ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DIPARTIMENTO DI INGEGNERIA INDUSTRIALE

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA MECCANICA

TESI DI LAUREA

in

SPERIMENTAZIONE E CALIBRAZIONE
DI MOTORI A COMBUSTIONE INTERNA

SVILUPPO DI UN AMBIENTE INTEGRATO PER LA CALIBRAZIONE SEMIAUTOMATICA DI FUNZIONI DI CONTROLLO MOTORE

CANDIDATO
Jacopo Tomei

RELATORE
Chiar.mo Prof. Nicolò Cavina

CORRELATORI
Chiar.mo Prof. Davide Moro
Chiar.mo Prof. Enrico Corti
Ing. Marco Cangini

Anno Accademico 2016/2017

Sessione I

Sommario

Questo lavoro va ad inserirsi nell'ambito della collaborazione che, da qualche anno, Alma Automotive porta avanti con Maserati, e che prevede la realizzazione di un ambiente integrato basato su tecnologia National Instruments e sviluppato in LabVIEW per la calibrazione semiautomatica delle principali funzioni di controllo motore.

Le attività svolte finora si sono concentrate sulla calibrazione del controllo lambda, la cui ottimizzazione sarà peraltro trattata in questa sede, mentre lo scopo principale del presente lavoro sarà lo sviluppo di un tool per l'automazione della calibrazione del controllo VVT per un motore 2.9 V6 turbocompresso che tuttora è in fase di aggiornamento nelle sale prova di Maserati.

Per quanto concerne il presente lavoro di tesi, la fase iniziale, di fondamentale importanza, sarà incentrata sullo studio del software di controllo per poter comprendere il funzionamento del sistema di attuazione e per individuare le curve e le mappe che dovranno essere calibrate.

Si passerà quindi alla fase pratica nella quale si effettueranno delle acquisizioni di dati in sala prove per poter definire gli algoritmi di analisi e validazione dei risultati, responsabili dell'individuazione dei valori da riportare nelle mappe.

In seguito ci si concentrerà sulla realizzazione del tool per l'automazione, non prima di aver definito il procedimento di calibrazione per ogni mappa sulla base di quanto consigliato dal costruttore.

La parte finale del lavoro vedrà l'implementazione dell'interfacciamento tra il tool sviluppato e i vari software deputati alla gestione del banco prova e della centralina, a cui farà seguito la fase di test dell'ambiente di calibrazione così ottenuto.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 Il TAC: struttura e funzionamento

Il TAC (*Tool per Automazione e Controllo*) è stato sviluppato con l'idea di realizzare un sistema in grado di gestire in modo integrato tutto l'hardware presente nelle sale di prova Maserati, rappresentando una sorta di collettore per i dati trasmessi tra i vari devices necessari al controllo del propulsore. Tali dispositivi possono essere suddivisi come segue:

- hardware e software per il controllo e la calibrazione della ECU;
- hardware e software per il controllo del banco prova, costituito dal freno e dai vari sensori presenti in sala;
- hardware e software per l'analisi in tempo reale della combustione.

Nelle sale prova Maserati gli strumenti utilizzati per assolvere i compiti appena elencati sono forniti, rispettivamente, da ETAS per il controllo della centralina (il software è INCA e l'hardware sono i moduli ETK, ES592 e ES910) e da AVL per il controllo del banco (AVL PUMA) e per l'analisi dei dati *indicating* (AVL INDICOM).

Lo sviluppo del TAC è stato portato avanti, fin dall'inizio, sfruttando la piattaforma messa a disposizione da National Instruments, sia per quanto riguarda l'hardware che per quanto riguarda il software, e ciò ha consentito di realizzare un sistema di controllo altamente evoluto ed efficiente in grado di svolgere le seguenti funzioni:

- comunicazione in realtime con tutti i devices presenti in sala;
- sviluppo e integrazione di strategie per l'automazione delle prove;
- sviluppo e integrazione di tool per la calibrazione automatica delle varie funzioni presenti in ECU;

- gestione di tutti i dati scambiati tra i vari devices.

L'obiettivo è chiaro, visto quanto elencato in precedenza: il TAC consente di ridurre notevolmente le operazioni richieste agli operatori di sala e soprattutto i tempi richiesti dai test e dalle procedure di calibrazione, permettendo di conseguenza un aumento dell'efficienza e una riduzione dei costi di esercizio della sala.

La versione iniziale del software prevedeva la comunicazione con la centralina, tramite il tool ETAS MCE, e con il sistema di analisi della combustione consentendo nella pratica lo sviluppo di strategie di automazione e controllo relative al solo motore, mentre la gestione del banco era demandata al personale di sala. In seguito è stata implementata la comunicazione con il sistema di controllo del banco rendendo così possibile automatizzare anche il settaggio del punto motore e il controllo di tutte le operazioni necessarie a mantenere in sicurezza il funzionamento del propulsore, arrivando nella pratica allo schema rappresentato in figura 1.1.

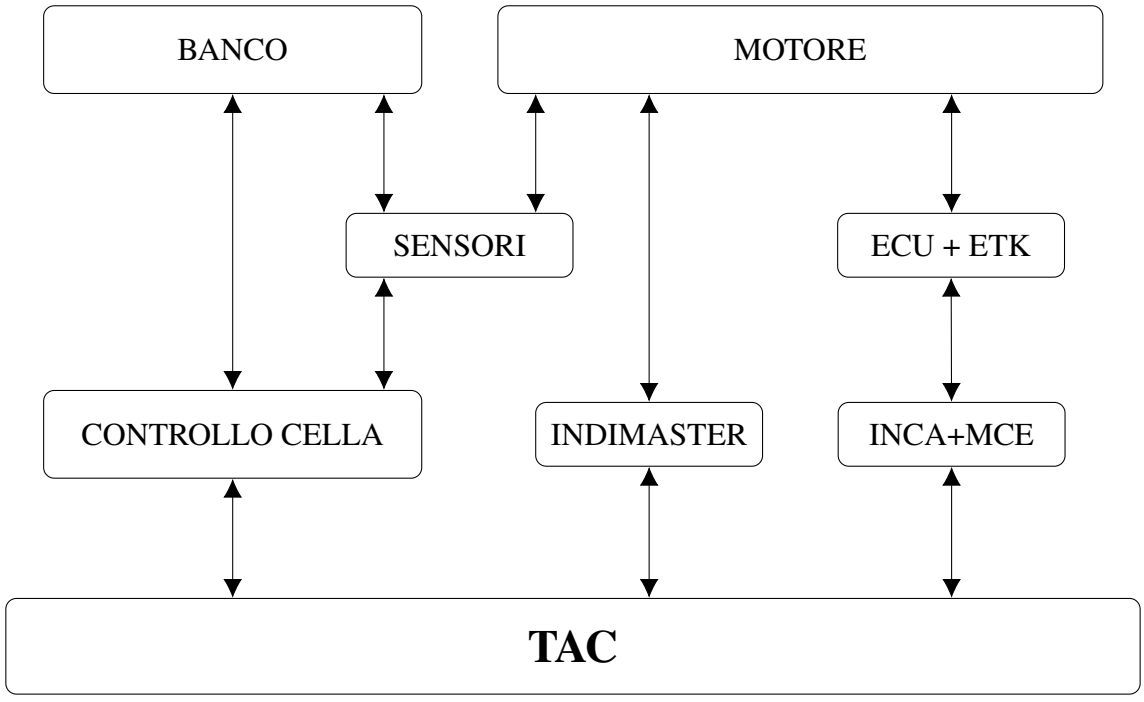


Figura 1.1: Schema applicativo del TAC

Allo stato attuale il TAC è costituito da diversi tool, tra i quali vale la pena ricordare quelli per l'esecuzione di Piani Quotati e Design of Experiment e per la calibrazione del controllo Lambda, ed è in continua espansione: nel presente lavoro verranno descritti nel dettaglio lo sviluppo del tool per la calibrazione della funzione deputata al controllo VVT,

che rappresenta l'attività principale svolta nel periodo di tesi, e l'aggiornamento del tool per la calibrazione della funzione deputata al controllo Lambda.

1.2 Il protocollo di comunicazione XCP

Lo standard attualmente utilizzato per la comunicazione bidirezionale tra la ECU e l'host che gestisce la calibrazione è il protocollo universale XCP, definito dall'ASAM (*Association for Standardization of Automation and Measuring Systems*). Tale protocollo consente l'accesso in tempo reale e durante il funzionamento a tutte le variabili presenti in centralina, e può essere implementato sfruttando diverse tipologie di bus, tra cui CAN ed Ethernet (da qui la lettera 'X' nel nome del protocollo). L'accesso ad una variabile avviene tramite la richiesta da parte del master dell'indirizzo di memoria che la variabile di interesse occupa nello slave, e tale indirizzo è definito per tutte le variabili previste dal software dell'ECU in un file di tipo *a2l*; questo file contiene inoltre le informazioni necessarie alla corretta interpretazione dei dati trasmessi via XCP, tra cui le unità di misura delle varie grandezze, la loro risoluzione, il range di valori impostabili per ogni calibrazione e le dimensioni di tutte le curve e di tutte le mappe.

Il protocollo XCP nasce dal protocollo CCP, che invece era specifico per la comunicazione via CAN, ed è stato sviluppato con i seguenti obiettivi:

- riduzione del carico richiesto al dispositivo slave, in termini di utilizzo della CPU, della RAM e della memoria fisica;
- elevata velocità di trasmissione dei dati;
- riduzione dell'utilizzo del bus di comunicazione;
- configurazione plug-and-play;
- scalabilità, ovvero la possibilità di utilizzo senza dover implementare tutti i comandi descritti nello standard;
- trasferibilità, ovvero la possibilità di essere utilizzato su diversi tipi di bus.

1.2.1 Il modello di comunicazione: *master* e *slave*

Il protocollo XCP si basa su una gerarchia di tipo *master-slave*: il sistema di calibrazione e di controllo è il dispositivo master, mentre la ECU è il dispositivo slave. È interessante sottolineare che ogni slave può comunicare con un solo master alla volta, mentre il master può comunicare contemporaneamente con diversi slave.

I dati vengono trasmessi via XCP tramite un sistema basato su messaggi, cioè tutti i pacchetti inviati sono contenuti in un frame del sistema di trasporto. Tale frame è costituito da tre parti come si può vedere nella figura 1.2: *XCP header*, *XCP packet* e *XCP tail*. Di queste tre solo l'XCP packet è indipendente dal protocollo di trasporto, ed è a sua volta costituito da altrettanti parti che sono l'*Identification Field*, il *Timestamp Field* e il *Data Field*.

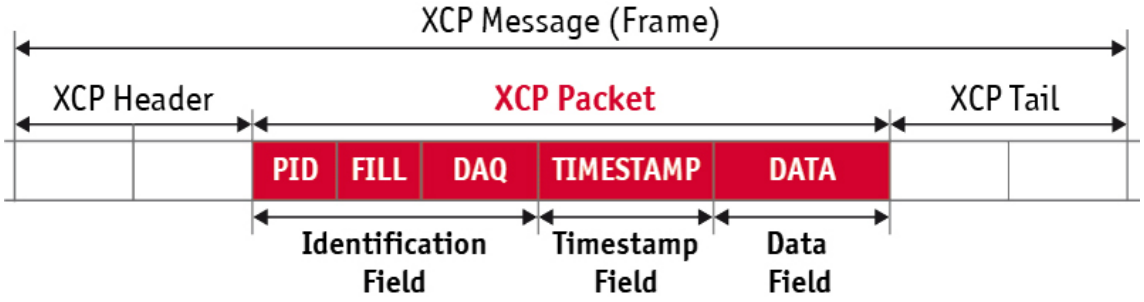


Figura 1.2: Struttura di un messaggio inviato tramite XCP

1.2.2 L'acquisizione dei dati

Come accennato in precedenza l'utilizzo del protocollo XCP consente di abilitare l'accesso ai dati contenuti nella memoria della ECU, e tale accesso può essere di due tipi:

- *measurement*, ovvero in lettura, che consente all'utente di leggere i valori che assumono nel tempo le variabili della ECU;
- *calibration*, ovvero in scrittura, che consente all'utente di modificare e ottimizzare i parametri delle funzioni di controllo.

La lettura delle *measurements* può avvenire in due modi:

- tramite *polling*, ovvero inviando richieste periodiche dal master allo slave; l'aspetto negativo di questo metodo è che l'invio di un comando dal master e la ricezione della risposta dallo slave comporta la generazione di almeno due messaggi, con la conseguenza che il tempo di acquisizione non risulta sincronizzato con il tempo di ciclo interno della ECU;
- tramite *synchronous data acquisition* (*DAQ*): questo metodo consente di superare il problema tipico della modalità di polling in quanto il dispositivo slave memorizza tutti i valori delle *measurements* in un buffer, che poi invia al dispositivo master non appena questo richiede il campionamento dei dati. Questa configurazione prevede una prima fase di impostazione della *DAQ list*, che consiste nell'invio da parte del

dispositivo master di una serie di comandi per definire le variabili che interessa acquisire; dopo che questa fase è stata completata il master può richiedere l'invio delle *measurements*. Lo slave, da parte sua, non appena riceve la richiesta del master inizia a mandare tutte le variabili definite nella DAQ list, e continuerà la trasmissione senza ulteriori richieste da parte del dispositivo master.

