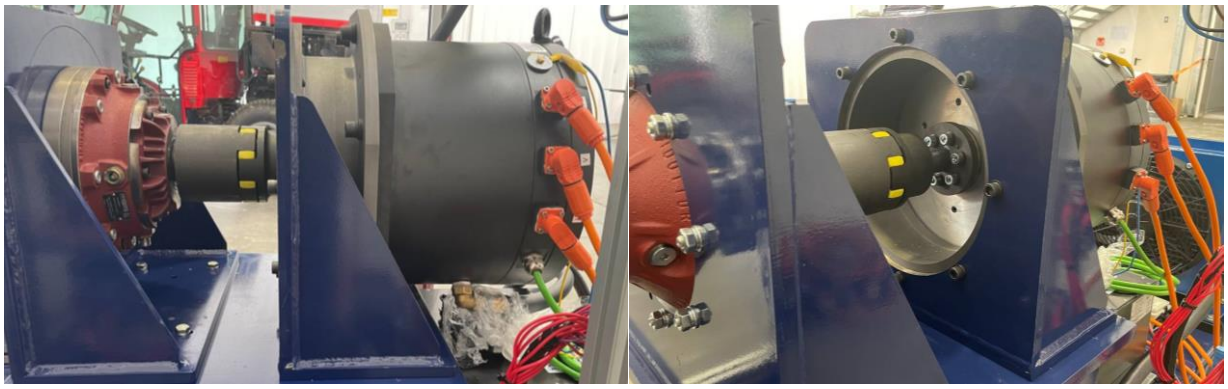


Figura 3.16: ***Vista frontale ed assonometrica degli accoppiamenti realizzati tra motore sotto test e banco reale***

Si riporta quindi, in Figura 3.17, una vista frontale dell'intero banco prova, dalla quale sono visibili anche tutte le componentistiche non riportate sui vari modelli CAD per ovvi motivi di semplicità realizzativa degli stessi.

Figura 3.17: ***Vista frontale completa del banco prova realizzato con motore sotto test accoppiato***

Rimane non visibile in Figura l'armadio con all'interno i moduli batterie, disposti posteriormente al banco prova.

Nonostante ciò, è possibile, da tale figura, mettere in evidenza i blocchi dei vari componenti a banco e porla a confronto con lo schema già riportato in precedenza in Figura 3.11.

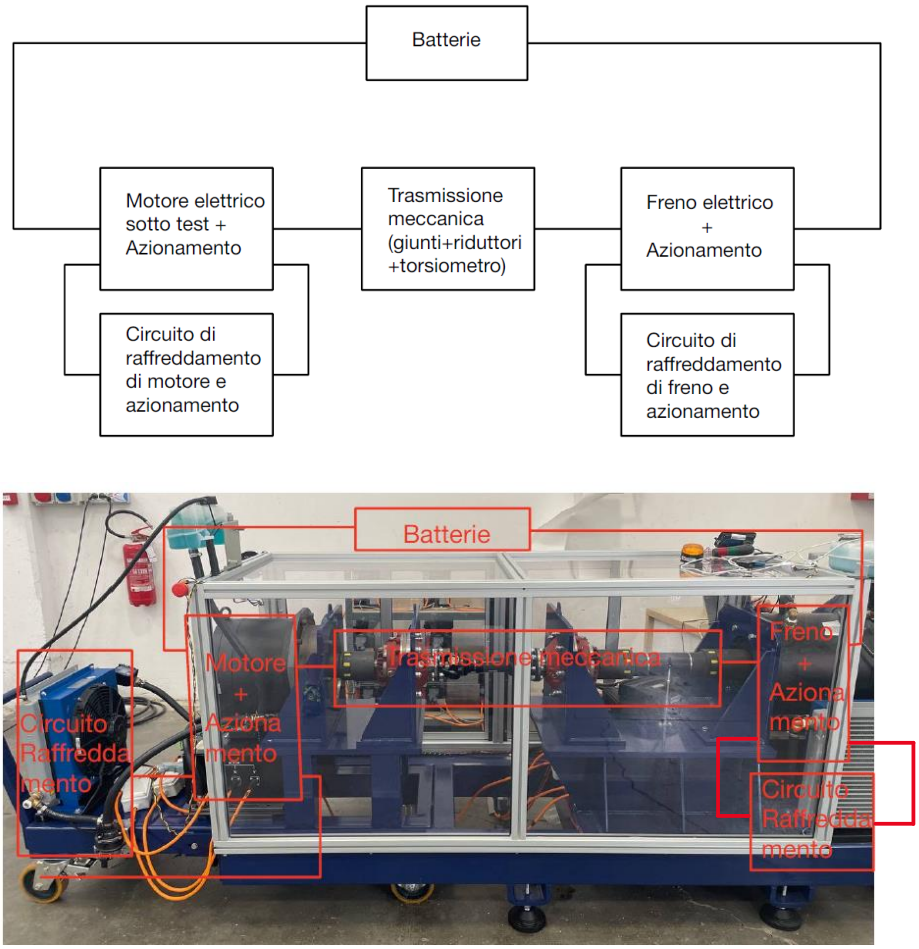


Figura 3.18: ***Confronto tra schema puramente a blocchi e banco prova fisico***

Schema HiL del banco prova realizzato

Si sottolinea, infine, come il sistema costruito rispetti perfettamente i requisiti di un banco per simulazioni HiL.

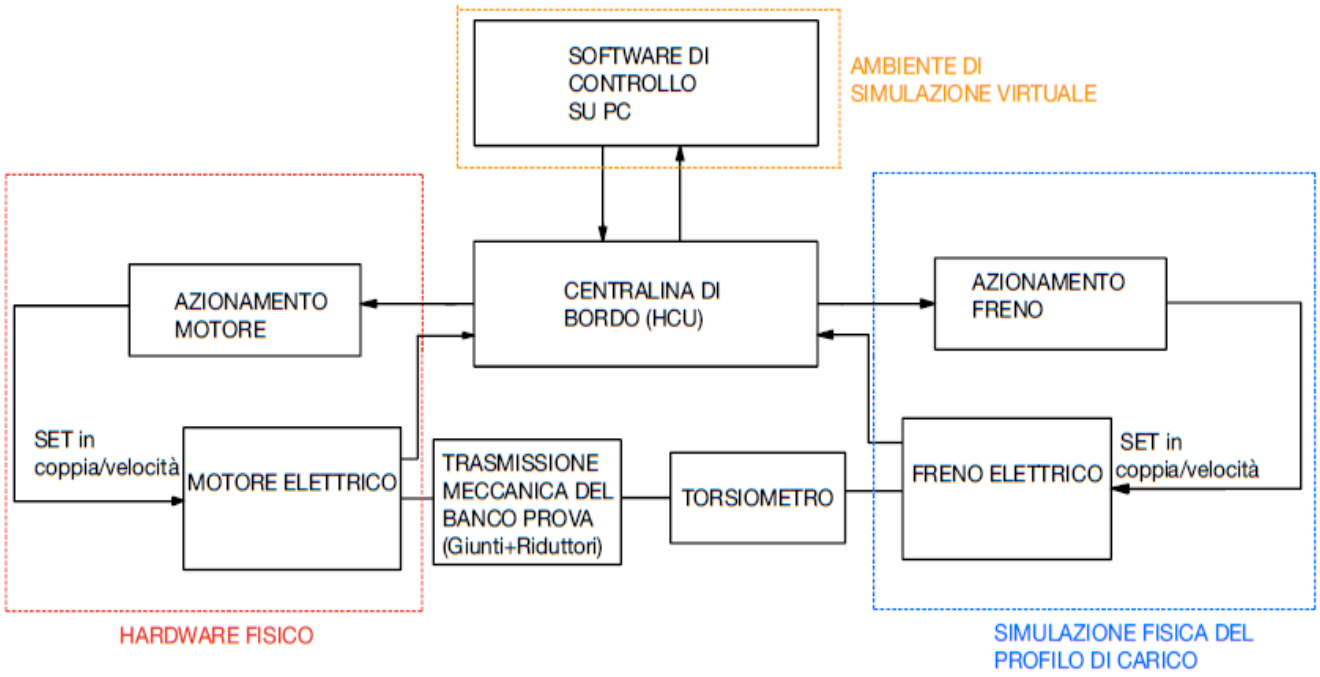


Figura 3.19: ***Schema HiL del banco prova realizzato***

Tutti gli scenari a cui l'hardware in test potrà andare incontro nella sua vita operativa verranno infatti simulati agendo sul controllo del freno/generatore del banco, sul quale si possono simulare i profili di missione tipici (costituiti da un certo andamento nel tempo della coppia resistente che il motore deve sostenere a determinati regimi di velocità operativa), potendo in questo modo anche testare il sistema di controllo su software.

Capitolo 4 : Valutazione delle prestazioni del banco prova e del motore elettrico sotto test

Caratterizzazione della linea di trasmissione meccanica del banco

Poiché il primo utilizzo effettivo del banco prova (esclusi i test preliminare per verificare il corretto funzionamento dei vari sistemi) coincide con questa attività di tesi, allora per poter valutare correttamente le caratteristiche tecniche di qualunque motore che viene montato a banco per poter essere testato, è necessario preliminarmente valutare quelle che sono le dissipazioni di potenza che vengono generate dalla trasmissione meccanica del banco prova.

Per poter far ciò, è possibile utilizzare il banco prova in una condizione di funzionamento "a vuoto", vale a dire disaccoppiando meccanicamente il motore dalla linea e facendo muovere la stessa a differenti step di velocità attraverso un controllo in velocità effettuato freno-generatore del banco. Il torsiometro è infatti stato disposto tra la linea meccanica ed il freno, pertanto lettura su di esso in questa configurazione fornirà la coppia necessaria a mantenere esclusivamente la linea meccanica ad una determinata velocità, al netto anche del rendimento di conversione elettromeccanica del freno in quanto la lettura effettuata dal trasduttore avviene direttamente sull'albero del generatore : il controllo in velocità del freno infatti farà sì che, dato un SET di velocità, quest'ultimo (agente in questa prova in realtà da motore, consumando un certo quantitativo di corrente) andrà ad erogare il quantitativo di coppia minima e necessaria per mantenere l'intera catena a quella determinata velocità, corrispondente alla somma delle coppie assorbite dai vari elementi del sistema stesso, ovvero riduttori e giunti meccanici vari. Non avendo predisposto delle misure puntuali di coppia in tutti i punti della trasmissione in cui questa ha delle variazioni significative (e quindi tra monte e valle di entrambi i riduttori del banco) non è possibile effettuare ricavare sperimentalmente i rendimenti delle singole parti, bensì solo quello relativo all'intero sistema di trasmissione.

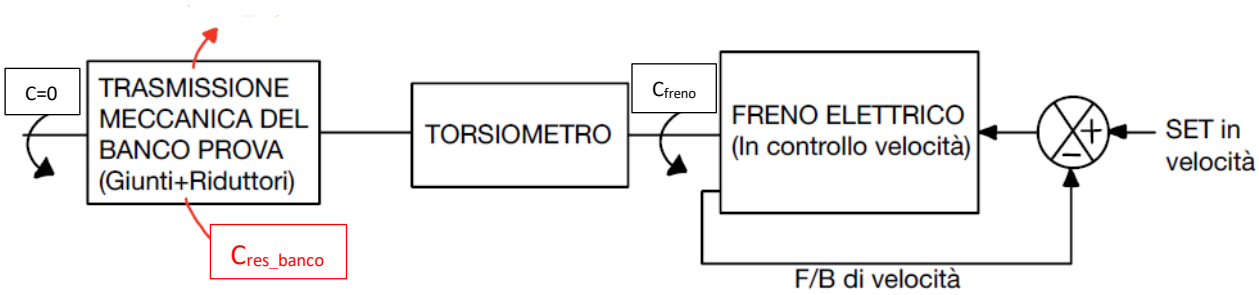


Figura 4.1: ***Schema di controllo del banco prova durante test di caratterizzazione delle perdite del banco stesso***

Per quanto riguarda la procedura di prova, il seguente test è stato eseguito per step di velocità, partendo da 0 rpm e procedendo per multipli di 360 rpm fino alla velocità di 2880 rpm, corrispondente al range di funzionamento operativo del banco prova stesso (si è infatti detto nel capitolo precedente come il punto debole in tal senso del sistema sia costituito dal giunto di Cardano, ed in particolare dalla sua velocità limite di rotazione).

Si riportano quindi i risultati ottenuti a seguito delle acquisizioni effettuate.

Velocità (rpm)	Coppia_res_banco (Nm)	Potenza_res_Banco (kW)
360	4	0,15
720	5	0,38
1080	5,5	0,62
1440	6,5	0,98
1800	7,5	1,41
2160	8	1,81
2520	9	2,38
2880	10	3,02

Tabella 4.1: ***Acquisizioni a banco per la caratterizzazione a vuoto dello stesso***

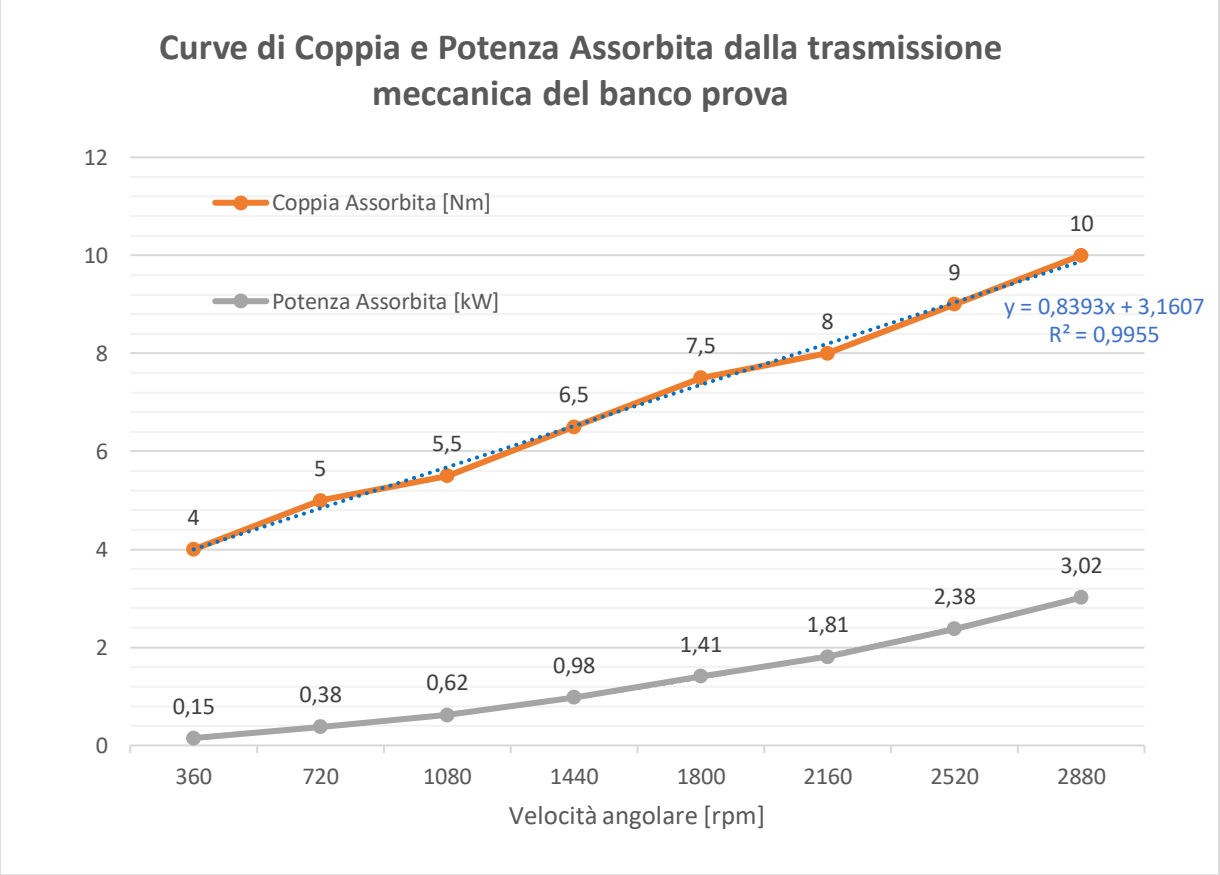


Figura 4.2: ***Andamenti di Coppia e Potenza resistente della trasmissione del banco al variare della sua velocità angolare***

Dai risultati ottenuti si può notare come la coppia assorbita dalla trasmissione abbia un andamento circa lineare al variare della velocità angolare della stessa, e conseguentemente la potenza assorbita, ottenuta come prodotto della coppia per la velocità angolare, è costituita da una funzione quadratica della velocità angolare.
Questa tipologia di prova non tiene chiaramente conto dell'effetto dovuto a variazioni di temperatura, in particolare dei riduttori epicicloidali e del loro olio lubrificante, essendo eseguita in condizioni di carico nulla: gli eventuali aumenti di temperatura che si possono generare negli ingranaggi passando dalle prove a bassa ad alta velocità sono infatti in questo caso trascurabili e poco significativi essendo questa tipologia di test piuttosto rapida e soprattutto a coppie applicate piuttosto ridotte.

Caratterizzazione del motore elettrico testato a banco

Una volta noti gli assorbimenti di potenza meccanica del banco a vari regimi di velocità, è stato possibile effettuare dei test per caratterizzare le prestazioni del motore elettrico PSMS riportato in Figura 3.12.

In una prima fase si eseguono su quest'ultimo due tipologie di prove:

1- **Test BEMF:**

E' una tipologia di prova per motori elettrici caratterizzati da rotori a magneti permanenti, come i PSMS in questione.

Durante un test BEMF (Back ElectroMotive Force) si va a misurare la tensione indotta generata tra le fasi del motore quando il suo rotore viene messo in rotazione tramite un trascinamento rotatitivo attuato da un organo esterno. Il test BEMF di un motore a magneti permanenti consente di rilevare eventuali problemi costruttivi intrinsechi del rotore, dei magneti e di rilevare eventuali disallineamenti angolari: il motore viene quindi trascinato a diverse velocità di rotazione, appuntando quella che è la tensione generata sulle tre fasi. Per ognuna delle fasi è possibile rilevare il valore della costante elettrica K_e e di distorsione armonica, entrambi parametri indicatori di un corretto funzionamento del motore.

La conformazione del banco prova costruito permette di eseguire questa prova molto agevolmente: si collega meccanicamente, tramite giunto elastico, il motore alla linea di trasmissione del banco prova, che viene portato in rotazione effettuando un controllo di velocità sul freno/generatore, come indicato dallo schema in Figura 4.3.

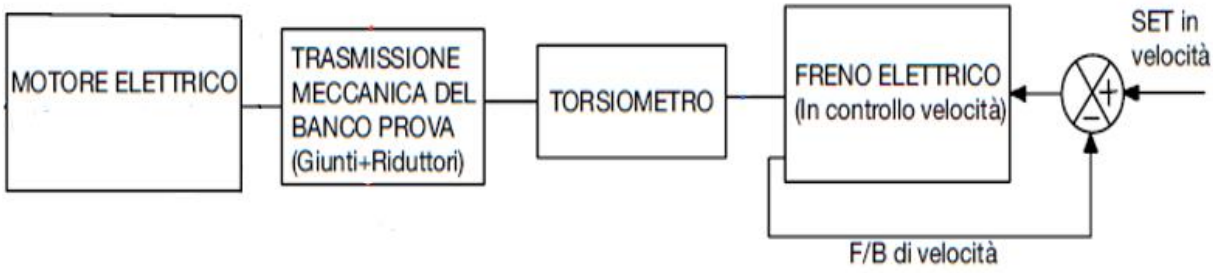


Figura 4.3: ***Schema di controllo del banco prova durante test BEMF del motore elettrico***

Il motore è collegato solo meccanicamente al banco ma non elettricamente, in quanto le 3 fasi devono restare libere per eseguire le misure di differenza di potenziale elettrico tra le fasi prese a coppie, alle diverse velocità di rotazione.

Anche in questo caso, come per le prove "a vuoto" di caratterizzazione del solo banco, la procedura di test consiste nel procedere per step di velocità crescenti: partendo da 0 rpm sì procede per multipli di 360 rpm fino alla velocità di 2880 rpm.

Portando quindi il banco alle diverse velocità sopra riportate si sono ricavati i seguenti dati:

Velocità di rotazione [rpm]	BEMF [VRms]
360	45,8
720	92,3
1080	138
1440	184,4
1800	230,5
2160	276
2520	322
2880	368

Tabella 4.2: ***Acquisizioni a banco per test BEMF del motore elettrico in prova***

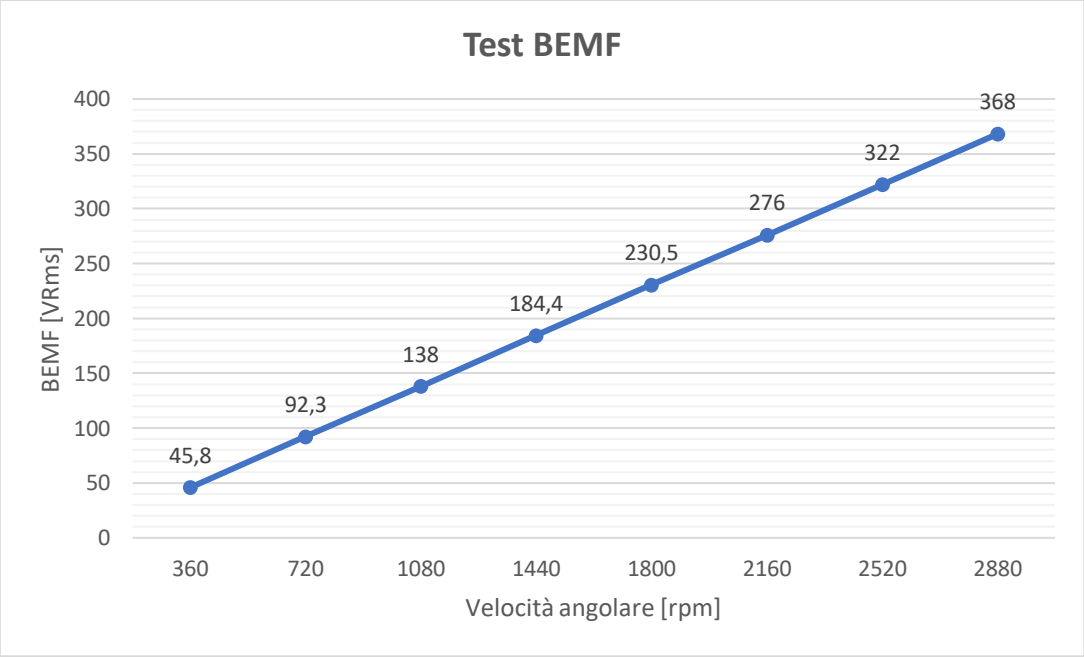


Figura 4.4: ***Test BEMF eseguito sul motore elettrico in prova a banco***

Dai risultati ottenuti si può notare come il motore testato presenti una BEMF perfettamente lineare al variare della velocità angolare del suo rotore: questa linearità risulta particolarmente desiderata, in quanto semplifica notevolmente il controllo del motore, rendendo prevedibile la sua risposta a delle variazioni di velocità, essendo la costante di velocità del motore effettivamente costante in tutto il suo range di utilizzo. Da un punto di vista costruttivo, un risultato come quello ottenuto e riportato in Figura 4.4 è quindi un primo indicatore di buona qualità realizzativa.

2- Ricostruzione della Curva Caratteristica

In generale i motori elettrici sono caratterizzati da una curva di potenza che cresce con l'aumentare del numero di giri, fino a stabilizzarsi ad un certo valore (tratto di Isopotenza). La curva di coppia, invece, ha il suo massimo già a numero di giri nullo e si mantiene costante (tratto di Isocoppia) fintanto che la potenza del motore continua ad aumentare, per poi diminuire fino ad un valore minimo in corrispondenza del massimo numero di giri ammissibile per il motore stesso. Un andamento tipico è quindi quello riportato a seguire in Figura 4.5.

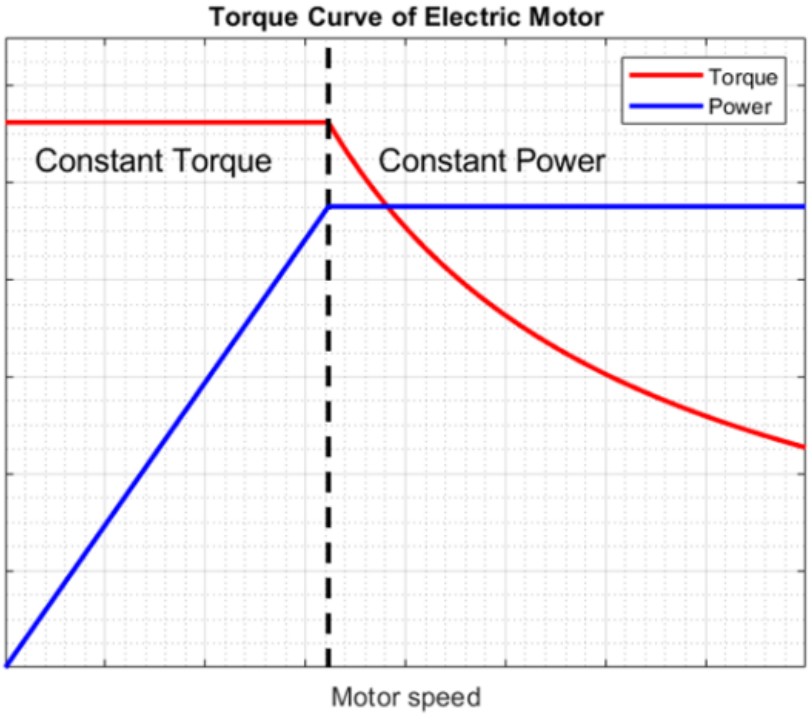


Figura 4.5: ***Curva caratteristica di un generico motore elettrico***

I dati di targa del motore in Figura 3.12 indicano una potenza massima di 35 kW misurati ad una velocità di 2880 rpm, corrispondenti quindi ad una coppia massima erogabile di circa 117 Nm. Durante questa prova si è andati dunque a ricostruire la sua curva caratteristica, adottando in questo caso la seguente strategia di controllo:

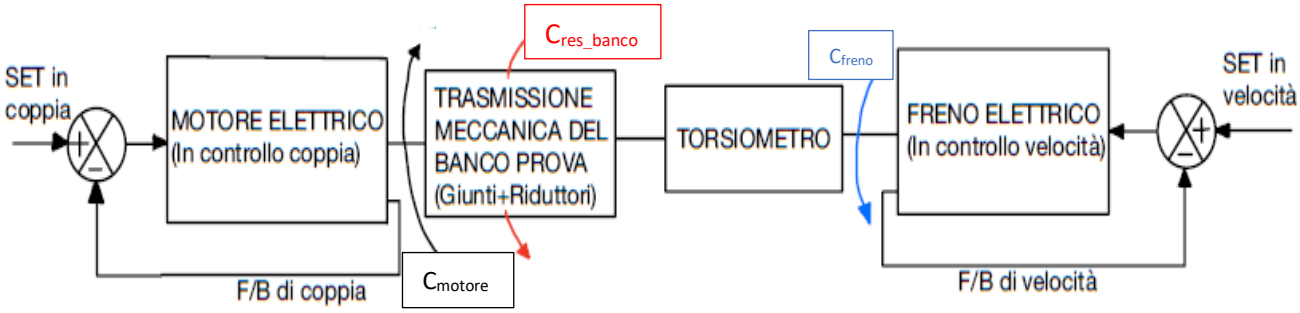


Figura 4.6: ***Schema di controllo del banco prova durante test di caratterizzazione del motore elettrico in prova***

Il freno elettrico viene nuovamente controllato in velocità ma in questo test agisce effettivamente come freno generatore: il motore controllato in coppia erogherà la massima coppia possibile ad ogni regime di velocità, coppia che risulterà notevolmente superiore alla coppia assorbita della linea di trasmissione e che tenterebbe dunque ad accelerare; il controllo in velocità del freno però impedisce che ciò che si verifichi, andando ad applicare a valle della linea di trasmissione del banco la coppia resistente (che si somma a quella propria del banco precedentemente determinata) necessaria a mantenere l'intero sistema alla velocità imposta dal controllo. Agendo effettivamente da freno, quest'ultimo rigetterà la corrente generata all'interno del pacco batterie del banco.
In questo caso, la coppia che verrà letta dal torsiometro corrisponderà dunque proprio alla coppia

resistente applicata dal freno. Per risalire quindi alla coppia effettivamente erogata dal motore sarà necessario andare a sommare alla coppia letta quella precedentemente ricavata allo stesso regime di velocità. Come per la prova di caratterizzazione della linea di trasmissione del banco, l'effetto della temperatura risulta trascurabile data la rapidità del test eseguito.

Si riportano dunque i risultati ottenuti a valle delle acquisizioni.

Velocità (rpm)	Coppia Torsiometro (Nm)	Coppia_res_Banco (Nm)	Coppia Erogata Motore a 70ARms (Nm)	Potenza Motore (kW)
360	113	4	117	4,41
720	112	5	117	8,82
1080	111	5,5	116,5	13,18
1440	108,5	6,5	115	17,34
1800	107,5	7,5	115	21,68
2160	106,5	8	114,5	25,90
2520	105,5	9	114,5	30,22
2880	104,5	10	114,5	34,53

*Tabella 4.3: **Acquisizioni a banco per la caratterizzazione del motore in prova***

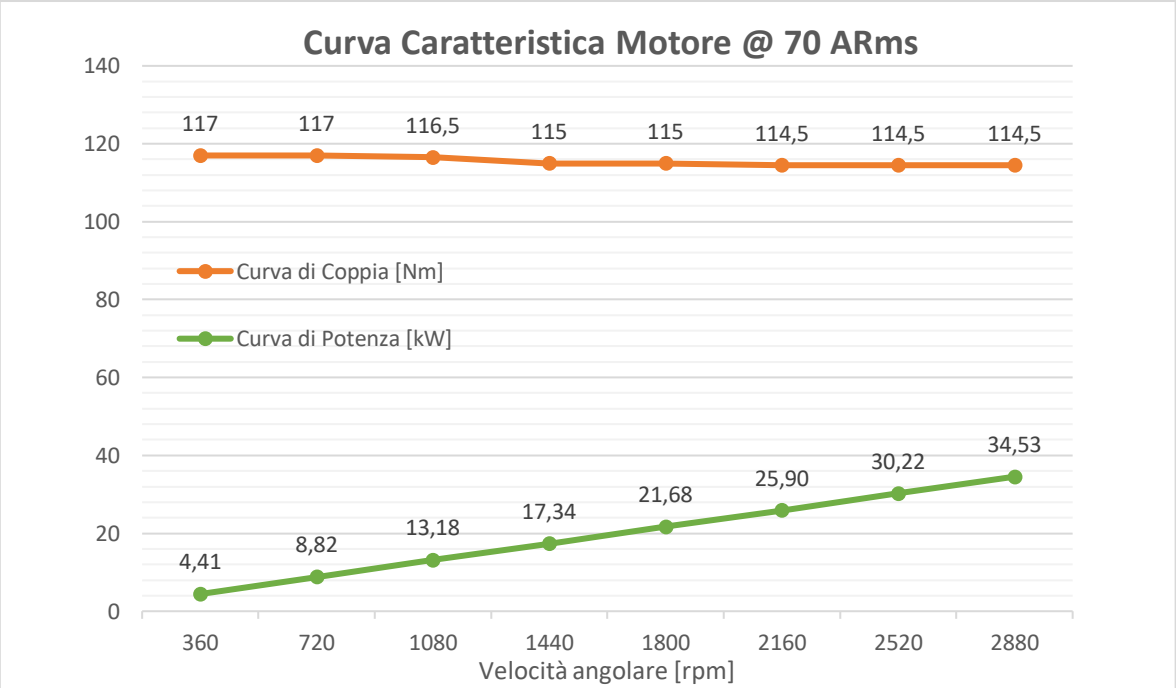


*Figura 4.7: **Curva caratteristica del motore elettrico testato a banco***

Come si può vedere dagli andamenti ricavati in Figura 4.7, confrontando i risultati ottenuti con la curva di un generico motore elettrico, riportata in Figura 4.5, si può sicuramente affermare che nel range di velocità tra 0 e 2880 rpm il motore testato è sostanzialmente in condizioni di iso-coppia, avendo registrato una variazione pari a soli 2.5 Nm tra i due estremi di velocità. Contestualmente la potenza massima erogabile aumenta di conseguenza linearmente con la velocità angolare.

Non è stato possibile ricavare la curva caratteristica completa, verificando anche la zona di iso-potenza ad elevate velocità per via dei limiti già discussi sulla velocità massima della linea di trasmissione del banco. Queste informazioni sarebbero state comunque di poco interesse dato che il punto di funzionamento nominale per tale motore, una volta montato sul veicolo a cui è destinato, è a 2000 rpm, con una richiesta di potenza in tali condizioni di 23 kW che viene perfettamente rispettata, come è visibile da Figura 4.7.

Test di durata delle prestazioni di motore in prova e del banco. Verifica del corretto funzionamento dei circuiti di raffreddamento

Una volta terminate le prove di caratterizzazione sulla trasmissione del banco e del motore su di esso montato, si è proceduto effettuando questa volta dei test di durata in modo da verificare contemporaneamente:

- in che modo variano le prestazioni del motore e conseguentemente del banco prova a seguito degli innalzamenti di temperatura;
- l'efficacia del sistema di raffreddamento installato a banco per azionamento e motore e del software che gestisce l'attivazione della ventola del radiatore per aumentarne lo scambio di calore;

La logica di controllo del banco durante questo test è la stessa del test precedente: un controllo in velocità per il freno-generatore ed un controllo in coppia per il motore in test.

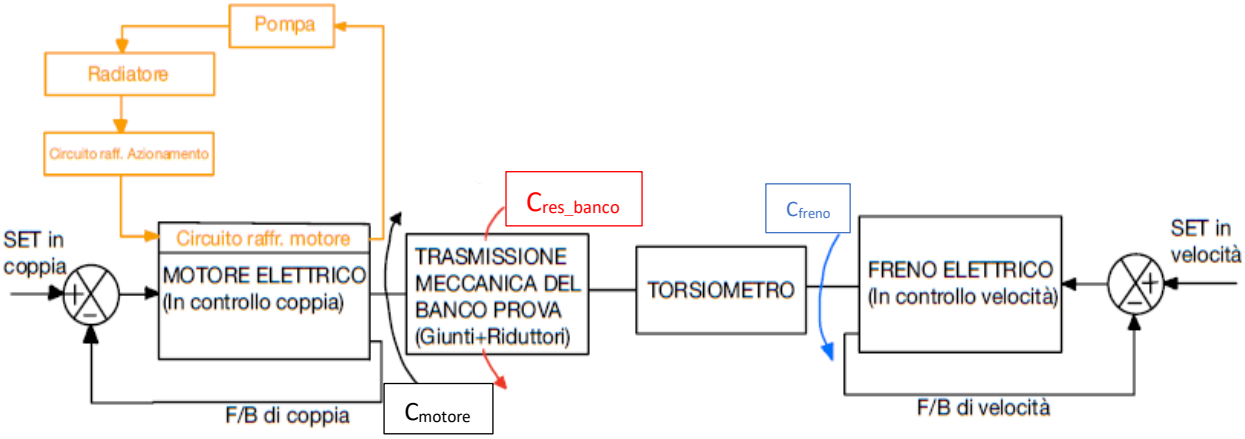


Figura 4.8: ***Schema di controllo del banco prova durante test di durata***

Come si può vedere dalla Figura 4.8, si è deciso di far transitare il fluido termo-refrigerante prima all'interno del circuito dell'azionamento e poi nel circuito interno del motore elettrico: questa scelta deriva dal fatto che l'azionamento, contenente tutta la componentistica elettronica, risulta essere più sensibile ad eventuali innalzamenti di temperatura: facendo transitare in esso il liquido appena uscito dal radiatore (e quindi più freddo) si garantisce in esso un maggior scambio termico e conseguentemente una temperatura più bassa.
Il movimento del fluido nell'impianto è garantito da una pompa centrifuga. La portata risultante dal suo funzionamento è letta attraverso un flussometro installato tra radiatore ed ingresso nell'azionamento.

Nello schema riportato in Figura 4.8 non viene riportato il circuito di raffreddamento per azionamento e freno del banco, perfettamente equivalente a quello riportato sul lato motore: ai fini di questa prova inoltre non si è effettuato alcun monitoraggio delle temperature lato generatore in quanto quest'ultimo è notevolmente sovradimensionato rispetto a quelle che sono le caratteristiche tecniche del motore in prova, pertanto questo lavorerà sicuramente in condizioni di sicurezza sia dal punto di vista delle prestazioni fornite sia dal punto di vista delle temperature di esercizio.

L'obiettivo di questa tipologia di test è quindi quella di verificare che i magneti permanenti installati nel motore non raggiungano, a seguito di un uso prolungato dello stesso, una temperatura tale per cui

si possa verificare un malfunzionamento o addirittura arrivare alla loro smagnetizzazione completa. In generale, infatti, i magneti permanenti perdono la loro magnetizzazione intrinseca all'aumentare della temperatura. La perdita di magnetizzazione può essere temporanea o permanente, a seconda della temperatura raggiunta e della qualità costruttiva e dei materiali utilizzati.

I dati di targa del motore riportano, in tal senso, che il motore può lavorare fino ad una temperatura di 120°C misurata in prossimità dei magneti. Con queste prove si va dunque a leggere tramite un sensore posto sullo statore la temperatura locale e si cercherà di fare in modo che dopo un tempo prolungato, questa temperatura raggiunga un asintoto, ovvero che si stabilizzi al di sotto della temperatura critica dichiarata dal costruttore.

La temperatura dell'azionamento invece viene tenuta sotto controllo attraverso il software fornito dal costruttore dello stesso.

Si riportano dunque i risultati relativi alle prove eseguite, relative a due differenti regimi di velocità:

- **Prova 1: Velocità di rotazione di 1600 [rpm], Portata di fluido di 19.6 L/min**

Tempo di prova [min]	Temperatura Motore a 56 ARms [°C]	Temperatura Azionamento [°C]	Coppia Torsiometro [Nm]
0	44	38	-
1	45	38	105
2	46	38	-
3	49	38	-
4	51	38	NOTA: Ventola radiatore ON
5	52	39	100
6	53	39	-
7	54	39	-
8	56	39	99
9	56	39	-
10	56	39	-
11	57	39	-
12	58	39	-
13	59	40	-
14	60	40	-
15	60	40	-
16	61	40	98
17	62	40	-
18	62	40	-
19	63	41	-
20	63	41	-
21	63	41	97
22	64	41	-
23	64	41	-
24	65	41	-
25	65	41	-
26	66	41	-
27	66	41	-
28	66	41	96
29	66	41	-
30	67	41	-
31	67	41	96

Tabella 4.4: ***Acquisizioni a banco prova durante test di durata di motore e trasmissione a 1600 [rpm]***

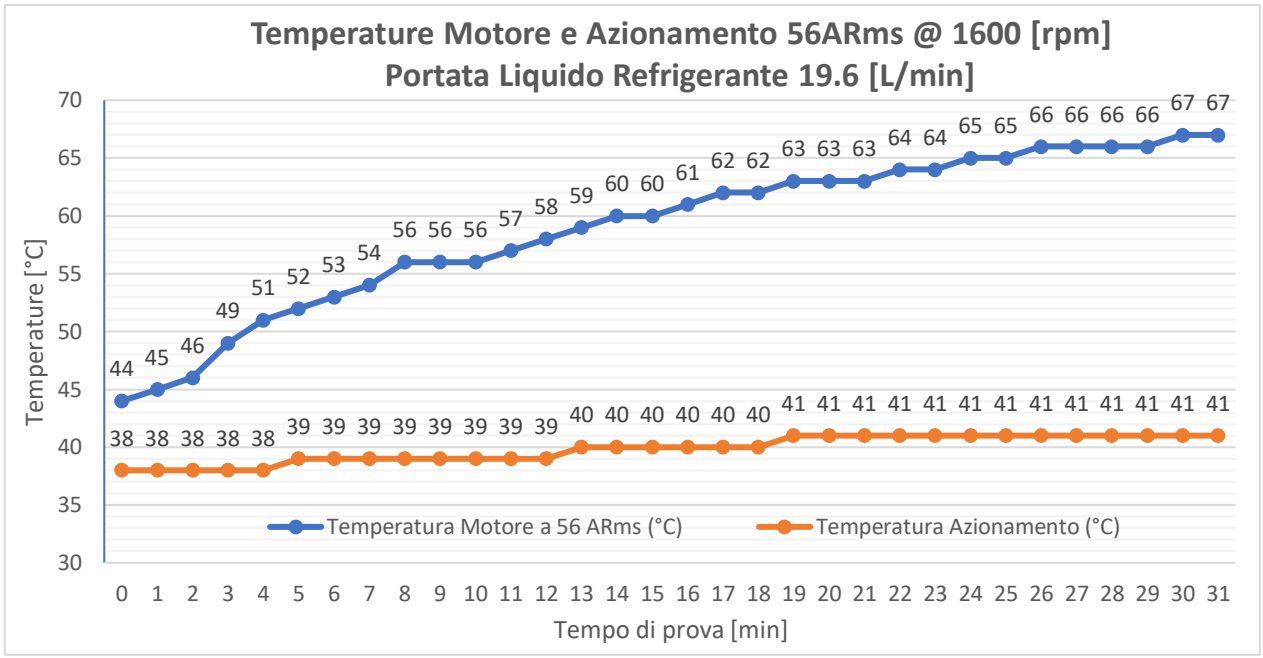


Figura 4.9: ***Andamento delle Temperature di Motore e Azionamento in test a banco a 1600 [rpm] in funzione del tempo di prova***

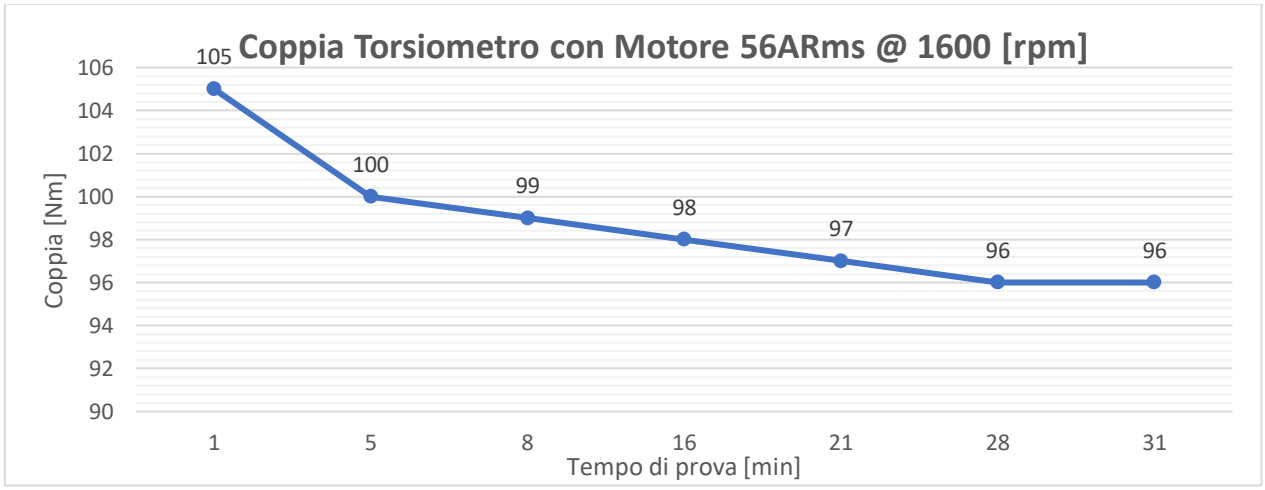


Figura 4.10: ***Andamento della coppia misurata dal Torsiometro a 1600 [rpm] in funzione del tempo di prova***

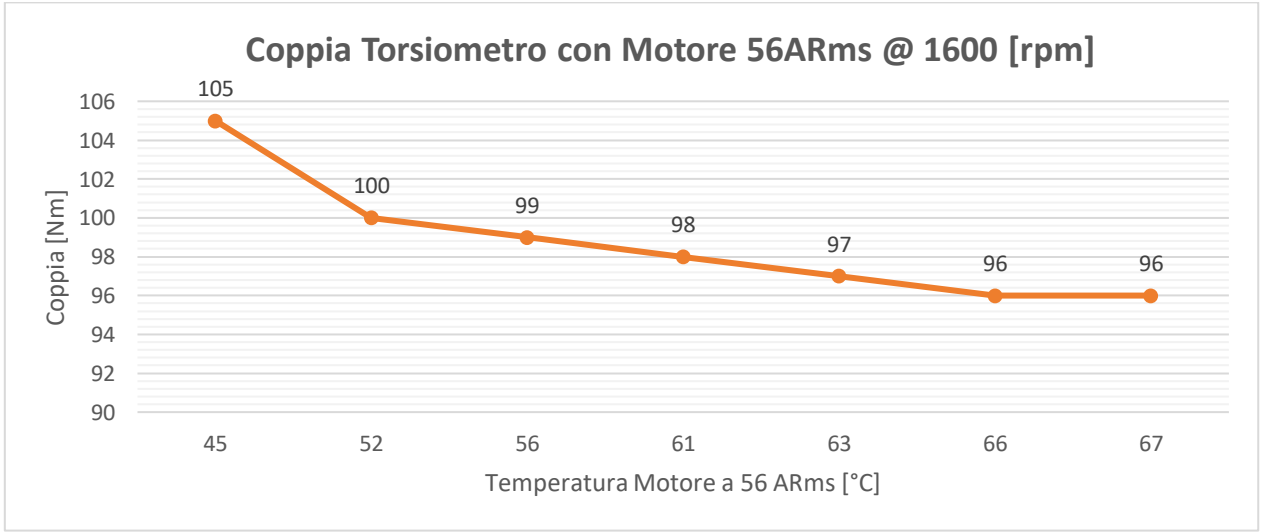


Figura 4.11: ***Andamento della coppia misurata dal Torsiometro a 1600 [rpm] in funzione della Temperatura Motore***

- **Prova 2: Velocità di rotazione di 2600 [rpm], Portata di fluido di 19.6 L/min**

Tempo di prova [min]	Temperatura Motore a 56 ARms [°C]	Temperatura Azionamento [°C]	Coppia Torsiometro [Nm]
0	39	39	-
1	39	39	103
2	44	39	-
3	48	40	100
4	51	40	NOTA: Ventola radiatore ON
5	54	40	-
6	57	40	99
7	59	41	-
8	61	41	-
9	63	41	98
10	65	41	-
11	66	41	97
12	67	42	-
13	68	42	-
14	69	42	-
15	70	42	96
16	71	42	-
17	71	42	96

*Tabella 4.5: **Acquisizioni a banco prova durante test di durata di motore e trasmissione a 2600 [rpm]***

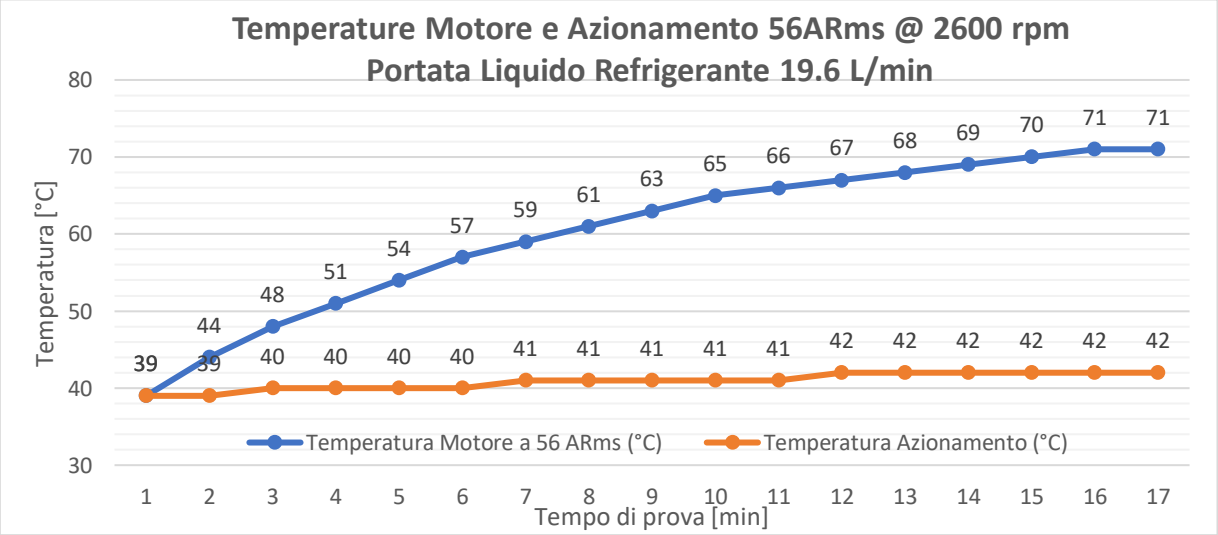


*Figura 4.12: **Andamento delle Temperature di Motore e Azionamento in test a banco a 2600 [rpm] in funzione del tempo di prova***

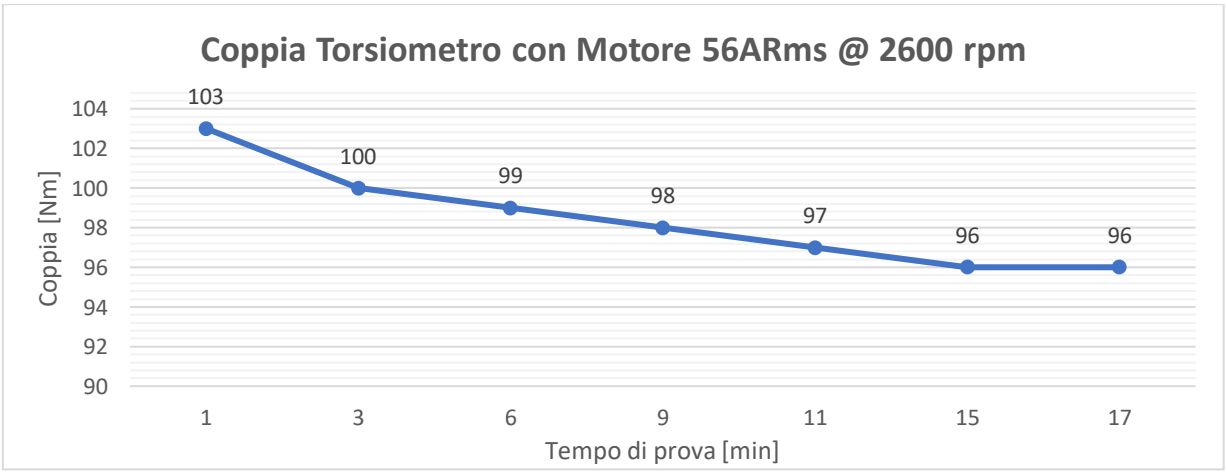


*Figura 4.13: **Andamento della coppia misurata dal Torsiometro a 2600 [rpm] in funzione del tempo di prova***

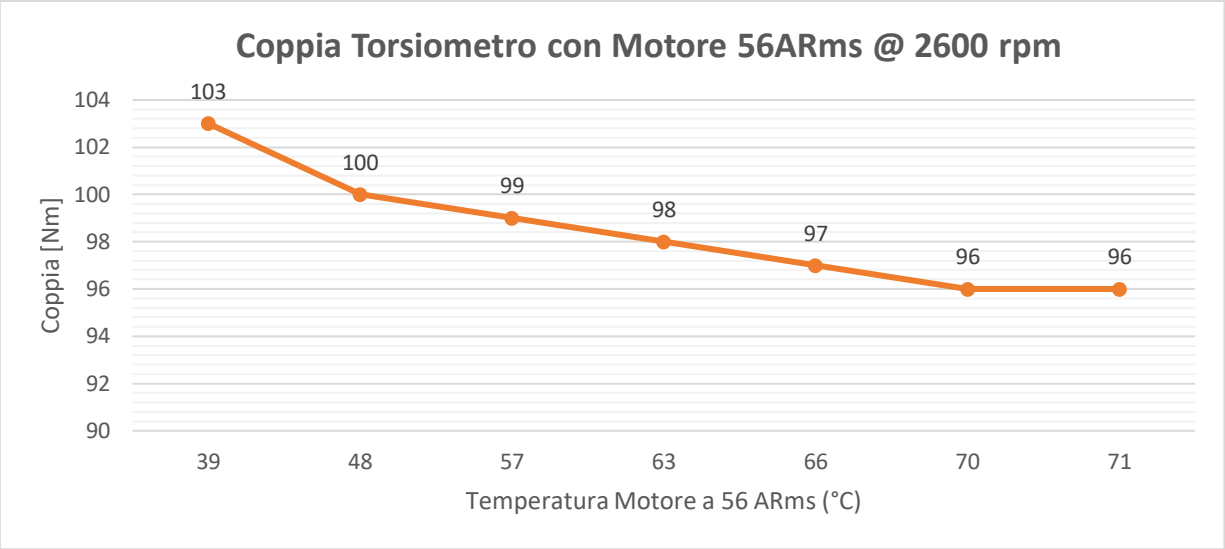


Figura 4.14: ***Andamento della coppia misurata dal Torsiometro a 2600 [rpm] in funzione della Temperatura Motore***

Dai dati acquisiti nel corso delle due prove a diverse velocità si possono estrapolare le seguenti considerazioni:

1. Il motore ed il relativo azionamento rispondono bene all'intervento del circuito di raffreddamento predisposto. Infatti, in entrambe le prove, sia la Temperatura del motore che quella dell'azionamento raggiungono un asintoto orizzontale ben al di sotto dei valori critici sopracitati. Di conseguenza sia i circuiti di raffreddamento interni di Motore ed Azionamento sia quello predisposto sul banco prova lavorano correttamente.
2. Con l'avanzare del tempo di utilizzo in modo continuativo del banco, si nota una leggera perdita di prestazioni, come si può facilmente visualizzare sia dalla Figura 4.10 sia dalla Figura 4.13.
   Questo fenomeno è derivante da un effetto combinato di due fattori:
   - Perdita di prestazioni del lubrificante utilizzato nei riduttori epicicloidali montati a banco: non essendo infatti al momento predisposti dei circuiti di raffreddamento dedicati al contenimento delle temperature degli oli, la rotazione prolungata degli ingranaggi provoca un innalzamento delle temperature elevato, che potrebbe superare i limiti ottimali (un aumento di temperatura contenuto entro certi limiti infatti produrrebbe anche un miglioramento delle prestazioni degli ingranaggi, dovuto al fatto che un olio meno viscoso genera un film più sottile e maggiormente scorrevole tra le parti mobili, riducendo le perdite per attrito) e di conseguenza un abbassamento della viscosità eccessivo, che non garantisce più una lubrificazione adeguata. Non essendo al momento previsto un sistema di monitoraggio della Temperatura all'interno degli ingranaggi, è difficile stabilire però in che misura questo aspetto influisce sulle perdite, congiuntamente a quelle riportate al punto a seguire, di cui invece è possibile effettuare una analisi più dettagliata.
   - Perdita delle prestazioni del motore elettrico in prova all'aumentare della Temperatura: per spiegare il fenomeno fisico legato a questo aspetto, è sufficiente considerare un circuito monofase equivalente del motore elettrico, riportato in Figura 4.15.

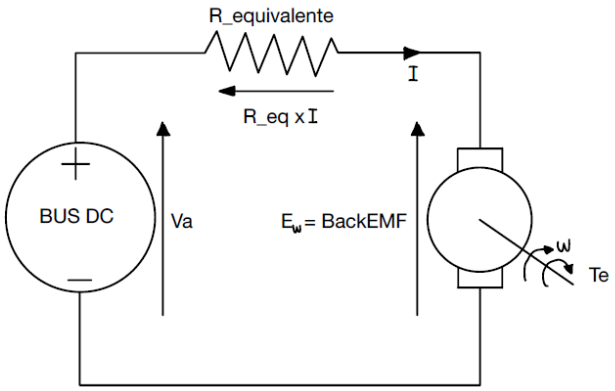


Figura 4.15: ***Circuito monofase equivalente di un generico motore elettrico***

Tenendo conto del fatto che le prove vengono eseguite a velocità angolare costante, allora sulla singola fase la tensione Back-EMF risulterà costante, essendo quest'ultima linearmente dipendente dalla velocità stessa.

Anche la tensione di alimentazione proveniente dal BUS-DC risulta essere circa costante, in quanto legata al pacco batterie.

Avendo indicato con $R_{equivalente}$ la Resistenza Elettrica complessiva di cavi di alimentazione e quelle statoriche e rotoriche, quest'ultima risulta essere fortemente sensibile a variazioni di Temperatura, ed in particolare i materiali conduttori sono caratterizzati da un abbassamento di conduttività all'aumentare della Temperatura: la resistenza elettrica, quindi, aumenterà con l'aumentare della Temperatura.

Dato quindi il sistema di equazioni che governa il sistema e dati gli andamenti di Temperatura registrati e riportati in Figura 4.9 e 4.12:

$$\begin{cases} Va - Ew - Req * I = 0 \\ Va = costante \\ Ew = BEMF = costante \\ Req = Req(T) \\ Te = Kt * I \end{cases} \tag{4.1}$$

L'aumento di Resistenza equivalente all'aumentare della Temperatura, per rispettare l'equazione alla maglia del circuito, comporta quindi una diminuzione di Corrente Elettrica circolante, che si traduce in una minor Coppia erogata (Te) dal motore.

Questo spiega inoltre come ci sia una perfetta corrispondenza tra gli andamenti riportati in Figura 4.9,4.10 e 4.11 e tra gli andamenti riportati in Figura 4.12,4.13 e 4.14: per tutto l'arco di tempo durante il quale la temperatura del motore continua a crescere la coppia erogata tende invece a decrescere. Quando invece l'andamento di temperatura raggiunge un andamento asintotico anche la coppia erogata assume lo stesso comportamento.

Capitolo 5 : Protocollo di comunicazione CAN-Bus adottato sul banco prova

Introduzione al protocollo CAN-Bus

Il CAN-Bus, abbreviazione di Controller Area Network Bus, è un protocollo di comunicazione seriale utilizzato per il trasferimento affidabile e rapido di dati tra centraline di controllo ECU (Electronic Control Unit) di dispositivi all'interno di un sistema distribuito e chiuso. E' stato sviluppato originariamente dalla società tedesca Bosch negli anni '80 per funzionare in ambienti fortemente disturbati dalla presenza di onde elettromagnetiche ed è tutt'oggi ampiamente utilizzato nell'industria automobilistica, nell'automazione industriale ed in molte altre applicazioni di tipo embedded per via della sua elevata affidabilità e per la sua robustezza, caratteristiche che lo rendono quindi particolarmente adatto per applicazioni critiche come quelle sopracitate, nelle quali è fondamentale il collegamento in qualunque circostanza tra sensori, attuatori e l'unità di controllo dell'intero sistema.

Il sistema è pensato su due cavi, identificati con il nome CAN_H (high) e CAN_L (low), intrecciati tra loro ed in controfasce, vale a dire che trasportano la stessa informazione, uguale ma opposta: mentre nel cavo CAN_H vi è l'informazione binaria 0 sul cavo CAN_L vi è l'informazione binaria 1. Tipicamente entrambe i segnali hanno ampiezza 1V e sono centrati su di una tensione pari a 2,5V (ovvero lo stato HIGH del canale CAN_L e lo stato LOW del canale CAN_H si trovano a 2,5V). La rete si basa quindi sulla differenza tra il segnale CAN_H ed il segnale CAN_L, con il risultato che può essere solamente 0V o 2V. Questo si traduce nel vantaggio di avere un sistema immune ai disturbi di natura elettromagnetica: infatti essendo i due cavi intrecciati tra loro, l'eventuale presenza di un disturbo si ripercuote in ugual modo su entrambi i canali e di conseguenza la differenza tra i due segnali porta sempre ad avere 0V o 2V.

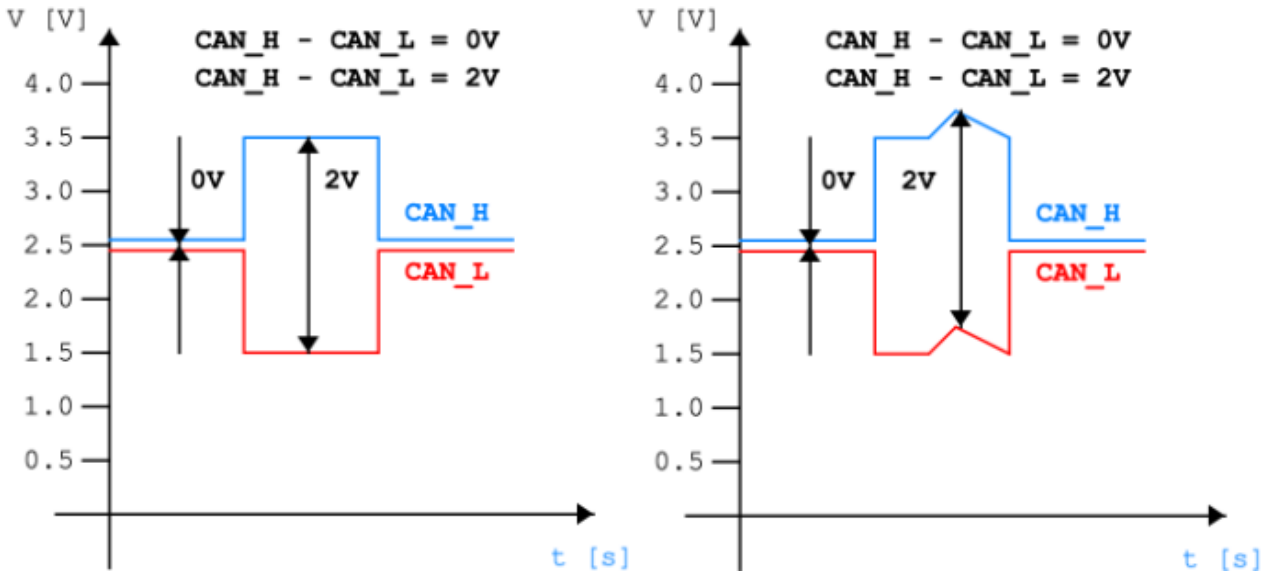


Figura 5.1: ***Segnali CAN in normali condizioni di funzionamento (a sinistra) e in condizioni di disturbo (a destra)***

Le principali caratteristiche di questo protocollo di comunicazione sono quindi:

- Utilizzo di una topologia di rete ad anello o bus lineare. I vari dispositivi si collegano quindi ad un'unica linea di trasmissione connessa a tutti i nodi della rete stessa.

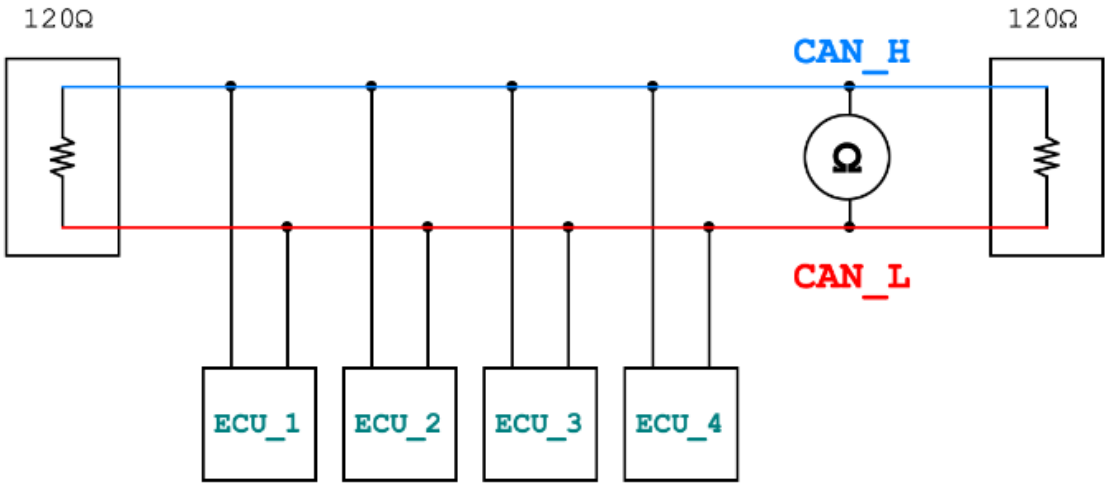


Figura 5.2: ***Topologia di una generica rete CAN-BUS***

Come si può vedere dalla Figura 5.2, il sistema adotta a monte e a valle del circuito due impedenze costituite da due resistenze a 120 Ω: tutte le unità vengono connesse in parallelo e si può verificare il corretto funzionamento del circuito inserendo un tester ohm-metrico tra i canali CAN_H e CAN_L che rileverà una resistenza equivalente di 60 Ω in caso di circuito sano oppure una resistenza equivalente di 120 Ω in caso una delle due resistenze non chiuda il circuito (circuito guasto).

- Velocità di comunicazione: il protocollo supporta differente velocità, che sono comunemente a 125 kbps, 250 kbps, 500 kbps ed a 1 Mbps e che dipendono dalle esigenze specifiche dell'applicazione a cui il protocollo è destinato e dalla lunghezza dei cavi. Le periferiche più critiche vengono collegate sul bus più veloce.

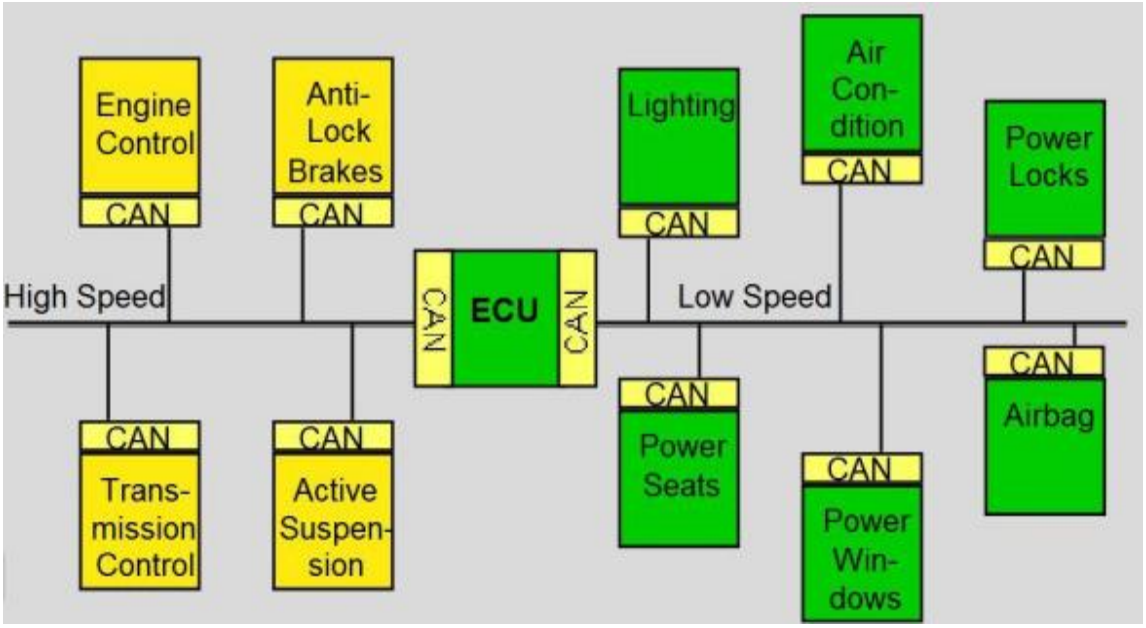


Figura 5.3: ***Esempio di rete CAN-Bus a diverse velocità di comunicazione***

- I dati sono inviati mediante messaggi denominati “frame”, che sono principalmente di due tipi, vale a dire il frame di dati (Data Frame, che contiene i dati effettivi) ed il frame remoto (Remote Frame, utilizzato per richiedere dati dagli altri nodi della rete).
  Dal punto di vista costruttivo, il frame che compone un messaggio completo è strutturato come segue:
  1. Start of Frame: bit responsabile di dichiarazione di inizio comunicazione. Se assume valore binario 0 (bit dominante) una comunicazione sta per iniziare.
  2. Arbitration Field: insieme di 12 bit (11 per la sezione Identifier ed 1 per l’RTR) che definiscono la priorità del messaggio. Poiché il bit 0 è quello dominante, più basso sarà il valore binario e maggiora sarà la priorità di quel messaggio.
  3. Control Field: composto da 6 bit che descrivono il controllo del pacchetto dati e permette di decodificare se il messaggio è un frame standard oppure esteso. Viene qui definita anche la dimensione del Data Field.
  4. Data Field: può estendersi sino ad un massimo di 8 byte (ovvero 64 bit) e contiene l’informazione che il messaggio vuole trasmettere.
  5. CRC Field: composto da 16 bit che interessano il nodo ricevente, il quale può dedurre se il messaggio inviato è corretto e se sono presenti errori nella trasmissione.
  6. ACK Field: composto da 2 bit ed indica se il messaggio è stato ricevuto correttamente oppure no.
  7. End of Frame: composto da 7 bit recessivi (bit 1) consecutivi.

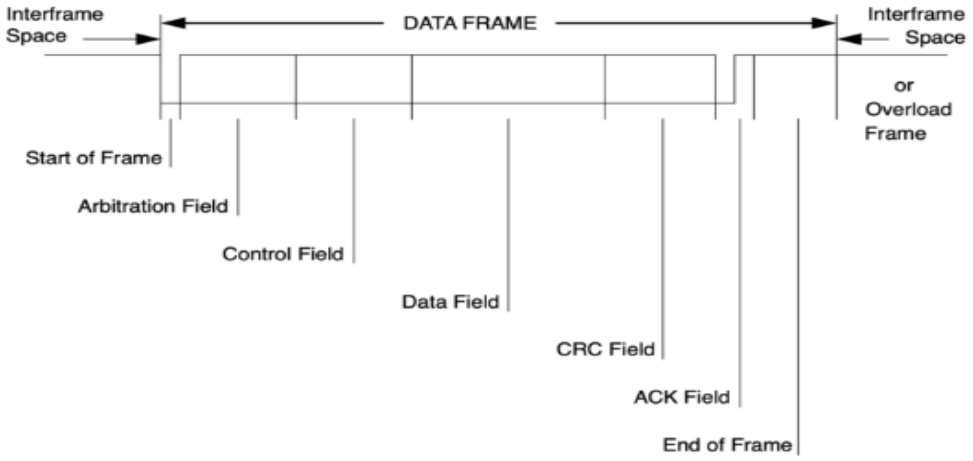


Figura 5.4: ***Costruzione di un Frame che trasporta un messaggio CAN-BUS***

- Utilizzo di un metodo di accesso al canale noto come “arbitraggio bit-wise”: i nodi che vogliono trasmettere dati sulla rete ascoltano la linea per assicurarsi che non vi siano altre trasmissioni in corso e quindi inviano i loro dati sulla base della loro priorità. In caso di collisioni, il nodo con priorità più elevata avrà successo nell’invio dei dati.

- Rete progettata per rilevare e correggere errori di trasmissione, che utilizza un meccanismo di checksum per garantire l’integrità dei dati trasmessi.

- Supporto della topologia multimaster, vale a dire che più nodi possono trasmettere dati sulla rete senza la necessità che vi sia un nodo master centrale.

- Protocollo di tipo half-duplex, ovvero i nodi possono sia trasmettere sia ricevere dati, ma non entrambi contemporaneamente sulla stessa linea.

Cablaggio CAN-Bus realizzato sul banco prova

La comunicazione tra i sottosistemi del banco prova realizzato avviene secondo il protocollo CAN-Bus appena descritto, in modo tale da simulare e testare anche quella che sarà la reale trasmissione dati a bordo del veicolo finito.

In particolare, durante il funzionamento del banco, dal software Simulink su pc vengono inviati due messaggi, uno destinato al motore ed uno destinato al freno. Questi due messaggi sono però caratterizzati da stesso identificatore, pertanto se viaggiassero sulla stessa linea CAN risulterebbero in conflitto. La centralina di controllo a bordo del banco, essendo un nodo prioritario del sistema, andrà quindi a intercettare questi due messaggi, smistandoli ed introducendo un piccolo ritardo sulla comunicazione, secondo quanto segue:

- Sulla Linea CAN 1 continuerà a viaggiare il messaggio destinato al Motore in prova;
- Sulla Linea CAN 2 verrà invece indirizzato il messaggio destinato al Freno.

Sulle due linee sono inoltre collegati anche il torsiometro, che indirizza i dati relativi a coppia e velocità angolare lette su un messaggio che viaggia sulla linea CAN 1 e che vengono letti tramite software sul pc di comando, ed il display del banco collegato alla linea CAN 2 sul quale leggere alcuni dati come lo stato di carica del pacco batteria.

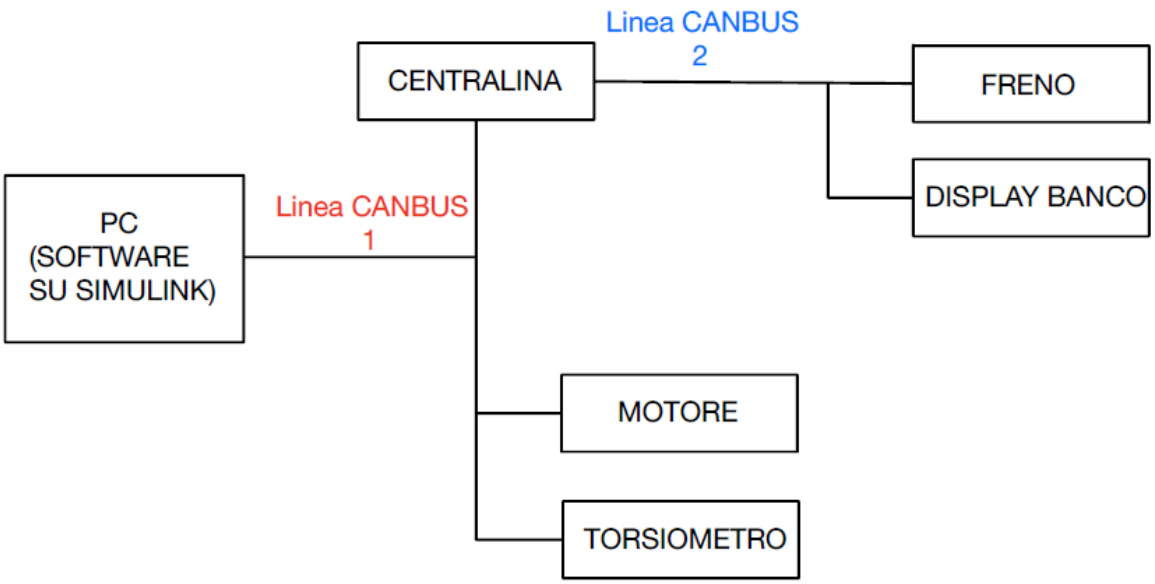


Figura 5.5: ***Schema semplificato delle linee CAN realizzate sul banco prova***

Sulle due linee CAN sono inoltre stati predisposti degli ulteriori nodi (prevedendo dunque già la possibilità di ampliare i dati acquisibili) che risultano al momento non utilizzati e che quindi non sono stati riportati nello schema semplificato proposto in Figura 5.5.

L’interfacciamento tra l’Hardware a banco e i vari software su PC avviene attraverso dei dispositivi di conversione CAN-USB della Peak System, come quelli riportati in Figura 5.6:

Figura 5.6: ***Dispositivi PCAN-USB della Peak System***

Capitolo 6 : Sviluppo di una GUI per invio e ricezione di segnali tramite il CAN-Bus del banco prova

Necessità della creazione di una Graphic User Interface (GUI) per il controllo e le acquisizioni a banco prova

Nella configurazione attuale del banco prova, per eseguirne il controllo è necessario avere piena padronanza del software che comunica con la centralina di bordo. Inoltre, anche l'acquisizione dei dati durante le prove, come quelle effettuate e riportate nel Capitolo 4, avviene in modo semi-manuale in quanto i parametri che si vogliono estrarre devono essere registrati manualmente dopo averli letti dai software.

Per questi motivi, si è allora pensato di creare una Interfaccia Grafica (GUI), che mascheri all'utente il lavoro svolto dal software di controllo: questa GUI, infatti, dovrà presentare una serie pulsanti/cursori che consentono all'operatore di decidere esclusivamente in che modo controllare il motore in prova ed il freno del banco (per entrambi è consentito infatti sia il controllo in velocità sia il controllo in coppia). Questi Input Set verranno quindi trasmessi al software di comando (che a questo punto lavorerà in modo del tutto trasparente rispetto all'utente che esegue la prova), che a sua volta sarà in comunicazione con la HCU montata sul banco prova. Al tempo stesso sulla schermata dell'interfaccia dovranno essere graficati gli andamenti delle principali grandezze di interesse ottenute come feedback dai sottosistemi montati a banco, nonché la possibilità di salvarli per poter eseguire gli opportuni post-processing.

Schematizzando quindi il flusso delle informazioni, la situazione da ricreare è quella riportata in Figura 6.1:

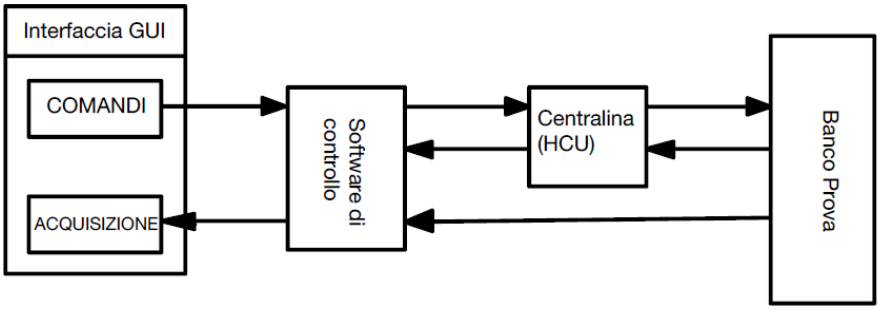


Figura 6.1: ***Schema del flusso dei dati da realizzare***

Per realizzare quanto indicato è possibile utilizzare il protocollo di comunicazione stesso con cui è stato cablato il banco prova, ovvero il CAN-Bus descritto nel Capitolo 5. Il compito dell'Interfaccia Grafica sarà dunque quello di:

1- Ricevere in ingresso i segnali da passare al software di controllo, il quale tramite CAN-Bus comunicherà con la centralina del banco;
2- Intercettare, dalle linee CAN-Bus fisiche realizzate sul banco, i messaggi contenenti i feedback dei vari sottosistemi per poterli graficare e salvare.