Per quanto riguarda le *calibrations*, invece, queste sono parametri costanti sui quali il calibratore agisce in modo da ottimizzare il funzionamento del propulsore; per dare la possibilità di modificare tali parametri durante il funzionamento stesso della ECU è necessario allocare ulteriore spazio nella memoria RAM. L'approccio più efficiente consiste nell'inizializzare in uno spazio dedicato della RAM tutti i parametri previsti dal software dell'ECU con i valori iniziali memorizzati nella flash memory; dopodiché, terminata la fase di booting della centralina, il master potrà accedere a tale spazio di memoria, chiamato per questo *calibration RAM*, per modificare i parametri desiderati.

Il punto di forza del protocollo XCP risiede nella possibilità di acquisire *measurements* e *calibrations* dalla memoria dell'ECU solo a specifici istanti di tempo oppure in funzione di determinati eventi. I rasters, che poi altro non sono che le DAQ lists nominate in precedenza, messi a disposizione dall'hardware ETAS sono tre:

- *time synchronous* a 10 millisecondi;
- *time synchronous* a 100 millisecondi;
- *segment synchronous*: prevede un riferimento angolare, varia con il regime di rotazione del motore.

1.3 Il protocollo di comunicazione iLinkRT

Il protocollo iLinkRT è un'estensione del protocollo XCP ed è configurato come segue:

- il time-out che lo slave deve attendere dopo aver ricevuto un comando dal master a cui verrà risposto negativamente è pari a 1 secondo;
- l'ordine dei bit che compongono un byte segue la logica MSB (*Most Significant Bit*), che prevede il posizionamento del bit più significativo (ovvero quello di valore maggiore) all'inizio del byte;
- sono supportate solo le DAQ list statiche, e inoltre il dispositivo slave non ha la possibilità di configurare le DAQ list autonomamente che devono essere modificate solo tramite INCA;

- per ogni evento (o raster, che sono tre) esiste una sola DAQ list, ed ognuna di queste DAQ list può contenere al massimo 256 variabili;
- la comunicazione con il modulo ES910 può essere solo di tipo interleaved, e non standard o a blocchi. Le differenze tra questi tre tipi di comunicazione sono rappresentate in figura 1.3:
  - la modalità *standard* prevede che ogni pacchetto di richiesta sia seguito da un pacchetto di risposta, e inoltre il dispositivo master non può inviare un nuovo comando finché non ha ricevuto dallo slave la risposta del comando precedente;
  - la modalità *interleaved* consente di velocizzare l'invio e la ricezione di pacchetti dato che il master può continuare a mandare comandi anche senza aver ricevuto le risposte precedenti;
  - la modalità *block* viene utilizzata per trasferire grandi quantità di dati, e permette di ottenere diversi pacchetti in risposta a un singolo comando.

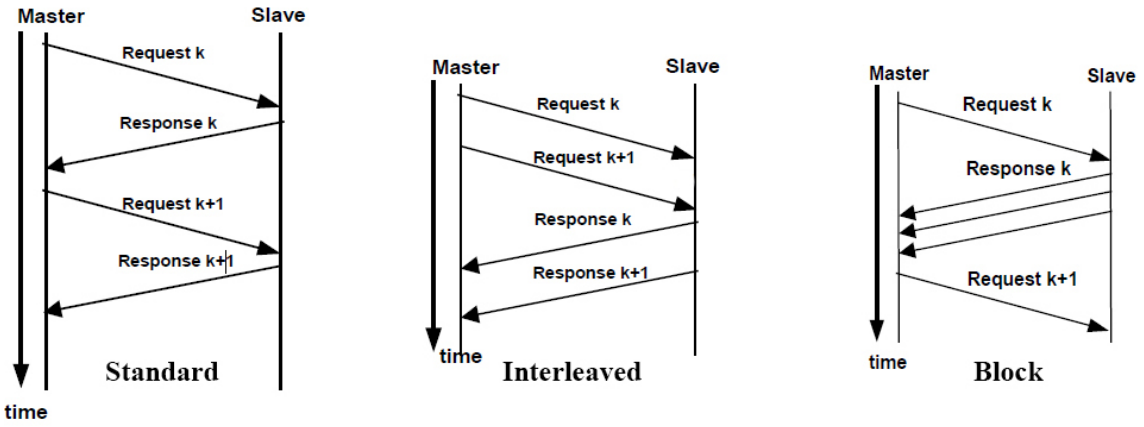


Figura 1.3: Tipologie di comunicazione disponibili nell'XCP

1.3.1 I comandi disponibili in iLinkRT

Il protocollo iLinkRT riprende dall'XCP la possibilità di comunicare in parallelo tramite due porte differenti, delle quali una è destinata all'acquisizione delle *measurements*, mentre l'altra viene impiegata per inviare e ricevere i comandi e le *calibrations*. I comandi disponibili in iLinkRT sono suddivisi in due set:

- il primo deriva dallo standard XCP e prevede tra gli altri i seguenti comandi:
  - *CONNECT*: stabilisce la connessione tra master e slave;

  - *GET_STATUS*: ottiene dallo slave tutte le informazioni sul suo stato;
  - *START_STOP_DAQ_LIST*: avvia o arresta l'invio delle *measurements*;

- il secondo è stato implementato da ETAS e AVL per ottimizzare la comunicazione ed è costituito dai seguenti comandi:

  - *READ_DAQ_EXTENDED*: richiede allo slave informazioni sulle variabili trasmesse nella DAQ list, tra cui il nome, il valore e l'unità di misura;
  - *CAL_DOWNLOAD_ADVANCED*: scarica dal master sullo slave i valori dei parametri specificati;
  - *READ_CAL_EXTENDED_V2*: richiede allo slave informazioni sui parametri che possono essere modificati;
  - *CAL_UPLOAD_ADVANCED_V2*: trasmette dallo slave al master il valore di calibrazione attuale dei parametri specificati.

Può essere utile descrivere nel dettaglio i comandi presenti nel set custom di ETAS dato che il team di Alma Automotive ha realizzato una serie di VI, ampiamente utilizzati nel TAC, per implementarli in ambiente LabVIEW.

READ_DAQ_EXTENDED

Il comando READ_DAQ_EXTENDED deve essere inviato dal dispositivo master seguendo la struttura riportata nella tabella 1.1, mentre la risposta positiva da parte dello slave avrà la struttura riportata nella tabella successiva 1.2:

Posizione	Tipo	Descrizione	Valore
0	BYTE	PID = 0xF1	241
1	BYTE	Subcommand 0x02	2

Tabella 1.1: Struttura del comando READ_DAQ_EXTENDED

Posizione	Tipo	Descrizione
0	BYTE	PID = 0xFF (risposta positiva)
1	BYTE	Estensione dell'indirizzo
$2-5$	4 BYTE	Indirizzo dell'elemento nella DAQ list
6..n	STRING	Nome della DAQ label
n + 1..m	STRING	Unità della DAQ label
m + 1..k	STRING	Descrizione della DAQ label
k + 1	BYTE	Data Type Body

Tabella 1.2: Struttura della risposta al comando READ_DAQ_EXTENDED

CAL_DOWNLOAD_ADVANCED

È il comando previsto da iLinkRT per scrivere in centralina i valori di calibrazione impostati dall'utente; prevede due modalità di funzionamento:

- *flat mode*: viene generalmente utilizzato per gli scalari ma è utile anche per spianare una curva o una mappa al valore prescelto;
- *area mode*: permette di modificare singolarmente ogni elemento di una curva o di una mappa, oppure di intervenire contemporaneamente su una porzione di essa passando al comando i valori da caricare sotto forma di vettore orientato per righe.

Le tabelle seguenti riportano le strutture per implementare il comando, sia in modalità *flat* che in modalità *area*, mentre la risposta dello slave non contiene alcun dato se non la conferma che la scrittura dei parametri è andata a buon fine.

Posizione	Tipo	Descrizione	Valore
0	BYTE	PID = 0xF1	241
1	BYTE	Subcommand 0x04	4
2	BYTE	Estensione dell'indirizzo	0
$3-6$	4 BYTE	Indirizzo	a a a a
7	BYTE	Modalità	0
8..	ELEMENT	Dati da caricare	z z z z

Tabella 1.3: Struttura del comando CAL_DOWNLOAD_ADVANCED in modalità *flat*

Posizione	Tipo	Descrizione	Valore
0	BYTE	PID = 0xF1	241
1	BYTE	Subcommand 0x04	4
2	BYTE	Estensione dell'indirizzo	0
$3-6$	4 BYTE	Indirizzo	a a a a
7	BYTE	Modalità	3
8,9	WORD	Start x	x x
10,11	WORD	Start y	y y
12,13	WORD	Dimensione lungo x	X X
14,15	WORD	Dimensione lungo y (1 per curve)	Y Y
16..	ELEMENTS	Dati da caricare	z z z z...

Tabella 1.4: Struttura del comando CAL_DOWNLOAD_ADVANCED in modalità *area*

Posizione	Tipo	Descrizione
0	BYTE	PID = 0xFF (risposta positiva)

Tabella 1.5: Struttura della risposta al comando CAL_DOWNLOAD_ADVANCED

READ_CAL_EXTENDED_V2

Questo comando deve essere inviato almeno una volta prima di poter avere accesso alle *calibrations*: bisogna comunque sottolineare che l'indice dell'elemento richiesto, corrispondente ai byte 3 e 4 in tabella 1.6, se non specificato viene incrementato automaticamente ogni volta che si invia il comando, cosa che permette di leggere tutte le variabili presenti senza la necessità di definire un puntatore come invece accade per il comando READ_DAQ_EXTENDED. Si fa inoltre notare che il byte 2 può essere impostato a 0 oppure a 1 in funzione del numero di centraline che controllano il motore; nel nostro caso, dato che il propulsore in prova possiede due centraline, è possibile conoscere i valori delle *calibrations* sia della ECU 1 che della ECU 2 semplicemente cambiando tale byte.

Posizione	Tipo	Descrizione	Valore
0	BYTE	PID = 0xF1	241
1	BYTE	Subcommand 0x06	6
2	BYTE	Device ID	0/1
3,4	WORD	Numero della *calibration*	n

Tabella 1.6: Struttura del comando READ_CAL_EXTENDED_V2

Posizione	Tipo	Descrizione
0	BYTE	PID = 0xFF (risposta positiva)
1	BYTE	Estensione dell'indirizzo
$2-5$	4 BYTE	Indirizzo della *calibration*
6..n	STRING	Nome della calibration label
n + 1..m	STRING	Unità della calibration label
m + 1..k	STRING	Descrizione della calibration label
k + 1	BYTE	Dimensioni del parametro, tipo di label
k + 2	BYTE	Data Type Body
k + 3	BYTE	Metodi di calibrazione accettati
k + 4	WORD	Dimensioni dell'asse x per mappe, curve e array
k + 6	WORD	Dimensioni dell'asse y per mappe e array 2D
k + 8	BYTE	Data Type Body per mappe e curve
k + 9	BYTE	Data Type Body per mappe

Tabella 1.7: Struttura della risposta al comando READ_CAL_EXTENDED_V2

CAL_UPLOAD_ADVANCED_V2

Dopo aver ottenuto la lista dei parametri di calibrazione disponibili con il comando precedente è possibile leggerne i valori tramite il comando CAL_UPLOAD_ADVANCED_V2, il quale prevede due modalità:

- *mode 3*, che non prevede l'invio dei valori degli assi di riferimento della curva o della mappa richiesta;
- *mode 4*, che invece prevede anche l'invio degli assi; è quella di default utilizzata nel TAC.

Il comando ha la struttura riportata nella tabella 1.8, mentre la risposta inviata dallo slave ha la struttura mostrata nella tabella 1.9. Si noti che tale comando permette di importare a scelta tutta la mappa, impostando 'Start x' e 'Start y' a 0, oppure una porzione di essa; nel caso in cui si voglia importare uno scalare è sufficiente impostare entrambi i valori di 'Size' a 1:

Posizione	Tipo	Descrizione	Valore
0	BYTE	PID = 0xF1	241
1	BYTE	Subcommand 0x07	7
2	BYTE	Estensione dell'indirizzo	0
$3-6$	4 BYTE	Indirizzo	a a a a
7	BYTE	Modalità	4
8,9	WORD	Start x	x x
10,11	WORD	Start y	y y
12,13	WORD	Dimensione lungo x	X X
14,15	WORD	Dimensione lungo y (1 per curve)	Y Y

Tabella 1.8: Struttura del comando CAL_UPLOAD_ADVANCED_V2

Posizione	Tipo	Descrizione
0	BYTE	PID = 0xFF (risposta positiva)
6..n	ELEMENTS	Scalare, curva o mappa richiesta (dimensione $\mathrm{x} \cdot \mathrm{y}$)
$\mathrm{m}+1$..n	ELEMENTS	Dati lungo x
$\mathrm{n}+1$..o	ELEMENTS	Dati lungo y

Tabella 1.9: Struttura della risposta al comando CAL_UPLOAD_ADVANCED_V2

1.3.2 L'implementazione dei comandi in LabVIEW

Per ogni comando si è scelto di realizzare due SubVI: uno per la generazione del comando stesso da inviare al dispositivo slave a partire dagli input scelti dall'utente (identificato con la lettera 'W'), e l'altro per la decodifica della risposta fornita dallo slave (identificato con la lettera 'R').

Nelle figure che seguono si riportano i due VI relativi al comando READ_DAQ_EXTENDED, in figura 1.4, i due VI relativi al comando CAL_DOWNLOAD_ADVANCED, in figura 1.5, i due VI relativi al comando READ_CAL_EXTENDED, in figura 1.6, e infine i due VI relativi al comando CAL_UPLOAD_ADVANCED, in figura 1.7.