Utilizzo di Matlab App Designer e del Vehicle Network Toolbox per la creazione della GUI

Per poter realizzare questa Interfaccia ci si è avvalsi dell'uso congiunto del Software Matlab App Designer fornito da Mathworks, specificatamente sviluppato per facilitare la creazione di schermate di controllo e visualizzazione dati, all'interno del quale sono stati implementati dei comandi specifici del Vehicle Network Toolbox (VNT) , altro prodotto Mathworks, sviluppato per facilitare la comunicazione tra sistemi secondo gli standard del mondo automotive.

Il tool VNT contiene al suo interno tutti i principali protocolli di comunicazione del settore in questione: CAN, CAN FD, XCP e J1939, prevedendo inoltre la possibilità di eseguire la comunicazione sia in ambiente Matlab, tramite dei comandi con una sintassi specifica, sia in ambiente Simulink attraverso degli appositi blocchi che replicano quanto può essere fatto tramite linee di codice.

Per quanto riguarda la comunicazione tramite CAN, che è quella di interesse, il tool mette inoltre a disposizione due altri strumenti fondamentali per effettuare simulazioni delle comunicazioni in ambiente virtuale e per visualizzare il corretto trasferimento dei dati: sono infatti previsti due canali di comunicazione virtuali ed uno strumento denominato CanExplorer.

Per visualizzare i canali CAN disponibili è sufficiente implementare, tramite la linea di comando di Matlab, la funzione *canChannelList*:

```
info = canChannelList

info =

  14×6 table

    Vendor           Device           Channel      DeviceModel       ProtocolMode   SerialNumber
    _____________    _____________    _______    _________________    ____________    ____________

    "MathWorks"      "Virtual 1"          1      "Virtual"            "CAN, CAN FD"     "0"
    "MathWorks"      "Virtual 1"          2      "Virtual"            "CAN, CAN FD"     "0"
    "Vector"         "VN1610 1"           1      "VN1610"             "CAN, CAN FD"     "18959"
    "Vector"         "VN1610 1"           2      "VN1610"             "CAN, CAN FD"     "18959"
    "Vector"         "Virtual 1"          1      "Virtual"            "CAN, CAN FD"     "0"
    "Vector"         "Virtual 1"          2      "Virtual"            "CAN, CAN FD"     "0"
    "PEAK-System"    "PCAN-USB Pro"       1      "PCAN-USB Pro"       "CAN, CAN FD"     "0"
    "PEAK-System"    "PCAN-USB Pro"       2      "PCAN-USB Pro"       "CAN, CAN FD"     "0"
    "Kvaser"         "USBcan Professional 1"  1  "USBcan Professional"  "CAN"           "10680"
    "Kvaser"         "USBcan Professional 1"  1  "USBcan Professional"  "CAN"           "10680"
    "Kvaser"         "Virtual 1"          1      "Virtual"            "CAN, CAN FD"     "0"
    "Kvaser"         "Virtual 1"          2      "Virtual"            "CAN, CAN FD"     "0"
    "NI"             "9862 CAN/HS (CAN1)"  1     "9862"               "CAN, CAN FD"     "17F5094"
    "NI"             "9862 CAN/HS (CAN2)"  1     "9862"               "CAN, CAN FD"     "17F50B2"
```

Figura 6.2: ***Canali CAN utilizzabili con il VNT***

Dalla Figura 6.2, si può vedere la possibilità di utilizzare sia canali CAN virtuali sia canali di dispositivi hardware connessi al PC, come il PCAN-USB riportato in Figura 5.6, anche utilizzato sul banco fisico.

In entrambi i casi, comunque, il VNT lavorerà seguendo lo schema riportato in Figura 6.3: uno dei nodi sulla linea CAN sarà un CAN device collegato al VNT.

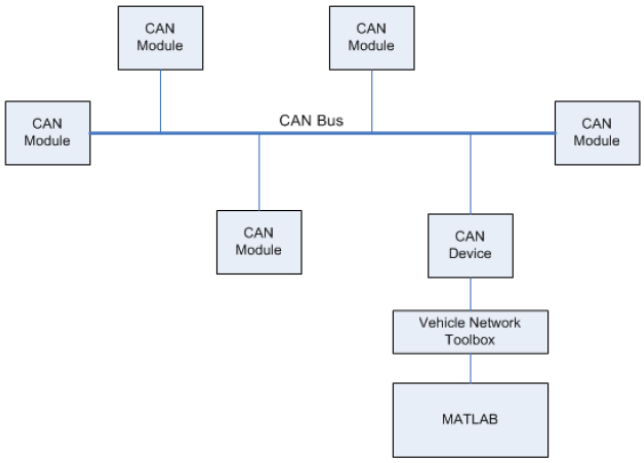


Figura 6.3: ***Schema di inserimento del VNT all'interno di una generica linea CAN-Bus***

Tramite Matlab, quindi, è possibile configurare un canale sul CAN device (in questo caso i dispositivi PCAN-USB) che permette di:

- Trasmettere messaggi alla linea CAN;
- Ricevere messaggi dalla linea CAN;
- Caricare e registrare messaggi e analizzarli in Matlab;
- Eseguire un replay della sequenza di messaggi registrati in Matlab;
- Costruire modelli Simulink da connettere alla linea CAN (come avviene per il software di controllo della centralina) e simulare/realizzare il traffico di messaggi;
- Monitorare il traffico dei messaggi sulla linea mediante CanExplorer.

I comandi specifici utilizzati su linea di codice per questa attività verranno poi illustrati e spiegati nel seguito della trattazione. L'utilizzo degli stessi comandi è replicabile in ambiente Simulink mediante i blocchi equivalenti presenti nella relativa libreria:

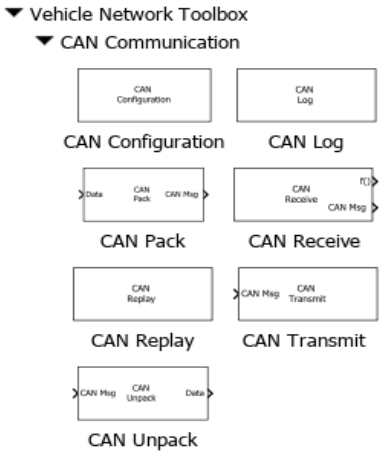


Figura 6.4: ***Blocchi VNT per la comunicazione CAN in ambiente Simulink***

Per quanto riguarda l'ultimo punto, lo strumento CanExplorer consente di visualizzare tutti i messaggi transitanti sullo specifico dispositivo hardware o virtuale connesso al VNT:

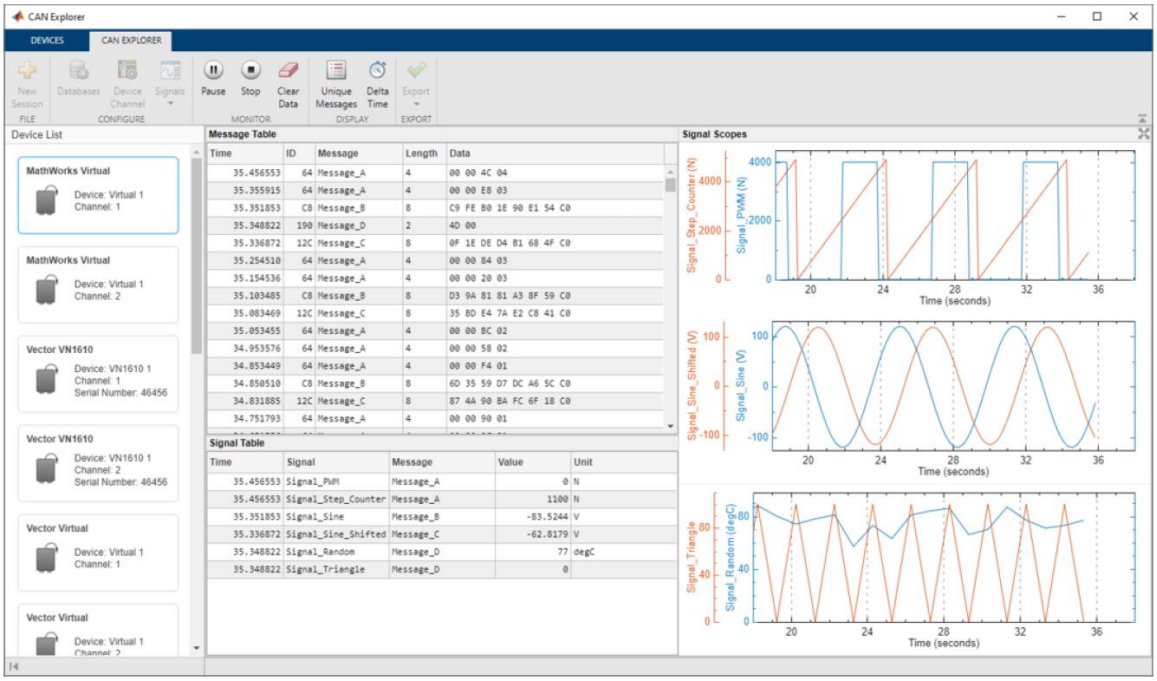


Figura 6.5: ***Monitoraggio della comunicazione CAN tramite CanExplorer***

Per quanto riguarda invece il Software Matlab App Designer, alla prima apertura quest'ultimo si presenta come riportato in Figura 6.6:

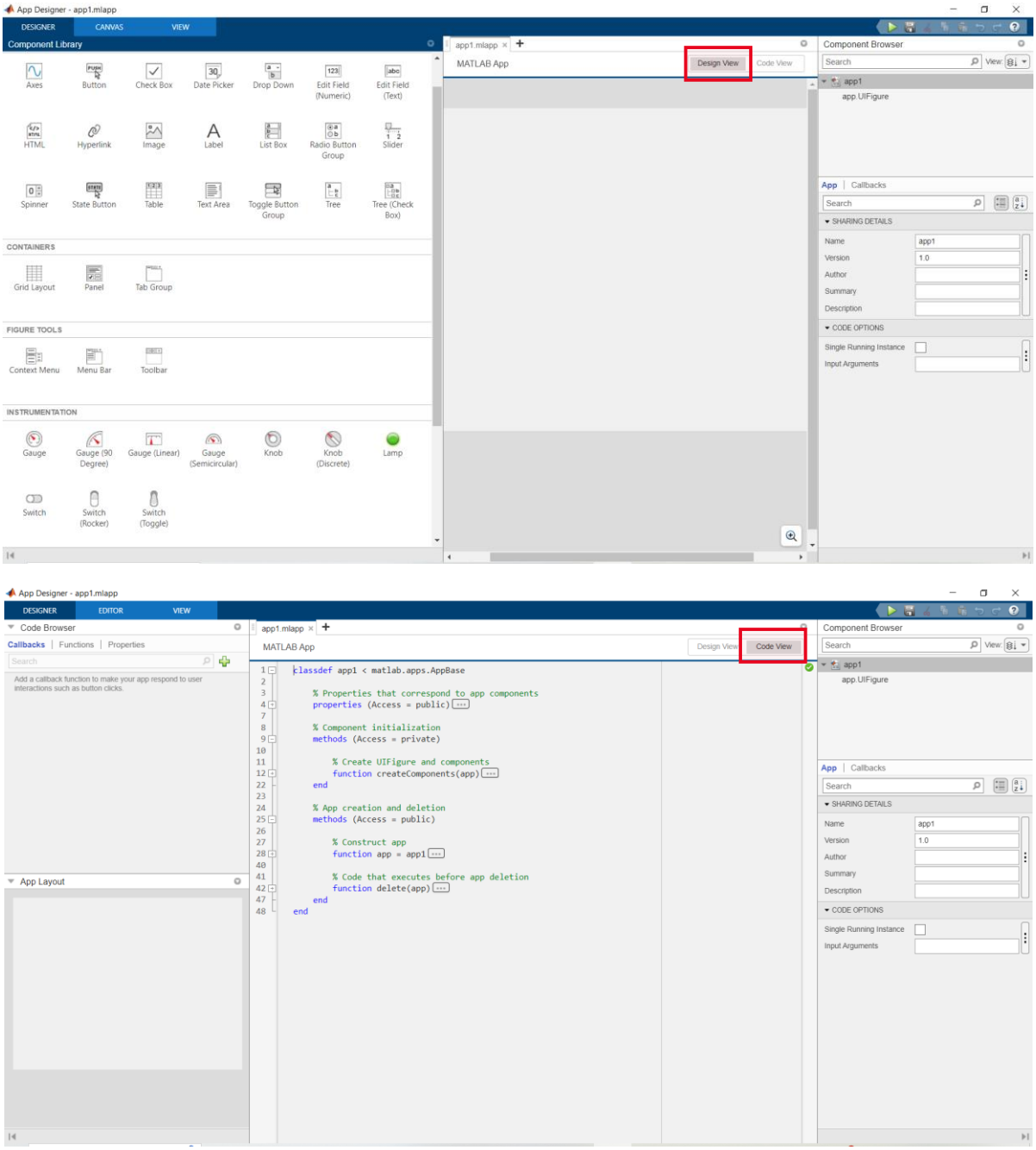


Figura 6.6: ***Schermate principali del software Matlab App Designer***

Come si può notare, attraverso la "Design View" il software consente di creare agevolmente la parte grafica della GUI utilizzando le componenti di librarie già predisposte. Questi andranno poi richiamati attraverso delle "Callback" nella "Code View", in cui si va a programmare la parte software della GUI stessa. I vantaggi fondamenti di questo strumento, per questa applicazione, oltre alla intuibilità di utilizzo, sono principalmente:

1. La possibilità di utilizzare all'interno le funzioni del tool VNT per la comunicazione CAN;
2. La possibilità, quando la GUI è stata ultimata e testata, di crearne un file eseguibile .exe, che diventa dunque indipendente dalla presenza o meno, sul PC su cui viene utilizzata, di Matlab.

Caratteristiche e funzionalità principali della GUI sviluppata

Si riporta quindi la schermata principale della GUI sviluppata, indicandone le parti principali che verranno successivamente descritte con maggior dettaglio, evidenziando le parti di codice che vengono eseguite da ciascuna di esse, secondo l'ordine numerico riportato:

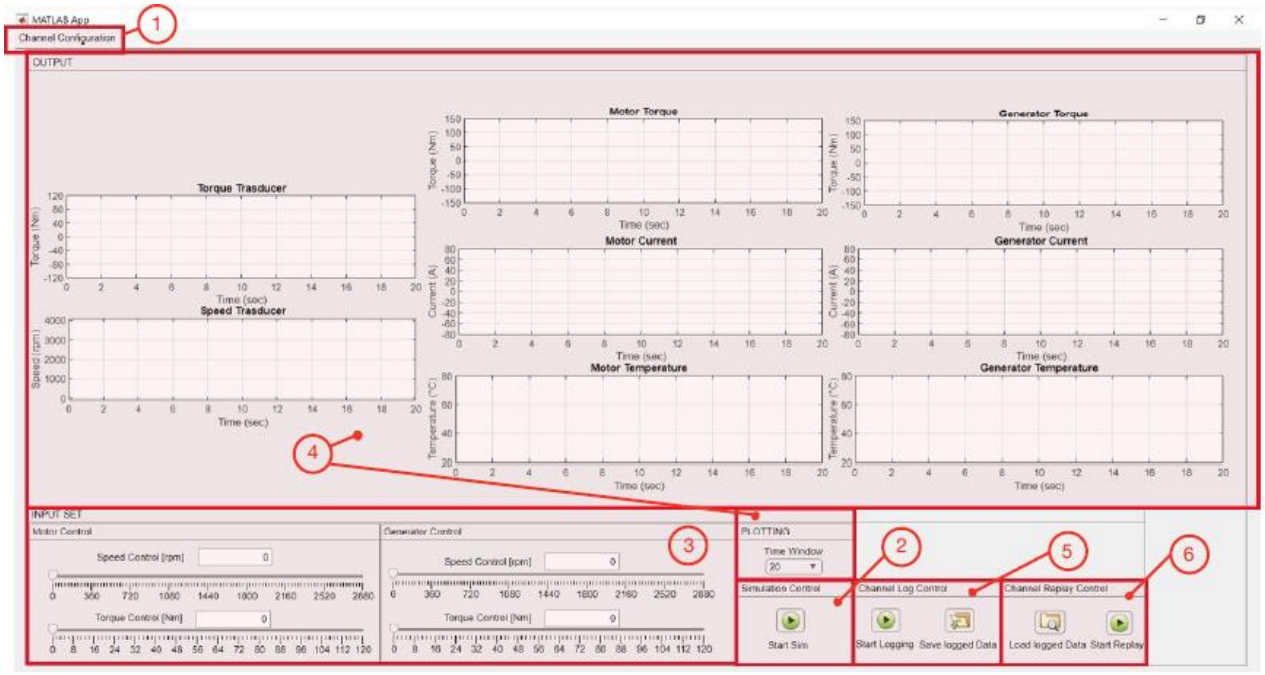


Figura 6.7: ***Schermata principale della GUI realizzata***

1) **Channel Configuration**: cliccando su questa voce, si apre un menù, come riportato in Figura 6.8.

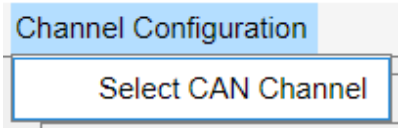


Figura 6.8: ***Menù per la selezione del canale CAN a cui collegarsi***

Cliccando ulteriormente sulla voce "Select CAN Channel" nel menù apertosi, si avrà la possibilità di procedere con la scelta di un canale CAN, come illustrato in Figura 6.9:

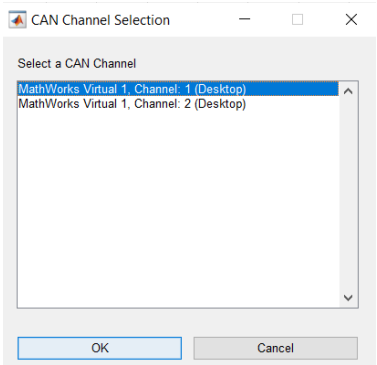


Figura 6.9: ***Selezione del canale CAN a cui collegarsi tra quelli disponibili***

Come si può vedere dalla Figura 6.9, non essendoci in questo specifico istante connessi dispositivi hardware come il PCAN-USB, risulteranno in elenco solamente i canali virtuali messi a disposizione dal tool VNT per effettuare simulazioni di comunicazione CAN. Nel momento in cui vengono collegati al PC anche dei dispositivi CAN fisici, una volta installati gli specifici driver, il codice sottostante al menù di selezione rileverà anche questi ultimi, consentendone la scelta.

Le funzioni specifiche del codice che consentono di realizzare quanto riportato sono:

```
function canChannelConstructor = formatCANChannelConstructor(app, index)

function canChannelDeviceStr = formatCANChannelDeviceStr(app, index)

function updateModelWithSelectedCANChannel(app, index)

function displayText = formatCANChannelEntryForDisplay(app)
```

Inoltre, qualora l'utente, durante un test virtuale, dimenticasse di selezionare uno dei due specifici canali virtuali, il codice provvede alla selezione di default del "MathWorks Virtual 1, Channel 1" attraverso la `function initializeCANChannelSelections(app).`

2) **Simulation Control**: controlla lo Start e lo Stop della simulazione qualora quest'ultima fosse completamente virtuale. In quest'ultimo caso, per iniziare a eseguire una simulazione, è necessario definirne un tempo di inizio (cliccando su Start Sim) ed un tempo di fine simulazione (cliccando sul pulsante Stop Sim, che si andrà a sostituire allo start quando quest'ultimo è stato eseguito). Nel caso in cui nella Channel Configuration precedentemente descritta si scelga di utilizzare la GUI su un canale CAN fisico, questa intera sezione perde di significato in quanto l'applicazione inizierà a lavorare automaticamente allineandosi in Real Time al tempo fisico. Gli aspetti sopra descritti vengono gestiti nel codice attraverso tre funzioni specifiche:

```
function setModelStartStopButtonEnableStatus(app)

    function startSimApplication(app, index)

    function stopSimApplication(app, index)
```

In particolare, la prima funzione effettua un check sul tipo di canale CAN a cui si è collegati e qualora quest'ultimo fosse un canale Hardware va a disabilitare il pulsante di Start e Stop Simulation, allineandosi automaticamente al tempo reale di prova. Di conseguenza anche le due function successive, collegate in Callback ai pulsanti sopracitati, potranno entrare in funzione solamente se la simulazione è di tipo virtuale. In quest'ultimo caso quindi il compito dei pulsanti di Start e Stop Sim è anche di avviare e di interrompere la comunicazione sul canale virtuale precedentemente selezionato, definendo anche la velocità di trasmissione dei messaggi della linea CAN, attraverso le seguenti funzioni:

```
function setupCANChannel(app)

function startCANChannel(app)

 function stopCANChannel(app)
```

3) **Input Set**: attraverso questa sezione si forniscono i comandi sia al motore che al freno del banco, prevedendo per entrambi sia il controllo in velocità sia il controllo in coppia. Come si spiegherà anche nel capitolo successivo, questa sezione della GUI è quella che subirà le maggiori modifiche passando dal test SiL, effettuato per questa attività di tesi, rispetto alla successiva applicazione dell'interfaccia al banco prova fisico: infatti ai fini del test SiL, in cui non viene simulata la presenza della centralina di bordo, i comandi dati attraverso i cursori della GUI entreranno direttamente come dati trasferiti tramite CAN-Bus nei blocchi di simulazione del motore e del banco. Nella realtà, invece, e come riportato anche in Figura 6.1, gli input qui forniti dovranno essere prima elaborati dal software che controlla la centralina del banco, e sarà poi questa che genererà i segnali CAN poi smistati ai sottosistemi. Sostanzialmente quindi, quella che è la catena di trasferimento effettiva delle informazioni nella realtà, nel modello di simulazione viene condensata nei soli cursori che intervengono direttamente nei controlli motore e freno. Pertanto, nelle simulazioni qui svolte i valori numerici associati al movimento dei cursori sono direttamente impacchettati all'interno di messaggi CAN utilizzando le funzionalità del VNT attraverso le funzioni richiamate in Callback dal movimento dei cursori stessi:

```
function updateEditFieldAndSliderValues_MotorSpeedControl(app)

function updateSliderAndEditFieldValues_MotorSpeedControl(app)

function updateEditFieldAndSliderValues_MotorTorqueControl(app)

function updateSliderAndEditFieldValues_MotorTorqueControl(app)

function updateEditFieldAndSliderValues_GeneratorSpeedControl(app)

function updateSliderAndEditFieldValues_GeneratorSpeedControl(app)

function updateEditFieldAndSliderValues_GeneratorTorqueControl(app)

function updateSliderAndEditFieldValues_GeneratorTorqueControl(app)
```

All'interno di tutte le funzioni viene utilizzato il comando "pack" del tool VNT, il quale presenta la seguente sintassi:

```
pack Load a raw signal value into message data.

  pack(MESSAGE, VALUE, STARTBIT, SIGNALSIZE, BYTEORDER) takes
  a set of input parameters to load raw signal VALUE(s) into MESSAGE(s).

  Inputs:
      MESSAGE - The message in which to pack the signal.
      VALUE - The raw signal value to pack.
      STARTBIT - The signal's starting bit location in the message data
          with a value between 0 and 63. The STARTBIT represents the
          least significant bit position in the CAN data of the signal.
      SIGNALSIZE - The signal's length in bits with a value between 1 and 64.
      BYTEORDER - The signal's endian format as 'LittleEndian' or
          'BigEndian'. This is often referenced as Intel for
          'LittleEndian' and Motorola for 'BigEndian'.

  Note that MESSAGE can be an array of CAN messages. When given as an
  array, the raw signal value is packed into each entry in the array
  individually. These signal values are then returned as an array of
  equal size to MESSAGE. The VALUE array is ordered the same as in the
  original MESSAGE array. Each MESSAGE in the array must have the
  same definition.
```

Figura 6.10: ***Sintassi del comando pack del VNT***

Il riempimento dei messaggi CAN segue la logica:

$$VALUE = RAW * Scaling + Offset \tag{6.1}$$

Dove:

- *VALUE* : è il valore fisico associato al dato effettivamente trasportato sul CAN;
- *RAW* : è il valore numerico effettivamente associato ai bit trasportati come dato sul CAN;
- *Scaling ed Offset* : sono dei fattori di conversione utilizzati per riuscire a contenere all'interno di un certo numero di bit un valore fisico che altrimenti non potrebbe esservi trasportato (si pensi ad esempio a valori numerici molto elevati, che andrebbero ad occupare un grande numero di bit se non scalati, oppure a valori numerici decimali, che per essere trasportati necessitano di essere scalati).

Una volta immagazzinati i dati da trasportare sul CAN-Bus all'interno degli specifici messaggi, è necessario predisporre la loro effettiva trasmissione, ed in particolare per questa tipologia di applicazione è necessario effettuare una trasmissione di tipo periodica, eseguita attraverso la funzione:

```
function setupCANTransmitMessages(app)
```

All'interno di quest'ultima viene utilizzato il comando "transmitPeriodic" del tool VNT, con la seguente sintassi:

```
transmitPeriodic Manage a message for periodic transmit.

  transmitPeriodic(CHANNEL, MESSAGE, 'On', PERIOD) enables the
  MESSAGE for periodic transmit at the provided PERIOD on the
  CHANNEL. The MESSAGE is sent periodically when the CHANNEL
  is online. Note that if a PERIOD is not provided, a default
  value of 0.500 seconds is used.
```

Figura 6.11: ***Sintassi del comando transmitPeriodic del VNT***

Analogamente, quando si terminerà la simulazione, premendo il pulsante Stop Sim la trasmissione dei messaggi CAN sul canale viene interrotta (ponendo la transmitPeriodic su 'Off' e poi chiudendo definitivamente il canale di comunicazione CAN), così come viene interrotta la ricezione dei messaggi dal canale, che verrà spiegata a seguire.

4) **Output e Plotting**: i vari dati di output di prova (reale o simulata) devono essere ricevuti ed estratti dai messaggi CAN corrispondenti che li trasportano, in modo da poter essere visualizzati in tempo reale ed essere eventualmente salvati. Per poter realizzare quanto descritto è stato necessario effettuare diversi passaggi:
   - Creare un timer per temporizzare la ricezione di messaggi CAN, così definito:

     ```
     app.receiveCANmsgsTimer = timer('Period', 0.1,'ExecutionMode',
     'fixedSpacing','TimerFcn', @(~,~)receiveCANmsgsTimerCallback(app));
     ```

     Il comando definisce ogni quanto tempo, in questo caso 0.1 secondi, richiamare la funzione riportata all'interno del comando, spiegata nel punto successivo.

- Esecuzione periodica della `function receiveCANmsgsTimerCallback(app)`, all'interno della quale viene utilizzato il comando "receive" del tool VNT.

```
receive Receive messages from a CAN bus.

  MSG = receive(CH, MESSAGESREQUESTED) returns received
  message(s) MSG equal to or less than MESSAGESREQUESTED in
  count from the channel CH.

  MESSAGESREQUESTED specifies the maximum count of messages to receive. It
  must be a nonzero, positive value or Inf to indicate return all
  available messages. If less messages are available to be received than
  MESSAGESREQUESTED specifies, the amount currently available will be
  returned. If no messages are available to receive, an empty array is
  returned.
```

Figura 6.12: ***Sintassi del comando receive del VNT***

Dato che il comando "receive" non è in grado di distinguere tra messaggi CAN che vengono forniti come input piuttosto che gli output di simulazione, allora all'interno della stessa funzione è richiamata un'ulteriore funzione per estrarre i soli messaggi CAN di output dalla simulazione ed infine una funzione per eseguire il plot di tali dati sulla schermata della GUI.

- Esecuzione periodica della `function newOutputData=getOutputCANmessage(app, msg)` per effettuare l'estrazione dei soli messaggi CAN di output attraverso il comando "extractAll" del tool VNT:

```
extractAll Returns all occurrences of the specified CAN message(s).

  [EXTRACTED, REMAINDER] = extractAll(MESSAGE, NAME) parses the given
  array MESSAGE and returns all messages found with the matched NAME(s).

  [EXTRACTED, REMAINDER] = extractAll(MESSAGE, ID, EXTENDED) parses the
  given array MESSAGE and returns all messages found with the matched
  ID/EXTENDED value(s).
```

Figura 6.13: ***Sintassi del comando extractAll del VNT***

In questo caso specifico, l'estrazione è stata effettuata sfruttando l'ID assegnato ai vari messaggi.

Una volta estratti tutti i messaggi di output provenienti dai diversi sottosistemi del banco è possibile effettuare l'estrazione dei dati contenuti negli stessi utilizzando invece il comando "unpack" del tool VNT:

```
unpack Unload a raw signal value from message data.

  VALUE = unpack(MESSAGE, STARTBIT, SIGNALSIZE, BYTEORDER, DATATYPE)
  takes a set of input parameters to unload raw signal VALUE(s) from the
  MESSAGE(s) and returns them as output.

  Inputs:
      MESSAGE - The CAN message(s) from which to unpack the signal(s).
      STARTBIT - The signal's starting bit location in the message data
          with a value between 0 and 63. The STARTBIT represents the
          least significant bit position in the CAN data of the signal.
      SIGNALSIZE - The signal's length in bits with a value between 1 and 64.
      BYTEORDER - The signal's endian format as 'LittleEndian' or
          'BigEndian'. This is often referenced as Intel for
          'LittleEndian' and Motorola for 'BigEndian'.
      DATATYPE - The desired type of the returned VALUE as 'int8',
          'int16', 'int32', 'int64', 'uint8', 'uint16', 'uint32',
          'uint64', 'single', or 'double'.
```

Figura 6.14: ***Sintassi del comando unpack del VNT***

Il comando lavora analogamente al comando “pack”, seguendo anche in questo caso la regola definita nell’Equazione (6.1) per effettuare l’estrazione corretta del valore fisico associato al valore trasportato dai bit sul CAN-Bus.

Dopo aver effettuato l’estrazione prima dei messaggi e poi dei dati contenuti in questi messaggi, all’interno della funzione viene predisposta anche la creazione di matrici di tipo “timeseries”, necessarie per il plotting dei dati a schermo: per ognuna delle variabili che si è voluto riportare, viene dunque creata una matrice formata da una prima colonna in cui è riportato il tempo e una seconda colonna in cui invece è riportato il valore della variabile a quello specifico tempo. In queste matrici vi saranno tante righe quanti sono i valori ricevuti nell’istante che passa tra una ricezione e la successiva, tempo che dipende dall’impostazione data al timer precedentemente descritto.

A questo punto è possibile utilizzare i dati estratti per eseguire il plot delle grandezze a schermo: per questa fase iniziale, si è scelto di rappresentare i parametri caratteristici dei tre blocchi principali del banco prova:

- Per il trasduttore: andamento di velocità del sistema e coppia sull’albero.
- Per il motore: andamento di coppia, corrente e temperatura.
- Per il freno: andamento di coppia, corrente e temperatura.

Gli aspetti legati alla rappresentazione dei dati estratti sono tutti gestiti attraverso la `function updatePlots(app)`, al cui interno viene richiamata una ulteriore funzione per la corretta gestione dei grafici, quale la `function setAxesLimitsAndTitles(app, XLim0,timeWindow)`.

5) **Channel Log Control**: la seguente sezione permette di effettuare la registrazione sia dei comandi dati come input al sistema (importanti anche per la successiva ed ultima sezione “Channel Replay Control”) sia dei risultati di simulazione.

   Per avviare la registrazione dei dati è necessaria premere sul pulsante predisposto Start Logging, il quale richiama in callback la `function startCANLogging(app)`,al cui interno viene a sua volta richiamata la `function startCANLogChannel(app)`.

   La `function retrieveLoggedCANMessages(app)` a questo punto creerà dei buffer temporanei all’interno dei quali verranno immagazzinati tutti i messaggi CAN ricevuti sempre tramite l’utilizzo del comando “receive” del tool VNT, e successivamente separati utilizzando l’ID che li contraddistingue attraverso il comando “extractAll” del tool VNT, con le stesse modalità descritte in precedenza.

   Nel momento in cui l’utente preme sull’icona “save Logged Data”, viene richiamata in callback la `function savedLoggedCANDataToFile(app)`, che provvede a salvare tutti i dati registrati nei vari buffer in un file di estensione .mat .

Per effettuare il salvataggio viene utilizzato il comando "canMessageReplayBlockStruct" del tool VNT, che converte i dati CAN in strutture di vettori compatibili anche con il comando/blocco VNT Replay.

```
canMessageReplayBlockStruct Converts CAN messages for use as Replay block output.

  MSGSTRUCTOFARRAYS = canMessageReplayBlockStruct(MSGS) formats the input
  CAN messages for use with the CAN Replay block. The Replay block
  requires a specific format for CAN messages, which is a structure of
  arrays containing the ID, Extended, Data, and other message elements.
  MGS may be provided as a CAN message timetable or an array of CAN message
  objects.

  Use this function to create a variable for the CAN Replay block. After
  creating the structure, you must save this variable to a MAT file. The
  Replay block mask allows selection of this MAT file and the variable
  within it to replay CAN messages in a Simulink model.
```

Figura 6.15: ***Sintassi del comando canMessageReplayBlockStruct del VNT***

Effettuato il salvataggio, i vari buffer vengono liberati dai dati precedentemente immagazzinati, pronti per essere nuovamente riempiti con delle eventuali nuove registrazioni.

Per interrompere completamente la registrazione l'utente dovrà selezionare lo Stop Logging, il quale richiamerà in callback la `function stopCANLogging(app).`

6) **Channel Replay Control**: quando l'utente seleziona il pulsante "Load logged Data", viene richiamata in callback la `function loadLoggedCANDataFromFile(app)`, che effettua il caricamento dei dati CAN di Input dal file selezionato. Poiché il comando di Replay viene gestito da codice, è necessario riconvertire i dati CAN che erano stati immagazzinati come struttura di vettori in una timetable (mentre il blocco Simulink è in grado di gestire direttamente queste strutture di vettori), utilizzando il comando "canMessageTimetable".

```
canMessageTimetable Converts CAN messages into a timetable.

  MSGTIMETABLE = canMessageTimetable(MSG) takes the input MSG as either an
  existing CAN message timetable, an array of CAN message objects, or a CAN
  message structure from the CAN Log block and creates a CAN message
  timetable. The timetable contains the raw message information (ID, Extended,
  Data, etc.) from the messages. If CAN message objects are input which
  contain decoded information, that decoding is retained in the CAN
  message timetable.
```

Figura 6.16: ***Sintassi del comando canMessageTimetable del VNT***

Nel momento in cui l'utente clicca sullo Start Replay, viene richiamata la `function startCANReplay(app)`, che richiama la `function startPlaybackOfLoggedCANData(app)`, la quale blocca la trasmissione periodica di messaggi CAN dai cursori dell'Input Set, ed abilita il replay dei dati CAN di input precedentemente caricati in memoria da file.

Al contrario, quando l'utente seleziona lo Stop Replay, la `function stopCANReplay(app)`, attraverso la `function stopPlaybackOfLoggedCANData(app)`, blocca il replay dei messaggi CAN e riabilita la trasmissione periodica degli stessi tramite i cursori dell'Input Set.

Per non appesantire eccessivamente la trattazione, non è stata riportata la sintassi di tutte le function sviluppate ed utilizzate in questa parte del documento. Per comodità di realizzativa, l'intero codice sottostate la GUI può essere consultato alla fine del seguente documento, nel Capitolo "Appendice ed allegati", Sottocapitolo "Script della GUI sviluppata per test SiL".

Capitolo 7 : Test SiL della GUI su modello Matlab-Simulink del banco prova

Generalità sui test Software in the Loop (SiL) ed il caso particolare della GUI sviluppata

Il Software in the Loop testing è una tipologia di test ampiamente utilizzata per provare dei software all'interno di un ambiente di simulazione. Questa necessità nasce dal fatto che in particolari applicazioni il software è fondamentale per il funzionamento di sistemi complessi.

Con questa tipologia di prove il software viene testato all'interno di un ambiente di simulazione completamente virtuale, che però rappresenta il comportamento del sistema nel mondo reale. L'ambiente di simulazione può variare di complessità, con lo scopo di fornire un contesto sempre più realistico in cui effettuare le simulazioni. In questo modo il programma viene isolato dall'hardware fisico, con il vantaggio di poter ripetere i test per poter valutare la coerenza del comportamento a differenti scenari e non correndo rischi di danneggiamento dei componenti reali.

Nel caso particolare della GUI sviluppata per poter essere interfacciato con il banco prova, prima di poterla collegare all'hardware fisico si rende necessario un test in ambiente di simulazione per verificare il corretto funzionamento del codice sviluppato, identificando e risolvendo eventuali problemi e riducendo così il rischio di compiere errori costosi sull'interno sistema fisico: è stato quindi sviluppato un modello in Matlab-Simulink che replica in un ambiente completamente virtuale il comportamento del banco prova, generando gli stessi segnali che si vogliono monitorare nelle prove reali.

Nei paragrafi successivi verrà quindi riportato e descritto il modello di simulazione creato, evidenziandone anche le principali limitazioni rispetto al banco reale e cosa comporta questo a livello software rispetto a quella che sarà la sua versione definitiva per l'accoppiamento all'hardware fisico.

Verranno quindi anche riportati dei risultati di simulazione che consentono di verificare il corretto funzionamento sia della GUI sia del modello creato.

Modello Simulink del banco prova: descrizione e semplificazioni rispetto al sistema fisico

I blocchi caratteristici utilizzati per la creazione del modello fanno parte del tool Simscape, che consente di effettuare simulazioni multi-fisiche consentendo dunque di riprodurre in modo abbastanza fedele il comportamento del banco prova reale.

Un modello a blocchi rappresentativo del banco prova viene nuovamente riportato in Figura 7.1, nel quale:

- Le linee a doppia freccia in colore blu rappresentano i collegamenti elettrici tra le batterie e le due macchine elettriche; la presenza della doppia punta è voluta, ed indica la possibilità sia di fornire che di ricevere corrente da entrambe le macchine, a seconda della modalità in cui stanno lavorando.
- Le linee a doppia freccia verdi indicano i collegamenti meccanici, corrispondenti agli alberi uscenti dai diversi componenti. Anche in questo caso la doppia freccia indica la possibilità di effettuare trasferimento di potenza meccanica in entrambi i versi, a seconda della modalità di funzionamento delle due macchine elettriche.
- Le linee in arancio indicano invece i collegamenti termofluidi tra le macchine elettriche ed i relativi dissipatori di potenza termica, per contenerne la temperatura di funzionamento.

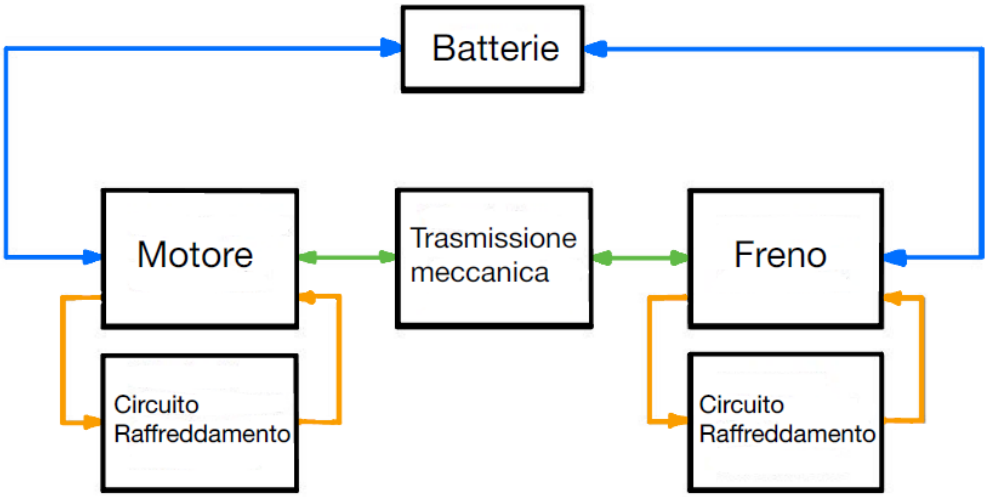


Figura 7.1: ***Schema a blocchi del banco***

Nella Figura 7.2 a seguire viene quindi mostrato lo schema equivalente costruito in Simulink sfruttando la libreria Simscape.

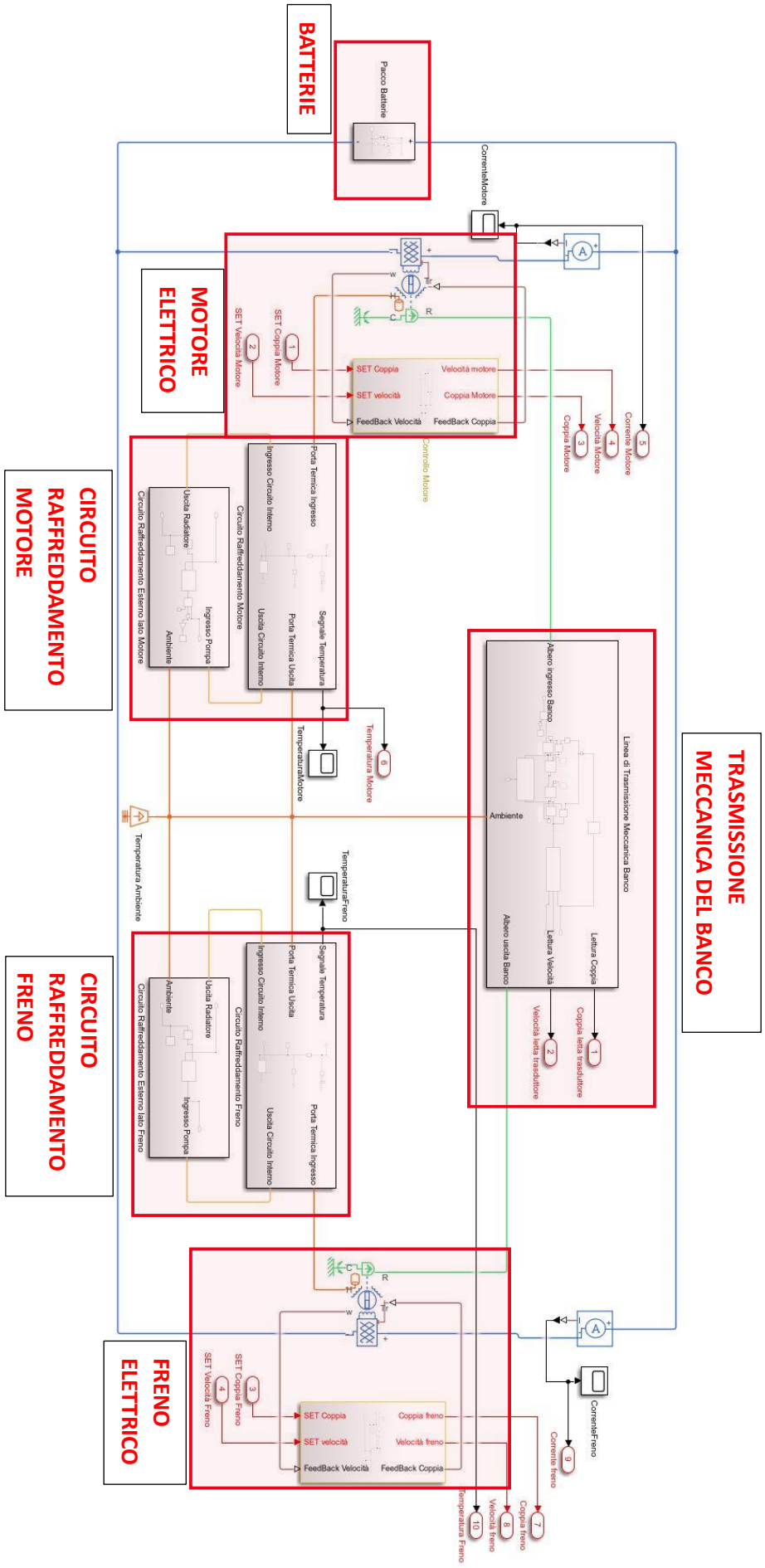


Figura 7.2: ***Schema a blocchi Simulink del banco prova***

Si può notare come le Figure 7.1 e 7.2 sono molto simili, in quanto il software e la libreria utilizzata consentono di semplificare notevolmente la costruzione dei modelli utilizzando blocchi all'apparenza piuttosto semplici, ma che al loro interno riescono a racchiudere molte informazioni e leggi fisiche del sistema reale.

Analizzando quindi nel dettaglio i diversi sottoblocchi:

- **Pacco Batterie**

  I motori e il freno del banco vengono tutti alimentati dalle stesse batterie, dalle caratteristiche già riportate in Tabella 3.1. Ai fini del modello, piuttosto che considerare 8 moduli di batterie da 80 V ciascuna poste in serie come nel banco prova reale, si è considerato un unico blocco equivalente di una tensione nominale di 640 V e di capacità pari a 25 Ah.

  Quando il freno genera una coppia negativa, quest'ultimo agisce da generatore fornendo corrente che verrà raccolta dalle batterie stesse. In condizioni ideali e con rendimenti dei motori, della batteria e degli azionamenti (inverter non rappresentati nel modello a blocco) unitari si avrebbe la tensione della batteria ad un valore costante per tutta la durata della prova. In queste condizioni, infatti, il freno genererebbe una quantità di corrente esattamente pari a quella consumata dal motore, senza modificare lo stato di carica delle batterie. Per avere un modello che rispecchi quanto più possibile il banco prova fisico, sono stati introdotti in quest'ultimo i rendimenti delle diverse parti.

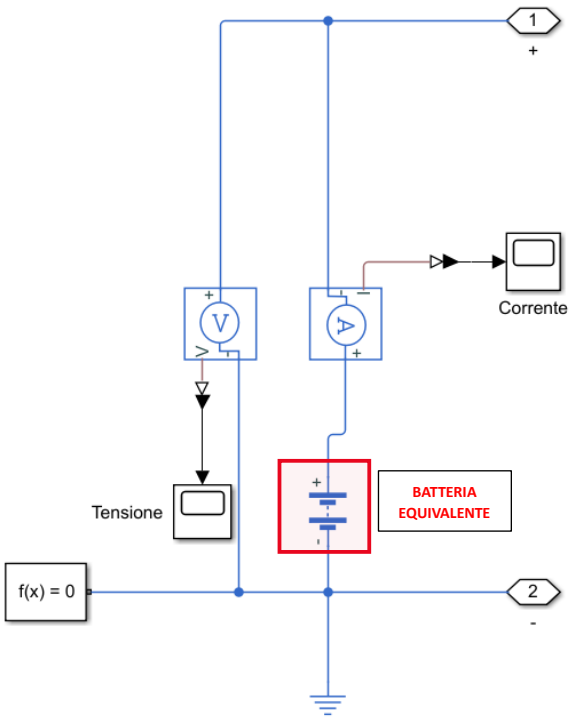


Figura 7.3: ***Schema a blocchi equivalente del pacco batterie***

- **Trasmissione meccanica del banco prova**

Si è detto che la linea di trasmissione del banco è costituita da due riduttori posti in contrapposizione, separati da un doppio giunto di Cardano. A valle segue un trasduttore per la misura di coppia e velocità della linea meccanica. Il modello quindi realizzato, tenendo conto delle caratteristiche tecniche dei riduttori installati e riportate in Tabella 3.1, è riportato in Figura 7.4.

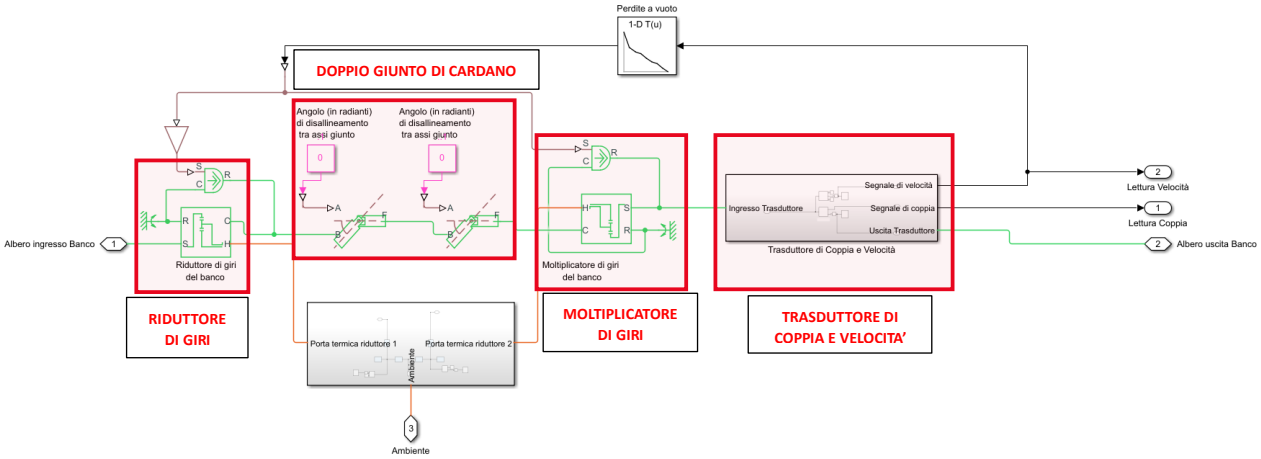


Figura 7.4: ***Schema a blocchi della linea di trasmissione meccanica del banco prova***

I blocchi necessari per rappresentare i riduttori epicicloidali del banco prova sono stati calibrati in modo tale da rappresentare:

1. Il rapporto di ingranamento effettivo, ricavato dal catalogo del costruttore, già riportato in Tabella 3.1;
2. La coppia a vuoto necessaria a mantenere il sistema in rotazione ad uno specifico set di velocità: per far ciò, e per essere il più coerenti possibile con il comportamento del banco reale, si è preferito riprodurre questo effetto interpolando i dati sperimentali ricavati e discussi nel Capitolo 4;
3. Il riscaldamento degli stessi a seguito di un loro utilizzo prolungato, data l'assenza di un circuito di raffreddamento dedicato all'olio lubrificante. La stima dei parametri per modellare questo comportamento tiene conto dei pochi dati sperimentali ottenuti in tal senso mediante delle misure di temperatura mediante termocamera, effettuate durante l'utilizzo del banco prova fisico.

Per quanto riguarda invece i blocchi relativi al doppio giunto di Cardano, questo può essere modellato utilizzando un doppio Universal Joint, che consente anche di inserire come input l'angolo di disallineamento tra gli assi di ingresso e uscita del giunto stesso.

Infine, per quanto riguardo il trasduttore di coppia e velocità, quest'ultimo può essere modellato utilizzando un sensore di coppia ideale (posto in serie alla linea meccanica) ed un sensore di velocità ideale (posto in parallelo alla linea meccanica), dato che anche nella realtà le perdite associate alla presenza sulla linea di tali elementi di misura sono sostanzialmente trascurabili.

- **Blocco Motore e Blocco Freno con relativi controlli**

  Sia il motore sia il freno elettrici montati a banco sono stati modellati utilizzando il blocco Motor&Drive.

  L'uso di questo blocco permette di omettere dal modello entrambi gli inverter/azionamenti, rispettivamente di motore e freno: infatti nonostante siano entrambe delle macchine PMSM a corrente alternata, questo è comunque in grado di rappresentarle includendo i driver all'interno del blocco e collegandoli direttamente ad una fonte di tensione continua DC, come il pacco batterie riportato in Figura 7.3.

  Diventa in questo modo particolarmente semplice anche modellare i rispettivi controlli: per entrambi, come per il banco reale, sono stati previsti sia un controllo coppia sia un controllo velocità. Nel caso di controllo coppia viene passato in ingresso al blocco Motor&Drive direttamente il valore di coppia che si desidera ottenere, in quanto tale blocco è concepito per effettuare esplicitamente controlli coppia (o equivalentemente in corrente). Nel caso invece del controllo velocità, per ricondurre quest'ultimo ad un controllo coppia equivalente voluto dal blocco di simulazione, si utilizza un controllore PID che elabora l'errore tra la velocità di Set e quella effettiva di Feedback generando un segnale di coppia verso la macchina elettrica necessaria a vincere quell'errore. La presenza di parte integrativa nel controllo garantisce inoltre che anche in condizioni di errore nullo (ossia quando si è raggiunta la velocità di Set voluta) il motore/freno continua a erogare la coppia necessaria a mantenere quella condizione di regime di velocità.

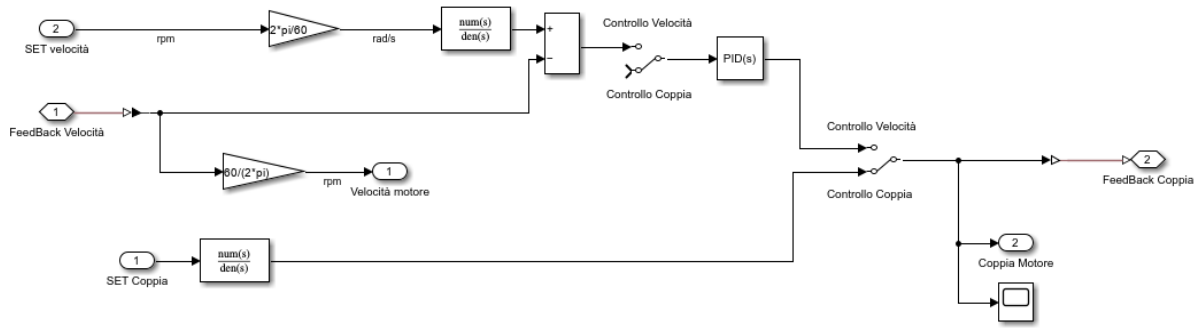


Figura 7.5: ***Controllo coppia e velocità per i blocchi Motor&Drive di Motore e Freno del banco***

L'assenza nel modello dei blocchi relativi agli azionamenti, sommata all'assenza, in questa prima fase di test della GUI, del software di gestione della centralina del banco, fanno sì che i segnali di SET sia di coppia sia di velocità provengano direttamente dall'utente che agirà, durante il test SiL, sui cursori presenti sulla schermata dell'interfaccia grafica, seppur questi segnali vengano comunque fatti transitare in messaggi CAN per simulare in ogni caso il reale trasferimento di dati dalla centralina ai sottosistemi del banco. Quest'aspetto verrà ancora ulteriormente ripreso, per una maggiore comprensione, nel paragrafo a seguire.

- **Circuiti di raffreddamento di Motore e Freno**

Per poter simulare anche il trasferimento sulla rete CAN dei segnali di temperatura, specialmente per il blocco motore (parametro monitorato anche durante le acquisizioni sperimentali), è stata abilitata per entrambi i blocchi Motor&Drive la thermal port, che consente di valutare quali effetti termici si producono a seguito di un utilizzo prolungato delle suddette macchine. E' stato quindi modellato anche lo scambio termico che motore e freno hanno con l'ambiente circostante e con il circuito di raffreddamento predisposto a banco. Data l'assenza nel modello di blocchi che simulano gli azionamenti, a differenza delle prove sperimentali in cui era stato valutato l'andamento della temperatura anche di questi ultimi, ai fini di questi test SiL si è andati a valutare l'andamento della sola temperatura di motore e freno (e non dei rispettivi azionamenti in quanto assenti).

Si riportano quindi in Figura 7.6 gli schemi a blocchi della parte di circuito termofluido interno a motore/freno e della parte di circuito termofluido esterno, in cui il calore viene dissipato per mezzo di un radiatore. L'obiettivo, come già discusso nel Capitolo 4, è quello di contenere, anche nel modello di simulazione, le temperature al di sotto delle temperature limite di funzionamento, per danneggiare o smagnetizzare totalmente i magneti permanenti.

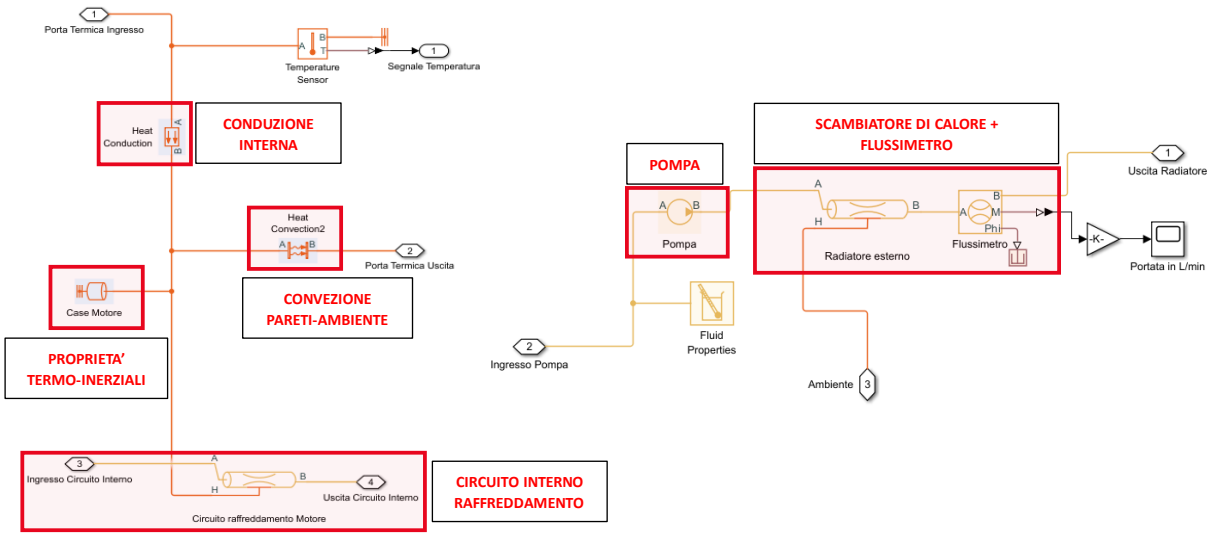


Figura 7.6: ***Circuito completo di raffreddamento di Motore e Freno***

Si riporta il circuito del solo lato motore in quanto dal lato freno risulta perfettamente equivalente nella costruzione. La differenza risiede nei soli parametri all'interno dei vari blocchi, essendo le dimensioni geometriche delle macchine piuttosto differenti.

Ricordando anche lo schema riportato in Figura 4.8, il fluido entra nel circuito di raffreddamento dell'azionamento (qui non riportato) per poi entrare in quello interno alla camicia del motore/freno. Uscendo viene rimesso in circolazione da una pompa (qui simulata attraverso un generatore di portata ideale, dato il consumo irrisorio di questo elemento rispetto al resto del sistema) per entrare poi all'interno del radiatore. Il modello di simulazione, quindi, rispecchia il percorso descritto.

Collegamento del modello Simulink creato alla GUI per esecuzione del test SiL

Poiché uno degli obiettivi è quello di testare l'Interfaccia grafica creata, è necessario allora crearne un collegamento con il modello sviluppato. Per far ciò è necessario procedere secondo quanto riportato in Figura 7.7.

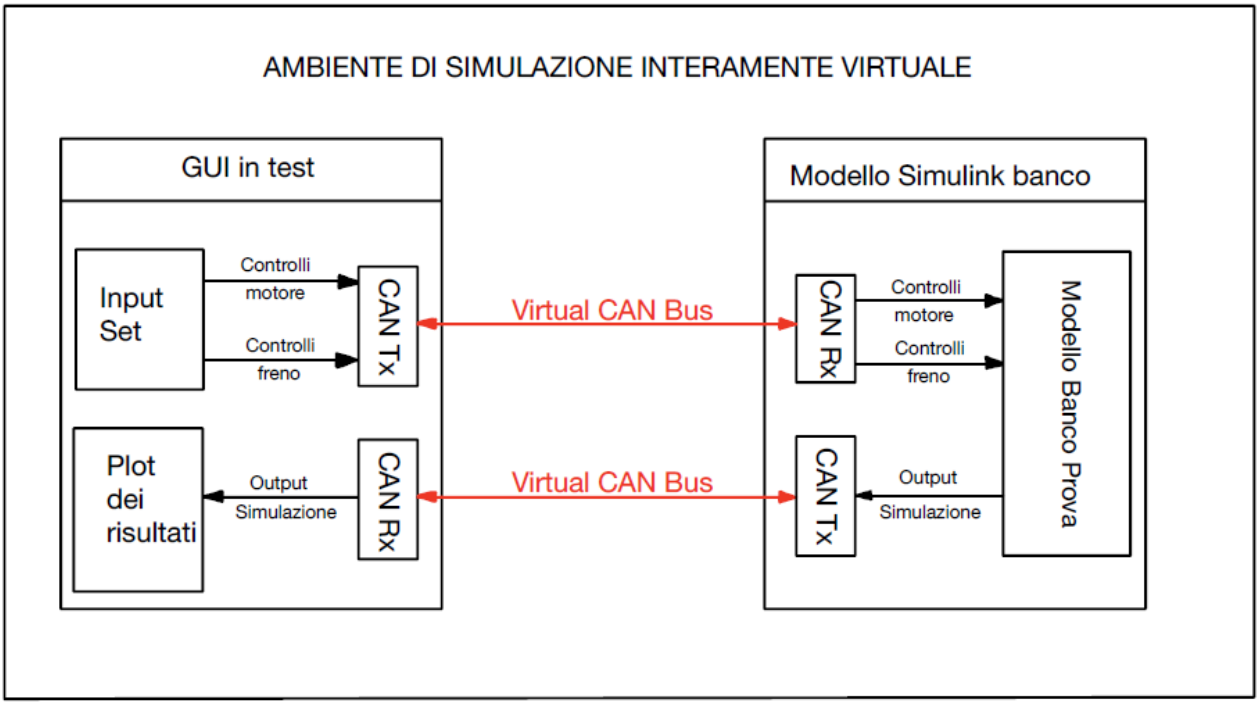


Figura 7.7: ***Schema di collegamento tra GUI e modello Simulink del banco prova***

Come già più volte ripetuto nei capitoli precedenti, in questa prima fase di test del Software si sta effettuando una forte semplificazione, non considerando che nella realtà tra il banco prova e l'Interfaccia sarà presente la centralina, con relativo Software di controllo: di fatto in queste prime prove quello che è il compito della centralina (ossia di generare i segnali verso motore e freno del banco) viene intrinsecamente incluso all'interno dei cursori di comando riportati nella GUI, inviando al modello dei segnali come se questi uscissero appunto da una HCU.

Tutti gli aspetti relativi alla costruzione della GUI sono stati già trattati nel Capitolo 6, sia per quanto riguarda la trasmissione sia per quanto riguarda la ricezione di messaggi CAN.
A seguire verrà quindi solamente illustrato in che modo i comandi effettuati sui cursori della GUI vengono recepiti dal modello di simulazione ed in che modo quest'ultimo invia i risultati della simulazione all'Interfaccia.

A differenza della GUI, in cui la gestione dei messaggi CAN deve essere effettuata interamente da linea di codice, in ambiente Simulink i messaggi CAN sono piuttosto semplici da utilizzare in quanto gestibili graficamente attraverso i blocchi messi a disposizione dal tool VNT.

In particolare, sono stati utilizzati:

1. Il blocco "CAN Configuration", che permette di sincronizzare il modello di simulazione al canale CAN scelto attraverso la GUI nel menù CAN Channel selection;

2. Dei blocchi “CAN Receive” per ricevere all’interno del modello i messaggi CAN transitanti nel canale precedentemente selezionato;
3. Dei blocchi “CAN Unpack” per estrarre dai messaggi ricevuti i dati da questi ultimi trasmessi e necessari al funzionamento del modello;
4. Dei blocchi “CAN Pack” per raggruppare i risultati di simulazione all’interno di specifici messaggi CAN;
5. Dei blocchi “CAN Transmit” per trasmettere questi ultimi messaggi contenenti i risultati di simulazione sul canale CAN, messaggi che verranno poi rielaborati all’interno della GUI per eseguire la visualizzazione e il salvataggio dei dati.

Con l’aggiunta di questi blocchi di collegamento CAN all’Interfaccia, il modello Simulink completo diventa come riportato in Figura 7.8:

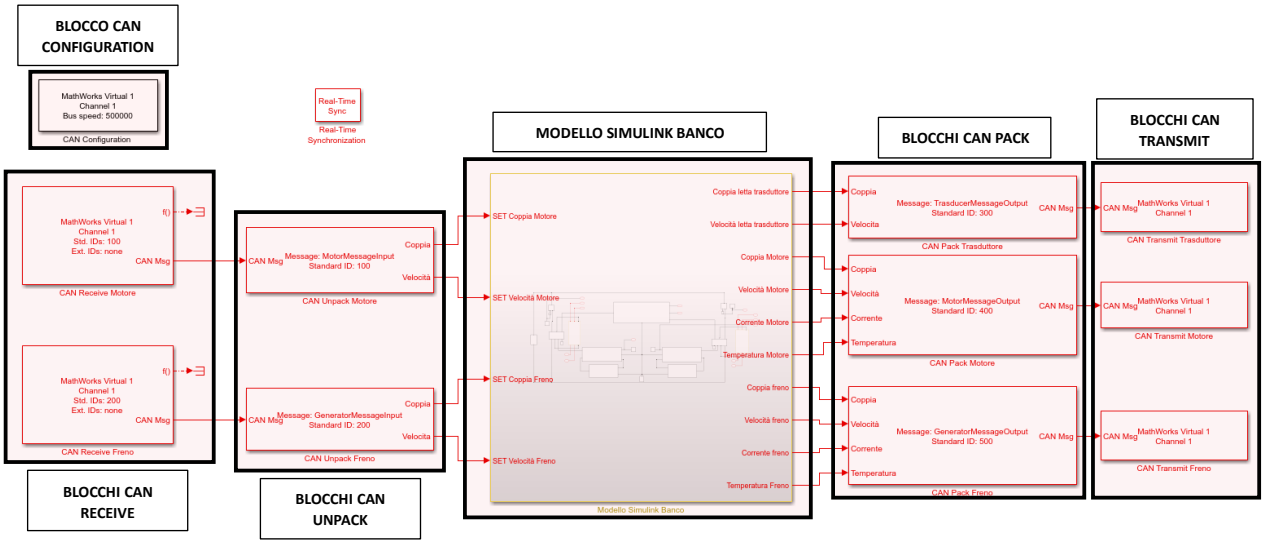


Figura 7.8: ***Modello Simulink del banco con collegamento alla GUI***

Nel modello è inoltre stato inserito il blocco “Real-Time Sync” per sincronizzare il tempo di simulazione del solutore con il tempo fisico reale con cui i messaggi CAN viaggiano sul Bus virtuale, necessario quando si vogliono realizzare dei trasferimenti di dati in tempo reale.

Il collegamento tra GUI e modello avviene quindi gestendo opportunamente gli ID dei messaggi CAN, che dovranno essere in corrispondenza tra linee di codice e blocco Simulink: gli Input Set di per i blocchi Motore e Freno vengono trasmessi tramite messaggi CAN a cui sono stati assegnati, tramite i comandi “pack” nel codice della GUI, rispettivamente gli Standard ID 100 e 200; all’interno del modello quindi i blocchi CAN Receive vengono così compilati:

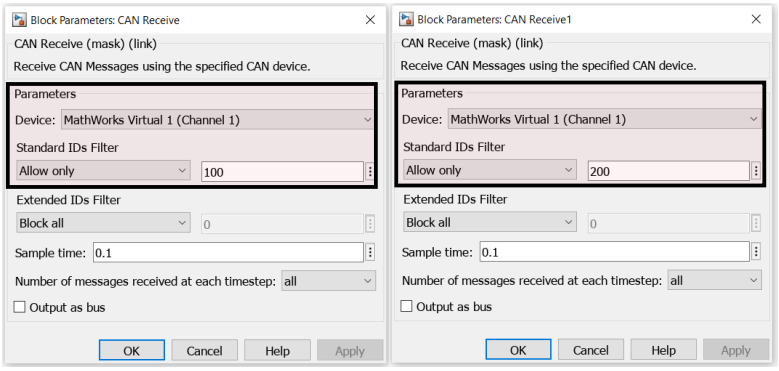


Figura 7.9: ***Compilazione dei blocchi Simulink CAN Receive***

I blocchi CAN Receive vengono quindi collegati ai blocchi Simulink CAN Unpack, all'interno dei quali viene eseguita l'estrazione dei segnali trasportati dai rispettivi messaggi CAN. Sia per il blocco motore sia per il blocco freno, i segnali di input trasportati possono essere o di coppia o di velocità richiesta. Pertanto, i blocchi CAN Unpack sono stati compilati come riportato in Figura 7.10:

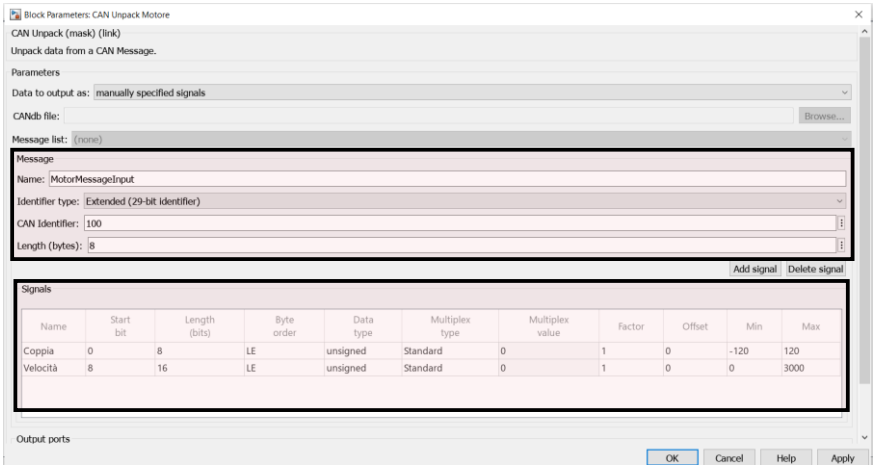


Figura 7.10: ***Compilazione del blocco Simulink CAN Unpack Motore***

Viene riportato in Figura 7.10 solo l'unpack relativo al blocco motore in quanto quello relativo al freno è del tutto identico, a meno del differente ID utilizzato per identificare il messaggio.

A questo punto, i segnali sono pronti per essere elaborati dal modello del banco prova.
I segnali di Output dal modello che si è scelto di monitorare, in coerenza con quanto effettuato anche durante le prove sperimentali sul banco reale, vengono quindi raggruppati in base al blocco di simulazione da cui provengono, andando in questo modo a costruire 3 messaggi CAN di output:

- un messaggio CAN con Standard ID 300 relativo al trasduttore
- un messaggio CAN con Standard ID 400 relativo al motore
- un messaggio CAN con Standard ID 500 relativo al freno

Per quanto concerne il trasduttore, essendo questo di coppia e di velocità, possono essere solo estratti un segnale di velocità ed uno di coppia. Questi segnali devono essere raggruppati all'interno del relativo messaggio attraverso il blocco CAN Pack, come riportato in Figura 7.11:

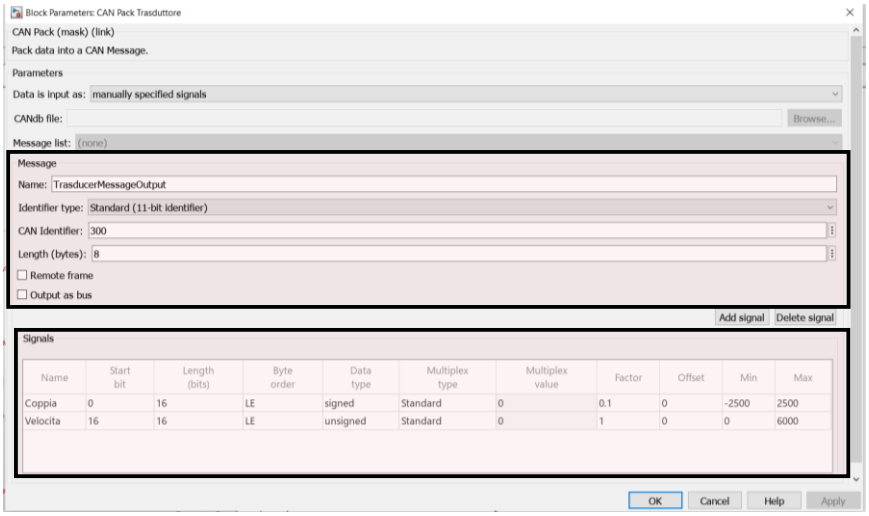


Figura 7.11: ***Compilazione del blocco Simulink CAN Pack Trasduttore***

Per quanto riguarda invece motore e freno, da questi si sono potuti raccogliere dati in merito alle correnti erogata/assorbita ed alle temperature di esercizio. Il blocco CAN Pack in questo caso è quindi stato compilato come riportato nella Figura 7.12 a seguire:

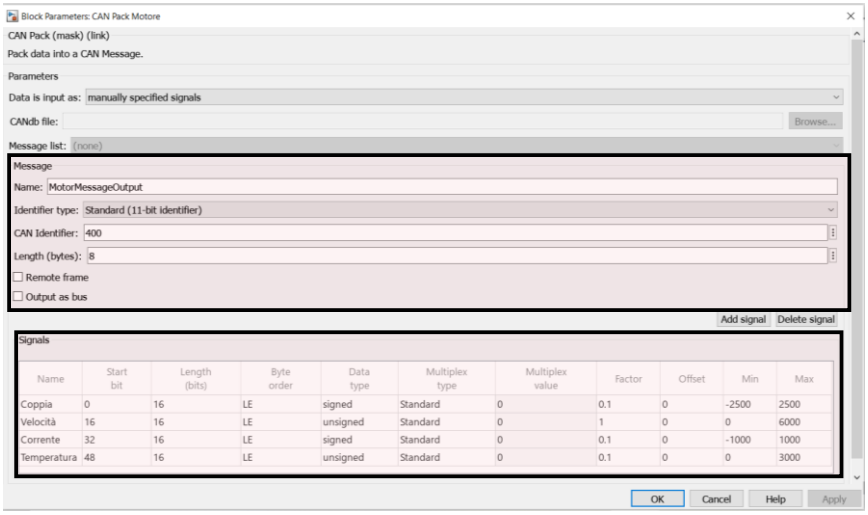


Figura 7.12: ***Compilazione del blocco Simulink CAN Pack Motore***

Viene riportato in Figura 7.12 solo il pack relativo al blocco motore in quanto quello relativo al freno è del tutto identico, a meno del differente ID utilizzato per generare il messaggio.

Come si può vedere da Figura 7.8, tutti i blocchi CAN Pack vengono collegati in uscita con dei blocchi CAN Transmit, attraverso i quali si effettua la trasmissione periodica dei messaggi CAN contenenti gli Output di simulazione provenienti da Trasduttore, Motore e Freno verso la GUI, per effettuarne il post-processing opportuno. Per effettuare una trasmissione dei messaggi di tipo periodica, il blocco Simulink CAN Transmit deve essere configurato come riportato in Figura 7.13 a seguire:

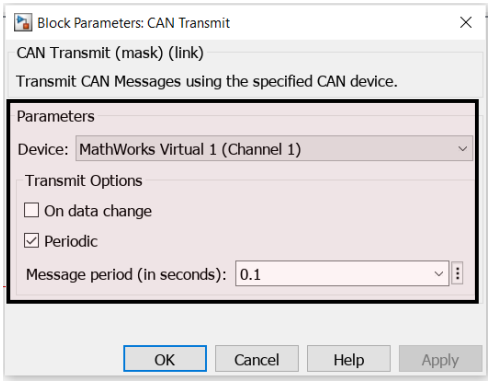


Figura 7.13: ***Compilazione di uno dei blocchi Simulink CAN Transmit***

Altre funzioni specifiche, come la `function updateModelWithSelectedCANChannel(app, index)`, sono implementate all'interno della GUI per consentire un collegamento ottimale tra Interfaccia e modello Simulink.

Come tutte le altre funzioni precedentemente descritte, anche quest'ultima può essere trovata in fondo al seguente documento, nel Capitolo "Appendice ed allegati", Sottocapitolo "Script della GUI sviluppata per test SiL".

Lo schema effettivo del test SiL realizzato per la GUI è quindi riportato in Figura 7.14 a seguire.