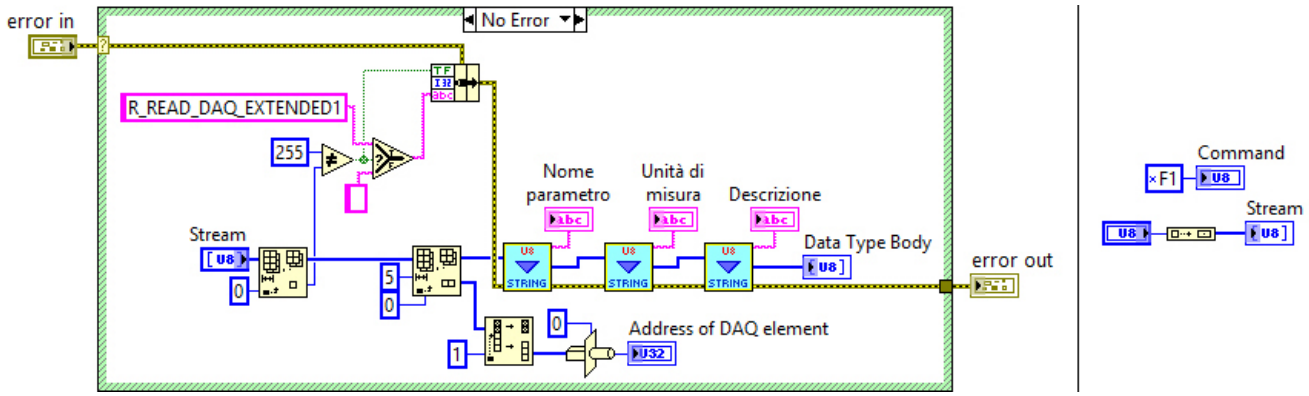


Figura 1.4: VI per il comando READ_DAQ_EXTENDED (‘R’ a sinistra e ‘W’ a destra)

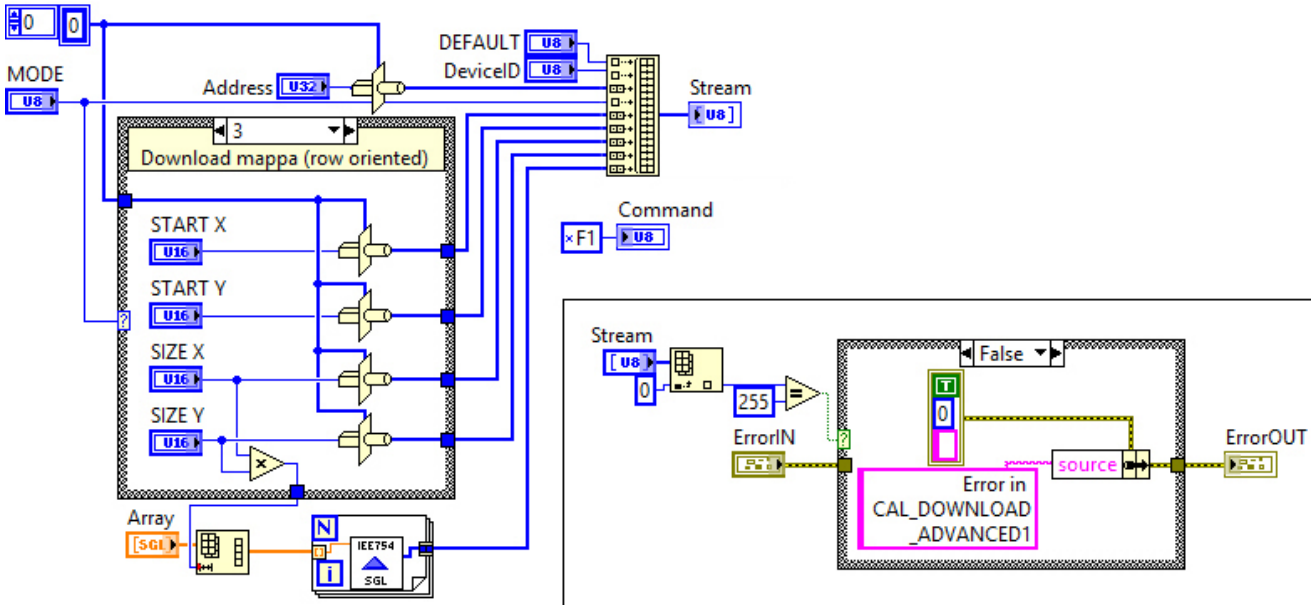


Figura 1.5: VI per il comando CAL_DOWNLOAD_ADVANCED (‘W’ a sinistra e ‘R’ a destra)

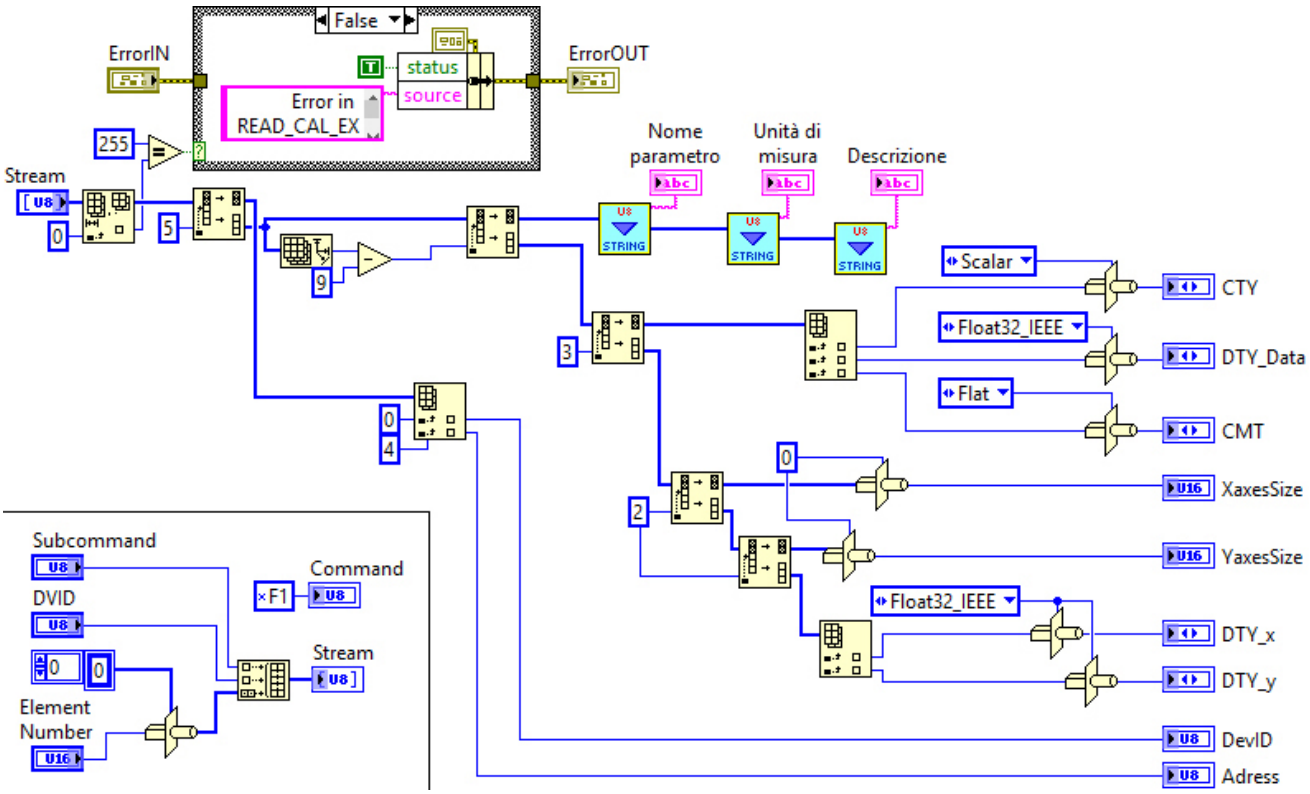


Figura 1.6: VI per il comando READ_CAL_EXTENDED (‘W’ a sinistra e ‘R’ a destra)

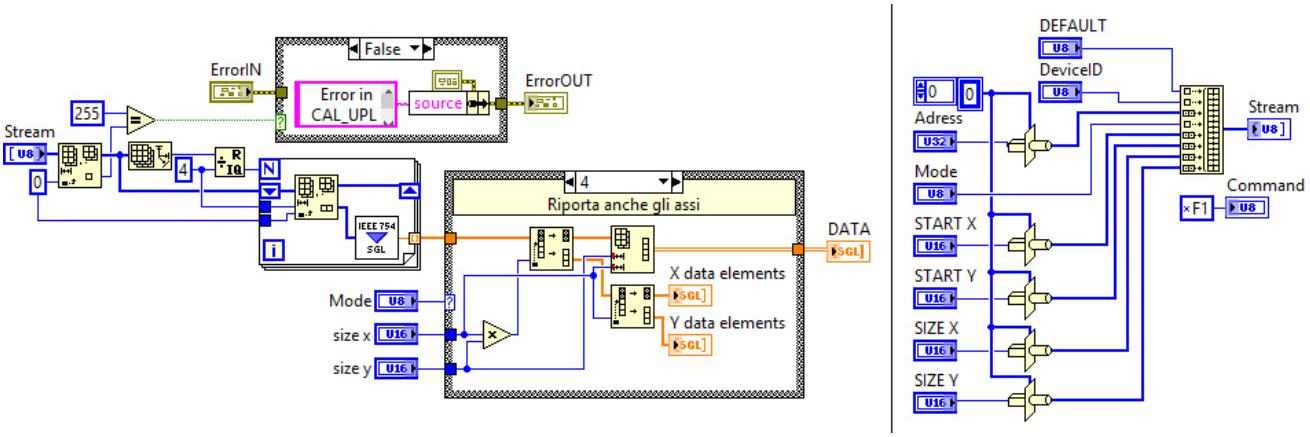


Figura 1.7: VI per il comando CAL_UPLOAD_ADVANCED (‘R’ a sinistra e ‘W’ a destra)

Il comando READ_DAQ_EXTENDED non ha input controllabili dall'esterno, pertanto il VI 'W' contiene solo la specifica per il comando e il subcomando; il VI 'R' prevede la divisione dell'array di byte in input in più parti separando le varie informazioni che contiene, dopodiché ogni parte viene opportunamente convertita tramite il SubVI 'ConvertU8toString'.

Per quanto riguarda il comando CAL_DOWNLOAD_ADVANCED è interessante notare che nella modalità 3 la mappa in input viene convertita in array tramite il blocco 'Reshape Array' prima di essere trasformata in notazione IEEE754.

Si vuole infine sottolineare che in tutti i VI illustrati è presente un'accurata gestione degli errori, implementata tramite le apposite *case structures*, in modo da consentire un'efficiente operazione di debug allo sviluppatore e tenere sempre aggiornato l'utente sul corretto flusso di dati; qualora uno dei VI incontri un errore in seguito all'invio di un comando errato o alla ricezione di una risposta negativa è sempre possibile stabilire quale VI ne sia stato la causa grazie alla generazione di messaggi di errore personalizzati.

1.3.3 La conversione dei dati trasmessi

Prima di procedere con la trattazione è necessario premettere che iLinkRT sfrutta lo standard IEEE per la formattazione e la codifica dei dati: tale standard consente di rappresentare un numero di tipo *single-precision floating point* secondo la notazione che segue:

$$(-1)^s \cdot 2^E \cdot M$$

dove s rappresenta il segno, E è l'esponente e M è la mantissa. Ogni numero richiede 32 bit per la sua rappresentazione, ovvero 4 byte, e tali bit vengono divisi in tre parti come segue:

- 1 bit per il segno;
- 8 bit per l'esponente;
- 23 bit per la mantissa.

I dati inviati e ricevuti tramite il protocollo iLinkRT devono pertanto essere codificati come array di byte e quindi si è reso necessario realizzare un VI ('SGLtoIEEE754') in grado di convertire nel formato richiesto i valori passati dall'utente come input ai comandi, e un VI ('IEEE754toSGL') per convertire i valori ottenuti come risposta dalla ECU in numeri decimali; si è dovuto inoltre creare un VI ('ConvertU8toString') per trasformare gli array di byte in stringhe in modo da poter decodificare i nomi delle variabili. Questi VI sono rappresentati, dall'alto verso il basso, in figura 1.8.