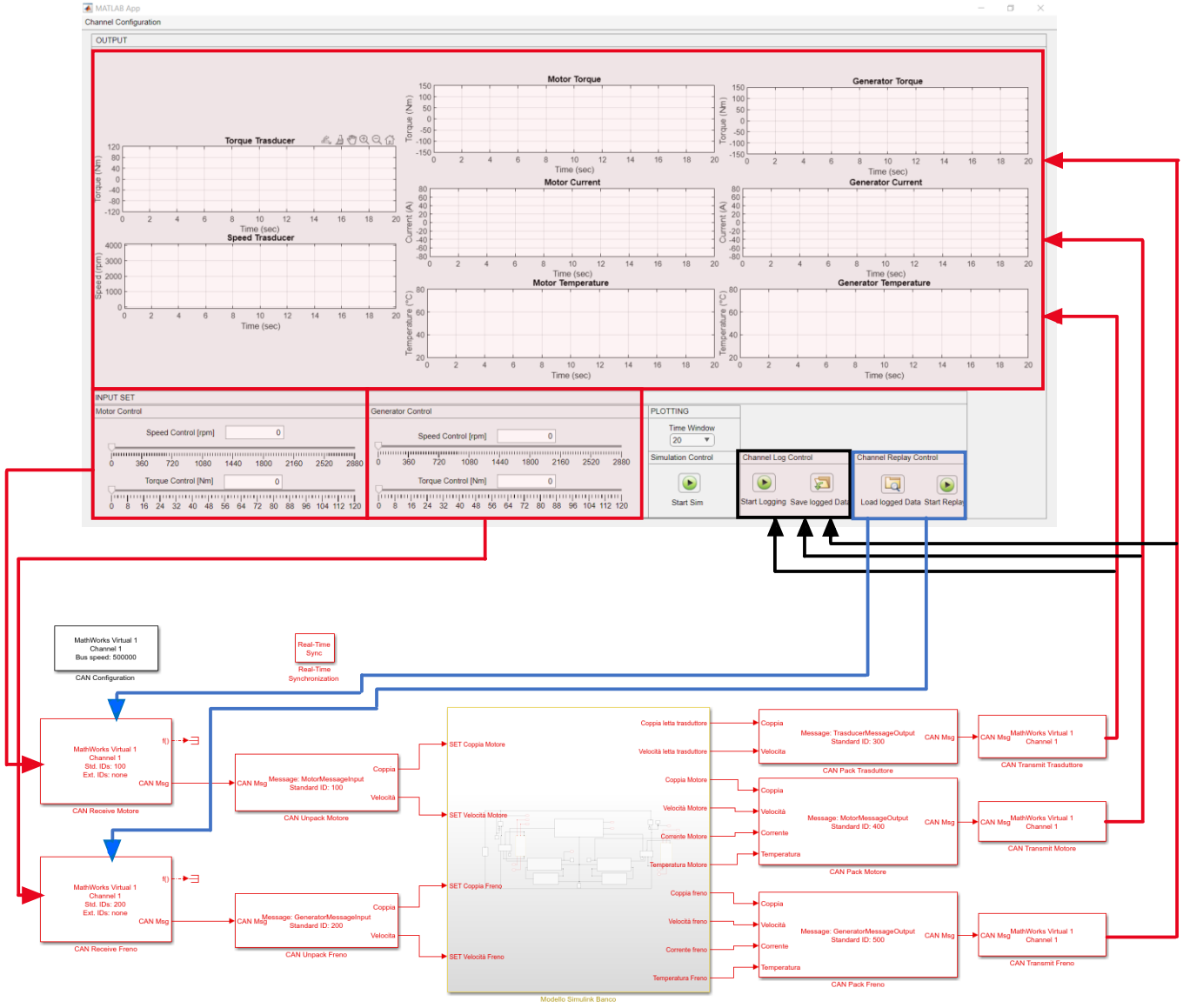


Figura 7.14: ***Schema del test SiL implementato***

Verifica del funzionamento della GUI e del modello Simulink del banco prova: esecuzione dei test SiL ed analisi dei risultati ottenuti

Terminata la costruzione del modello ed effettuato il collegamento all'Interfaccia Grafica, è possibile effettuare delle simulazioni SiL con le quali andare a verificare contemporaneamente:

- il corretto funzionamento della GUI sviluppata, in particolar modo per quanto riguarda la comunicazione CAN con il modello di simulazione;
- la risposta del modello di simulazione del banco a differenti input, verificando quindi se c'è corrispondenza tra il comportamento fisico del banco reale e quello simulato. Il modello infatti è stato calibrato sulla base dei dati di targa dei componenti del banco reale, facendo in modo che i risultati ottenuti in simulazione siano il più possibile in linea con gli quelli verificati nella realtà durante le diverse acquisizioni sperimentali eseguite.

Si procede quindi aprendo la GUI, andando innanzitutto a selezionare tramite l'apposito menù il canale CAN su cui eseguire la comunicazione: essendo un test SiL si procede utilizzando uno dei due canali CAN virtuali forniti dal tool VNT.
L'utilizzo della `function updateModelWithSelectedCANChannel(app,index)` nel codice dell'Interfaccia garantisce che nel momento in cui si effettua una variazione nella scelta del canale, quest'ultima venga trasmessa al modello Simulink in modo da aggiornare automaticamente questa scelta anche all'interno del blocco CAN Configuration.
Essendo questo un test virtuale, per far partire la comunicazione CAN ma anche per avviare il modello è necessario utilizzare lo Start Sim: come si vedrà nelle Figure a seguire, in realtà la comunicazione CAN inizia immediatamente quando viene premuto il suddetto pulsante, mentre il modello di simulazione inizierà a girare con un ritardo intrenseco del Software Simulink nella compilazione del modello stesso. I messaggi CAN di output da quest'ultimo, quindi, inizieranno ad essere visibili solo dopo alcuni secondi.

Nel corso delle simulazioni effettuate e riportate in questo documento, è stato inoltre predisposto il salvataggio dei messaggi utilizzando il "Channel Log Control", testandone così l'effettivo funzionamento. Questo dà modo, inoltre, di poter infine anche testare la funzione "Channel Replay Control" rilanciando successivamente le simulazioni ma dando in input al modello i dati salvati: questa funzionalità può risultare utile applicata poi al banco prova fisico, in quanto può essere utilizzata ad esempio per effettuare dei test a banco utilizzando dati CAN ricavati però da un veicolo operante in campo, testando quindi i motori basandosi su dati raccolti in field test reali.

Si riportano quindi a seguire le simulazioni eseguite in alcuni scenari tipici di utilizzo del banco prova:

1. Motore in controllo Coppia e Freno in controllo Velocità e confronto con l'equivalente prova sperimentale.
2. Motore in controllo Velocità e Freno in controllo Coppia.
3. Motore in controllo Velocità e Freno in controllo Coppia, con Motore in frenata rigenerativa.

I profili di carico scelti ,ed eseguiti agendo sugli Input Set della GUI, vengono spiegati nel dettaglio nei rispettivi paragrafi; in generale si è cercato di coprire il maggior numero di scenari possibili, con il fine di verificare il funzionamento corretto dell'intero sistema in qualunque circostanza esso si possa trovare. Per verificare la corretta trasmissione dei messaggi CAN, si può monitorare il Bus virtuale utilizzato per le simulazioni mediante lo strumento CanExplorer.

1. **Simulazione con Motore in controllo Coppia e Freno in controllo Velocità a 2600 rpm e confronto con l'equivalente prova sperimentale riportata nel Capitolo 4.**

La prima simulazione effettuata è volta a replicare l'intera prova di durata eseguita sul banco prova fisico sul motore. In particolare, si è scelto di riprodurre la prova eseguita alla velocità di regime di 2600 rpm, essendo quella più gravosa in termini di potenze in gioco eseguita.

La durata della simulazione è stata quindi calibrata sulla base della prova sperimentale di riferimento, e quindi pari a circa 1100 secondi (18 minuti).

Al suo termine, la schermata della GUI si presenta come riportato in Figura 7.15:

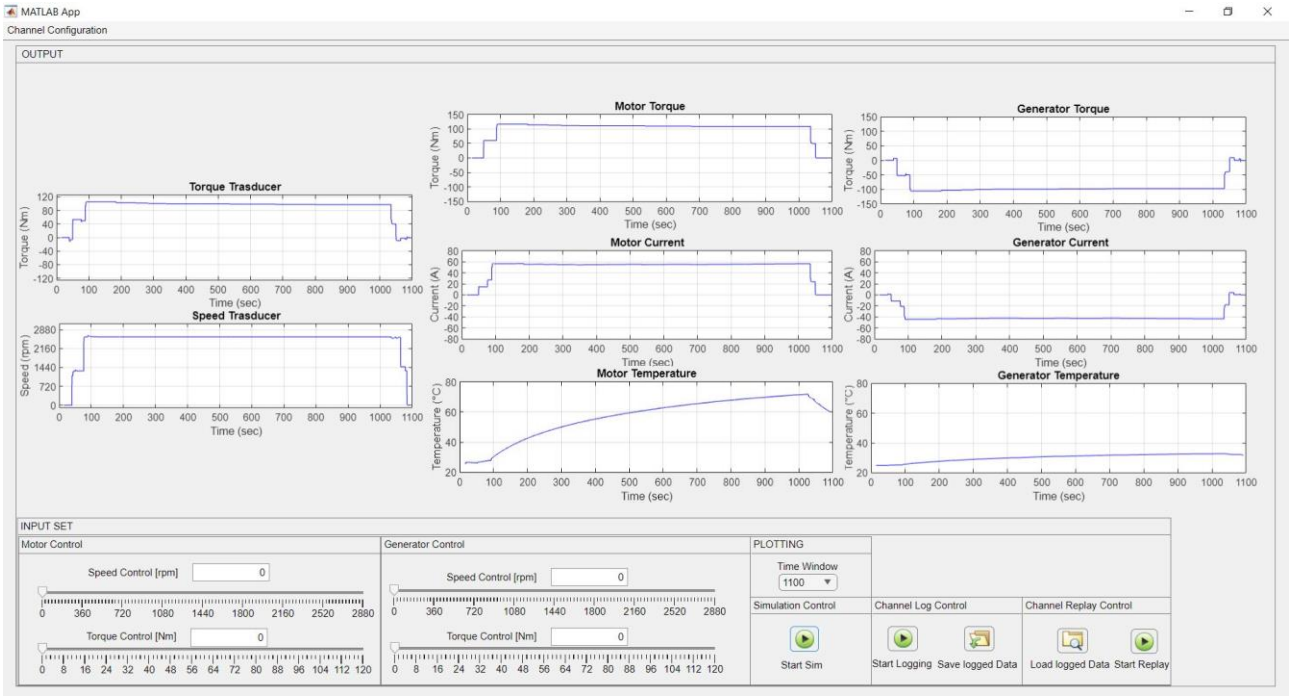


Figura 7.15: ***Schermata della GUI al termine della 1° simulazione effettuata***

Un estratto della trasmissione dei messaggi e dei dati CAN è, invece, riportata in Figura 7.16:

Message Table

Time	ID	Message	Length	Data
662.022926	C8	Input Freno	8	00 8C 0A 00 00 00 00 00
662.022908	64	Input Motore	8	78 00 00 00 00 00 00 00
662.008933	190	Output Motore	8	B0 04 8C 0A 5E 02 3B 02
661.993968	1F4	Output Freno	8	C5 FB 8C 0A 2E FE 32 01
661.987003	12C	Output Trasduttore	8	3B 04 8C 0A 00 00 00 00

Figura 7.16: ***Estratto dei messaggi transitanti sul Bus virtuale utilizzato in simulazione***

Quanto riportato in Figura 7.16 conferma quanto riportato nelle Figure 7.10 e 7.12; infatti:

- Il blocco Motore ha un input (ID=100 che in esadecimale equivale a ID=64) di Coppia, il cui segnale occupa il primo byte (primi 8 bit del Data message). In output (ID=400 che in esadecimale equivale a ID=190) si trasportano i segnali di Coppia nei primi 2 byte (primi 16 bit), di Velocità nel terzo e quarto byte (16 bit totali a partire dal bit numero 16), di Corrente nel quinto e sesto byte (16 bit totali a partire dal bit numero 32) ed infine la temperatura nel settimo e ottavo byte (16 bit totali a partire dal bit numero 48).
- Il blocco Freno ha un input (ID=200 che in esadecimale equivale a ID=C8) di Velocità, il cui segnale occupa il secondo e il terzo byte (16 bit totali del Data Message a partire dal bit numero 8). In output la costruzione del messaggio è identica a quella eseguita per il Motore, a meno dell'identificativa (che in questo caso è ID=500, in esadecimale ID=1F4).
- Il trasduttore in output (ID=300 che in esadecimale equivale a ID=12C) trasporta un segnale di Coppia nei primi due byte (primi 16 bit) e di Velocità nel terzo e quarto byte (16 bit totali a partire dal bit numero 16).

Il salvataggio dei dati CAN inviati e ricevuti può essere verificato aprendo nel Workspace di Matlab il file .mat di salvataggio, come riportato in Figura 7.17:

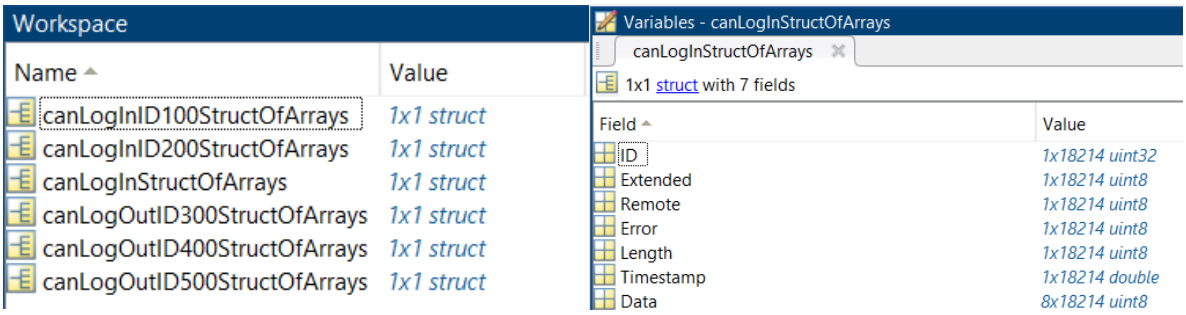


Figura 7.17: ***Salvataggio in memoria dei messaggi CAN***

Come si può vedere, i messaggi CAN vengono raccolti singolarmente a seconda dell'ID da cui sono caratterizzati. Al tempo stesso è però predisposta, nella struct canLogInStructOfArrays, la raccolta di tutti i messaggi CAN di Input al modello di simulazione, necessari per il corretto funzionamento della funzione di Replay dei messaggi.

Il funzionamento di quest'ultima parte di codice può quindi essere verificata, senza interrompere la simulazione in corso, procedendo al caricamento in memoria del file precedentemente salvato e da cui verrà estratta la struct precedentemente citata e avviando successivamente lo Start Replay. Il codice sottostante provvede quindi a ripulire i plot riportati sulla schermata della GUI e a interrompere la comunicazione CAN tramite i cursori, avviando il replay dei dati CAN invece caricati in memoria. Al termine di questa ulteriore simulazione, l'Interfaccia apparirà esattamente come esattamente come riportato in Figura 7.15 al termine della prima simulazione: questo conferma il corretto funzionamento della GUI, nonché la replicabilità dei risultati ottenuti mediante il modello creato.

Per quanto riguarda invece il comportamento del modello di simulazione, l'analisi dell'andamento delle grandezze estratte tramite il CAN-Bus virtuale consente di trarne delle conclusioni.

Si riporta quindi in Tabella 7.1 il set di comandi dati al modello nel corso della simulazione:

Tempo di Simulazione [s]	Controllo Motore in Coppia [Nm]	Controllo Freno in Velocità [rpm]	
0 (Start Sim)	0	0	Step 1
30	60	1300	
60	117	2600	
120	117	2600	Step 2
180	117	2600	
240	117	2600	
300	117	2600	
360	117	2600	
420	117	2600	
480	117	2600	
540	117	2600	
600	117	2600	
660	117	2600	
720	117	2600	
780	117	2600	
840	117	2600	
900	117	2600	
960	117	2600	
1020	117	2600	Step 3
1080 (Stop Sim)	0	0	

Tabella 7.1: ***Input Set al modello di simulazione - 1° simulazione***

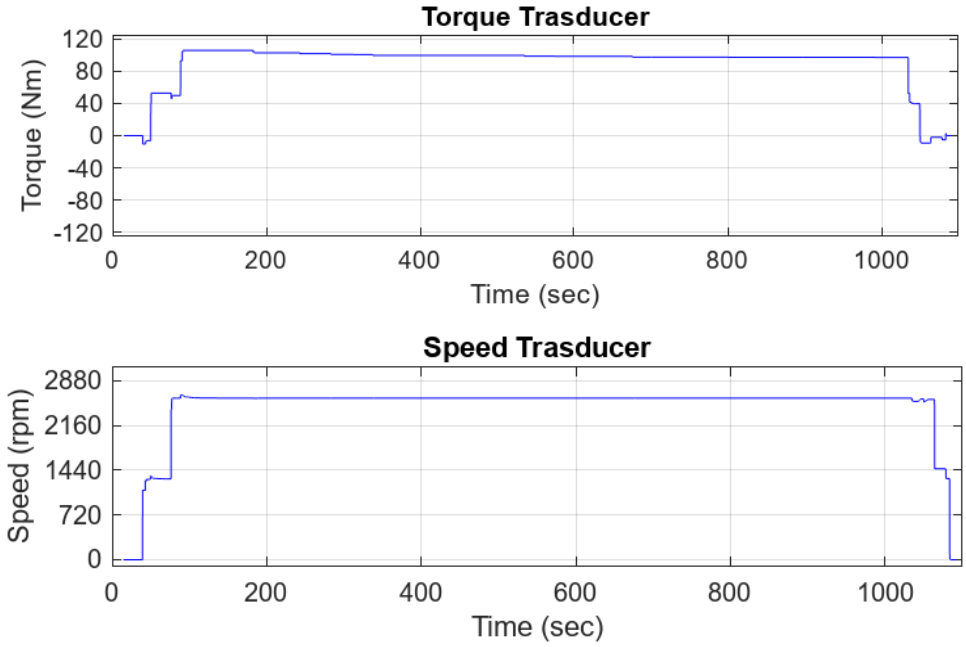


Figura 7.18: ***Coppia e Velocità lette dal modello del trasduttore - 1° simulazione***

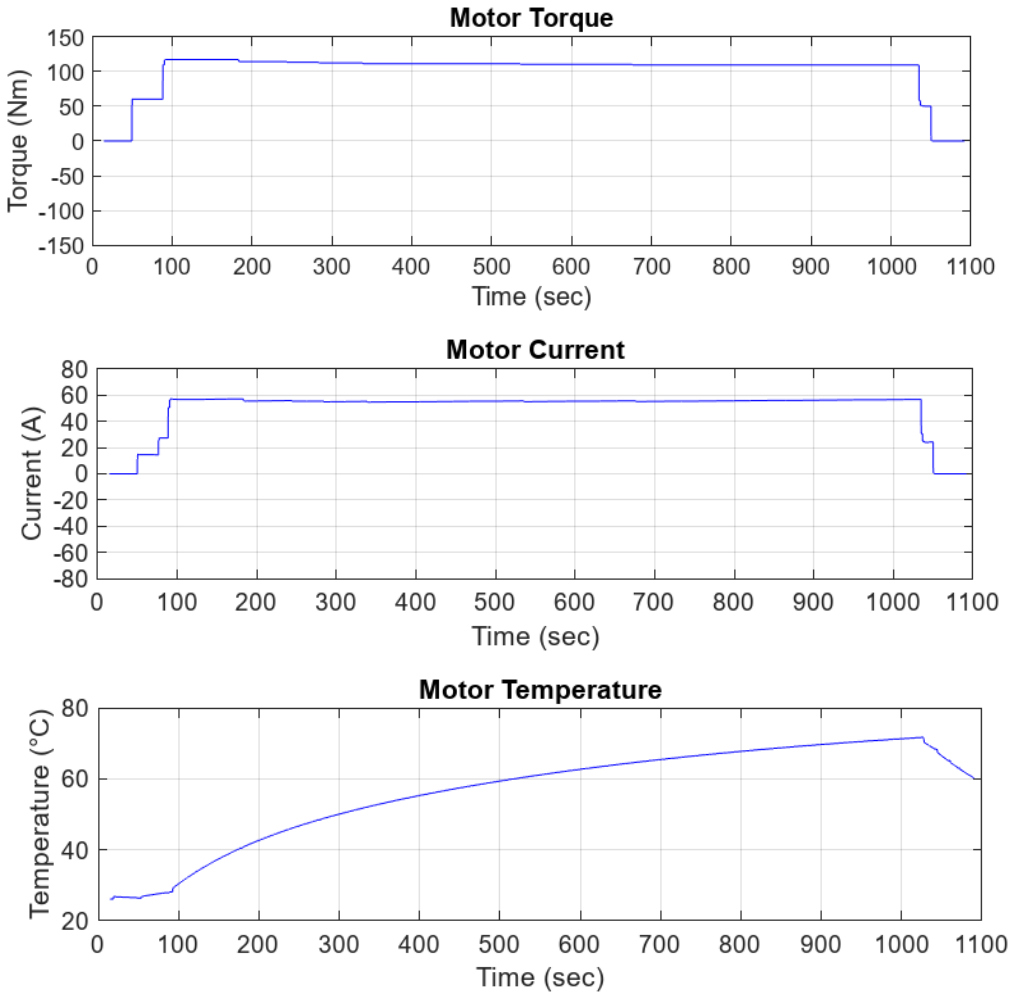
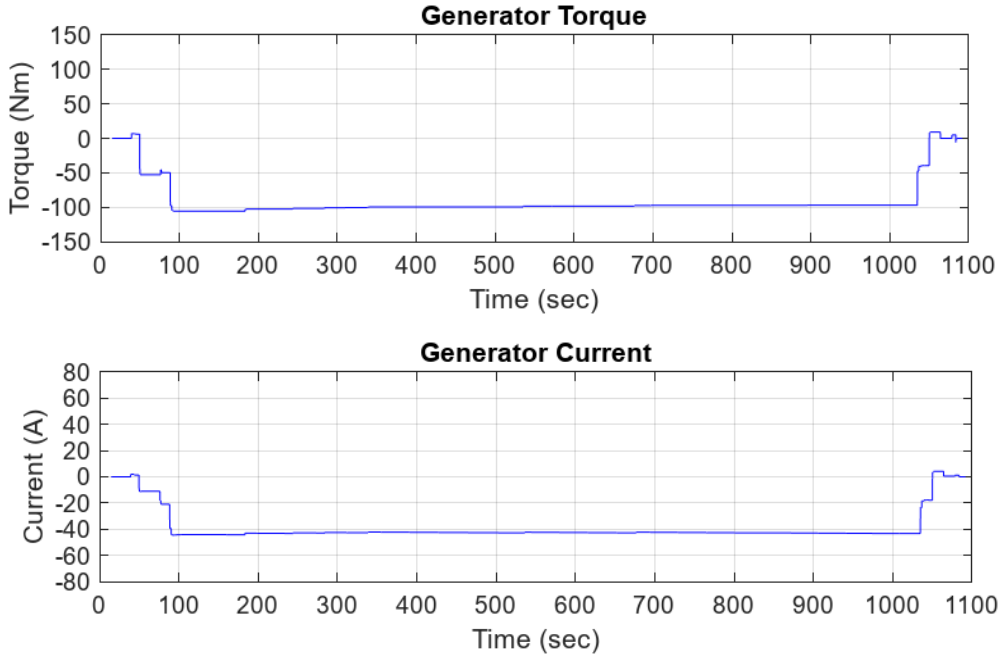


Figura 7.19: ***Coppia erogata e Corrente assorbita dal modello del motore e sua Temperatura - 1° simulazione***



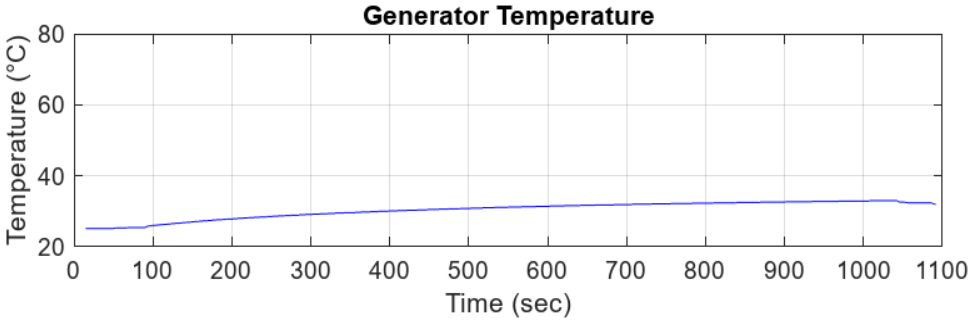


Figura 7.20: ***Coppia assorbita e Corrente erogata dal modello del freno e sua Temperatura - 1° simulazione***

Per una maggiore comprensione, la simulazione è stata suddivisa in 3 step, di seguito esplicati:

- Step 1 [0-120 s]: il primo step della simulazione eseguita serve a portare il modello di simulazione nelle condizioni di regime operativo di prova: si interviene quindi progressivamente sia sul controllo velocità del freno sia sul controllo coppia del motore fino a portarli rispettivamente nelle condizioni riportate in Tabella 7.3, vale a dire il freno ad una velocità costante di 2600 rpm e il motore in erogazione di coppia pari a 117 Nm, corrispondente alla sua coppia massima nominale.

- Step 2 [120-1020 s]: come avviene nella prova sperimentale descritta nel Capitolo 4, si procede mantenendo sia il motore in controllo alla massima coppia nominale erogabile, sia il freno in controllo velocità alla velocità di set imposta per un periodo di tempo prolungato per verificare l'evoluzione del comportamento del sistema. Con il progredire della simulazione, l'aumento della temperatura provoca un aumento di resistenza elettrica (simulata nel blocco Motor&Drive attraverso un coefficiente di variazione di resistenza in funzione della temperatura). A parità di tensione di alimentazione del blocco motore, la corrente assorbita da quest'ultimo tende a diminuire fintanto che la temperatura tende a stabilizzarsi. La coppia erogata dal modello del motore, proporzionale alla corrente, tenderà quindi anch'essa a diminuire, nonostante sul Set si continui a richiedere la massima coppia nominale.

  Dal punto di vista del modello del freno, invece questo effetto risulta non visibile in quanto la sua temperatura si stabilizza a valori molto inferiori, in quanto quest'ultimo lavora, in condizioni di regime, a potenze ben al di sotto di quella nominale: in tali condizioni quindi il circuito di raffreddamento predisposto è in grado di smaltire il calore più efficacemente, garantendo alla macchina delle temperature di esercizio inferiori.

- Step 3 [1020-1080 s]: si procede a fermare il sistema portando prima a zero il controllo in coppia del motore e poi a ridurre progressivamente a zero anche il controllo in velocità del freno. Si può notare come la temperatura del modello del motore inizi progressivamente a scendere quando questo non sta più assorbendo corrente.

Nella Tabella 7.2 a seguire si riportano quindi i principali risultati ottenuti ai vari step dalla simulazione:

Tempo [s]	Controllo Freno in Velocità [rpm]	Controllo Motore in Coppia [Nm]	Coppia Torsiometro [Nm]	Coppia Effettiva Motore [Nm]	Corrente Assorbita Motore [A]	Temperatura Motore [°C]	Coppia Resistente Freno [Nm]	Corrente Erogata Freno [A]
0 (Start Sim)	0	0	0	0	0	25	0	0
30	1300	60	53	60	17	26,5	-53	-12
60	2600	117	107,5	117	57	27	-107,5	-44,6
120	2600	117	105,6	115,1	56,5	33,4	-105,6	-44
180	2600	117	102,8	112,3	56	40,7	-102,8	-43,6
240	2600	117	101,8	111,3	55,7	46	-101,8	-43,4
300	2600	117	100,7	110,2	55,5	50,1	-100,7	-43,2
360	2600	117	99,7	109,2	55	53,5	-99,7	-43
420	2600	117	99,5	109	54,9	56,3	-99,5	-42,7
480	2600	117	99	108,7	54,7	58,8	-99	-42,6
540	2600	117	98,6	108,2	55,5	60,8	-98,6	-42,5
600	2600	117	98,2	107,7	54,4	62,7	-98,2	-42,4
660	2600	117	97,5	107,3	54,3	64,5	-97,5	-42,3
720	2600	117	97,4	107,3	54,3	66	-97,4	-42,2
780	2600	117	97,3	107,2	54,2	67,4	-97,3	-42,2
840	2600	117	97,3	107,2	54,1	68,7	-97,3	-42,2
900	2600	117	97,2	107,1	54,1	69,8	-97,2	-42,1
960	2600	117	97,1	107	54	70,8	-97,1	-42
1020	2600	117	97,1	107	54	71,5	-97,1	-42
1080 (Stop Sim)	0	0	0	0	0	61	0	0

Tabella 7.2: ***Risultati – 1° simulazione***

Avendo eseguito la simulazione nelle stesse condizioni operative di una delle prove sperimentale, è possibile effettuare un confronto tra i risultati ottenuti sperimentalmente e in simulazione, riportandoli sugli stessi grafici:

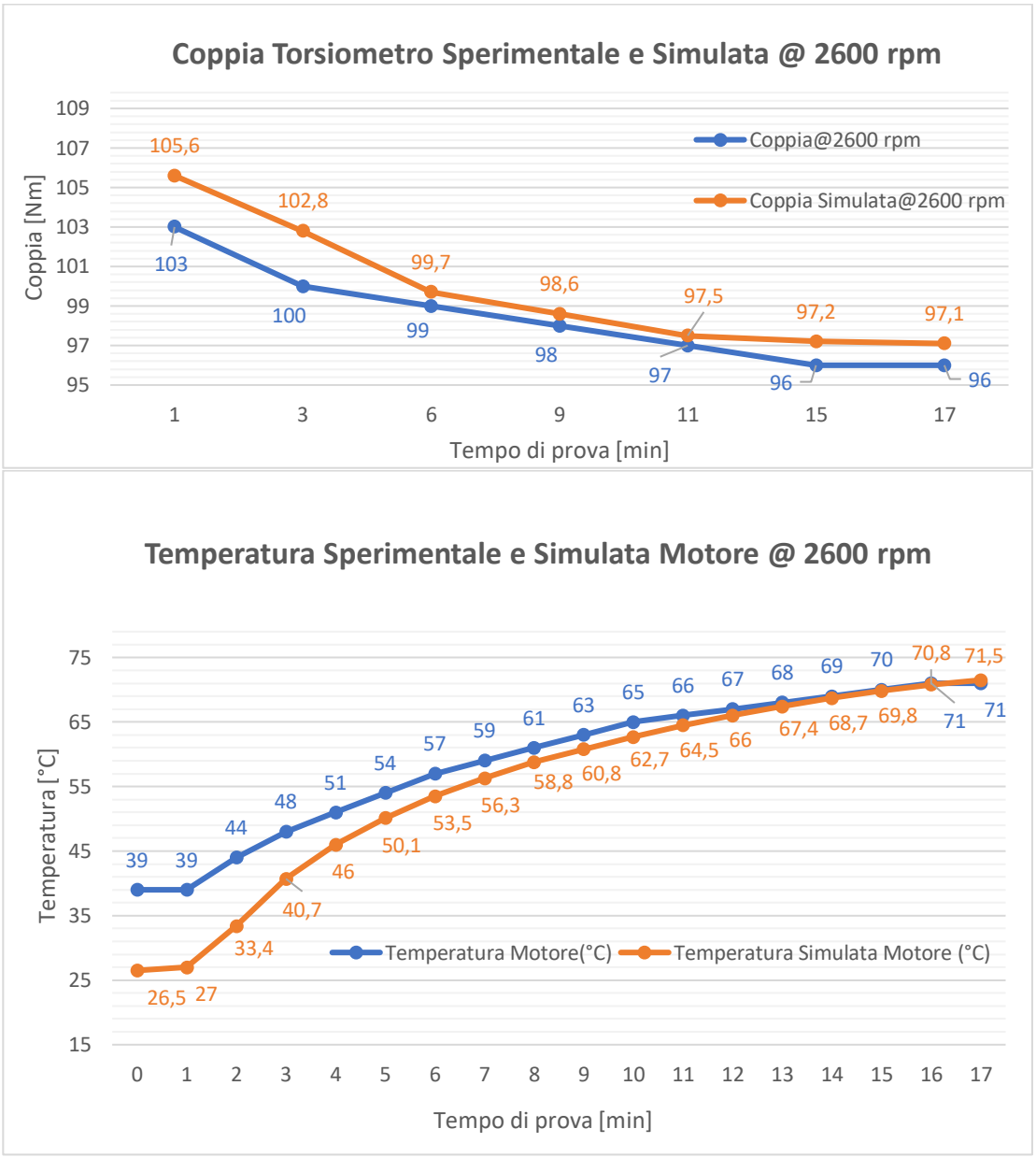


Figura 7.21: ***Confronto tra i risultati sperimentali e di simulazione ottenuti***

Si può notare come gli andamenti risultino piuttosto simili e tendenti a valori tra loro confrontabili. I risultati fanno presagire quindi ad una corretta modellazione del sistema.

Oltre alla simulazione necessaria per effettuare il confronto tra i dati sperimentali e quelli simulati, sono state effettuate altre due simulazioni per evidenziare la possibilità, anche nel modello, di poter variare la tipologia di controllo delle due macchine elettriche a banco. In questi casi però, i risultati ottenuti dalla simulazione non sono correlabili ad alcun risultato sperimentale, non avendo implementato nelle fasi test reali queste specifiche possibilità di controllo. I risultati ottenuti dalla simulazione possono però essere utilizzati per prevedere il comportamento del banco prova fisico nelle medesime condizioni operative in cui sono stati condotti questi test virtuali.

2. Simulazione con Motore in controllo Velocità e Freno in controllo Coppia

L'inversione dei comandi sulle due macchine elettriche consente di verificare il funzionamento del motore montato a banco anche in controllo velocità, vale a dire se questo è in grado di erogare la coppia necessaria a mantenere la linea meccanica del banco a quella determinata velocità di set: controllando in coppia la macchina a valle del banco e in velocità il motore, quest'ultimo eserciterà quindi sulla trasmissione la coppia motrice (o resistente, come si vedrà nella terza ed ultima simulazione) necessaria a mantenere il sistema alla velocità di set, a seconda del set di coppia resistente (o motrice, come si vedrà nella terza ed ultima simulazione) imposto a valle.

In questo caso, la durata della simulazione è pari a circa 1000 secondi.

Non si riportano in questo caso delle spiegazioni dettagliate in merito alla trasmissione dei messaggi CAN ed al loro salvataggio e replay, essendo sostanzialmente simili al caso precedente: la differenza sostanziale è solo nei messaggi che trasportano gli Input Set, essendo stati invertiti i comandi.

Si effettua invece nuovamente una analisi sul comportamento del modello di simulazione, riportando in Tabella 7.3 i set di comandi dati durante la simulazione e nelle Figure a seguire gli andamenti delle grandezze estratte dalla schermata dell'Interfaccia.

Tempo di Simulazione [s]	Controllo Motore in Velocità [rpm]	Controllo Freno in Coppia [Nm]	
0 (Start Sim)	0	0	Step 1
35	360	0	
50	720	0	
65	1080	0	
80	1440	0	
100	1800	0	
110	2160	0	
120	2520	0	
140	2880	0	
160	1440	0	Step 2
190	1440	-30	
205	1440	-60	
220	1440	-90	
240	1440	-105	
255	1800	-105	Step 3
275	2160	-105	
300	2520	-105	
330	2700	-105	
fino a 440	2700	-105	
440	2880	-105	
fino a 860	2880	-105	
860	2880	-90	Step 4
870	2880	-60	
890	2880	-30	
910	2880	0	
930	2160	0	
935	1080	0	
955	0	0	
fino a 990 (Stop Sim)	0	0	

Tabella 7.3: ***Input Set al modello di simulazione - 2° simulazione***

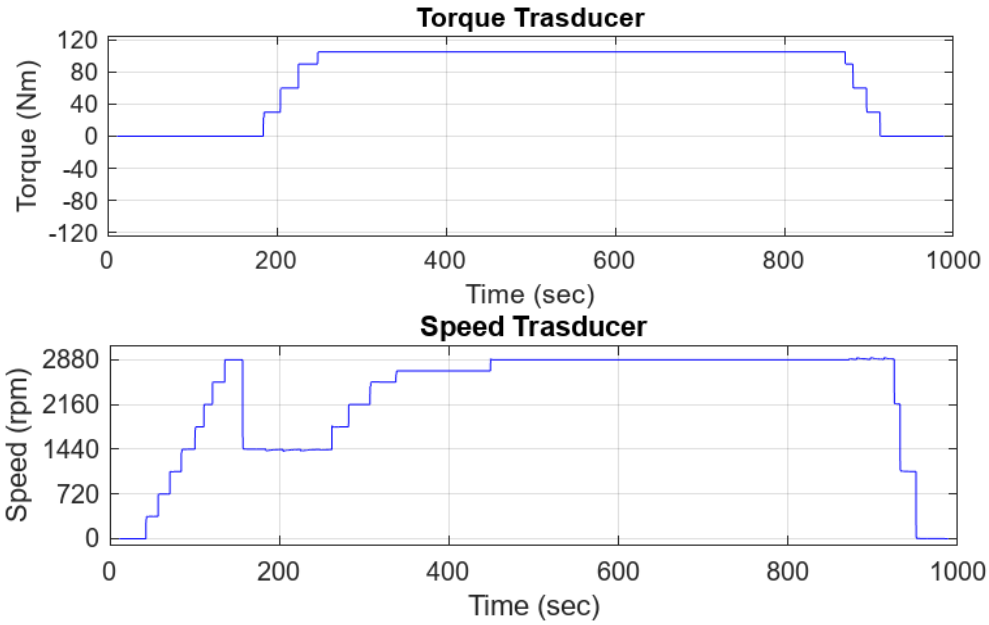


Figura 7.22: ***Coppia e Velocità lette dal modello del trasduttore - 2° simulazione***

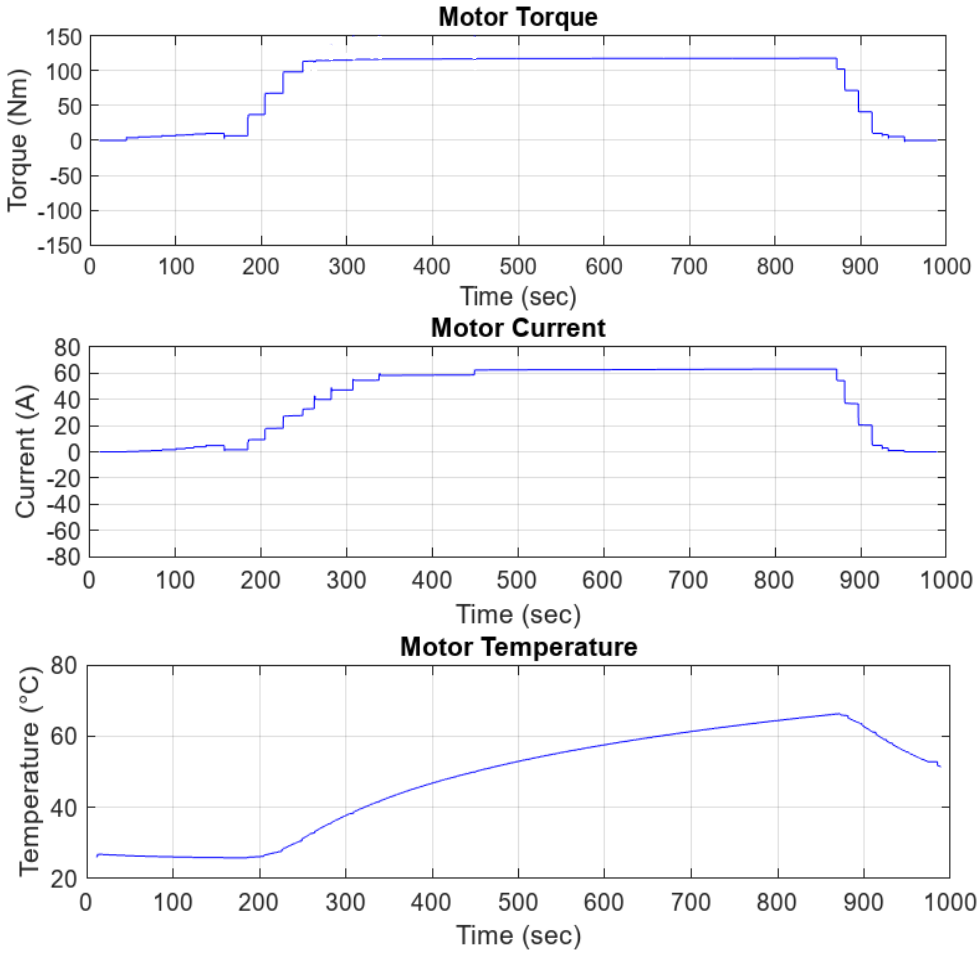


Figura 7.23: ***Coppia erogata e Corrente assorbita dal modello del motore e sua Temperatura - 2° simulazione***

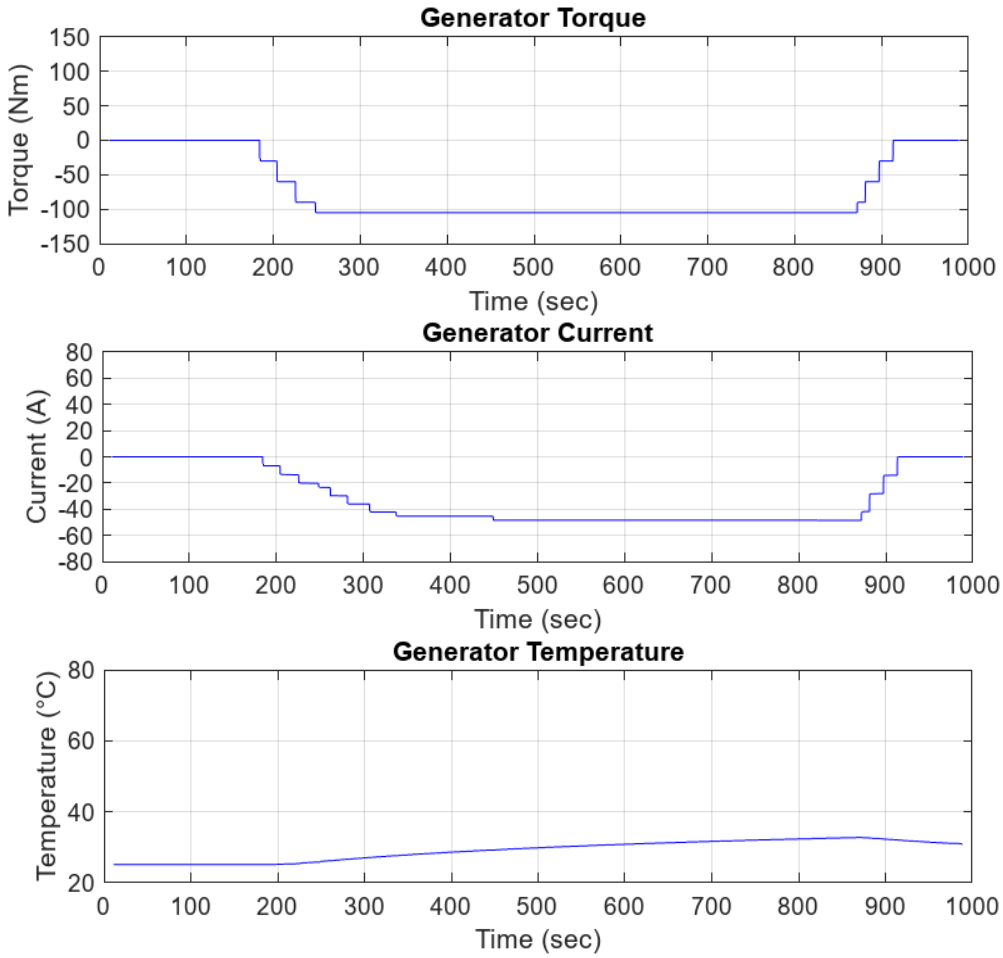


Figura 7.24: ***Coppia assorbita e Corrente erogata dal modello del freno e sua Temperatura - 2° simulazione***

Per una maggiore comprensione, anche in questo caso la simulazione è stata suddivisa in 4 step:

- Step 1 [0-160 s]: Si agisce sul controllo velocità del motore per portare l'intera linea meccanica ai vari regimi di velocità, mantenendo a zero invece il controllo coppia sul freno. Osservando l'andamento delle coppie e correnti: il freno del banco, non applicando alcuna coppia, ha una erogazione di corrente nulla. Il freno motore invece, dovendo trascinare alla velocità voluta l'intero sistema, deve in questa fase erogare una coppia pari a quella necessaria a mantenere in banco a quella specifica velocità, assorbendo quindi corrente. Sul modello del trasduttore si leggerà però una coppia nulla, in quanto posizionato a valle della trasmissione, che in questa fase assorbe l'intera potenza erogata dal motore.

- Step 2 [160-255 s]: Si porta inizialmente il sistema ad una velocità di 1440 rpm agendo sul controllo motore. Successivamente si procede per incrementi progressivi portando il freno ad assorbire una coppia massima pari a 105 Nm. Il motore quindi inizierà ad aumentare la coppia erogata per mantenere a velocità costante l'intero sistema, aumentando l'assorbimento di corrente dalle batterie; queste ultime dovranno infatti, per chiudere il ricircolo di potenza, erogare una corrente pari alla differenza tra la corrente media assorbita dal motore e quella generata dal freno, a causa delle perdite di conversione elettromeccaniche nelle due macchine elettriche e la potenza richiesta per tenere in rotazione la trasmissione meccanica. Si può notare infatti facilmente come la corrente recuperata tramite il generatore sia sempre inferiore a quella erogata sul motore.
  Il valore di coppia letta sul trasduttore è quindi positiva perché diretta da motore verso freno,

con valore pari alla differenza tra la coppia erogata e quella assorbita, a quella velocità, dalla linea meccanica del banco prova. Il valore di coppia letto sarà, in valore assoluto, pari alla coppia resistente esercitata dal freno.

- Step 3 [255-860 s]: mantenendo il freno in controllo alla coppia resistente precedentemente raggiunta, si aumenta la velocità del sistema agendo sul controllo motore fino a portare quest'ultimo ad una velocità di regime di 2880 rpm. Per ogni step crescente di velocità, coppia erogata e resistente si mantengono costanti, mentre variano le correnti assorbite ed erogate da motore e freno. Si mantiene quindi questo set di parametri per un tempo sufficientemente lungo, al fine di verificare in questa fase anche l'andamento della temperatura del blocco di simulazione del motore: si può notare, da questo punto di vista, che questa tende ad un valore asintotico pari a circa 70°C verso il termine della simulazione, a conferma di una corretta modellazione del circuito di raffreddamento. Non si riporta l'andamento della temperatura del blocco freno essendo valide le considerazioni già fatte in merito a ciò per la prima simulazione.

- Step 4 [860-990 s]: si procede a fermare il sistema portando prima a zero il controllo in coppia del freno e poi a ridurre progressivamente a zero anche il controllo in velocità del motore. Si può inoltre notare come la temperatura del blocco motore diminuisce con il suo progressivo inutilizzo.

Nella Tabella 7.4 si riportano quindi i principali risultati ottenuti ai vari step della simulazione:

Tempo [s]	Controllo Motore in Velocità [rpm]	Controllo Freno in Coppia [Nm]	Coppia Torsiometro [Nm]	Coppia Erogata Motore [Nm]	Corrente Assorbita Motore [A]	Temperatura Motore [°C]	Coppia Resistente Effettiva Freno [Nm]	Corrente Erogata Freno [A]
0 (Start Sim)	0	0	0	0	0	25	0	0
35	360	0	4,1	4,1	0,24	26,5	0	0
50	720	0	5,1	5,1	0,6	26,5	0	0
65	1080	0	5,6	5,6	0,99	26,5	0	0
80	1440	0	6,6	6,6	1,57	26,5	0	0
100	1800	0	7,6	7,6	2,27	26,5	0	0
110	2160	0	8,1	8,1	2,9	26,5	0	0
120	2520	0	9,1	9,1	3,8	26,5	0	0
140	2880	0	10,1	10,1	4,8	26,5	0	0
160	1440	0	6,6	6,6	1,57	26,5	0	0
190	1440	30	37,1	37,1	9,37	27	30	-7,05
205	1440	60	67,6	67,6	18,16	27,5	60	-13,8
220	1440	90	98,2	98,2	28,08	29	90	-20,4
240	1440	105	113,4	113,4	33,5	31	105	-23,8
255	1800	105	114,5	114,5	41,1	32	105	-30,2
275	2160	105	115,1	115,1	48,6	34,5	105	-36,7
300	2520	105	116,2	116,2	56,5	37	105	-43,3
330	2700	105	116,7	116,7	60,9	39	105	-46,7
fino a 440	2700	105	117	117	61,2	50	105	-46,5
440	2880	105	117,3	117,3	62	50	105	-47
fino a 860	2880	105	117,5	117,5	62,5	68	105	-47,5
860	2880	90	102,3	102,3	57,5	68	90	-44,2
870	2880	60	71,5	71,5	39,1	65	60	-29,9
890	2880	30	40,9	40,9	21,7	63	30	-15,1
910	2880	0	10,2	10,2	5,2	60,5	0	0
930	2160	0	8,1	8,1	3,2	59	0	0
935	1080	0	5,6	5,6	1,1	57	0	0
955	0	0	0	0	0	55	0	0
990 (Stop Sim)	0	0	0	0	0	50	0	0

Tabella 7.4: ***Risultati – 2° simulazione***

3. Simulazione con Motore in controllo Velocità e Freno in controllo Coppia, con Motore in frenata rigenerativa

L'inversione dei comandi sulle due macchine elettriche consente anche di verificare il funzionamento del motore montato a banco, sempre in controllo coppia, anche in condizioni di funzionamento da generatore: controllando in coppia la macchina a valle del banco e in velocità il motore, quest'ultimo andrà ad esercitare sulla trasmissione la coppia resistente necessaria a mantenere il sistema alla velocità di set, a seconda del set di coppia motrice imposto a valle. Durante questa prova sostanzialmente si invertono anche i funzionamenti delle due macchine, con il freno che agisce da motore erogando coppia ed il motore che agisce da freno assorbendo coppia.

Anche in questo caso, la durata della simulazione è pari a circa 1000 secondi.

Non si riportano anche in questo caso delle spiegazioni dettagliate in merito alla trasmissione dei messaggi CAN ed al loro salvataggio e replay, essendo sostanzialmente simili al caso precedente (cambia solo il segno della coppia entrante nel blocco di simulazione del freno).

Si effettua invece nuovamente una analisi sul comportamento del modello di simulazione, riportando in Tabella 7.5 i set di comandi dati durante la simulazione e nelle Figure a seguire gli andamenti delle grandezze estratte dalla schermata dell'Interfaccia.

Tempo di Simulazione [s]	Controllo Motore in Velocità [rpm]	Controllo Freno in Coppia [Nm]	
0 (Start Sim)	0	0	Step 1
35	360	0	
50	720	0	
60	1080	0	
75	1440	0	
85	1800	0	
110	2160	0	
125	2520	0	
135	2880	0	
170	1440	0	Step 2
190	1440	30	
205	1440	60	
225	1440	90	
250	1440	120	
280	1800	120	Step 3
305	2160	120	
330	2520	120	
345	2700	120	
fino a 480	2700	120	
480	2880	120	
fino a 860	2880	120	
860	2880	80	Step 4
870	2880	40	
885	2880	0	
895	1800	0	
910	900	0	
915	0	0	
fino a 990 (Stop Sim)	0	0	

Tabella 7.5: ***Input Set al modello di simulazione - 3° simulazione***

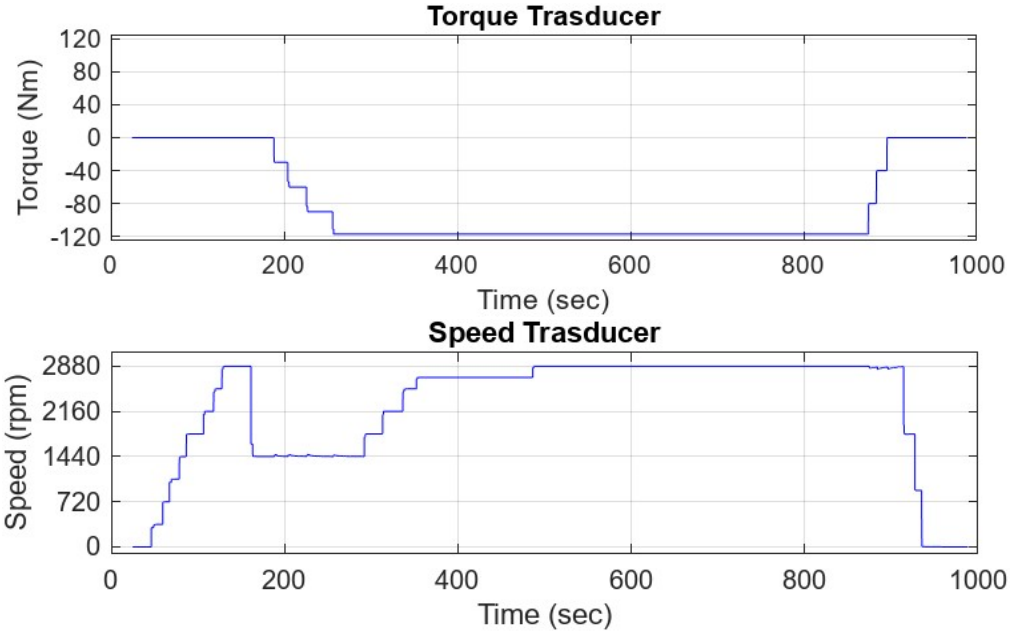


Figura 7.25: ***Coppia e Velocità lette dal modello del trasduttore - 3° simulazione***

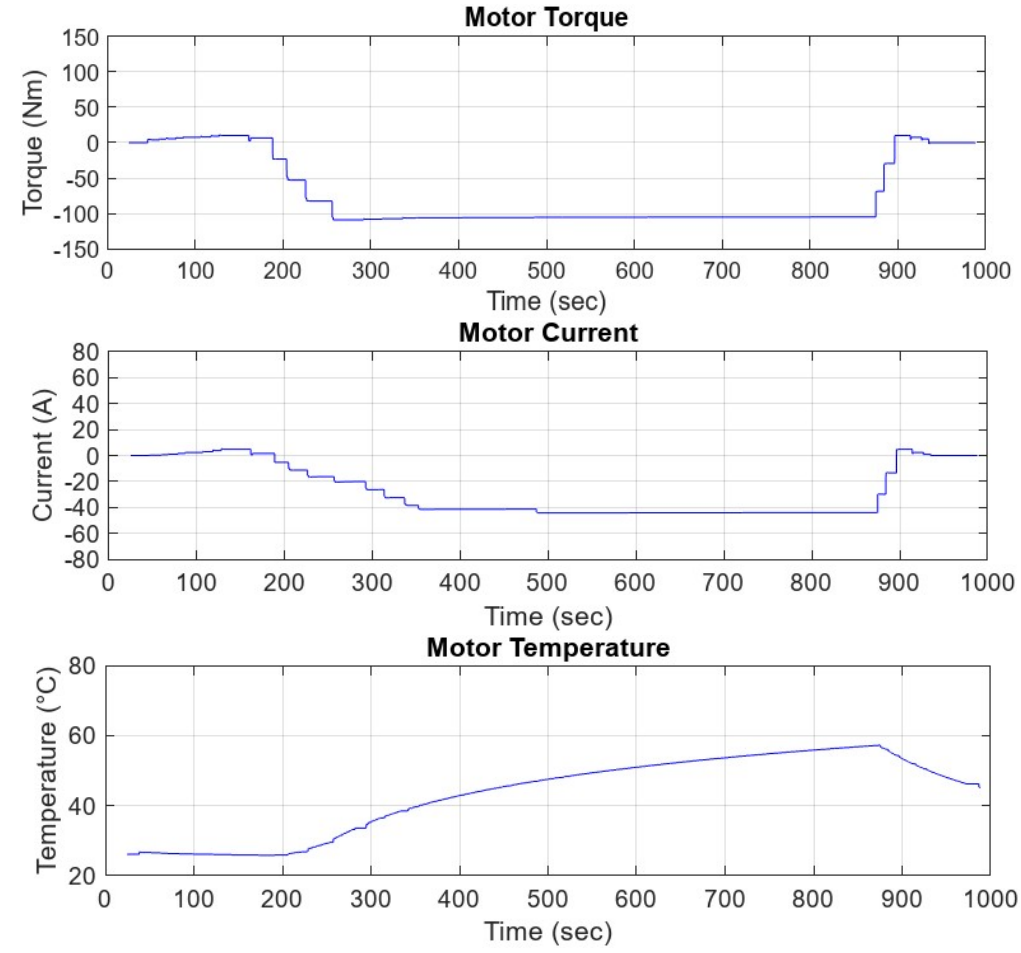


Figura 7.26: ***Coppia assorbita e Corrente erogata dal modello del motore e sua Temperatura - 3° simulazione***

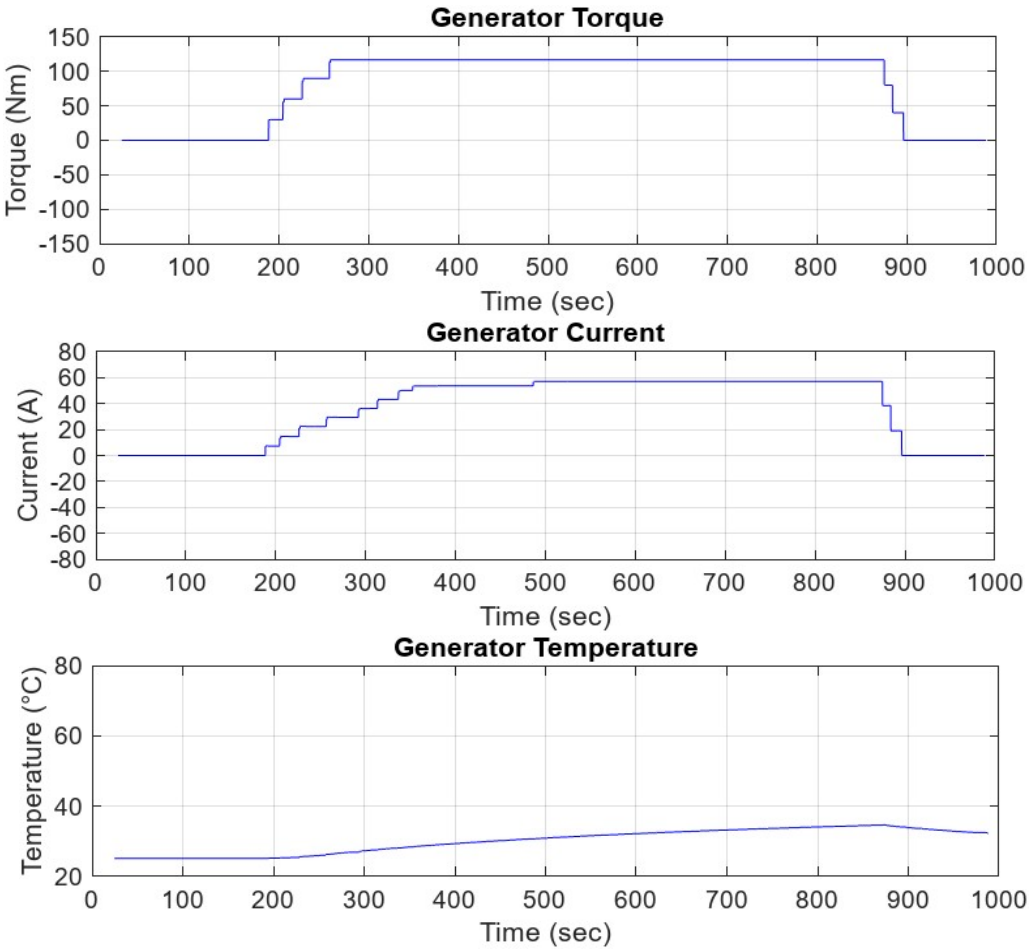


Figura 7.27: ***Coppia erogata e Corrente assorbita dal modello del freno e sua Temperatura - 3° simulazione***

Anche in questo caso, si è suddiviso l'andamento della simulazione in 4 step:

- Step 1 [0-170 s]: Si agisce sul controllo velocità del motore per portare l'intera linea meccanica ai vari regimi di velocità, mantenendo a zero invece il controllo coppia sul freno. Osservando l'andamento delle coppie e correnti: il freno del banco, non applicando alcuna coppia, ha una erogazione di corrente nulla. Il freno motore invece, dovendo trascinare alla velocità voluta l'intero sistema, deve in questa fase erogare una coppia pari a quella necessaria a mantenere in banco a quella specifica velocità, assorbendo quindi corrente. Sul modello del trasduttore si leggerà però una coppia nulla, in quanto posizionato a valle della trasmissione, che in questa fase assorbe l'intera potenza erogata dal motore.

- Step 2 [170-280 s]: Si porta inizialmente il sistema ad una velocità di 1440 rpm agendo sul motore. Successivamente si procede per incrementi successivi portando il freno ad erogare una coppia massima pari a 120 Nm. Il freno quindi inizierà ad assorbire corrente, con il motore che per mantenere a velocità costante il sistema applicherà sullo stesso una coppia resistente, lavorando quindi come generatore ed erogando corrente verso le batterie; queste ultime dovranno infatti, per chiudere il ricircolo di potenza, erogare una corrente pari alla differenza tra la corrente media assorbita dal freno e quella della frenata rigenerativa del motore in prova, a causa delle perdite di conversione elettromeccaniche nelle due macchine elettriche e la potenza richiesta per tenere in rotazione la trasmissione meccanica. Si può notare infatti

facilmente come la corrente recuperata tramite il motore sia sempre inferiore a quella erogata sul freno.
Il valore di coppia letta sul trasduttore è quindi negativa perché diretta da freno del banco verso motore in test, con valore pari alla coppia erogata dal freno.

- Step 3 [280-860]: mantenendo il freno in controllo alla coppia erogata raggiunta in precedenza, si aumenta la velocità del sistema agendo sul controllo motore fino a portare quest'ultimo ad una velocità di regime di 2880 rpm. Per ogni step crescente di velocità, coppia resistente e corrente erogata dal motore (verso le batterie) aumentano per evitare accelerazioni dell'intero sistema. Si mantiene quindi questo set di parametri per un tempo sufficientemente lungo, al fine di verificare in questa fase anche l'andamento della temperatura del blocco di simulazione del motore: si può notare, da questo punto di vista, che questa tende ad un valore asintotico pari a circa 60°C verso il termine della simulazione, a conferma di una corretta modellazione del circuito di raffreddamento. Non si riporta l'andamento della temperatura del blocco freno essendo valide le considerazioni già fatte in merito a ciò per la prima simulazione.

- Step 4 [850-980 s]: si procede a fermare il sistema portando prima a zero il controllo in coppia del freno e poi a ridurre progressivamente a zero anche la velocità del motore.

Nella Tabella 7.6 a seguire si riportano quindi i principali risultati ottenuti:

Tempo [s]	Controllo Motore in Velocità [rpm]	Controllo Freno in Coppia [Nm]	Coppia Torsiometro [Nm]	Coppia Resistente Motore [Nm]	Corrente Erogata Motore [A]	Temperatura Motore [°C]	Coppia Erogata Effettiva Freno [Nm]	Corrente Assorbita Freno [A]
0 (Start Sim)	0	0	0	0	0	25	0	0
35	360	0	0	4,1	0,24	26,5	0	0
50	720	0	0	5,1	0,6	26,5	0	0
60	1080	0	0	5,6	0,99	26,5	0	0
75	1440	0	0	6,6	1,57	26,5	0	0
85	1800	0	0	7,6	2,27	26,5	0	0
110	2160	0	0	8,1	2,9	26,5	0	0
125	2520	0	0	9,1	3,8	26,5	0	0
135	2880	0	0	10,1	4,8	26,5	0	0
170	1440	0	0	6,6	1,57	26,5	0	0
190	1440	-30	-30	-23,1	-5,2	26,5	-30	7,2
205	1440	-60	-60	-52,6	-11,2	27	-60	14,6
225	1440	-90	-90	-82,1	-16,3	28	-90	22,3
250	1440	-120	-120	-111,5	-20,5	29	-120	30,3
280	1800	-120	-120	-110,5	-27	31	-120	37,6
305	2160	-120	-120	-110	-33,5	34	-120	45
330	2520	-120	-120	-108,9	-40	37	-120	52,4
345	2700	-120	-120	-108,3	-43,2	40	-120	56,4
fino a 480	2700	-120	-120	-108	-43	47	-120	56,8
480	2880	-120	-120	-107,6	-46,7	47	-120	60,9
fino a 860	2880	-120	-120	-107	-47,6	57	-120	62
860	2880	-80	-80	-68,2	-32,1	57	-80	41,5
870	2880	-40	-40	-29,2	-14,4	56	-40	20,5
885	2880	0	0	10,2	5,3	55	0	0
895	1800	0	0	7,6	2,5	54	0	0
910	900	0	0	5,3	0,87	53	0	0
915	0	0	0	0	0	52	0	0
990 (Stop Sim)	0	0	0	0	0	47	0	0

Tabella 7.6: ***Risultati – 3° simulazione***

Tra la simulazione 1 e le simulazioni 2 e 3 si può notare un comportamento molto differente del blocco motore:

- Nella prima simulazione questo veniva controllato in coppia e l'innalzamento di temperatura ne provocava una diminuzione di coppia erogata, con il freno che, controllato in velocità, adattava la coppia sul suo albero per mantenere il set di velocità imposto su tutta la linea meccanica, situazione confermata anche dalle evidenze sperimentali.
- Nella seconda e terza simulazione invece è il freno che impone la coppia al sistema, ed essendo poco soggetto a variazioni di temperatura in questo range di valori di utilizzo (lavorando su valori di potenza ben al di sotto delle sue massime prestazioni) allora questo è in grado di erogare sempre la coppia richiesta. Il motore invece, essendo controllato qui in velocità, dovrà sempre fornire al sistema la coppia necessaria affinché l'intero sistema non subisca accelerazioni: di conseguenza nonostante il suo incremento di temperatura (che lo porterebbe naturalmente ad abbassare la corrente circolante, che però comporterebbe un abbassamento di coppia) la corrente da esso assorbita/erogata in questo caso tende ad aumentare, così da tenere pressoché costante la coppia resistente/motrice sul suo albero. Non avendo eseguito questa strategia di controllo sul banco prova fisico e non avendo monitorato l'effettiva temperatura del freno, non è possibile stabilire a priori se questa situazione si replichi anche nella realtà, aspetto che può essere monitorato in utilizzi futuri dello stesso.

Nonostante ciò, sulla base dei risultati ottenuti nella prima delle simulazioni svolte e considerando anche quanto osservato sperimentalmente sul banco prova fisico i cui risultati sono riportati nel Capitolo 4, si può affermare che il modello di simulazione creato risponde correttamente alle diverse modalità di controllo.

Anche in merito alla GUI e quindi alla trasmissione dei messaggi CAN realizzata il riscontro è positivo avendo osservato in tutte le casistiche di simulazione un corretto comportamento delle funzionalità predisposte.

I risultati ottenuti dal modello nei tre scenari di possibile utilizzo del banco prova consentono inoltre di eseguire una prima stima sul rendimento dell'intero sistema e sul ricircolo di potenza. Considerando infatti le condizioni di regime raggiunte durante le 3 simulazioni svolte, una tensione disponibile massima di 640V ed una tensione del pacco batterie, al termine delle simulazioni, pari in tutti i casi a 600 V (si è quindi considerato nel calcolo del rendimento una tensione mediata):

Simulazione n°	Velocità a regime [rpm]	P_media Albero Motore [kW]	P_media Albero Freno [kW]	P_diss trasmissione [kW]	Corrente media Motore [A]	Corrente media Freno [A]	Delta corrente medio [A]	P_media batterie [kW]	Rendimento medio [-]
1	2600	30,5	-27,9	2,7	55,5	-43,3	12,2	7,6	0,80
2	2880	35,4	-31,7	3,7	62,0	-47	14,9	9,2	0,79
3	2880	-32,5	36,2	3,8	-47,2	61,5	14,3	8,9	0,80

Tabella 7.7: ***Stima rendimenti del banco prova per le 3 simulazioni svolte***

In cui il rendimento è stato calcolato, in ogni casistica, come:

$$\eta = \frac{P_{utile}}{P_{spesa}} = \frac{P_{utile}}{P_{utile} + P_{batterie}} \tag{7.1}$$

Con P_{utile} pari alla potenza motrice della linea di trasmissione meccanica del banco.

Sviluppi futuri per la GUI creata

Dopo aver eseguito i test SiL dell'interfaccia, attraverso i quali è stata anche effettuata la validazione del modello di simulazione del banco, le attività future da realizzare prevedono il collegamento della stessa al Software di Controllo del banco prova fisico.

Per far ciò sarà sicuramente necessario effettuare delle modifiche al codice sottostante sviluppato: infatti piuttosto che essere collegato al modello Simulink di simulazione del banco e ad un canale di comunicazione CAN virtuale, questo dovrà essere collegato al modello Simulink del software di comunicazione con l'HCU, con il CAN-Bus che dovrà essere quello effettivo del banco realizzato, secondo lo schema già riportato in Figura 6.1, e di seguito riproposta.

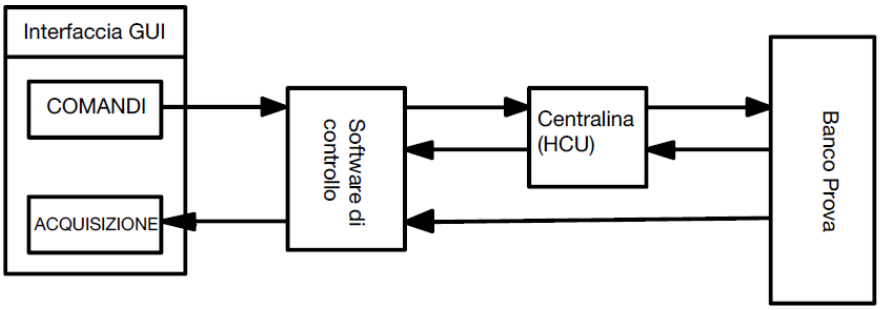


Figura 7.28: ***Schema di collegamento della GUI con il software e hardware del banco prova***

Conclusioni

L'attività di tesi svolta prevedeva di caratterizzare le prestazioni di un banco prova HiL a ricircolo di potenza elettrica per testare la parte elettrica ed elettronica di powertrain ibridi/elettrici per veicoli non stradali NRMM. Questa attività è stata eseguita sperimentalmente durante il primo utilizzo effettivo del banco stesso per il testing di un motore elettrico PMSM destinato all'impiego in una trattrice agricola full-electric in fase prototipale. Durante l'attività sperimentale è stato possibile, in primis, ricavare le caratteristiche intrinseche della linea di trasmissione meccanica del banco, costituita in questa configurazione da due riduttori epicicloidali identici ma posti in contrapposizione (ovvero uno agente da riduttore di giri ed uno invece da moltiplicatore di giri), interposti tra due giunti cardanici necessari per risolvere i problemi di disallineamento tra la parte fissa del banco e la parte mobile dello stesso, sulla quale vengono montati i componenti hardware da testare: la necessità di interporre i giunti sopracitati tra i due riduttori nasce dal fatto di dover contenere la velocità angolare dei giunti stessi entro i propri limiti strutturali, nel mentre però che il resto della banco ruota a velocità ben più elevate, e pari alle velocità operative dei componenti testati. Da questo punto di vista la modularità del banco consente di poter anche facilmente variare la configurazione della trasmissione, utilizzando altri riduttori a differente rapporto di trasmissione rispetto a quelli attualmente utilizzati, oppure aggiungendone degli altri a monte del motore testato: in entrambi i casi comunque il banco sarà utilizzabile per testate motori che lavorano a velocità ben più elevate della velocità limite del banco, che nella sua configurazione di base è di circa 3000 rpm.
Successivamente il banco è stato utilizzato per ricavare le principali caratteristiche dell'hardware sottoposto a test, quali: la caratteristica BEMF, la curva caratteristiche del motore elettrico e infine la variazione delle caratteristiche dell'intero sistema a seguito di un suo utilizzo prolungato.

Per i futuri utilizzi del banco prova e per rendere il comando e l'acquisizione dati dallo stesso più userfriendly per l'utente che esegue le prove e che non conosce il software di controllo della centralina del banco, contestualmente è stata sviluppata una GUI (Graphic User Interface) da collegare al software (in modo da mascherare quest'ultimo all'utente). Dato che tutti i sistemi montati a banco si basano su controlli eseguiti utilizzando il protocollo di comunicazione CAN-Bus, anche la GUI è stata interamente realizzata utilizzando il protocollo sopracitato, sfruttando le funzionalità del tool Vehicle Network Toolbox (VNT) della Mathworks. La schermata e il codice sottostante dell'Interfaccia è stata sviluppata all'interno dl Matlab App Designer, all'interno del quale possono essere utilizzate tutte le funzionalità del VNT e dal quale a GUI ultimata e testata può essere ricavato un file eseguibile, utilizzabile a questo punto su qualunque PC, senza la necessità di integrare nelle simulazioni Matlab.

Per utilizzare in completa sicurezza il software sviluppato sul sistema fisico, è però necessario preliminarmente verificarne il corretto funzionamento all'interno di un ambiente di simulazione interamente virtuale, senza alcun interfacciamento con dispositivi hardware: questa tipologia di test, nel mondo del software testing, prende il nome di test SiL. Per eseguire questa tipologia di prova per la GUI sviluppata è stato allora sviluppato un modello Simulink di simulazione del comportamento del banco prova, collegato all'Interfaccia Grafica mediante un canale di comunicazione CAN virtuale fornito dal VNT stesso.

Il modello di simulazione è stato creato a partire dai dati di targa dei diversi componenti montati sul banco prova fisico, in modo da simulare in maniera più veritiera possibile (seppur con alcune semplificazioni) ciò che accade sul sistema reale.

I test SiL eseguiti hanno quindi consentito contemporaneamente di:

- Verificare il corretto funzionamento di tutte le funzionalità CAN predisposte sulla GUI;
- Verificare, tramite gli output di simulazione, la risposta del modello rispetto al sistema reale per la sua validazione in condizioni operative simili a quelle delle prove sperimentali eseguite a banco. E' stata infatti eseguita una analisi comparativa tra i risultati di simulazione e quelli sperimentali nelle medesime condizioni di prova.
  Questo ha consentito anche di effettuare una stima sul ricircolo di potenza elettrica che avviene anche nella realtà mediante il calcolo dei rendimenti nei diversi scenari di simulazione eseguiti.

Le prove eseguite hanno evidenziato che l'interfaccia comunica correttamente, mediante protocollo CAN, con il modello creato e quest'ultimo risponde coerentemente a quanto evidenziato sperimentalmente.

Gli obiettivi prefissati sono quindi stati tutti raggiunti. L'obiettivo futuro, per completare il lavoro avviato, è quindi di completare il codice sviluppato per la GUI per renderlo compatibile con il software di gestione dell'HCU e con le linee CAN-Bus fisiche realizzate sul banco.