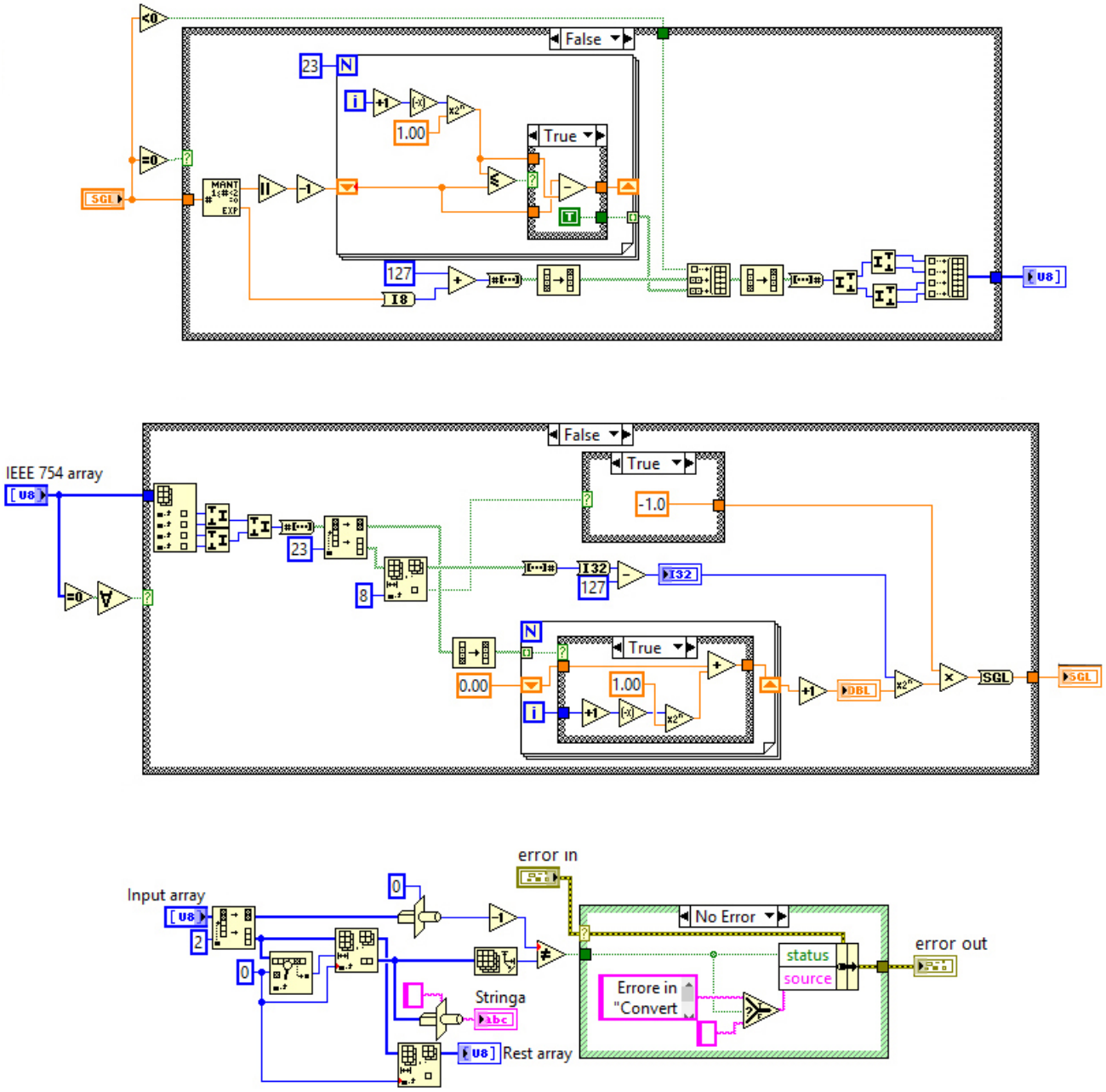


Figura 1.8: VI per la conversione dei dati: 'SGLtoIEEE754', 'IEEE754toSGL' e 'ConvertU8toString'

1.3.4 La comunicazione UDP/IP

L'ultima fase necessaria a stabilire la comunicazione tra il TAC e la centralina è l'implementazione del sistema di comunicazione: si è scelto in tal senso di creare una comunicazione di tipo UDP (*User Datagram Protocol*), un protocollo di trasporto a pacchetto che consente un'elevata velocità di trasmissione, essendo privo di latenza, a fronte tuttavia di un'affidabilità non troppo elevata dato che non è in grado di verificare l'avvenuta trasmissione dei dati e di ritrasmettere gli eventuali pacchetti persi. Si tratta in definitiva di un protocollo decisamente flessibile che sfrutta generalmente il protocollo di rete IP: per stabilire un collegamento tra client e server è sufficiente impostare l'indirizzo IP del client e la porta UDP del server da cui verranno inviati i messaggi.

LabVIEW mette a disposizione una palette di strumenti per la creazione di una comunicazione UDP, e tale processo prevede l'apertura della porta UDP necessaria alla comunicazione (nel seguito si vedrà che tale porta è sempre la 8090) tramite il blocco 'Open UDP Port', quindi l'invio della richiesta di connessione da parte del TAC (client) verso il modulo ES910.3 (server) tramite il comando CONNECT; una volta avvenuta la connessione è possibile inviare e ricevere comandi tramite i due blocchi 'UDP Write' e 'UDP Read'. Si vuole inoltre sottolineare che al fine di prevenire eventuali errori di trasmissione è stato necessario implementare un VI che verificassero la corretta trasmissione dei pacchetti tramite il confronto dei counter dei messaggi inviati e ricevuti.

1.4 La sala prove Maserati: dispositivi e collegamenti

1.4.1 I dispositivi utilizzati

In figura 1.9 è rappresentato il layout schematico dei collegamenti tra i vari dispositivi presenti nella sala prove Maserati utilizzata nell'ambito di questo lavoro. Il cuore dell'intero sistema di calibrazione è rappresentato dall'hub ethernet, al quale vengono collegati tutti i devices necessari alla gestione del motore e del processo di calibrazione:

- ECU xETK: è la centralina di controllo del motore, alla quale è stato aggiunto l'xETK. Si tratta di un modulo aggiuntivo applicato all'hardware della ECU standard che consente l'accesso a tutte le variabili e le calibrazioni lette e scritte dalla centralina durante il suo funzionamento, e prevede inoltre un'interfaccia Ethernet in grado di comunicare tramite il protocollo XCP. Questo permette a più dispositivi host esterni di collegarsi simultaneamente alla ECU, fino a un massimo di quattro, consentendo inoltre la trasmissione dei dati ad alta velocità e la possibilità di connettersi direttamente a un PC esterno senza la necessità di ulteriori moduli. In figura

sono rappresentate due ECU con xETK in quanto il motore Alfa Romeo possiede una centralina per ogni bancata;

- ES592: è un device prodotto da ETAS per la calibrazione, diagnostica e programmazione della centralina. Possiede interfacce per collegamenti via CAN, LIN ed Ethernet e consente la connessione tra INCA e l'xETK per l'acquisizione delle *measurements* e la modifica delle *calibrations*;

- ES910.3: è un device prodotto da ETAS per la prototipazione rapida di software per l'automazione, evoluzione del modulo ES592. In altri termini l'ES910.3 permette l'accesso all'ECU da parte di applicativi esterni alla centralina stessa e diversi da INCA, consentendo nella pratica lo scambio di dati tra la ECU, il tool per la calibrazione automatica e il banco tramite il protocollo iLinkRT; necessita di un tool di INCA chiamato MCE che verrà illustrato nel capitolo seguente;

- PC host/RT: è il PC realtime, basato su hardware National Instruments, sul quale viene eseguito il tool per la calibrazione automatica delle funzioni motore;

- PC INCA: è il PC sul quale viene eseguito INCA, il software proprietario prodotto da ETAS per il controllo e la gestione della ECU.

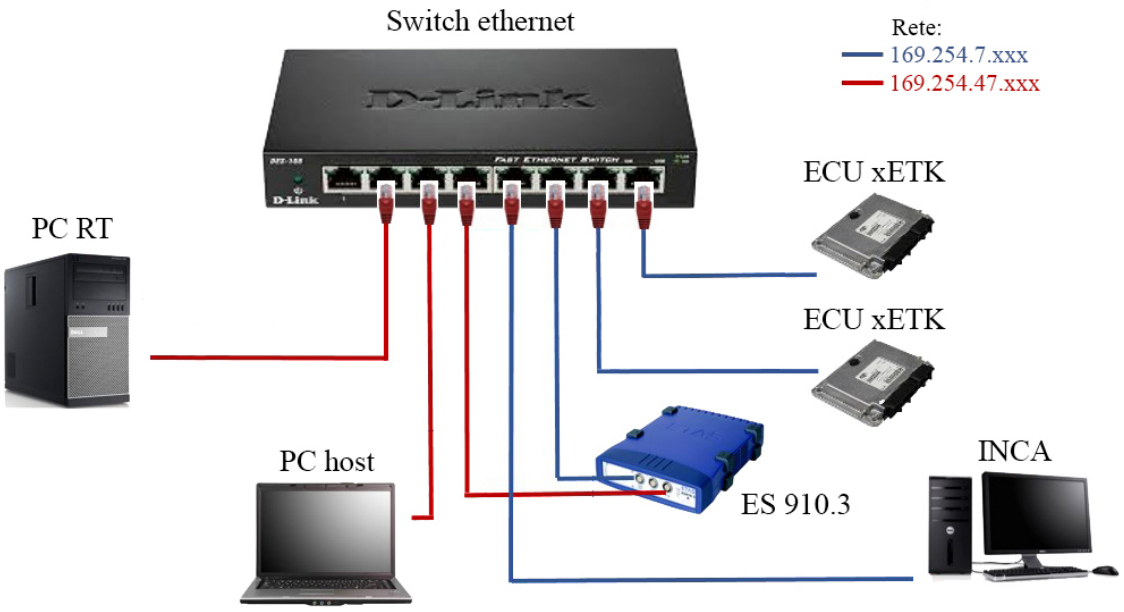


Figura 1.9: Schema dei collegamenti della sala prove in Maserati

È opportuno evidenziare la presenza di due sottoreti diverse, alle quali appartengono dispositivi diversi: la sottorete rossa, costituita dai PC host e realtime, e la sottorete blu,

costituita dal modulo ETAS, dal PC INCA e dalle due ECU. Si noti inoltre che il modulo ETAS è connesso a entrambe le sottoreti, come è lecito aspettarsi, in modo da poter mettere in comunicazione la centralina motore sia con il PC realtime che con INCA.

1.4.2 Il layout di base per la calibrazione

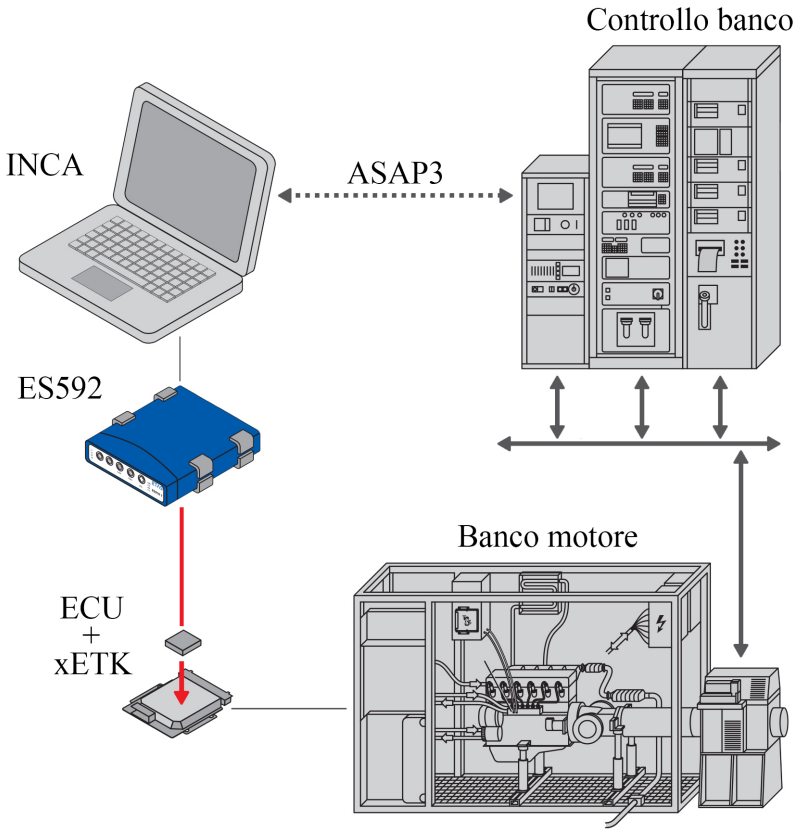


Figura 1.10: Layout con il modulo ES592

Il sistema di calibrazione di base utilizzato nelle sale prova Maserati è riportato nella figura 1.10: con questa configurazione il calibratore può accedere alla ECU tramite INCA e il modulo ES592, mentre INCA è a sua volta controllato dal software di gestione del banco. Un simile layout presenta alcuni limiti che dovranno essere superati se si vuole implementare un sistema di automazione della calibrazione, e che possono essere riassunti come segue:

- velocità di trasferimento dei dati relativamente bassa (sono necessari 100 ms per l'upload di una mappa 16x16 in ECU;
- tempi di upload variabili a causa del jitter, ovvero dei ritardi introdotti dal sistema di collegamento a causa di una gestione poco efficiente delle code;

- ASAP3 è un protocollo di comunicazione a blocchi, e ciò impedisce l'acquisizione delle *measurements* durante la scrittura delle *calibrations* e viceversa;

- ASAP3 non supporta una gerarchia multi-master, impedendo di fatto l'accesso allo slave da più di un master alla volta.