Riferimenti bibliografici

Capitolo 1

[1.1] E. Spessa, *Controllo delle emissioni di inquinanti: slides Formazione NO_x*, 2021

[1.2] E. Spessa, *Controllo delle emissioni di inquinanti: slides Formazione CO*, 2021

[1.3] E. Spessa, *Controllo delle emissioni di inquinanti: slides Formazione HC*, 2021

[1.4] E. Spessa, *Controllo delle emissioni di inquinanti: slides Formazione PM*, 2021

[1.5] Parlamento Europeo: *REGOLAMENTO (UE) 2016/1628 DEL PARLAMENTO EUROPEO E DEL CONSIGLIO,* 2016

[1.6] F. Mocera, *Comparative Analysis of Hybrid Electric Architectures for Specialized Agricultural Tractors, 2022*

[1.7] E. Spessa, *Controllo delle emissioni di inquinanti: slides Trattamento delle emissioni allo scarico dei motori CI: DOC e DPF*, 2021

[1.9] Quattroruote.it, *Componenti auto: Filtro antiparticolato*

[1.10] W. Addy Majewski, *Diesel Oxidation Catalyst*, 2021

[1.11] E. Spessa, *Controllo delle emissioni di inquinanti: slides Trattamento delle emissioni allo scarico dei motori CI: catalizzatori riducenti per NOx*, 2021

[1.12] F. Millo, *Propulsori termici: slides Motori diesel: aftertreatment*, 2021

[1.13] F. Millo, *Propulsori termici: slides La sovralimentazione: complementi*, 2021

[1.14] A. Somà, *Trends and Hybridization Factor for Heavy-Duty Working Vehicles*, 2017

[1.15] *lifeatena.com*

[1.16] *omnitrattore.it/news/626993/antonio-carraro-trattori-elettrici-ibridi/*

Capitolo 2

[2.1] E. Spessa, *Controllo delle emissioni di inquinanti: slides Hybrid Electric Vehicles*, 2021

[2.2] A. Somà, *Trends and Hybridization Factor for Heavy-Duty Working Vehicles*, 2017

[2.3] S. Vaschetto, *Architetture veicoli elettrici ed ibridi*, 2020

[2.4] L. Rolando, *Powertrain electrification*, 2019

[2.5] *e-nsight.com/2019/10/11/librido-misto-o-complesso/*

[2.6] A. Somà, N. Bosso, A. Merlo, *Electrohydraulic hybrid lifting vehicle. Patent, WO2011128772,*2011

[2.7] A. Somà, *Trends and Hybridization Factor for Heavy-Duty Working Vehicles*, 2017

[2.8] H. Wang, L. Liu, G. Zheng, X. Liu, et al., *Study of two-motor hybrid bulldozer*, 2014

[2.9] J. Flint, *A different kind of hybrid from John Deere*, 2013

[2.10] E. Anderson, *John Deere 644K Hybrid Drivetrain Overview, Performance, & Developmental Analysis*, 2013

[2.11] A. Somà, F. Bruzzese, E. Viglietti, *Hybridization factor and performances of hybrid electric telescopic heavy vehicles*, 2015

[2.12] F. Mocera, *Study of hybrid electric architectures for industrial vehicle applications using Hardware In the Loop techniques*, 2019

Capitolo 3

[3.1] *rw-italia.it*

[3.2] *wikipedia.org/wiki/Hardware-in-the-loop*

[3.3] F. Mocera, *Study of hybrid electric architectures for industrial vehicle applications using Hardware In the Loop techniques*, 2019

Capitolo 5

[5.1] *mst-tutorial.it/rete-di-comunicazione-can*

[5.2] *Bosch, CAN specification*

[5.3] *"Reti E Sistemi Per L'automazione", Bosch's Controller Area Network, Antonello Galanti*

[5.4] *peak-system.com*

Capitolo 6

[6.1] *mathworks.com/help/vnt*

[6.2] *VNT documentation, vnt_ug.pdf*

Capitolo 7

[7.1] *mathworks.com/help/vnt*

[7.2] *VNT documentation, vnt_ug.pdf*

Appendice ed allegati

Script della GUI sviluppata per test SiL

```
classdef GUIsimulinkbancoDefinitivo < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                        matlab.ui.Figure
        ChannelConfigurationMenu        matlab.ui.container.Menu
        SelectCANChannel                matlab.ui.container.Menu
        OutputsPlotPanel                matlab.ui.container.Panel
        InputsPanel                     matlab.ui.container.Panel
        plottingPanel                   matlab.ui.container.Panel
        plotTimeWindowDropDown          matlab.ui.control.DropDown
        LabelDropDown                   matlab.ui.control.Label
        ChannelReplayControlPanel       matlab.ui.container.Panel
        StartReplayLabel                matlab.ui.control.Label
        LoadloggedDataLabel             matlab.ui.control.Label
        loadCANLogDataButton            matlab.ui.control.Button
        canReplayStartStopButton        matlab.ui.control.StateButton
        ChannelLogControls              matlab.ui.container.Panel
        SaveloggedDataLabel             matlab.ui.control.Label
        StartLoggingLabel               matlab.ui.control.Label
        saveLoggedDataButton            matlab.ui.control.Button
        canLoggingStartStopButton       matlab.ui.control.StateButton
        SimulationControlPanel          matlab.ui.container.Panel
        StartSimLabel                   matlab.ui.control.Label
        SimStartStopButton              matlab.ui.control.StateButton
        GeneratorControlPanel           matlab.ui.container.Panel
        NumericEditField_Generator_TorqueControl  matlab.ui.control.NumericEditField
        LabelNumericEditField2_6        matlab.ui.control.Label
        NumericEditField_Generator_SpeedControl  matlab.ui.control.NumericEditField
        LabelNumericEditField2_5        matlab.ui.control.Label
        Slider_Generator_TorqueControl  matlab.ui.control.Slider
        Slider_Generator_SpeedControl   matlab.ui.control.Slider
        MotorControlPanel               matlab.ui.container.Panel
        NumericEditField_Motor_TorqueControl  matlab.ui.control.NumericEditField
        LabelNumericEditField2_4        matlab.ui.control.Label
        NumericEditField_Motor_SpeedControl  matlab.ui.control.NumericEditField
        LabelNumericEditField2          matlab.ui.control.Label
        Slider_Motor_TorqueControl      matlab.ui.control.Slider
        Slider_Motor_SpeedControl       matlab.ui.control.Slider
        TemperaturaFrenoPlot            matlab.ui.control.UIAxes
        TemperaturaMotorePlot           matlab.ui.control.UIAxes
        CorrenteFrenoPlot               matlab.ui.control.UIAxes
        CoppiaFrenoPlot                 matlab.ui.control.UIAxes
        CorrenteMotorePlot              matlab.ui.control.UIAxes
        CoppiaMotorePlot                matlab.ui.control.UIAxes
        VelocitaTorsiometroPlot         matlab.ui.control.UIAxes
        CoppiaTorsiometroPlot           matlab.ui.control.UIAxes
    end

    properties (Access = private)
        receiveCANmsgsTimer                                     % Timer object for receiving CAN data
        CoppiaTorsiometroFb                                     % CoppiaTorsiometro feedback
        CoppiaTorsiometroLine                                   % CoppiaTorsiometro plot line handle
        VelocitaTorsiometroFb                                   % VelocitaTorsiometro feedback
        VelocitaTorsiometroLine                                 % VelocitaTorsiometro plot line handle
        CoppiaMotoreFb                                          % CoppiaMotore feedback
        CoppiaMotoreLine                                        % CoppiaMotore plot line handle
        CorrenteMotoreFb                                        % CorrenteMotore feedback
        CorrenteMotoreLine                                      % CorrenteMotore plot line handle
        TemperaturaMotoreFb                                     % TemperaturaMotore feedback
        TemperaturaMotoreLine                                   % TemperaturaMotore plot line handle
        CoppiaFrenoFb                                           % CoppiaFreno feedback
        CoppiaFrenoLine                                         % CoppiaFreno plot line handle
        CorrenteFrenoFb                                         % CorrenteFreno feedback
        CorrenteFrenoLine                                       % CorrenteFreno plot line handle
        TemperaturaFrenoFb                                      % TemperaturaFreno feedback
        TemperaturaFrenoLine                                    % TemperaturaFreno plot line handle
        mdl                                                     % Test harness model
        plotTimeWidth                                           % Width of plots in seconds
        canChannelInfo                                          % Table of info about available CAN channels
        canChannelObj                                           % CAN channel object
        numConstructors                 % Number of possible CAN channel object constructor strings - depends on connected hardware
        availableCANChannelsForDisplay                  % String array of available CAN channels to populate CAN Channel list box
        canChannelConstructorSelected               % Selected CAN channel constructor from list of available CAN Channel Constructors
        canChannelDeviceSelected       % Selected CAN channel Device and DeviceMenu strings from list of available CAN channel Devices
        canChannelIndex                                 % index of selected CAN channel from list of available CAN channels
        canLogChannelObj                                        % CAN channel object for logging CAN data
        canLogMsgInID100Buffer                                  % Variable to hold Input CAN message data with specific ID
        canLogMsgInID200Buffer                                  % Variable to hold Input CAN message data with specific ID
        canLogMsgOutID300Buffer                                 % Variable to hold Output CAN message data with specific ID
        canLogMsgOutID400Buffer                                 % Variable to hold Output CAN message data with specific ID
        canLogMsgOutID500Buffer                                 % Variable to hold Output CAN message data with specific ID
        canLogMsgInBuffer                                       % Variable to hold all Input CAN message data
        MotorMessageInput                                       % CAN message container for Motor
        GeneratorMessageInput                                   % CAN message container for Generator
        General_Factor_CANmessage_Coppia_Corrente_Temperatura=0.1;      %CAN data factor for Torque,Current and Temperature signal
    end
```

```
methods (Access = private)

    function setupInitialVals(app)
         % set initial target slider and display values
        app.Slider_Motor_SpeedControl.Value = 0;
        app.Slider_Motor_TorqueControl.Value = 0;
        app.NumericEditField_Motor_SpeedControl.Value = app.Slider_Motor_SpeedControl.Value;
        app.NumericEditField_Motor_TorqueControl.Value = app.Slider_Motor_TorqueControl.Value;
         app.Slider_Generator_SpeedControl.Value = 0;
        app.Slider_Generator_TorqueControl.Value = 0;
        app.NumericEditField_Generator_SpeedControl.Value = app.Slider_Generator_SpeedControl.Value;
        app.NumericEditField_Generator_TorqueControl.Value = app.Slider_Generator_TorqueControl.Value;
    end

    function setUpPlotGridAndHoldStatus(app)

        grid(app.CoppiaTorsiometroPlot, 'on');
        hold(app.CoppiaTorsiometroPlot,'on');

        grid(app.VelocitaTorsiometroPlot, 'on');
        hold(app.VelocitaTorsiometroPlot,'on');

        grid(app.CoppiaMotorePlot, 'on');
        hold(app.CoppiaMotorePlot,'on');

        grid(app.CorrenteMotorePlot, 'on');
        hold(app.CorrenteMotorePlot,'on');

        grid(app.TemperaturaMotorePlot, 'on');
        hold(app.TemperaturaMotorePlot,'on');

        grid(app.CoppiaFrenoPlot, 'on');
        hold(app.CoppiaFrenoPlot,'on');

        grid(app.CorrenteFrenoPlot, 'on');
        hold(app.CorrenteFrenoPlot,'on');

        grid(app.TemperaturaFrenoPlot, 'on');
        hold(app.TemperaturaFrenoPlot,'on');

        % initialize handles to  plot lines
        % set handle visibility to off so calls to cla() don't delete these handles
         app.CoppiaTorsiometroLine = plot(app.CoppiaTorsiometroPlot, nan, nan,'b', 'HandleVisibility', 'off');% initialize handle   to plot
      line
        app.VelocitaTorsiometroLine = plot(app.VelocitaTorsiometroPlot, nan, nan,'b', 'HandleVisibility', 'off');     % initialize handle to
      plot line
        app.CoppiaMotoreLine = plot(app.CoppiaMotorePlot, nan, nan,'b', 'HandleVisibility', 'off');     % initialize handle to plot line
        app.CorrenteMotoreLine = plot(app.CorrenteMotorePlot, nan, nan,'b', 'HandleVisibility', 'off');     % initialize handle to plot line
        app.TemperaturaMotoreLine = plot(app.TemperaturaMotorePlot, nan, nan,'b', 'HandleVisibility', 'off');   % initialize  handle to plot
      line
        app.CoppiaFrenoLine = plot(app.CoppiaFrenoPlot, nan, nan,'b', 'HandleVisibility', 'off');     % initialize handle to plot line
        app.CorrenteFrenoLine = plot(app.CorrenteFrenoPlot, nan, nan,'b', 'HandleVisibility', 'off');     % initialize handle to plot line
        app.TemperaturaFrenoLine = plot(app.TemperaturaFrenoPlot, nan, nan,'b', 'HandleVisibility', 'off');     % initialize handle to plot
      line
    end

    function setAxesLimitsAndTitles(app, XLim0,timeWindow)
        % set X axis limits
        app.CoppiaTorsiometroPlot.XLim = [XLim0, XLim0+timeWindow];
        app.VelocitaTorsiometroPlot.XLim = [XLim0, XLim0+timeWindow];

        app.CoppiaMotorePlot.XLim = [XLim0, XLim0+timeWindow];
        app.CorrenteMotorePlot.XLim = [XLim0, XLim0+timeWindow];
        app.TemperaturaMotorePlot.XLim = [XLim0, XLim0+timeWindow];

        app.CoppiaFrenoPlot.XLim = [XLim0, XLim0+timeWindow];
        app.CorrenteFrenoPlot.XLim = [XLim0, XLim0+timeWindow];
        app.TemperaturaFrenoPlot.XLim = [XLim0, XLim0+timeWindow];

        % update X axis ticks
        XTick_VelocitaTorsiometro = app.VelocitaTorsiometroPlot.XTick;
        XTick_VelocitaTorsiometro = [XTick_VelocitaTorsiometro(1):timeWindow/10:XTick_VelocitaTorsiometro(end)];
        % the tick mode is set to "auto" and it always displays 10 ticks for whatever time range I pick.  Looks like I don't need
        % to adjust the tics if auto mode always displays 10 ticks.

         % set Y axis limits
        app.VelocitaTorsiometroPlot.YLim = [-100, 4100];

         % set Y axis ticks
        app.VelocitaTorsiometroPlot.YTick = [0;1000;2000;3000;4000];
        app.VelocitaTorsiometroPlot.YGrid = 'on';

        % add axis annotations
        xlabel(app.VelocitaTorsiometroPlot,'Time (sec)');
        ylabel(app.VelocitaTorsiometroPlot,'Speed (rpm)');
        title(app.VelocitaTorsiometroPlot, 'Speed Trasducer');

          % update X axis ticks
        XTick_CoppiaTorsiometro = app.CoppiaTorsiometroPlot.XTick;
        XTick_CoppiaTorsiometro = [XTick_CoppiaTorsiometro(1):timeWindow/10:XTick_CoppiaTorsiometro(end)];
        % the tick mode is set to "auto" and it always displays 10 ticks for whatever time range I pick.  Looks like I don't need
        % to adjust the tics if auto mode always displays 10 ticks.

         % set Y axis limits
        app.CoppiaTorsiometroPlot.YLim = [-120, 120];

         % set Y axis ticks
        app.CoppiaTorsiometroPlot.YTick = [-120;-80;-40;0;40;80;120];
        app.CoppiaTorsiometroPlot.YGrid = 'on';
```

```
% add axis annotations
xlabel(app.CoppiaTorsiometroPlot,'Time (sec)');
ylabel(app.CoppiaTorsiometroPlot,'Torque (Nm)');
 title(app.CoppiaTorsiometroPlot, 'Torque Trasducer');

   % update X axis ticks
XTick_CorrenteMotore = app.CorrenteMotorePlot.XTick;
XTick_CorrenteMotore = [XTick_CorrenteMotore(1):timeWindow/10:XTick_CorrenteMotore(end)];
% the tick mode is set to "auto" and it always displays 10 ticks for whatever time range I pick.  Looks like I don't need
% to adjust the tics if auto mode always displays 10 ticks.

 % set Y axis limits
 app.CorrenteMotorePlot.YLim = [-80,80];

 % set Y axis ticks
app.CorrenteMotorePlot.YTick = [-80;-60;-40;-20;0;20;40;60;80];
app.CorrenteMotorePlot.YGrid = 'on';

% add axis annotations
xlabel(app.CorrenteMotorePlot,'Time (sec)');
ylabel(app.CorrenteMotorePlot,'Current (A)');
 title(app.CorrenteMotorePlot, 'Motor Current');

  % update X axis ticks
XTick_CoppiaMotore = app.CoppiaMotorePlot.XTick;
XTick_CoppiaMotore = [XTick_CoppiaMotore(1):timeWindow/10:XTick_CoppiaMotore(end)];
% the tick mode is set to "auto" and it always displays 10 ticks for whatever time range I pick.  Looks like I don't need
% to adjust the tics if auto mode always displays 10 ticks.

 % set Y axis limits
 app.CoppiaMotorePlot.YLim = [-150, 150];

 % set Y axis ticks
app.CoppiaMotorePlot.YTick = [-150;-100;-50;0;50;100;150];
app.CoppiaMotorePlot.YGrid = 'on';

% add axis annotations
xlabel(app.CoppiaMotorePlot,'Time (sec)');
ylabel(app.CoppiaMotorePlot,'Torque (Nm)');
 title(app.CoppiaMotorePlot, 'Motor Torque');

  % update X axis ticks
XTick_TemperaturaMotore = app.TemperaturaMotorePlot.XTick;
XTick_TemperaturaMotore = [XTick_TemperaturaMotore(1):timeWindow/10:XTick_TemperaturaMotore(end)];
% the tick mode is set to "auto" and it always displays 10 ticks for whatever time range I pick.  Looks like I don't need
% to adjust the tics if auto mode always displays 10 ticks.

 % set Y axis limits
 app.TemperaturaMotorePlot.YLim = [20, 80];

 % set Y axis ticks
app.TemperaturaMotorePlot.YTick = [20;40;60;80];
app.TemperaturaMotorePlot.YGrid = 'on';

% add axis annotations
xlabel(app.TemperaturaMotorePlot,'Time (sec)');
ylabel(app.TemperaturaMotorePlot,'Temperature (°C)');
 title(app.TemperaturaMotorePlot, 'Motor Temperature');

 % update X axis ticks
XTick_CorrenteFreno = app.CorrenteFrenoPlot.XTick;
XTick_CorrenteFreno = [XTick_CorrenteFreno(1):timeWindow/10:XTick_CorrenteFreno(end)];
% the tick mode is set to "auto" and it always displays 10 ticks for whatever time range I pick.  Looks like I don't need
% to adjust the tics if auto mode always displays 10 ticks.

 % set Y axis limits
 app.CorrenteFrenoPlot.YLim = [-80,80];

 % set Y axis ticks
app.CorrenteFrenoPlot.YTick = [-80;-60;-40;-20;0;20;40;60;80];
app.CorrenteFrenoPlot.YGrid = 'on';

% add axis annotations
xlabel(app.CorrenteFrenoPlot,'Time (sec)');
ylabel(app.CorrenteFrenoPlot,'Current (A)');
 title(app.CorrenteFrenoPlot, 'Generator Current');

  % update X axis ticks
XTick_CoppiaFreno = app.CoppiaFrenoPlot.XTick;
XTick_CoppiaFreno = [XTick_CoppiaFreno(1):timeWindow/10:XTick_CoppiaFreno(end)];
% the tick mode is set to "auto" and it always displays 10 ticks for whatever time range I pick.  Looks like I don't need
% to adjust the tics if auto mode always displays 10 ticks.

 % set Y axis limits
 app.CoppiaFrenoPlot.YLim = [-150, 150];

 % set Y axis ticks
app.CoppiaFrenoPlot.YTick = [-150;-100;-50;0;50;100;150];
app.CoppiaFrenoPlot.YGrid = 'on';

% add axis annotations
xlabel(app.CoppiaFrenoPlot,'Time (sec)');
ylabel(app.CoppiaFrenoPlot,'Torque (Nm)');
 title(app.CoppiaFrenoPlot, 'Generator Torque');

  % update X axis ticks
XTick_TemperaturaFreno = app.TemperaturaFrenoPlot.XTick;
XTick_TemperaturaFreno = [XTick_TemperaturaFreno(1):timeWindow/10:XTick_TemperaturaFreno(end)];
% the tick mode is set to "auto" and it always displays 10 ticks for whatever time range I pick.  Looks like I don't need
% to adjust the tics if auto mode always displays 10 ticks.
```

```
     % set Y axis limits
    app.TemperaturaFrenoPlot.YLim = [20, 80];

     % set Y axis ticks
    app.TemperaturaFrenoPlot.YTick = [20;40;60;80];
    app.TemperaturaFrenoPlot.YGrid = 'on';

    % add axis annotations
    xlabel(app.TemperaturaFrenoPlot,'Time (sec)');
    ylabel(app.TemperaturaFrenoPlot,'Temperature (°C)');
     title(app.TemperaturaFrenoPlot, 'Generator Temperature');
end     % setAxesLimitsAndTitles

function findTestbenchModel(app)
    % hard code name of model to use with the UI.  Could expand this to a
    % model selection UI using uigetfile() for a more general solution
    app.mdl = 'ModelloCompleto_CAN';

    % if the selected model isn't already open, open it
    if ~bdIsLoaded(app.mdl)
        open_system(app.mdl)
    end
end

function initializeCANChannelSelections(app)
    % get the available CAN channels
    getAvailableCANChannelInfo(app);

    % pick default CAN channel constructor properties
    app.canChannelIndex = 1;
    app.canChannelConstructorSelected = formatCANChannelConstructor(app, app.canChannelIndex);
    app.canChannelDeviceSelected = formatCANChannelDeviceStr(app, app.canChannelIndex);

    % update simulation model with initialized CAN Channel selection
    updateModelWithSelectedCANChannel(app, app.canChannelIndex);
end

function setupCANTransmitMessages(app)
     % create  CAN message containers
    app.MotorMessageInput = canMessage(100,false,8,"ProtocolMode","CAN");
    app.GeneratorMessageInput = canMessage(200,false,8,"ProtocolMode","CAN");

    % fill the message containers with data
    pack(app.MotorMessageInput,app.Slider_Motor_TorqueControl.Value,0,8,'LittleEndian')
    pack(app.MotorMessageInput,app.Slider_Motor_SpeedControl.Value,8,16,'LittleEndian')

    pack(app.GeneratorMessageInput,app.Slider_Generator_TorqueControl.Value,0,8,'LittleEndian')
    pack(app.GeneratorMessageInput,app.Slider_Generator_SpeedControl.Value,8,16,'LittleEndian')

    % set up periodic transmission of this CAN message.  Actual transmission starts/stops with CAN channel start/stop
    transmitPeriodic(app.canChannelObj, app.MotorMessageInput, 'On', 0.1);
    transmitPeriodic(app.canChannelObj, app.GeneratorMessageInput, 'On', 0.1);
end

function receiveCANmsgsTimerCallback(app)
     % receiveCANmsgsTimerCallback Timer callback function for GUI updating

        try
        % receive available CAN messages

        %msg = receive(app.canChannelObj, Inf , 'OutputFormat', 'timetable')
        msg = receive(app.canChannelObj, Inf)';

        % update CAN message data
        newOutputData = getOutputCANmessage(app, msg);

        if ~newOutputData
            return;
        end

        % Update  plots with latest data from CAN bus
        updatePlots(app);

    catch err
        disp(err.message)
    end             % try/catch
end                 % receiveCANmsgsTimerCallback

function newOutputData = getOutputCANmessage(app, msg)

    % exit if no messages were received as there is nothing to update.
    if isempty(msg)
        newOutputData = false;
        return;
    end

    % Extract the Output CAN messages.
    msgOutTrasducer = extractAll(msg,300,false); %Output Message from Torque and Velocity trasducer
    msgOutMotor = extractAll(msg,400,false); %Output Message from Motor
    msgOutGenerator = extractAll(msg,500,false); %Output Message from Generator

% if no messages then just return as there is nothing to do
    if (isempty(msgOutTrasducer) && isempty(msgOutMotor) && isempty(msgOutGenerator))
        newOutputData = false;
        return;
    end

    if (~isempty(msgOutTrasducer) || ~isempty(msgOutMotor) || ~isempty(msgOutGenerator))

        for i=1:length(msgOutTrasducer)
```

```
            Coppia_torsiometro(i)=(double(unpack(msgOutTrasducer(i),0,16,'LittleEndian','int16')))*app.General_Factor_CANmessage_Coppia_
            Corrente_Temperatura;

            Velocita_torsiometro(i)=unpack(msgOutTrasducer(i),16,16,'LittleEndian','uint16');

            TimeOutID300(i)=msgOutTrasducer(i).Timestamp;
        end

        for j=1:length(msgOutMotor)

            Coppia_motore(j)=(double(unpack(msgOutMotor(j),0,16,'LittleEndian','int16')))*app.General_Factor_CANmessage_Coppia_Corrente_
            Temperatura;

            Velocita_motore(j)=unpack(msgOutMotor(j),16,16,'LittleEndian','uint16');

            Corrente_motore(j)=(double(unpack(msgOutMotor(j),32,16,'LittleEndian','int16')))*app.General_Factor_CANmessage_Coppia_Corren
            te_Temperatura;

            Temperatura_motore(j)=(double(unpack(msgOutMotor(j),48,16,'LittleEndian','int16')))*app.General_Factor_CANmessage_Coppia_Cor
            rente_Temperatura;

            TimeOutID400(j)=msgOutMotor(j).Timestamp;
        end

        for k=1:length(msgOutGenerator)

            Coppia_freno(k)=(double(unpack(msgOutGenerator(k),0,16,'LittleEndian','int16')))*app.General_Factor_CANmessage_Coppia_Corren
            te_Temperatura;

            Velocita_freno(k)=unpack(msgOutGenerator(k),16,16,'LittleEndian','uint16');

            Corrente_freno(k)=(double(unpack(msgOutGenerator(k),32,16,'LittleEndian','int16')))*app.General_Factor_CANmessage_Coppia_Cor
            rente_Temperatura;

            Temperatura_freno(k)=(double(unpack(msgOutGenerator(k),48,16,'LittleEndian','int16')))*app.General_Factor_CANmessage_Coppia_
            Corrente_Temperatura;

            TimeOutID500(k)=msgOutGenerator(k).Timestamp;
        end
        % received new Output messages, so create time series from CAN signal data

        % save the feedback signal
            if (isempty(app.CoppiaTorsiometroFb) || isempty(app.VelocitaTorsiometroFb) || isempty(app.CoppiaMotoreFb) ||
            isempty(app.CorrenteMotoreFb) || isempty(app.TemperaturaMotoreFb) || isempty(app.CoppiaFrenoFb) ||
            isempty(app.CorrenteFrenoFb) || isempty(app.TemperaturaFrenoFb))   % check if property has been initialized

            app.CoppiaTorsiometroFb = cell(2,1);
            app.VelocitaTorsiometroFb = cell(2,1);

            app.CoppiaMotoreFb = cell(2,1);
            app.CorrenteMotoreFb = cell(2,1);
            app.TemperaturaMotoreFb = cell(2,1);

            app.CoppiaFrenoFb = cell(2,1);
            app.CorrenteFrenoFb = cell(2,1);
            app.TemperaturaFrenoFb = cell(2,1);
            % it appears Simulink.SimulationData.Dataset class is not
            % compatible with MATLAB Compiler, so change the way we store data
            % from a Dataset format to cell array.

            % save actual data if it's the first acquisition
            % messages
            app.CoppiaTorsiometroFb = timeseries(Coppia_torsiometro',(TimeOutID300)','Name','Coppia Torsiometro');
            app.VelocitaTorsiometroFb = timeseries(Velocita_torsiometro',(TimeOutID300)','Name','Velocità Torsiometro');

            app.CoppiaMotoreFb = timeseries(Coppia_motore',(TimeOutID400)','Name','Coppia Motore');
            app.CorrenteMotoreFb = timeseries(Corrente_motore',(TimeOutID400)','Name','Corrente Motore');
            app.TemperaturaMotoreFb = timeseries(Temperatura_motore',(TimeOutID400)','Name','Temperatura Motore');

            app.CoppiaFrenoFb = timeseries(Coppia_freno',(TimeOutID500)','Name','Coppia Torsiometro');
            app.CorrenteFrenoFb = timeseries(Corrente_freno',(TimeOutID500)','Name','Corrente Torsiometro');
            app.TemperaturaFrenoFb = timeseries(Temperatura_freno',(TimeOutID500)','Name','Temperatura Freno');

        else  % add to existing data previously acquired
            app.CoppiaTorsiometroFb = timeseries([app.CoppiaTorsiometroFb.Data; Coppia_torsiometro'], ...
                [app.CoppiaTorsiometroFb.Time;(TimeOutID300)'],'Name','Coppia Torsiometro');
             app.VelocitaTorsiometroFb = timeseries([app.VelocitaTorsiometroFb.Data; Velocita_torsiometro'], ...
                [app.VelocitaTorsiometroFb.Time;(TimeOutID300)'],'Name','Velocità Torsiometro');

             app.CoppiaMotoreFb = timeseries([app.CoppiaMotoreFb.Data; Coppia_motore'], ...
                [app.CoppiaMotoreFb.Time;(TimeOutID400)'],'Name','Coppia Motore');
             app.CorrenteMotoreFb = timeseries([app.CorrenteMotoreFb.Data; Corrente_motore'], ...
                [app.CorrenteMotoreFb.Time;(TimeOutID400)'],'Name','Corrente Motore');
              app.TemperaturaMotoreFb = timeseries([app.TemperaturaMotoreFb.Data; Temperatura_motore'], ...
                [app.TemperaturaMotoreFb.Time;(TimeOutID400)'],'Name','Temperatura Motore');

             app.CoppiaFrenoFb = timeseries([app.CoppiaFrenoFb.Data; Coppia_freno'], ...
                [app.CoppiaFrenoFb.Time;(TimeOutID500)'],'Name','Coppia Freno');
             app.CorrenteFrenoFb = timeseries([app.CorrenteFrenoFb.Data; Corrente_freno'], ...
                [app.CorrenteFrenoFb.Time;(TimeOutID500)'],'Name','Corrente Freno');
             app.TemperaturaFrenoFb = timeseries([app.TemperaturaFrenoFb.Data; Temperatura_freno'], ...
                [app.TemperaturaFrenoFb.Time;(TimeOutID500)'],'Name','Temperatura Freno');
        end
        newOutputData = true;
    end
end% getOutputCANmessage()
```

```
function updatePlots(app)

    timeWindow = app.plotTimeWidth;

    %Trasducer
    time_Torsiometro = app.CoppiaTorsiometroFb.Time;

    endTime_Torsiometro = time_Torsiometro(end);
    startTime_Torsiometro = max((endTime_Torsiometro-timeWindow),0);

    % find the array index correpsonding to startTime
    [~, startNdx_Torsiometro] = min(abs(time_Torsiometro-startTime_Torsiometro));

    if isempty(startNdx_Torsiometro)
        startNdx_Torsiometro = 1;
    end

    % extract the timeWindow of sample points from the time vector
    time_Torsiometro = time_Torsiometro(startNdx_Torsiometro:end);

    % update the target speed plot using the line handle
    app.CoppiaTorsiometroLine.XData = time_Torsiometro;
    app.CoppiaTorsiometroLine.YData = app.CoppiaTorsiometroFb.Data(startNdx_Torsiometro:end);

    app.VelocitaTorsiometroLine.XData = time_Torsiometro;
    app.VelocitaTorsiometroLine.YData = app.VelocitaTorsiometroFb.Data(startNdx_Torsiometro:end);

    %Motor
    time_Motore = app.CoppiaMotoreFb.Time;

    endTime_Motore = time_Motore(end);
    startTime_Motore = max((endTime_Motore-timeWindow),0);

    % find the array index correpsonding to startTime
    [~, startNdx_Motore] = min(abs(time_Motore-startTime_Motore));

    if isempty(startNdx_Motore)
        startNdx_Motore = 1;
    end

    % extract the timeWindow of sample points from the time vector
    time_Motore = time_Motore(startNdx_Motore:end);

    % update the target speed plot using the line handle
    app.CoppiaMotoreLine.XData = time_Motore;
    app.CoppiaMotoreLine.YData = app.CoppiaMotoreFb.Data(startNdx_Motore:end);

    app.CorrenteMotoreLine.XData = time_Motore;
    app.CorrenteMotoreLine.YData = app.CorrenteMotoreFb.Data(startNdx_Motore:end);

    app.TemperaturaMotoreLine.XData = time_Motore;
    app.TemperaturaMotoreLine.YData = app.TemperaturaMotoreFb.Data(startNdx_Motore:end);

    %Generator
    time_Freno = app.CoppiaFrenoFb.Time;

    endTime_Freno = time_Freno(end);
    startTime_Freno = max((endTime_Freno-timeWindow),0);

    % find the array index correpsonding to startTime
    [~, startNdx_Freno] = min(abs(time_Freno-startTime_Freno));

    if isempty(startNdx_Freno)
        startNdx_Freno = 1;
    end

    % extract the timeWindow of sample points from the time vector
    time_Freno = time_Freno(startNdx_Freno:end);

    % update the target speed plot using the line handle
    app.CoppiaFrenoLine.XData = time_Freno;
    app.CoppiaFrenoLine.YData = app.CoppiaFrenoFb.Data(startNdx_Freno:end);

    app.CorrenteFrenoLine.XData = time_Freno;
    app.CorrenteFrenoLine.YData = app.CorrenteFrenoFb.Data(startNdx_Freno:end);

    app.TemperaturaFrenoLine.XData = time_Freno;
    app.TemperaturaFrenoLine.YData = app.TemperaturaFrenoFb.Data(startNdx_Freno:end);

    % reset axis limits based on current time window
    setAxesLimitsAndTitles(app, startTime_Torsiometro,timeWindow);

    % delete old data that preceeds the current time window
    if endTime_Torsiometro - timeWindow > 0         % data in buffer longer than timeWindow, ok to delete old data points
        app.CoppiaTorsiometroFb = delsample(app.CoppiaTorsiometroFb, 'Index', [1:(startNdx_Torsiometro-1)]);
        app.VelocitaTorsiometroFb = delsample(app.VelocitaTorsiometroFb, 'Index', [1:(startNdx_Torsiometro-1)]);
    end

    if endTime_Motore - timeWindow > 0         % data in buffer longer than timeWindow, ok to delete old data points
        app.CoppiaMotoreFb = delsample(app.CoppiaMotoreFb, 'Index', [1:(startNdx_Motore-1)]);
        app.CorrenteMotoreFb = delsample(app.CorrenteMotoreFb, 'Index', [1:(startNdx_Motore-1)]);
        app.TemperaturaMotoreFb = delsample(app.TemperaturaMotoreFb, 'Index', [1:(startNdx_Motore-1)]);
    end

    if endTime_Freno - timeWindow > 0         % data in buffer longer than timeWindow, ok to delete old data points
        app.CoppiaFrenoFb = delsample(app.CoppiaFrenoFb, 'Index', [1:(startNdx_Freno-1)]);
        app.CorrenteFrenoFb = delsample(app.CorrenteFrenoFb, 'Index', [1:(startNdx_Freno-1)]);
        app.TemperaturaFrenoFb = delsample(app.TemperaturaFrenoFb, 'Index', [1:(startNdx_Freno-1)]);
    end
end
```

```
function updateEditFieldAndSliderValues_MotorSpeedControl(app)
    % update value displayed on edit field to be an integer
    app.NumericEditField_Motor_SpeedControl.Value =round(app.NumericEditField_Motor_SpeedControl.Value);
    % update CAN message transmitted to ECU
    pack(app.MotorMessageInput,app.NumericEditField_Motor_SpeedControl.Value,8,16,'LittleEndian')
    app.Slider_Motor_SpeedControl.Value = app.NumericEditField_Motor_SpeedControl.Value;
end

function updateSliderAndEditFieldValues_MotorSpeedControl(app)
    % update value displayed on slider to be an integer
    app.Slider_Motor_SpeedControl.Value =round(app.Slider_Motor_SpeedControl.Value);
    % update CAN message transmitted to ECU
    pack(app.MotorMessageInput,app.Slider_Motor_SpeedControl.Value,8,16,'LittleEndian')
    app.NumericEditField_Motor_SpeedControl.Value = app.Slider_Motor_SpeedControl.Value;
end

function updateEditFieldAndSliderValues_MotorTorqueControl(app)
    % update value displayed on edit field to be an integer
    app.NumericEditField_Motor_TorqueControl.Value =round(app.NumericEditField_Motor_TorqueControl.Value);
    % update CAN message transmitted to ECU
    pack(app.MotorMessageInput,app.NumericEditField_Motor_TorqueControl.Value,0,8,'LittleEndian')
    app.Slider_Motor_TorqueControl.Value = app.NumericEditField_Motor_TorqueControl.Value;
end

function updateSliderAndEditFieldValues_MotorTorqueControl(app)
    % update value displayed on slider to be an integer
    app.Slider_Motor_TorqueControl.Value =round(app.Slider_Motor_TorqueControl.Value);
    % update CAN message transmitted to ECU
    pack(app.MotorMessageInput,app.Slider_Motor_TorqueControl.Value,0,8,'LittleEndian')
    app.NumericEditField_Motor_TorqueControl.Value = app.Slider_Motor_TorqueControl.Value;
end

function updateEditFieldAndSliderValues_GeneratorSpeedControl(app)
    % update value displayed on edit field to be an integer
    app.NumericEditField_Generator_SpeedControl.Value =round(app.NumericEditField_Generator_SpeedControl.Value);
    % update CAN message transmitted to ECU
    pack(app.GeneratorMessageInput,app.NumericEditField_Generator_SpeedControl.Value,8,16,'LittleEndian')
    app.Slider_Generator_SpeedControl.Value = app.NumericEditField_Generator_SpeedControl.Value;

end

function updateSliderAndEditFieldValues_GeneratorSpeedControl(app)
    % update value displayed on slider to be an integer
    app.Slider_Generator_SpeedControl.Value =round(app.Slider_Generator_SpeedControl.Value);
    % update CAN message transmitted to ECU
    pack(app.GeneratorMessageInput,app.Slider_Generator_SpeedControl.Value,8,16,'LittleEndian')
    app.NumericEditField_Generator_SpeedControl.Value = app.Slider_Generator_SpeedControl.Value;
end

function updateEditFieldAndSliderValues_GeneratorTorqueControl(app)
    % update value displayed on edit field to be an integer
    app.NumericEditField_Generator_TorqueControl.Value =round(app.NumericEditField_Generator_TorqueControl.Value);
    % update CAN message transmitted to ECU
    pack(app.GeneratorMessageInput,app.NumericEditField_Generator_TorqueControl.Value,0,8,'LittleEndian')
    app.Slider_Generator_TorqueControl.Value = app.NumericEditField_Generator_TorqueControl.Value;
end

function updateSliderAndEditFieldValues_GeneratorTorqueControl(app)
    % update value displayed on slider to be an integer
    app.Slider_Generator_TorqueControl.Value =round(app.Slider_Generator_TorqueControl.Value);
    % update CAN message transmitted to ECU
    pack(app.GeneratorMessageInput,app.Slider_Generator_TorqueControl.Value,0,8,'LittleEndian')
    % update vehicle speed numeric edit field with value entered from vehicle speed slider
    app.NumericEditField_Generator_TorqueControl.Value = app.Slider_Generator_TorqueControl.Value;
end

function startSimApplication(app, index)
    % start the model running on the desktop

    % check to see if hardware or virtual CAN channel is selected, otherwise
    % do nothing
    if app.canChannelInfo.DeviceModel(index) == "Virtual"
        % check to see if the model is loaded before trying to run
        if bdIsLoaded(app.mdl)
            % model is loaded, now check to see if it is already running
            if ~strcmp('running',get_param(app.mdl,'SimulationStatus'))
                % model is not already running, so start it
                % flush the CAN Receive message buffers
                app.CoppiaTorsiometroFb = [];
                app.VelocitaTorsiometroFb = [];
                app.CoppiaMotoreFb = [];
                app.CorrenteMotoreFb = [];
                app.TemperaturaMotoreFb = [];
                app.CoppiaFrenoFb = [];
                app.CorrenteFrenoFb = [];
                app.TemperaturaFrenoFb = [];
                % clear figure window
                cla(app.CoppiaTorsiometroPlot)
                cla(app.VelocitaTorsiometroPlot)
                cla(app.CoppiaMotorePlot)
                cla(app.CorrenteMotorePlot)
                cla(app.TemperaturaMotorePlot)
                cla(app.CoppiaFrenoPlot)
                cla(app.CorrenteFrenoPlot)
                cla(app.TemperaturaFrenoPlot)

                % start the CAN channels and update timer if it isn't already running
                startCANChannel(app);

                % start the model
```

```
                set_param(app.mdl, 'SimulationCommand', 'start');

                % set the sim start/stop button icon to the stop icon indicating the model has
                % been successfully started and is ready to be stopped at the next
                % button press
                app.SimStartStopButton.Icon = "IconEnd.png";
                app.StartSimLabel.Text = "Stop Sim";
            else
                % model is already running, inform the user
                warnStr = sprintf('Warning: Model %s is already running', app.mdl);
                warndlg(warnStr, 'Warning');
            end
        else
            % model is not yet loaded, so warn the user
            warnStr = sprintf('Warning: Model %s is not loaded\nPlease load the model and try again', app.mdl);
            % opts = struct('WindowStyle', 'modal');
            warndlg(warnStr, 'Warning');
        end
    end
end

function stopSimApplication(app, index)
    % stop the model running on the desktop
    try
        % check to see if hardware or virtual CAN channel is selected
        if app.canChannelInfo.DeviceModel(index) == "Virtual"
            % virtual channel selected, so issue a stop command to the
            % the simulation, even if it is already stopped
            set_param(app.mdl, 'SimulationCommand', 'stop')

            % stop the CAN channels and update timer
            stopCANChannel(app);
        end
        % set the sim start/stop button text to Start indicating the model has
        % been successfully stopped and is ready to start again at the next
        % button press
        app.SimStartStopButton.Icon = "IconPlay.png";
        app.StartSimLabel.Text = "Start Sim";
    catch
        % do nothing at the moment
    end
end

function startCANLogging(app)
    startCANLogChannel(app);
    % update the button icon and label
    app.canLoggingStartStopButton.Icon = 'IconEnd.png';
    app.StartLoggingLabel.Text = "Stop Logging";
end

function stopCANLogging(app)
    % stop the CAN Log channel
    stop(app.canLogChannelObj);

    % get the messages from the CAN log message queue
    retrieveLoggedCANMessages(app);

    % update the button icon and label
    app.canLoggingStartStopButton.Icon = 'IconPlay.png';
    app.StartLoggingLabel.Text = "Start Logging";
end

function startCANReplay(app)
    startPlaybackOfLoggedCANData(app);

    % update the button icon and label
    app.canReplayStartStopButton.Icon = 'IconEnd.png';
    app.StartReplayLabel.Text = "Stop Replay";
end

function stopCANReplay(app)
    stopPlaybackOfLoggedCANData(app);

    % update the button icon and label
    app.canReplayStartStopButton.Icon = 'IconPlay.png';
    app.StartReplayLabel.Text = "Start Replay";
end

function setupCANChannel(app)
    % create a CAN channel for sending and receiving messages
    app.canChannelObj = eval(app.canChannelConstructorSelected);

    % set the baud rate (can only do this if the UI has channel initialization access)
    if app.canChannelObj.InitializationAccess
        configBusSpeed(app.canChannelObj, 500000);
    end
end         % setupCANchannel()

function startCANChannel(app)
    % start the CAN channel if it isn't already running
    try
        if ~app.canChannelObj.Running
            start(app.canChannelObj);
        end
    catch
        % do nothing
    end

    % start the CAN receive processing timer - check to see if it is already running.  This allows us to change CAN channels
    % with or without starting and stopping the model running on the real time target.
```

```
    if strcmpi(app.receiveCANmsgsTimer.Running, 'off')
        start(app.receiveCANmsgsTimer);
    end
end

function stopCANChannel(app)
    % stop the CAN channel
    stop(app.canChannelObj);

    % stop the CAN message processing timer
    stop(app.receiveCANmsgsTimer);
end

function canChannelConstructor = formatCANChannelConstructor(app, index)
   canChannelConstructor = "canChannel(" + "'" + app.canChannelInfo.Vendor(index) + "'" + ", " + "'" + app.canChannelInfo.Device(index) +
   "'" + ", " + app.canChannelInfo.Channel(index) + ")";
end

function canChannelDeviceStr = formatCANChannelDeviceStr(app, index)
   canChannelDeviceStr = app.canChannelInfo.Vendor(index) + " " + app.canChannelInfo.Device(index) + " " + "(Channel " +
   app.canChannelInfo.Channel(index) + ")";
end

function updateModelWithSelectedCANChannel(app, index)
    % check to see if we are using a virtual CAN channel and whether the model is loaded
    if app.canChannelInfo.DeviceModel(index) == "Virtual" && bdIsLoaded(app.mdl)
        % using a virtual channel

        % find path to CAN configuration block
        canConfigPath = find_system(app.mdl,'Variants', 'AllVariants', 'LookUnderMasks', 'all',...
            'FollowLInks', 'on', 'Name', 'CAN Configuration');

        % find path to CAN transmit block
        canTransmitPath = find_system(app.mdl,'Variants', 'AllVariants', 'LookUnderMasks', 'all',...
            'FollowLInks', 'on', 'Name', 'CAN Transmit');

        % find path to CAN receive block
        canReceivePath = find_system(app.mdl,'Variants', 'AllVariants', 'LookUnderMasks', 'all',...
            'FollowLInks', 'on', 'Name', 'CAN Receive');

        % push the selected CAN channel into the simulation model CAN Configuration block
        set_param(canConfigPath{1}, 'Device', app.canChannelDeviceSelected);
        set_param(canConfigPath{1}, 'DeviceMenu', app.canChannelDeviceSelected);
        set_param(canConfigPath{1}, 'ObjConstructor', app.canChannelConstructorSelected);

        % push the selected CAN channel into the simulation model CAN Recieve block
        set_param(canReceivePath{1}, 'Device', app.canChannelDeviceSelected);
        set_param(canReceivePath{1}, 'DeviceMenu', app.canChannelDeviceSelected);
        set_param(canReceivePath{1}, 'ObjConstructor', app.canChannelConstructorSelected);

        % push the selected CAN channel into the simulation model CAN Transmit block
        set_param(canTransmitPath{1}, 'Device', app.canChannelDeviceSelected);
        set_param(canTransmitPath{1}, 'DeviceMenu', app.canChannelDeviceSelected);
        set_param(canTransmitPath{1}, 'ObjConstructor', app.canChannelConstructorSelected);
    end
end

function displayText = formatCANChannelEntryForDisplay(app)
    % pre-allocate Execution Context column for canChannelInfo table
    ExecutionContext = repelem(string(missing), height(app.canChannelInfo), 1);

    % add the pre-allocated Execution Context column to the canChannelInfo table
    app.canChannelInfo = addvars(app.canChannelInfo, ExecutionContext, 'NewVariableNames', 'ExecutionContext');

    % determine if the execution context is "Desktop" or "RealTime"
    app.canChannelInfo.ExecutionContext(app.canChannelInfo.DeviceModel == "Virtual") = "Desktop";
    app.canChannelInfo.ExecutionContext(app.canChannelInfo.DeviceModel ~= "Virtual") = "Realtime";

    % format CAN channel list for display
     displayText = app.canChannelInfo.Vendor + " " + app.canChannelInfo.Device + ", Channel: " + app.canChannelInfo.Channel + " " + "(" +
   app.canChannelInfo.ExecutionContext + ")";
end

function setModelStartStopButtonEnableStatus(app)

    if(app.canChannelInfo.DeviceModel(app.canChannelIndex) == "Virtual")
        % using a virtual channel, so talk to model on host, not on SLRT
        app.SimStartStopButton.Enable = 'on';
    else
        % using a hardware CAN channel, so talk to model on real time target, not host
        app.SimStartStopButton.Enable = 'off';
    end

    if isdeployed
        app.SimStartStopButton.Enable = 'off';
    end
end

function setupCANLogChannel(app)
    % create a CAN channel for sending and receiving messages
    app.canLogChannelObj = eval(app.canChannelConstructorSelected);

    % set the baud rate (can only do this if the UI has channel initialization access)
    if app.canLogChannelObj.InitializationAccess
        configBusSpeed(app.canLogChannelObj, 500000);
    end
end

function startCANLogChannel(app)
    % start the CAN Log channel if it isn't already running
```

```
    try
        if ~app.canLogChannelObj.Running
            start(app.canLogChannelObj);
        end
    catch
        % do nothing
    end
end

function stopCANLogChannel(app)
    % stop the CAN Log channel
    stop(app.canLogChannelObj);
end

function getAvailableCANChannelInfo(app)
    % get a table containing all available CAN channels and devices
    app.canChannelInfo = canChannelList;

    % format CAN channel information for display on the UI
    app.availableCANChannelsForDisplay = formatCANChannelEntryForDisplay(app);

    % save the number of available constructors
    app.numConstructors = numel(app.canChannelInfo.Vendor);

end

function retrieveLoggedCANMessages(app)
    try
        % receive available CAN message
        % initialize buffer to make sure it is empty
        app.canLogMsgInBuffer = [];

        app.canLogMsgInID100Buffer = [];
        app.canLogMsgInID200Buffer = [];
        app.canLogMsgOutID300Buffer = [];
        app.canLogMsgOutID400Buffer = [];
        app.canLogMsgOutID500Buffer = [];

        % receive available CAN messages
        %msg = receive(app.canLogChannelObj, Inf, 'OutputFormat',
        %'timetable');
        msg = receive(app.canLogChannelObj, Inf)';

        % fill the buffer with the logged Command CAN message data
        app.canLogMsgInBuffer = extractAll(msg,[100 200],[false false]);

        app.canLogMsgInID100Buffer = extractAll(msg,100,false);
        app.canLogMsgInID200Buffer = extractAll(msg,200,false);
        app.canLogMsgOutID300Buffer = extractAll(msg,300,false);
        app.canLogMsgOutID400Buffer = extractAll(msg,400,false);
        app.canLogMsgOutID500Buffer = extractAll(msg,500,false);

    catch err
        disp(err.message)
    end             % try/catch
end

function flushCANFbMsgQueue(app)
CoppiaTorsiometroFb = [];
VelocitaTorsiometroFb = [];
CoppiaMotoreFb = [];
VelocitaMotoreFb = [];
CorrenteMotoreFb = [];
TemperaturaMotoreFb = [];
CoppiaFrenoFb = [];
VelocitaFrenoFb = [];
CorrenteFrenoFb = [];
TemperaturaFrenoFb = [];
end

function savedLoggedCANDataToFile(app)
    % raise dialog box to prompt user for a CAN log file to store the logged data
    [FileName,PathName] = uiputfile('*.mat','Select a .MAT file to store CAN data');
    if FileName ~= 0
        % user did not cancel the file selection operation, so OK to save
        % convert the CAN log data from Timetable to struct of arrays so the data is compatible
        % with the VNT Simulink Replay block.
        canLogInStructOfArrays = canMessageReplayBlockStruct(app.canLogMsgInBuffer);

        canLogInID100StructOfArrays = canMessageReplayBlockStruct(app.canLogMsgInID100Buffer);
        canLogInID200StructOfArrays = canMessageReplayBlockStruct(app.canLogMsgInID200Buffer);
        canLogOutID300StructOfArrays = canMessageReplayBlockStruct(app.canLogMsgOutID300Buffer);
        canLogOutID400StructOfArrays = canMessageReplayBlockStruct(app.canLogMsgOutID400Buffer);
        canLogOutID500StructOfArrays = canMessageReplayBlockStruct(app.canLogMsgOutID500Buffer);
        % canLog = app.canLogMsgBuffer;

   save(fullfile(PathName,FileName),'canLogInStructOfArrays','canLogInID100StructOfArrays',"canLogInID200StructOfArrays",
   "canLogOutID300StructOfArrays","canLogOutID400StructOfArrays","canLogOutID500StructOfArrays");
        % clear the buffer after saving it
        app.canLogMsgInBuffer = [];

        app.canLogMsgInID100Buffer = [];
        app.canLogMsgInID200Buffer = [];
        app.canLogMsgOutID300Buffer = [];
        app.canLogMsgOutID400Buffer = [];
        app.canLogMsgOutID500Buffer = [];
    end
end
function loadLoggedCANDataFromFile(app)
    % raise dialog box to prompt user for a CAN log file to load
```

```
            [FileName,PathName] = uigetfile('*.mat','Select a CAN log file to load');

            figure(app.UIFigure)        % return focus to main UI after dlg closes

            if FileName ~= 0
                % user did not cancel the file selection operation, so OK to load
                % make sure the message buffer is empty before loading in the logged CAN data
                app.canLogMsgInBuffer = [];

                % upload the saved message data from the selected file
                canLogInMsgStructOfArrays = load(fullfile(PathName, FileName), 'canLogInStructOfArrays');

                % convert the saved message data into timetables for the replay command
                app.canLogMsgInBuffer = canMessageTimetable(canLogInMsgStructOfArrays.canLogInStructOfArrays);

            end
        end

        function startPlaybackOfLoggedCANData(app)
            % turn off periodic transmission of CAN message from UI controls.
             transmitPeriodic(app.canChannelObj,app.MotorMessageInput, 'Off');
             transmitPeriodic(app.canChannelObj,app.GeneratorMessageInput, 'Off');

            % stop the UI CAN channel so we can instead use if for playback
            stopCANChannel(app)

            % flush the existing CAN messages stored for plotting
            flushCANFbMsgQueue(app)

            % clear the existing plots
             cla(app.CoppiaTorsiometroPlot)
             cla(app.VelocitaTorsiometroPlot)
             cla(app.CoppiaMotorePlot)
             cla(app.CorrenteMotorePlot)
             cla(app.TemperaturaMotorePlot)
             cla(app.CoppiaFrenoPlot)
             cla(app.CorrenteFrenoPlot)
             cla(app.TemperaturaFrenoPlot)

            % start the CAN Channel and replay the logged CAN message data
            startCANChannel(app)

            % replay the logged CAN data on the UI CAN Channel
            replay(app.canChannelObj, app.canLogMsgInBuffer);
        end

        function stopPlaybackOfLoggedCANData(app)
            % stop the playback CAN channel
            stopCANChannel(app)

            % flush the existing CAN messages stored for plotting
            flushCANFbMsgQueue(app)

            % clear the existing plots
            cla(app.CoppiaTorsiometroPlot)
            cla(app.VelocitaTorsiometroPlot)
            cla(app.CoppiaMotorePlot)
            cla(app.CorrenteMotorePlot)
            cla(app.TemperaturaMotorePlot)
            cla(app.CoppiaFrenoPlot)
            cla(app.CorrenteFrenoPlot)
            cla(app.TemperaturaFrenoPlot)

            % re-enable periodic transmission CAN message from UI controls.
            transmitPeriodic(app.canChannelObj, app.MotorMessageInput, 'On', 0.1);
            transmitPeriodic(app.canChannelObj, app.GeneratorMessageInput, 'On', 0.1);

            % restart the CAN Channel from/To UI
            startCANChannel(app)
          end
        end

    methods (Access = public)

        function close(app)
            % clean up and close the UI
            closeUI(app)
        end
    end

    % Callbacks that handle component events
    methods (Access = private)

        % Code that executes after component creation
        function StartupFcn(app)

            % initialize radio button values
            setupInitialVals(app)

            % Setting up plot Displays
             setUpPlotGridAndHoldStatus(app)

            % initialize plot time width property
             plotTimeWindowWidth(app);

            % set axis limits
             setAxesLimitsAndTitles(app, 0, app.plotTimeWidth)

            % initialize a handle to the model running on the host PC
            try
```

```
        % app.mdl = bdroot;
        findTestbenchModel(app)
    catch
        % do nothing
    end

    % initialize CAN channel type string - assume first channel from list of available channels at startup
    initializeCANChannelSelections(app)

    % initialize start/stop button states
    app.SimStartStopButton.Value = 0;
    app.canLoggingStartStopButton.Value = 0;
    app.canReplayStartStopButton.Value = 0;

    % initialize the model and real time button enables based on type of CAN channel device: Virtual or Hardware
    setModelStartStopButtonEnableStatus(app)

    % setup the CAN channel
    setupCANChannel(app);

    % setup the CAN Log channel
    setupCANLogChannel(app);

    % setup periodic transmission of CAN messages to target
    setupCANTransmitMessages(app)

    % create a timer to receive CAN msgs
    app.receiveCANmsgsTimer = timer('Period', 0.1,...
                                    'ExecutionMode', 'fixedSpacing', ...
                                    'TimerFcn', @(~,~)receiveCANmsgsTimerCallback(app));
    % initialize a handle to the model running on the host PC
    try
        % app.mdl = bdroot;
        findTestbenchModel(app)
    catch
        % do nothing
    end

    % if deployed, disable the simulation start/stop buttons
    if isdeployed
        app.startSimButton.Enable = 'off';
        app.stopSimButton.Enable = 'off';
    end
end

% Menu selected function: SelectCANChannel
function selectCANChannel_callback(app, event)
      % get user selected CAN channel
    getAvailableCANChannelInfo(app)

    % display the list of available channels in a dialog box for the user to select from
    [index, tf] = listdlg('Name', 'CAN Channel Selection','PromptString', 'Select a CAN Channel',...
        'ListString', app.availableCANChannelsForDisplay, 'SelectionMode', 'single', ...
        'ListSize', [300, 200]);

    figure(app.UIFigure)        % return focus to main UI after listdlg closes

    if tf        % user made a selection
        % save the index of the selected CAN channel for other functions to use
        app.canChannelIndex = index;

        % format the CAN channel constructor.  This string will be used to create a CAN channel object
        % that will be stored in the app UI, as well as update the VNT blocks in the Simulink model with
        % the user selected CAN channel.
        app.canChannelConstructorSelected = formatCANChannelConstructor(app, app.canChannelIndex);

        % format the CAN channel Device and DeviceMenu strings.  These strings, along with the channel
        % constructor will be used to update the VNT blocks in the Simulink model with the user selected
        % CAN channel.
        app.canChannelDeviceSelected = formatCANChannelDeviceStr(app, app.canChannelIndex);

        % set model start/stop button enable status
        setModelStartStopButtonEnableStatus(app)

        % stop and delete existing CAN channel
        try
            stopCANChannel(app);

            % stop the CAN Log channel
            stop(app.canLogChannelObj);

            delete(app.canChannelObj)
            delete(app.canLogChannelObj)
        catch
            % do nothing
        end
        % create new channel
        setupCANChannel(app);
        setupCANLogChannel(app);

        % update the simulation model with the new CAN channel selection
        updateModelWithSelectedCANChannel(app, app.canChannelIndex);

        % setup periodic transmission of CAN messages to target
        setupCANTransmitMessages(app)

    else         % user did not make a selection, so just return and use previous selection
        return;
    end
end
```

```
% Value changed function: Slider_Motor_SpeedControl
function Slider_Motor_SpeedControlValueChanged(app, event)
    updateSliderAndEditFieldValues_MotorSpeedControl(app)
end

% Value changed function: NumericEditField_Motor_SpeedControl
function NumericEditField_Motor_SpeedControlValueChanged(app, event)
    updateEditFieldAndSliderValues_MotorSpeedControl(app)
end

% Value changed function: SimStartStopButton
function SimStartStopButton_callback(app, event)
     value = app.SimStartStopButton.Value;
    if value == 1           % button was pressed
        startSimApplication(app, app.canChannelIndex);
    else
        stopSimApplication(app, app.canChannelIndex);
    end
end

% Value changed function: canLoggingStartStopButton
function canLoggingStartStopButton_callback(app, event)
    value = app.canLoggingStartStopButton.Value;
    if value == 1           % button was pressed
        startCANLogging(app);
    else
        stopCANLogging(app);
    end
end

% Button pushed function: saveLoggedDataButton
function saveLoggedDataButtonPushed(app, event)
    savedLoggedCANDataToFile(app);
end

% Button pushed function: loadCANLogDataButton
function loadCANLogDataButtonPushed(app, event)
    loadLoggedCANDataFromFile(app)
end

% Value changed function: canReplayStartStopButton
function canReplayStartStopButton_callback(app, event)
value = app.canReplayStartStopButton.Value;
    if value == 1       % button was pressed
        startCANReplay(app);
    else
        stopCANReplay(app);
    end
end

% Value changed function: plotTimeWindowDropDown
function plotTimeWindowWidth(app, event)
app.plotTimeWidth = str2num(app.plotTimeWindowDropDown.Value);
end

% Close request function: UIFigure
function closeUI(app, event)
     % clean up CAN receive timer objects when UI window is closed
    try
        stop(app.receiveCANmsgsTimer);
        delete(app.receiveCANmsgsTimer);
    catch
        % do nothing
    end

    % stop and delete the CAN channel
    try
        stop(app.canChannelObj)
        delete(app.canChannelObj)
    catch
        % do nothing
    end
    delete(app)
end

% Value changed function: Slider_Motor_TorqueControl
function Slider_Motor_TorqueControlValueChanged(app, event)
   updateSliderAndEditFieldValues_MotorTorqueControl(app)
end

% Value changed function: NumericEditField_Motor_TorqueControl
function NumericEditField_Motor_TorqueControlValueChanged(app, event)
   updateEditFieldAndSliderValues_MotorTorqueControl(app)
end

% Value changed function: Slider_Generator_SpeedControl
function Slider_Generator_SpeedControlValueChanged(app, event)
    updateSliderAndEditFieldValues_GeneratorSpeedControl(app)
end

% Value changed function: NumericEditField_Generator_SpeedControl
function NumericEditField_Generator_SpeedControlValueChanged(app, event)
    updateEditFieldAndSliderValues_GeneratorSpeedControl(app)
end

% Value changed function: Slider_Generator_TorqueControl
function Slider_Generator_TorqueControlValueChanged(app, event)
    updateSliderAndEditFieldValues_GeneratorTorqueControl(app)
end
```

```
        % Value changed function: NumericEditField_Generator_TorqueControl
        function NumericEditField_Generator_TorqueControlValueChanged(app, event)
            updateEditFieldAndSliderValues_GeneratorTorqueControl(app)
        end
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Position = [100 100 1444 923];
            app.UIFigure.Name = 'MATLAB App';
            app.UIFigure.CloseRequestFcn = createCallbackFcn(app, @closeUI, true);

            % Create ChannelConfigurationMenu
            app.ChannelConfigurationMenu = uimenu(app.UIFigure);
            app.ChannelConfigurationMenu.Text = 'Channel Configuration';

            % Create SelectCANChannel
            app.SelectCANChannel = uimenu(app.ChannelConfigurationMenu);
            app.SelectCANChannel.MenuSelectedFcn = createCallbackFcn(app, @selectCANChannel_callback, true);
            app.SelectCANChannel.Text = 'Select CAN Channel';

            % Create OutputsPlotPanel
            app.OutputsPlotPanel = uipanel(app.UIFigure);
            app.OutputsPlotPanel.Title = ' OUTPUT';
            app.OutputsPlotPanel.Position = [16 22 1413 896];

            % Create CoppiaTorsiometroPlot
            app.CoppiaTorsiometroPlot = uiaxes(app.OutputsPlotPanel);
            xlabel(app.CoppiaTorsiometroPlot, 'X')
            ylabel(app.CoppiaTorsiometroPlot, 'Y')
            app.CoppiaTorsiometroPlot.PlotBoxAspectRatio = [4.21052631578947 1 1];
            app.CoppiaTorsiometroPlot.XTickLabelRotation = 0;
            app.CoppiaTorsiometroPlot.YTickLabelRotation = 0;
            app.CoppiaTorsiometroPlot.XGrid = 'on';
            app.CoppiaTorsiometroPlot.YGrid = 'on';
            app.CoppiaTorsiometroPlot.GridAlpha = 0.15;
            app.CoppiaTorsiometroPlot.MinorGridAlpha = 0.25;
            app.CoppiaTorsiometroPlot.Box = 'on';
            app.CoppiaTorsiometroPlot.Position = [2 503 455 258];

            % Create VelocitaTorsiometroPlot
            app.VelocitaTorsiometroPlot = uiaxes(app.OutputsPlotPanel);
            xlabel(app.VelocitaTorsiometroPlot, 'X')
            ylabel(app.VelocitaTorsiometroPlot, 'Y')
            app.VelocitaTorsiometroPlot.PlotBoxAspectRatio = [4.21052631578947 1 1];
            app.VelocitaTorsiometroPlot.XTickLabelRotation = 0;
            app.VelocitaTorsiometroPlot.YTickLabelRotation = 0;
            app.VelocitaTorsiometroPlot.XGrid = 'on';
            app.VelocitaTorsiometroPlot.YGrid = 'on';
            app.VelocitaTorsiometroPlot.GridAlpha = 0.15;
            app.VelocitaTorsiometroPlot.MinorGridAlpha = 0.25;
            app.VelocitaTorsiometroPlot.Box = 'on';
            app.VelocitaTorsiometroPlot.Position = [3 260 454 324];

            % Create CoppiaMotorePlot
            app.CoppiaMotorePlot = uiaxes(app.OutputsPlotPanel);
            xlabel(app.CoppiaMotorePlot, 'X')
            ylabel(app.CoppiaMotorePlot, 'Y')
            app.CoppiaMotorePlot.PlotBoxAspectRatio = [4.21052631578947 1 1];
            app.CoppiaMotorePlot.XTickLabelRotation = 0;
            app.CoppiaMotorePlot.YTickLabelRotation = 0;
            app.CoppiaMotorePlot.XGrid = 'on';
            app.CoppiaMotorePlot.YGrid = 'on';
            app.CoppiaMotorePlot.GridAlpha = 0.15;
            app.CoppiaMotorePlot.MinorGridAlpha = 0.25;
            app.CoppiaMotorePlot.Box = 'on';
            app.CoppiaMotorePlot.Position = [462 633 466 227];

            % Create CorrenteMotorePlot
            app.CorrenteMotorePlot = uiaxes(app.OutputsPlotPanel);
            xlabel(app.CorrenteMotorePlot, 'X')
            ylabel(app.CorrenteMotorePlot, 'Y')
            app.CorrenteMotorePlot.PlotBoxAspectRatio = [4.21052631578947 1 1];
            app.CorrenteMotorePlot.XTickLabelRotation = 0;
            app.CorrenteMotorePlot.YTickLabelRotation = 0;
            app.CorrenteMotorePlot.XGrid = 'on';
            app.CorrenteMotorePlot.YGrid = 'on';
            app.CorrenteMotorePlot.GridAlpha = 0.15;
            app.CorrenteMotorePlot.MinorGridAlpha = 0.25;
            app.CorrenteMotorePlot.Box = 'on';
            app.CorrenteMotorePlot.Position = [462 414 466 252];

            % Create CoppiaFrenoPlot
            app.CoppiaFrenoPlot = uiaxes(app.OutputsPlotPanel);
            xlabel(app.CoppiaFrenoPlot, 'X')
            ylabel(app.CoppiaFrenoPlot, 'Y')
            app.CoppiaFrenoPlot.PlotBoxAspectRatio = [4.21052631578947 1 1];
            app.CoppiaFrenoPlot.XTickLabelRotation = 0;
            app.CoppiaFrenoPlot.YTickLabelRotation = 0;
            app.CoppiaFrenoPlot.XGrid = 'on';
            app.CoppiaFrenoPlot.YGrid = 'on';
            app.CoppiaFrenoPlot.GridAlpha = 0.15;
            app.CoppiaFrenoPlot.MinorGridAlpha = 0.25;
            app.CoppiaFrenoPlot.Box = 'on';
```

```
app.CoppiaFrenoPlot.Position = [925 617 466 257];

% Create CorrenteFrenoPlot
app.CorrenteFrenoPlot = uiaxes(app.OutputsPlotPanel);
xlabel(app.CorrenteFrenoPlot, 'X')
ylabel(app.CorrenteFrenoPlot, 'Y')
app.CorrenteFrenoPlot.PlotBoxAspectRatio = [4.21052631578947 1 1];
app.CorrenteFrenoPlot.XTickLabelRotation = 0;
app.CorrenteFrenoPlot.YTickLabelRotation = 0;
app.CorrenteFrenoPlot.XGrid = 'on';
app.CorrenteFrenoPlot.YGrid = 'on';
app.CorrenteFrenoPlot.GridAlpha = 0.15;
app.CorrenteFrenoPlot.MinorGridAlpha = 0.25;
app.CorrenteFrenoPlot.Box = 'on';
app.CorrenteFrenoPlot.Position = [925 414 466 252];

% Create TemperaturaMotorePlot
app.TemperaturaMotorePlot = uiaxes(app.OutputsPlotPanel);
xlabel(app.TemperaturaMotorePlot, 'X')
ylabel(app.TemperaturaMotorePlot, 'Y')
app.TemperaturaMotorePlot.PlotBoxAspectRatio = [4.21052631578947 1 1];
app.TemperaturaMotorePlot.XTickLabelRotation = 0;
app.TemperaturaMotorePlot.YTickLabelRotation = 0;
app.TemperaturaMotorePlot.XGrid = 'on';
app.TemperaturaMotorePlot.YGrid = 'on';
app.TemperaturaMotorePlot.GridAlpha = 0.15;
app.TemperaturaMotorePlot.MinorGridAlpha = 0.25;
app.TemperaturaMotorePlot.Box = 'on';
app.TemperaturaMotorePlot.Position = [462 211 466 252];

% Create TemperaturaFrenoPlot
app.TemperaturaFrenoPlot = uiaxes(app.OutputsPlotPanel);
xlabel(app.TemperaturaFrenoPlot, 'X')
ylabel(app.TemperaturaFrenoPlot, 'Y')
app.TemperaturaFrenoPlot.PlotBoxAspectRatio = [4.21052631578947 1 1];
app.TemperaturaFrenoPlot.XTickLabelRotation = 0;
app.TemperaturaFrenoPlot.YTickLabelRotation = 0;
app.TemperaturaFrenoPlot.XGrid = 'on';
app.TemperaturaFrenoPlot.YGrid = 'on';
app.TemperaturaFrenoPlot.GridAlpha = 0.15;
app.TemperaturaFrenoPlot.MinorGridAlpha = 0.25;
app.TemperaturaFrenoPlot.Box = 'on';
app.TemperaturaFrenoPlot.Position = [925 211 466 252];

% Create InputsPanel
app.InputsPanel = uipanel(app.OutputsPlotPanel);
app.InputsPanel.Title = 'INPUT SET';
app.InputsPanel.Position = [2 1 1286 219];

% Create MotorControlPanel
app.MotorControlPanel = uipanel(app.InputsPanel);
app.MotorControlPanel.Title = 'Motor Control';
app.MotorControlPanel.Position = [1 1 391 196];

% Create Slider_Motor_SpeedControl
app.Slider_Motor_SpeedControl = uislider(app.MotorControlPanel);
app.Slider_Motor_SpeedControl.Limits = [0 2880];
app.Slider_Motor_SpeedControl.MajorTicks = [0 360 720 1080 1440 1800 2160 2520 2880];
app.Slider_Motor_SpeedControl.MajorTickLabels = {'0', '360', '720', '1080', '1440', '1800', '2160', '2520', '2880'};
app.Slider_Motor_SpeedControl.ValueChangedFcn = createCallbackFcn(app, @Slider_Motor_SpeedControlValueChanged, true);
app.Slider_Motor_SpeedControl.Position = [29 120 341 3];

% Create Slider_Motor_TorqueControl
app.Slider_Motor_TorqueControl = uislider(app.MotorControlPanel);
app.Slider_Motor_TorqueControl.Limits = [0 120];
app.Slider_Motor_TorqueControl.ValueChangedFcn = createCallbackFcn(app, @Slider_Motor_TorqueControlValueChanged, true);
app.Slider_Motor_TorqueControl.Position = [29 46 341 3];

% Create LabelNumericEditField2
app.LabelNumericEditField2 = uilabel(app.MotorControlPanel);
app.LabelNumericEditField2.HorizontalAlignment = 'right';
app.LabelNumericEditField2.VerticalAlignment = 'top';
app.LabelNumericEditField2.Position = [79 132 112 22];
app.LabelNumericEditField2.Text = 'Speed Control [rpm]';

% Create NumericEditField_Motor_SpeedControl
app.NumericEditField_Motor_SpeedControl = uieditfield(app.MotorControlPanel, 'numeric');
app.NumericEditField_Motor_SpeedControl.Limits = [0 Inf];
 app.NumericEditField_Motor_SpeedControl.ValueChangedFcn = createCallbackFcn(app, @NumericEditField_Motor_SpeedControlValueChanged,
                                                    true);
app.NumericEditField_Motor_SpeedControl.Position = [208 135 50 22];

% Create LabelNumericEditField2_4
app.LabelNumericEditField2_4 = uilabel(app.MotorControlPanel);
app.LabelNumericEditField2_4.HorizontalAlignment = 'right';
app.LabelNumericEditField2_4.VerticalAlignment = 'top';
app.LabelNumericEditField2_4.Position = [77 48 112 22];
app.LabelNumericEditField2_4.Text = 'Torque Control [Nm]';

% Create NumericEditField_Motor_TorqueControl
app.NumericEditField_Motor_TorqueControl = uieditfield(app.MotorControlPanel, 'numeric');
app.NumericEditField_Motor_TorqueControl.Limits = [0 Inf];
 app.NumericEditField_Motor_TorqueControl.ValueChangedFcn = createCallbackFcn(app,@NumericEditField_Motor_TorqueControlValueCh anged,
                                                    true);
app.NumericEditField_Motor_TorqueControl.Position = [206 51 50 22];

% Create GeneratorControlPanel
app.GeneratorControlPanel = uipanel(app.InputsPanel);
app.GeneratorControlPanel.Title = 'Generator Control';
app.GeneratorControlPanel.Position = [392 1 400 196];
```

```
        % Create Slider_Generator_SpeedControl
        app.Slider_Generator_SpeedControl = uislider(app.GeneratorControlPanel);
        app.Slider_Generator_SpeedControl.Limits = [0 2880];
        app.Slider_Generator_SpeedControl.MajorTicks = [0 360 720 1080 1440 1800 2160 2520 2880];
        app.Slider_Generator_SpeedControl.ValueChangedFcn = createCallbackFcn(app, @Slider_Generator_SpeedControlValueChanged, true);
        app.Slider_Generator_SpeedControl.Position = [16 120 341 3];

        % Create Slider_Generator_TorqueControl
        app.Slider_Generator_TorqueControl = uislider(app.GeneratorControlPanel);
        app.Slider_Generator_TorqueControl.Limits = [0 120];
           app.Slider_Generator_TorqueControl.ValueChangedFcn = createCallbackFcn(app, @Slider_Generator_TorqueControlValueChanged, true);
        app.Slider_Generator_TorqueControl.Position = [17 45 340 3];

        % Create LabelNumericEditField2_5
        app.LabelNumericEditField2_5 = uilabel(app.GeneratorControlPanel);
        app.LabelNumericEditField2_5.HorizontalAlignment = 'right';
        app.LabelNumericEditField2_5.VerticalAlignment = 'top';
        app.LabelNumericEditField2_5.Position = [74 124 112 22];
        app.LabelNumericEditField2_5.Text = 'Speed Control [rpm]';

        % Create NumericEditField_Generator_SpeedControl
        app.NumericEditField_Generator_SpeedControl = uieditfield(app.GeneratorControlPanel, 'numeric');
        app.NumericEditField_Generator_SpeedControl.Limits = [0 Inf];
    app.NumericEditField_Generator_SpeedControl.ValueChangedFcn = createCallbackFcn(app, @NumericEditField_Generator_SpeedControlValueChanged,
                                                                  true);
        app.NumericEditField_Generator_SpeedControl.Position = [203 127 50 22];

        % Create LabelNumericEditField2_6
        app.LabelNumericEditField2_6 = uilabel(app.GeneratorControlPanel);
        app.LabelNumericEditField2_6.HorizontalAlignment = 'right';
        app.LabelNumericEditField2_6.VerticalAlignment = 'top';
        app.LabelNumericEditField2_6.Position = [74 47 112 22];
        app.LabelNumericEditField2_6.Text = 'Torque Control [Nm]';

        % Create NumericEditField_Generator_TorqueControl
        app.NumericEditField_Generator_TorqueControl = uieditfield(app.GeneratorControlPanel, 'numeric');
        app.NumericEditField_Generator_TorqueControl.Limits = [0 Inf];
    app.NumericEditField_Generator_TorqueControl.ValueChangedFcn = createCallbackFcn(app, @NumericEditField_Generator_TorqueControlValueChanged,
                                                                  true);
        app.NumericEditField_Generator_TorqueControl.Position = [203 50 50 22];

        % Create SimulationControlPanel
        app.SimulationControlPanel = uipanel(app.InputsPanel);
        app.SimulationControlPanel.Title = 'Simulation Control';
        app.SimulationControlPanel.Position = [791 1 138 123];

        % Create SimStartStopButton
        app.SimStartStopButton = uibutton(app.SimulationControlPanel, 'state');
        app.SimStartStopButton.ValueChangedFcn = createCallbackFcn(app, @SimStartStopButton_callback, true);
        app.SimStartStopButton.Icon = 'IconPlay.png';
        app.SimStartStopButton.IconAlignment = 'center';
        app.SimStartStopButton.Text = '';
        app.SimStartStopButton.BackgroundColor = [0.9412 0.9412 0.9412];
        app.SimStartStopButton.Position = [48 48 34 30];

        % Create StartSimLabel
        app.StartSimLabel = uilabel(app.SimulationControlPanel);
        app.StartSimLabel.Position = [37 21 55 22];
        app.StartSimLabel.Text = 'Start Sim';

        % Create ChannelLogControls
        app.ChannelLogControls = uipanel(app.InputsPanel);
        app.ChannelLogControls.Title = 'Channel Log Control';
        app.ChannelLogControls.Position = [928 1 182 123];

        % Create canLoggingStartStopButton
        app.canLoggingStartStopButton = uibutton(app.ChannelLogControls, 'state');
        app.canLoggingStartStopButton.ValueChangedFcn = createCallbackFcn(app, @canLoggingStartStopButton_callback, true);
        app.canLoggingStartStopButton.Icon = 'IconPlay.png';
        app.canLoggingStartStopButton.Text = '';
        app.canLoggingStartStopButton.BackgroundColor = [0.9412 0.9412 0.9412];
        app.canLoggingStartStopButton.Position = [20 49 36 30];

        % Create saveLoggedDataButton
        app.saveLoggedDataButton = uibutton(app.ChannelLogControls, 'push');
        app.saveLoggedDataButton.ButtonPushedFcn = createCallbackFcn(app, @saveLoggedDataButtonPushed, true);
        app.saveLoggedDataButton.Icon = 'IconBrowse.png';
        app.saveLoggedDataButton.BackgroundColor = [0.9412 0.9412 0.9412];
        app.saveLoggedDataButton.Position = [111 49 36 30];
        app.saveLoggedDataButton.Text = '';

        % Create StartLoggingLabel
        app.StartLoggingLabel = uilabel(app.ChannelLogControls);
        app.StartLoggingLabel.Position = [1 23 73 22];
        app.StartLoggingLabel.Text = 'Start Logging';

        % Create SaveloggedDataLabel
        app.SaveloggedDataLabel = uilabel(app.ChannelLogControls);
        app.SaveloggedDataLabel.Position = [79 23 99 21];
        app.SaveloggedDataLabel.Text = 'Save logged Data';

        % Create ChannelReplayControlPanel
        app.ChannelReplayControlPanel = uipanel(app.InputsPanel);
        app.ChannelReplayControlPanel.Title = 'Channel Replay Control';
        app.ChannelReplayControlPanel.Position = [1108 1 178 123];

        % Create canReplayStartStopButton
        app.canReplayStartStopButton = uibutton(app.ChannelReplayControlPanel, 'state');
        app.canReplayStartStopButton.ValueChangedFcn = createCallbackFcn(app, @canReplayStartStopButton_callback, true);
```

```
            app.canReplayStartStopButton.Icon = 'IconPlay.png';
            app.canReplayStartStopButton.Text = '';
            app.canReplayStartStopButton.BackgroundColor = [0.9412 0.9412 0.9412];
            app.canReplayStartStopButton.Position = [130 45 32 30];

            % Create loadCANLogDataButton
            app.loadCANLogDataButton = uibutton(app.ChannelReplayControlPanel, 'push');
            app.loadCANLogDataButton.ButtonPushedFcn = createCallbackFcn(app, @loadCANLogDataButtonPushed, true);
            app.loadCANLogDataButton.Icon = 'IconFindFiles.png';
            app.loadCANLogDataButton.BackgroundColor = [0.9412 0.9412 0.9412];
            app.loadCANLogDataButton.Position = [42 46 37 30];
            app.loadCANLogDataButton.Text = '';

            % Create LoadloggedDataLabel
            app.LoadloggedDataLabel = uilabel(app.ChannelReplayControlPanel);
            app.LoadloggedDataLabel.Position = [10 22 100 22];
            app.LoadloggedDataLabel.Text = 'Load logged Data';

            % Create StartReplayLabel
            app.StartReplayLabel = uilabel(app.ChannelReplayControlPanel);
            app.StartReplayLabel.Position = [110 22 72 22];
            app.StartReplayLabel.Text = 'Start Replay';

            % Create plottingPanel
            app.plottingPanel = uipanel(app.InputsPanel);
            app.plottingPanel.Title = 'PLOTTING';
            app.plottingPanel.Position = [791 123 138 74];

            % Create LabelDropDown
            app.LabelDropDown = uilabel(app.plottingPanel);
            app.LabelDropDown.HorizontalAlignment = 'right';
            app.LabelDropDown.VerticalAlignment = 'top';
            app.LabelDropDown.Position = [28 31 73 15];
            app.LabelDropDown.Text = 'Time Window';

            % Create plotTimeWindowDropDown
            app.plotTimeWindowDropDown = uidropdown(app.plottingPanel);
            app.plotTimeWindowDropDown.Items = {'10', '20', '50', '100'};
            app.plotTimeWindowDropDown.ValueChangedFcn = createCallbackFcn(app, @plotTimeWindowWidth, true);
            app.plotTimeWindowDropDown.Position = [34 9 63 20];
            app.plotTimeWindowDropDown.Value = '20';

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = GUIsimulinkbancoDefinitivo

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            % Execute the startup function
            runStartupFcn(app, @StartupFcn)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```