1.4.3 Il layout avanzato per l'automazione

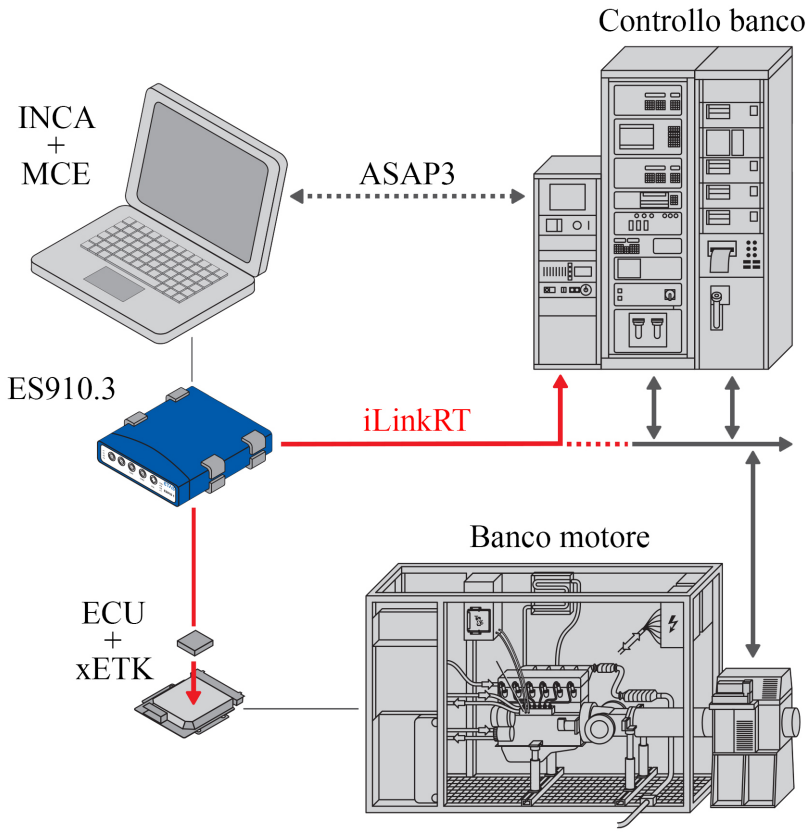


Figura 1.11: Layout con il modulo ES910.3

Grazie all'introduzione del modulo ES910.3 e all'adozione del protocollo iLinkRT ETAS ha reso possibile il superamento di tutti i problemi elencati in precedenza, ottenendo il layout rappresentato in figura 1.11 e sviluppando di fatto un nuovo sistema di comunicazione tra INCA e un qualsiasi software di automazione custom, rendendo tuttavia necessario l'utilizzo del tool MCE per la gestione e lo scambio di *measurements* e *calibrations* tra i vari devices presenti in sala.

La configurazione dell'intero sistema prevede che l'utente scelga quali variabili, sia in lettura che in scrittura, si vogliono controllare aggiungendole tramite MCE: le *measurements* e le *calibrations* scelte saranno rese disponibili a qualunque software di automazio-

ne in grado di accedere alla rete, il quale potrà acquisirle e modificarle ad elevata velocità grazie al protocollo iLinkRT.

In definitiva le due interfacce iLinkRT e ASAP3 svolgono compiti analoghi ma con modalità diverse, e nulla vieta loro di coesistere: si può scegliere di destinare MCE e iLinkRT all'uso con i sistemi di automazione, in modo da poter sviluppare algoritmi di controllo e calibrazione in realtime, lasciando al protocollo ASAM tutte le altre operazioni di routine, come la registrazione, la configurazione dell'hardware e l'upload di software. La sostanziale differenza tra i due tipi di collegamenti è che ASAP interviene sull'interfaccia host per la comunicazione, mentre iLinkRT/MCE agisce direttamente sul modulo ES910.3, riducendo sensibilmente i tempi di trasferimento dei dati e rendendo la connessione più stabile.

1.5 Il motore Alfa Romeo 2.9 V6 T

Il propulsore impiegato per tutte le attività di analisi e test descritte in questa tesi è quello che equipaggia l'Alfa Romeo Giulia in versione Quadrifoglio; di origine Ferrari, è un'unità moderna ad elevate prestazioni ed è controllata da due centraline Bosch. Presenta una cilindrata di 2.9 litri, possiede sei cilindri suddivisi in due bancate disposte a V di 90° ed è dotata di iniezione diretta e di sovralimentazione, realizzata mediante due turbocompressori ognuno dei quali alimenta una bancata. Nella tabella seguente sono riportati alcuni dati relativi al propulsore:

Cilindrata	2.891 cc
Architettura	6 cilindri a V di $90°$, 4 valvole per cilindro
Potenza massima	510 CV (375 kW) @ 6500 rpm
Coppia massima	600 Nm @ $2500 - 5500$ rpm
Alesaggio	86.5 mm
Corsa	82 mm
Rapporto di compressione	$9.3 : 1$
Distribuzione	DOHC con variatore di fase in aspirazione e scarico
Sistema di iniezione	Iniezione diretta (200 bar di pressione massima)
Alimentazione	Sovralimentazione mediante due turbocompressori Turbine single-scroll, intercooler aria/acqua
Costruzione	Testa: lega di alluminio; Banco: lega di alluminio con bronzine in acciaio; Basamento: lega di alluminio; Albero a gomiti: acciaio forgiato, nitrurazione finale; Pistoni: alluminio forgiato.

Tabella 1.10: Specifiche del motore Alfa Romeo 2.9 V6 T

Capitolo 2

Software utilizzati

2.1 MATLAB

MATLAB è un ambiente matematico di programmazione e analisi dati che è stato impiegato in questo lavoro per la messa a punto degli algoritmi di trattamento dei dati acquisiti al banco di prova. La caratteristica interessante di MATLAB consiste nel poter scrivere dei veri e propri programmi, definiti *script*, che integrano le più varie funzioni matematiche, interfacce grafiche e grafici di output. In figura 2.1 si può vedere l'ambiente di lavoro tipico, costituito dai seguenti elementi:

1. *Browser*: consente di esplorare il contenuto dell'hard disk, aprire e modificare script, caricare file di dati di tipo *mat* e aprire grafici di tipo *fig* salvati in precedenza;

2. *Workspace*: mostra tutte le variabili correntemente presenti nella memoria, che possono essere definite manualmente dall'utente tramite la *command window* oppure allocate dall'ultimo script eseguito. L'interfaccia utente del *workspace* consente inoltre di aprire e modificare i vettori e le matrici nonché di esplorare le strutture in modo semplice e intuitivo;

3. *Command Window*: è lo strumento principale di MATLAB in quanto consente all'utente di definire variabili, richiamare funzioni, visualizzare l'output degli script, aprire finestre grafiche ed eseguire in tempo reale qualsiasi comando implementato in MATLAB;

4. *Editor*: consente di creare e modificare gli script, definire nuove *functions* e creare classi e librerie di funzioni. Le versioni più recenti di MATLAB hanno introdotto funzionalità di formattazione avanzate per gli script in modo da poter esportare i listati in vari formati e integrarli in report e presentazioni con tanto di grafici;

5. *Graphics*: è la finestra che permette di vedere output grafici come plot, istogrammi, interfacce GUI e che contiene tutti i tool per la modifica e la formattazione di grafici 2D e 3D. In particolare è possibile gestire più grafici nella stessa finestra, modificare titolo del grafico e nomi degli assi, il colore, lo spessore e lo stile delle linee, il contenuto e la posizione della legenda.

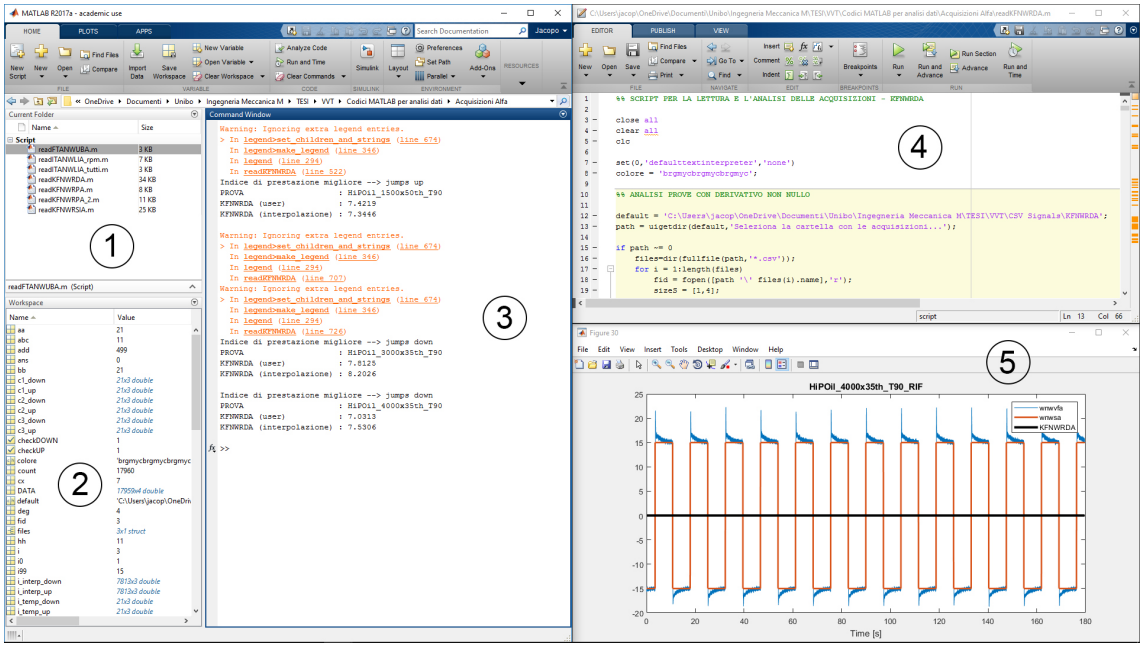


Figura 2.1: Interfaccia di MATLAB

MATLAB si è rivelato uno strumento fondamentale nella definizione degli algoritmi di analisi dei dati e trattamento dei segnali acquisiti durante i test al banco: grazie alla notevole flessibilità presentata dall'ambiente di lavoro è stato possibile scrivere dei codici che implementassero al tempo stesso la conversione delle acquisizioni in un formato compatibile con MATLAB, le operazioni preliminari di trattamento dei segnali e la parte vera e propria di analisi dei dati, in modo da seguire il processo di calibrazione dall'inizio alla fine in modo efficiente ed integrato. Gli script impiegati in questa attività sono riportati integralmente in appendice e dettagliatamente spiegati nel capitolo relativo alla calibrazione del sistema di controllo del VVT, e serviranno da base per la realizzazione del tool di calibrazione automatica realizzato in LabVIEW.

2.1.1 Simulink

Simulink è un software di programmazione grafica strutturata a blocchi presente in MATLAB che consente la progettazione e la simulazione di modelli matematici e sistemi fisici in modo interattivo. L'interfaccia utente, rappresentata in figura 2.2, offre un'ampia

libreria di blocchi in parte predefiniti e in parte modificabili e personalizzabili dall'utente, ed è totalmente integrata con lo spazio di lavoro MATLAB: è infatti possibile importare variabili di qualunque genere presenti nel *workspace*, generare output testuali nella *command window* ed esportare i risultati delle simulazioni sotto forma di grafici, variabili o file *mat*.

Le simulazioni condotte in Simulink prevedono l'impostazione di un intervallo temporale e di uno step di integrazione, che può essere fisso o variabile, per definire le caratteristiche del solutore, il quale a sua volta può essere selezionato tra diversi messi a disposizione da Simulink e che differiscono per precisione e rapidità di convergenza.

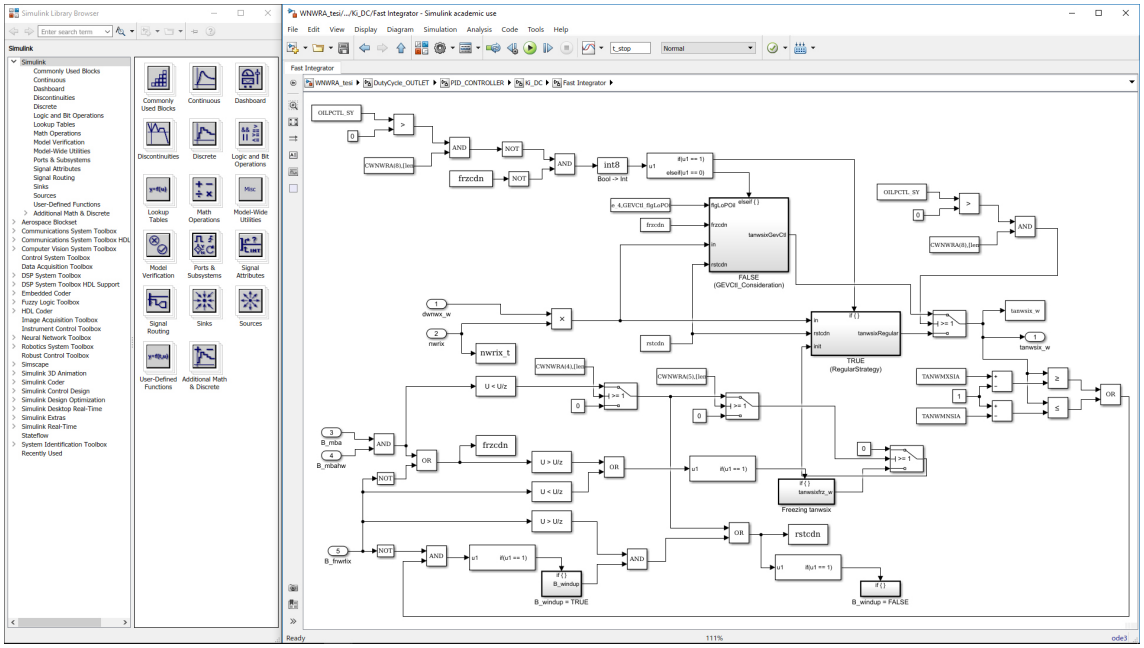


Figura 2.2: Interfaccia di Simulink

Nel presente lavoro di tesi Simulink è stato utilizzato nella fase iniziale delle attività per riprodurre lo schema a blocchi delle funzioni di controllo del sistema VVT. Tale lavoro è stato affrontato al fine di comprendere il funzionamento del software di centralina e per verificare il comportamento del controllore PID che governa il sistema di attuazione; avendo a disposizione i file contenenti le acquisizioni registrate in sala prove è stato possibile riprodurre il funzionamento della ECU e osservare la risposta del sistema in seguito alla richiesta di un target.

2.1.2 Guide

Guide è il tool di sviluppo di interfacce grafiche integrato in MATLAB, rappresentato in figura 2.3; consente di costruire passo per passo la GUI desiderata posizionando manualmente i vari controlli, tra cui si ricordano *textbox*, *listbox*, *command button* e *label*,

assegnando callbacks ai vari eventi generati dall'utente e scrivendo il codice associato ai vari comandi in modo relativamente semplice. Una volta definito il layout grafico dell'interfaccia Guide genera automaticamente il codice necessario all'inizializzazione e alla rappresentazione della finestra e dei comandi; sarà compito del programmatore integrare tale script con il codice che gestisce l'interazione tra l'utente e l'interfaccia. È possibile ricreare le stesse interfacce generate con Guide per via indiretta, definendo e posizionando i vari comandi tramite opportuno script, ma si tratta di un metodo decisamente più laborioso e meno intuitivo.

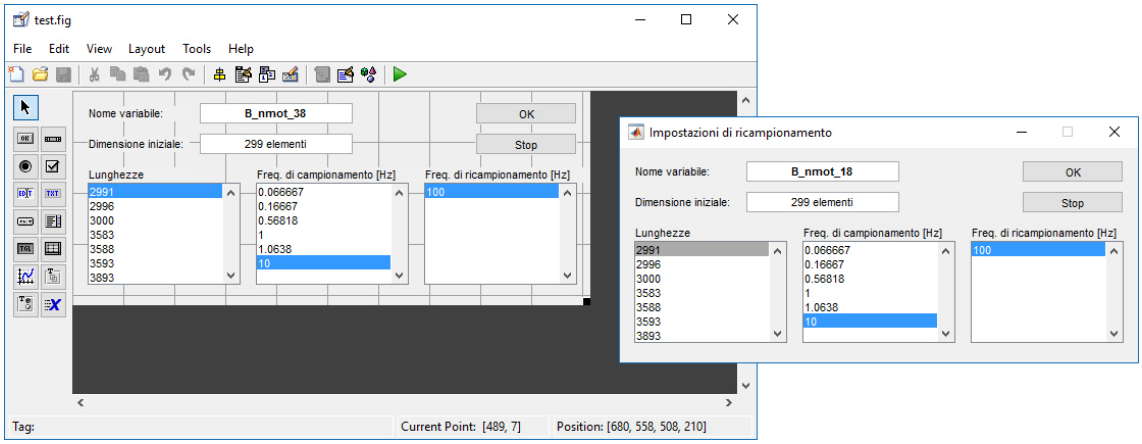


Figura 2.3: Interfaccia di Guide

Ai fini della trattazione si ricorda che Guide è stato impiegato in questo lavoro per programmare un piccolo tool per il ricampionamento e l'interpolazione di alcuni segnali acquisiti durante le prove ad una frequenza di campionamento errata.

2.2 LabVIEW

LabVIEW è l'ambiente di programmazione grafica distribuito da National Instruments; a differenza dei linguaggi tradizionali in LabVIEW le linee di codice sono sostituite da blocchi uniti tramite fili, mentre l'ordine dei comandi non è più definito da istruzioni sequenziali ma segue il flusso dei dati imposto dai blocchi. Ogni blocco è rappresentato da un'icona che ne indica la funzione svolta, e gli stessi collegamenti tra i vari blocchi presentano colore, spessore e tratto diverso a seconda del tipo di dato che trasportano. Così come in MATLAB si utilizzano gli script, i programmi di LabVIEW sono chiamati *Virtual Instruments*, o *VIs*, e sono costituiti da due elementi essenziali:

- *front panel* (figura 2.4): è l'interfaccia grafica che contiene i controlli utilizzabili dall'utente per interagire con il programma, sia in input per l'impostazione di valori (tramite *controls*) che in output per la visualizzazione dei risultati (tramite

indicators). È possibile inserire controlli numerici, caselle di testo, grafici, controlli booleani, interfacce per il collegamento con devices di acquisizione, pulsanti e switch;

- *block diagram* (figura 2.5): è la finestra che contiene il codice necessario a controllare e gestire tutti gli oggetti presenti nel *front panel*. Si vedano a sinistra le icone che rappresentano gli input del *VI* e a destra le icone relative agli output; si notino inoltre i vari colori utilizzati per distinguere il tipo dei dati: in blu i numeri *integer*, in arancione i numeri *double*, in verde i booleani e in viola le stringhe.

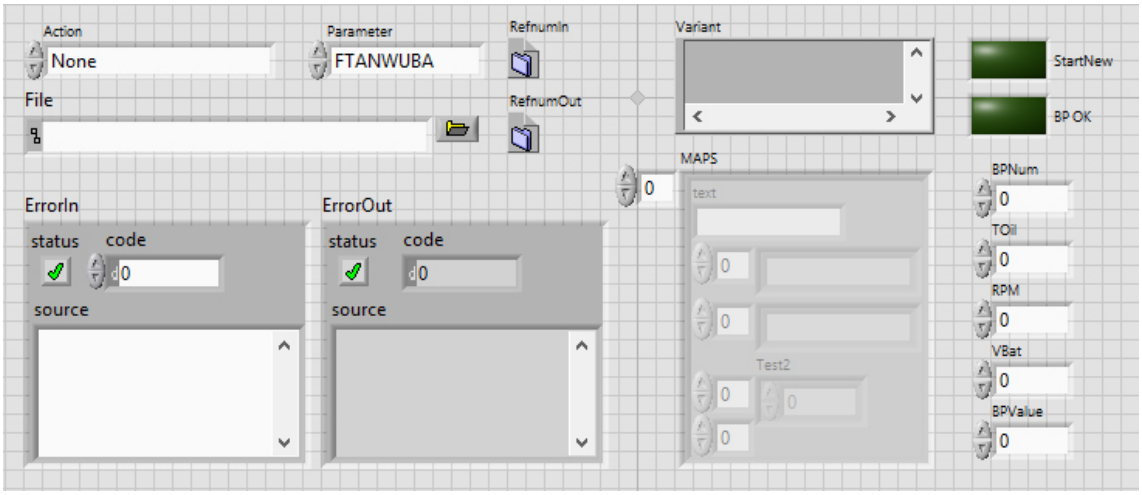


Figura 2.4: Interfaccia di LabVIEW: *front panel*

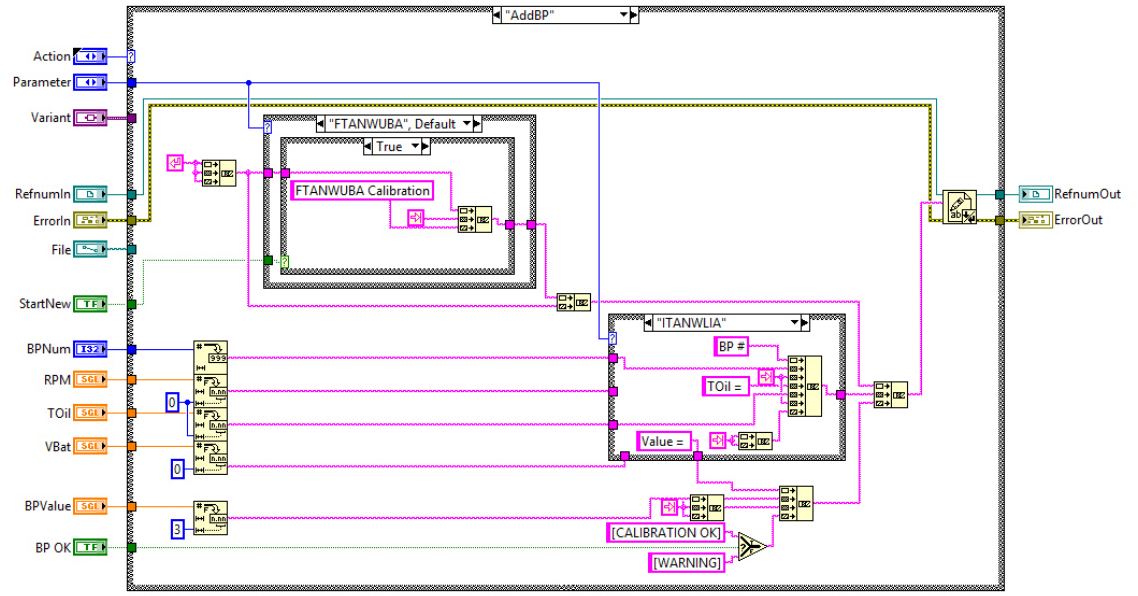


Figura 2.5: Interfaccia di LabVIEW: *block diagram*

Ogni *VI* può contenere altri *VI*, chiamati *subVI*, ed ognuno di questi avrà ingressi ed uscite come se fosse una function scritta in un linguaggio di programmazione tradizionale: le figure 2.4 e 2.5 rappresentano un *subVI* in grado di generare un report in formato *txt* al termine del processo di calibrazione. Il codice grafico ad alto livello utilizzato in LabVIEW consente di programmare in modo intelligente e flessibile, e ha consentito di racchiudere in un unico *subVI* operazioni diverse, che vanno dalla generazione del report, all'aggiunta dei vari breakpoints e al confronto tra le mappe di default e quelle calibrate; tali funzioni possono essere richiamate all'occorrenza utilizzando sempre lo stesso blocco con input di volta in volta diversi. A questo proposito si fa notare, in alto a destra nella figura 2.4, la presenza del pannello per il settaggio degli input e degli output del *subVI* e alla sua destra l'icona, personalizzabile dal programmatore, che identifica il *subVI* nel *block diagram*.

LabVIEW nasce come linguaggio per la programmazione dell'hardware prodotto da National Instruments e consente la creazione di librerie e la compilazione di programmi che possono essere eseguiti su dispositivi di diverso tipo. È opportuno ricordare che questa possibilità verrà sfruttata proprio in questo lavoro per la realizzazione del tool di automazione della calibrazione, che sarà illustrato nel capitolo 4.

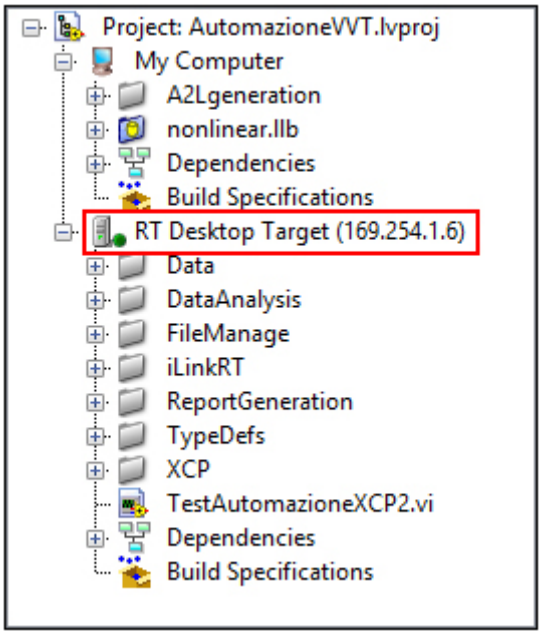


Figura 2.6: Struttura di un progetto LabVIEW

Un'ultima considerazione relativa alla struttura di un progetto LabVIEW, mostrata in figura 2.6: è opportuno sottolineare che è possibile eseguire *VI* sia in locale sul PC su cui è installato LabVIEW, aggiungendoli sotto 'My Computer', oppure su un qualunque dispositivo esterno. In figura è rappresentato il *project explorer* del tool di calibrazione

che verrà descritto nel seguito, e il rettangolo rosso indica che il *VI* principale, chiamato 'TestAutomazioneXCP2', viene eseguito sul PC realtime identificato dall'indirizzo IP indicato tra parentesi; tale struttura permette di tenere distinti tutti quei *VI* che per definizione richiedono di girare su destinazioni diverse, mantenendo in locale i programmi secondari che servono eventualmente in fase di debug oppure che devono essere eseguiti raramente, e mandare sui target solo il programma principale.

2.3 INCA

INCA è un software prodotto da ETAS per la gestione, la diagnosi e la calibrazione delle centraline motore, collegate al sistema di controllo tramite l'hardware prodotto dalla stessa ETAS. Si tratta in realtà di un pacchetto di software di cui INCA costituisce il prodotto base al quale è possibile aggiungere diversi tool tra cui si ricordano ETAS MCE e ETAS MDA, che verranno brevemente descritti nel seguito.

La ECU è in grado di gestire completamente e autonomamente il propulsore al quale è collegata tramite il software di gestione in essa presente; tale software si basa sulle informazioni provenienti dai vari sensori che inviano alla centralina le cosiddette *measurements*, ed effettua il controllo attuando dei valori che vengono chiamati *calibrations*. INCA permette all'utente di accedere alle *measurements* (in sola lettura) e alle *calibrations* (in lettura e scrittura), controllando il funzionamento del motore ed eventualmente modificando i parametri delle varie funzioni di controllo.

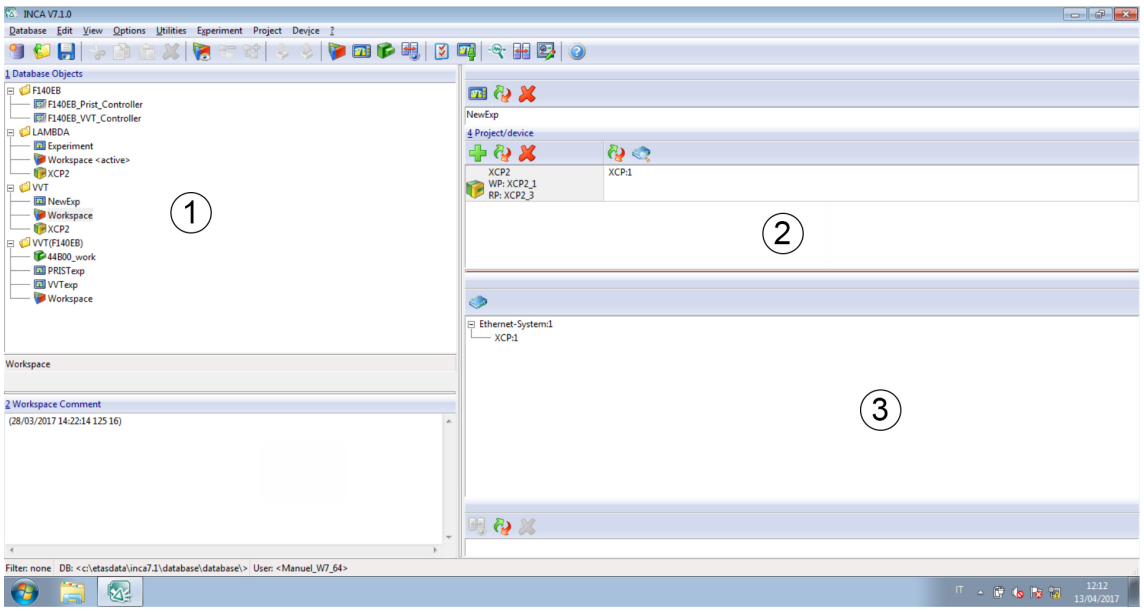


Figura 2.7: Interfaccia di INCA

In figura 2.7 è rappresentato l'ambiente di lavoro di INCA, costituito dai seguenti elementi:

1. *Database Objects*: il database manager è la finestra principale di INCA; tramite essa è possibile gestire tutti gli oggetti necessari alla gestione di una centralina, tra cui il *workspace*, l'*experiments* e l'*ECU project*. È possibile creare cartelle e sottocartelle per presentare una visualizzazione ordinata dei vari progetti, ed esportare qualunque elemento in modo da poter essere utilizzato in altri progetti;

2. *Experiment* e *Project/device*: contengono informazioni sull'*experiment* e sull'*ECU project* del *workspace* attualmente selezionato e permettono di eliminarli e sostituirli;

3. *Hardware*: mostra gli hardware correntemente attivi e collegati al progetto selezionato.

Ogni *workspace* a sua volta è costituito da tre elementi fondamentali, che devono essere creati e configurati prima di poter avere accesso alle *measurements* e alle *calibrations*:

- *ECU project*: è costituito da due files, uno di tipo *a2l* che contiene la descrizione della centralina, nonché i nomi e il tipo delle variabili in essa presenti, e uno di tipo *hex* che contiene il software di centralina. Entrambi vengono generalmente forniti dal produttore della ECU;

- *Hardware*: è il tipo di device che viene gestito dal software presente nell'*ECU project*. INCA permette di scegliere tra diversi dispositivi target, tuttavia l'hardware viene generalmente individuato automaticamente sulla base del contenuto dei file *a2l* e *hex*. L'interfaccia utente di INCA mette a disposizione l'*Hardware Configuration Editor*, rappresentato in figura 2.8, tramite il quale è possibile leggere i dati di fabbrica della centralina, verificare il collegamento con INCA e modificare l'accesso alle calibrazioni;

- *Experiment*: è l'interfaccia grafica sulla quale è possibile visualizzare *measurements* e *calibrations*. L'utente può scegliere quali variabili inserire nell'*experiment* e come visualizzarle: sotto forma di valori scalari, di curve, di mappe o di grafici, in modo da creare un ambiente di lavoro secondo le esigenze del calibratore. In figura 2.9 si può vedere un *experiment* creato per la calibrazione del controllo VVT: si noti la presenza di indicatori booleani che permettono di verificare rapidamente lo stato della calibrazione. È opportuno sottolineare che l'interfaccia di *experiment* mette a disposizione dell'utente due diverse modalità di lavoro:

– *reference page*, che permette la sola osservazione delle calibrazioni presenti in centralina, senza la possibilità di modificarle; è utile ad esempio per ripristinare velocemente le impostazioni di fabbrica dopo aver modificato una serie di parametri durante il funzionamento del motore;
– *working page*, che permette all'utente la modifica delle calibrazioni presenti in centralina; all'occorrenza è possibile impostare la *working page* corrente come nuova *reference page*.

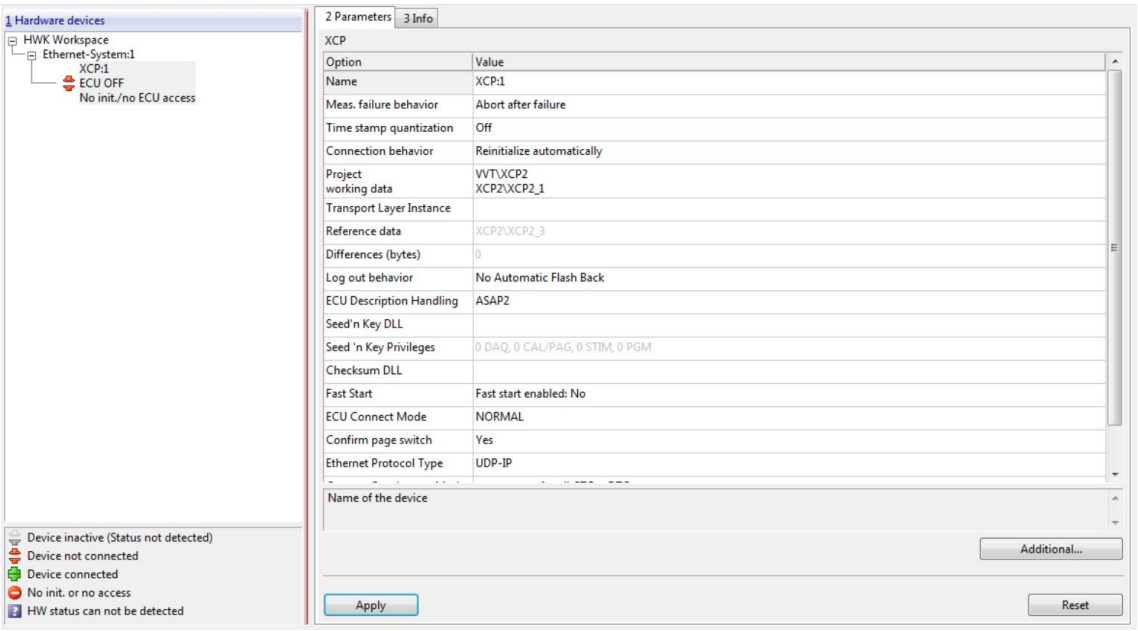


Figura 2.8: Interfaccia dell'*Hardware Configuration Editor*

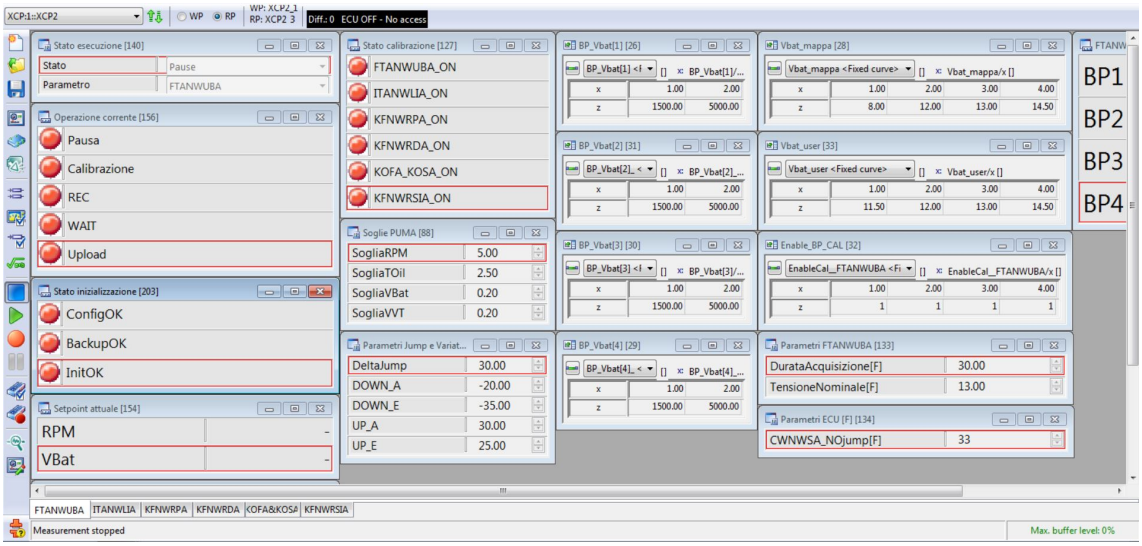


Figura 2.9: Interfaccia dell'*Experiment*

2.3.1 Tool MCE

È un add-on per INCA che, insieme al modulo ES910 prodotto dalla stessa ETAS, consente di stabilire una comunicazione real time con l'ECU per lo sviluppo e la messa a punto di sistemi automatici di calibrazione; il collegamento tra il PC host e la centralina avviene tramite il protocollo iLinkRT, di cui si è già parlato. Il software permette di scegliere le variabili a cui può accedere in realtime il software di calibrazione tramite il modulo ES910; uno dei vantaggi di questo approccio consiste nell'incremento di velocità per quanto riguarda il collegamento con la ECU e nella conseguente possibilità di gestire *measurements* e *calibrations* ad elevata frequenza.

2.3.2 Tool MDA

Questo tool permette di aprire i file *dat* contenenti i segnali acquisiti e di visualizzarli consentendo all'utente un controllo preliminare sull'esito della prova prima di convertire i file *dat* in formato *mat*.

Capitolo 3

Caratteristiche e metodo di calibrazione del controllo VVT

3.1 La fasatura variabile della distribuzione

Nei motori meno prestazionali le leggi di moto delle valvole dipendono esclusivamente dal profilo delle camme e dal meccanismo di distribuzione, sia per quanto riguarda la fasatura, ovvero l'istante in cui esse si aprono o si chiudono, sia per quanto riguarda l'alzata, e non possono essere modificate durante il funzionamento del motore.

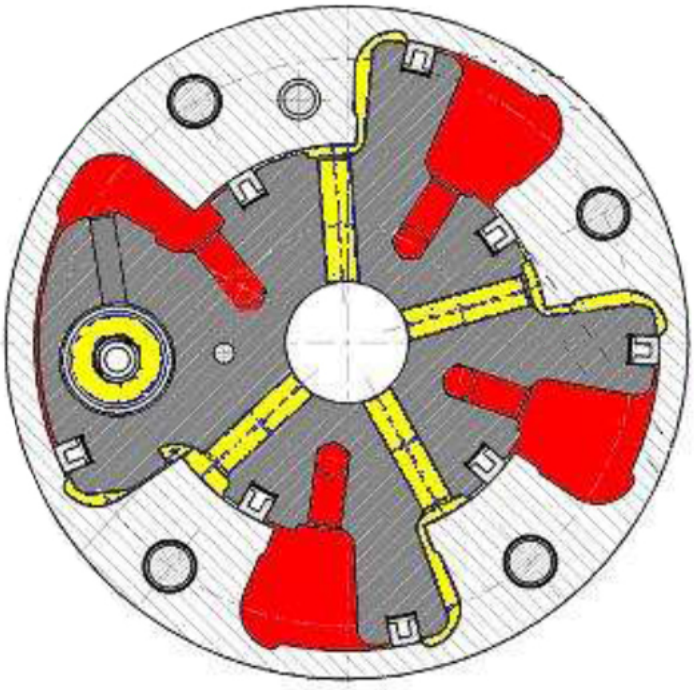


Figura 3.1: Sezione di un variatore di fase

I sistemi VVT (*Variable Valve Timing*), generalmente applicati ai propulsori più potenti e performanti, consentono di modificare la fasatura della distribuzione, anticipando o ritardando l'apertura e la chiusura delle valvole stesse e svincolando quindi parzialmente il moto degli alberi a camme da quello dell'albero a gomiti. Questo viene realizzato introducendo nella catena cinematica della distribuzione degli opportuni *variatori di fase*, di cui è riportato un esempio in figura 3.1: questi dispositivi sono costituiti da un rotore interno, solidale all'albero a camme, e da uno statore esterno, collegato all'albero motore tramite cinghia dentata o con una cascata di ingranaggi. Lo statore trasmette il moto al rotore sfruttando la resistenza offerta dall'olio in pressione di cui sono riempite le camere di separazione, rappresentate in colore rosso e giallo in figura; variando i volumi di tali camere è possibile ottenere un moto relativo del rotore rispetto allo statore, e quindi dell'albero a camme rispetto all'albero motore. In particolare, riempiendo la camera rossa e svuotando quella gialla si può anticipare l'azione delle camme sulle valvole, mentre riempiendo la camera gialla e svuotando quella rossa si può ritardare l'azione delle camme. Applicando un variatore di fase su ogni albero a camme è possibile ottimizzare gli istanti di apertura e chiusura delle valvole di aspirazione e scarico su tutto il campo di funzionamento del motore.

La regolazione del flusso d'olio alle camere viene gestita tramite un cassetto distributore del tipo rappresentato in figura 3.2; questo presenta quattro porte:

- P: alimentazione del cassetto, collegata alla mandata della pompa dell'olio;
- A: prima uscita del cassetto, collegata alle camere di anticipo;
- B: seconda uscita del cassetto, collegata alle camere di ritardo;
- T: scarico del cassetto.

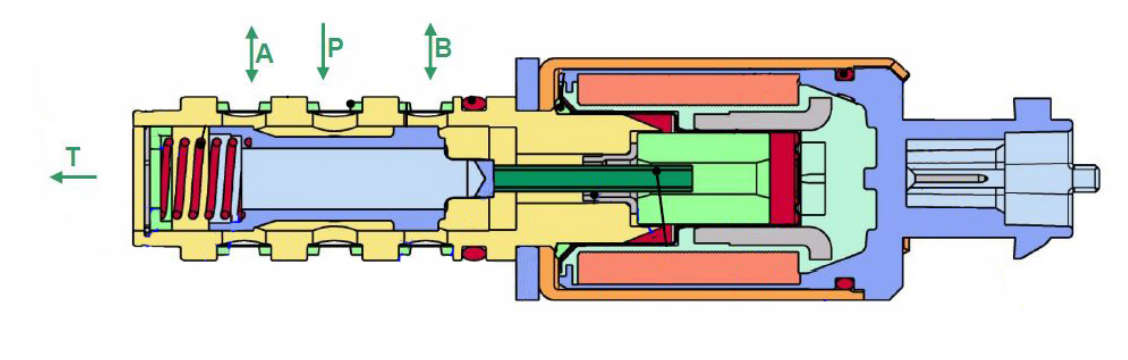


Figura 3.2: Cassetto di distribuzione per l'attuazione dei variatori di fase

Lo stelo del distributore, che comanda l'apertura e la chiusura delle varie porte del cassetto, viene mosso da un solenoide alimentato in regime PWM (*pulse-width modula-*

tion); a seconda del duty cycle imposto al solenoide lo stelo si sposterà dalla posizione di equilibrio variando la resistenza al passaggio dell'olio nelle due porte A e B e determinando di conseguenza la portata inviata alle camere di anticipo e ritardo. È interessante sottolineare che la caratteristica idraulica dei cassetti impiegati nei variatori del motore Alfa Romeo presenta la posizione di equilibrio, ovvero di mandata nulla alle camere dei variatori, in corrispondenza di un duty cycle pari al 47% circa; questo comportamento dovrà essere tenuto in considerazione in fase di calibrazione del controller PID che governa l'attuazione del cassetto, in particolare per quanto riguarda il parametro di precontrollo.

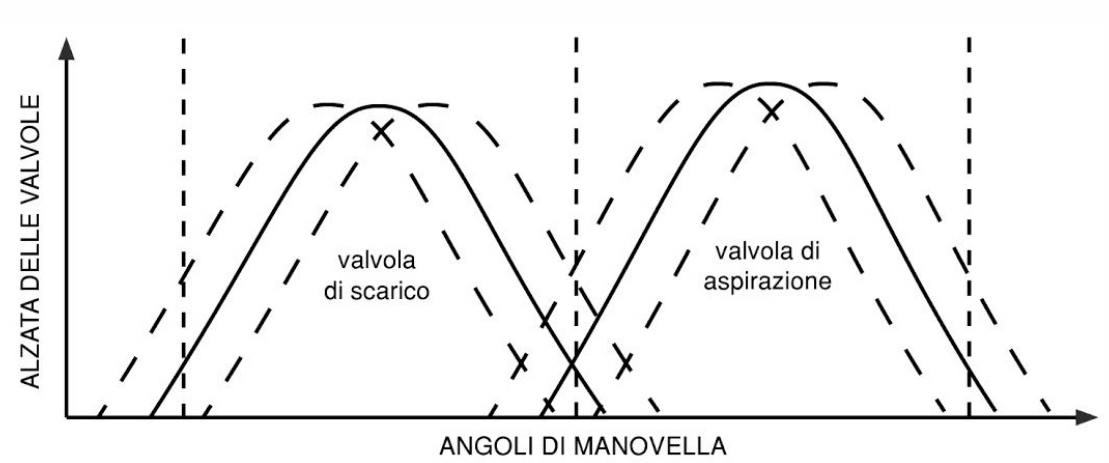


Figura 3.3: Leggi di alzata attuabili tramite variatori di fase

L'azione dei variatori di fase consente una rotazione relativa massima tra albero a camme e albero motore pari indicativamente a 50 °CA per questo tipo di motori permettendo di realizzare un ampio range di fasature, sia lato aspirazione che lato scarico, come si può vedere in figura 3.3. Si tratta di un sistema sicuramente complesso da gestire e di difficile calibrazione, come si vedrà nel paragrafo seguente, e tuttavia i vantaggi che esso consente di ottenere sono decisamente importanti, soprattutto se applicato a motori ad accensione comandata:

- in condizioni di regolazione si può gestire il carico controllando direttamente la quantità d'aria aspirata anticipando o ritardando la chiusura delle valvole di aspirazione (strategia *EIVC*), senza dover strozzare il flusso in ingresso con la tradizionale valvola a farfalla ed evitando di conseguenza le perdite di carico da essa introdotte;

- si può variare il rapporto di compressione effettivo, evitando che diminuisca troppo ai carichi parziali, gestendo l'EGR interno (strategia *LEVC*): in tal modo si può intervenire sul processo di combustione, riducendo in particolare le emissioni di NOx;

- si possono sfruttare al meglio i fenomeni dinamici dovuti alla non-stazionarietà del flusso d'aria e dei gas esausti che hanno luogo nei condotti di aspirazione e scarico per migliorare il riempimento dei cilindri a tutti i regimi di funzionamento del motore.

3.2 La strategia di controllo

3.2.1 Panoramica delle funzioni di controllo

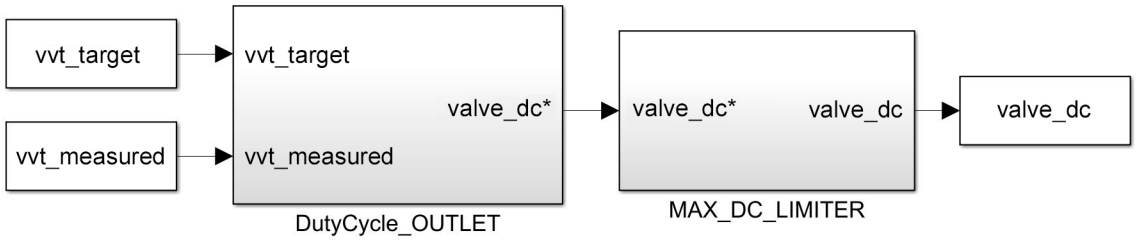


Figura 3.4: Schema a blocchi generale della funzione VVT1

Le funzioni motore implementate nella ECU e destinate alla gestione dei variatori di fase sono due:

- VVT1: funzione per il controllo dei variatori lato scarico;
- VVT2: funzione per il controllo dei variatori lato aspirazione.

Poiché si tratta di due funzioni identiche nel seguito si farà riferimento alla prima, con ovvia generalità della trattazione; laddove le due funzioni dovessero differire verrà opportunamente specificato. Vista inoltre la riservatezza del software si farà riferimento a una generica funzione, fornendo comunque una visione generale quanto più possibile completa dei modelli utilizzati. In figura 3.4 è rappresentato lo schema a blocchi complessivo realizzato in Simulink della funzione VVT1; si possono vedere chiaramente gli input e l'output di tale funzione, descritti nell'elenco che segue:

- INPUT
  - *vvt_target*: angolo target per la chiusura delle valvole di scarico imposto dalla ECU, in funzione del regime di rotazione e del carico del motore;
  - *vvt_measured*: angolo effettivo di posizione dell'albero a camme lato scarico. Viene calcolato partendo dall'angolo misurato dal sensore di fase opportunamente corretto in funzione della velocità di rotazione del motore; tale cor-

rezione viene effettuata data la scarsa risoluzione di misura che presenta il sensore di fase.

- OUTPUT
  - *valve_dc*: è il duty cycle imposto al solenoide del cassetto distributore tale da realizzare l'anticipo (o il ritardo) di fase desiderato.

Nella pratica la funzione VVT1 si occupa di controllare in tensione il cassetto distributore tramite il duty cycle in modo che l'angolo misurato o, per meglio dire, stimato, *vvt_measured* sia effettivamente uguale all'angolo *vvt_target*; a tal fine si utilizza un controllore PID, contenuto nel blocco DutyCycle_OUTLET che verrà descritto dettagliatamente nel seguito, mentre il blocco MAX_DC_LIMITER si occupa di prevenire un eventuale sovraccarico elettrico del solenoide limitando il duty cycle massimo.

3.2.2 Il blocco DutyCycle_OUTLET

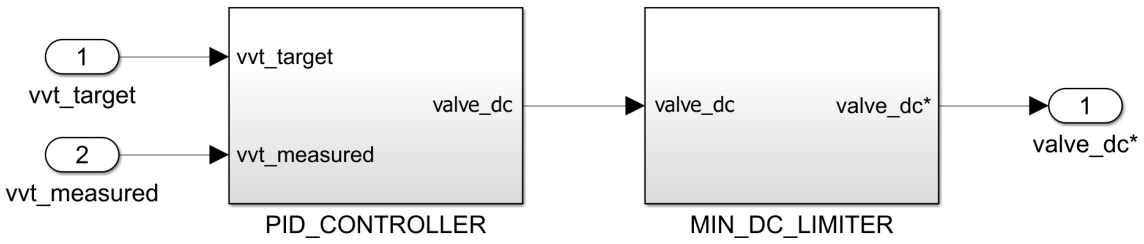


Figura 3.5: Schema del blocco DutyCycle_OUTLET

Nella figura soprastante è rappresentato lo schema interno del blocco DutyCycle_OUTLET; esso è a sua volta costituito dai seguenti blocchi:

- PID_CONTROLLER: è il blocco che contiene il controllore di posizione PID;
- MIN_DC_LIMITER: è il blocco preposto alla limitazione inferiore del duty cycle calcolato;

È evidente che il cuore del sistema di controllo sia rappresentato dal blocco PID_CONTROLLER, il cui output risulta essere proprio il duty cycle per il comando del solenoide del cassetto distributore; si noti che tale valore, prima di essere attuato, viene controllato ed eventualmente limitato dal blocco MIN_DC_LIMITER, il quale provvede a riportare i variatori nella posizione di riferimento, ovvero quella che corrisponde al massimo disincrocio della fasatura, nel caso in cui il duty cycle in output sia inferiore ad una determinata soglia.

3.2.3 Il blocco PID_CONTROLLER

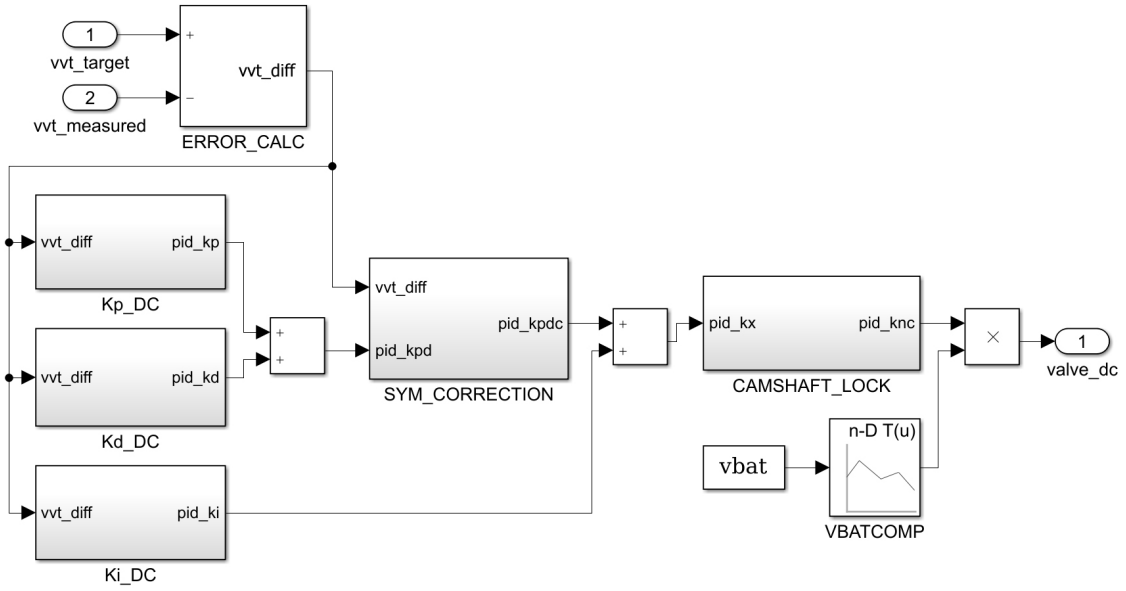


Figura 3.6: Schema del blocco PID_CONTROLLER

All'interno del blocco PID_CONTROLLER, illustrato in figura 3.6, si trovano i blocchi responsabili del calcolo di *valve_dc*:

- ERROR_CALC: questo blocco si occupa del calcolo dell'errore *vvt_diff* tra l'angolo misurato e l'angolo target, che servirà poi da input al PID, e verifica il bloccaggio dell'albero a camme qualora l'angolo target richiesto dal sistema di controllo corrisponda alla posizione di riferimento dei variatori;

- Kp_DC: contributo proporzionale del controllore PID. L'output *pid_kp* viene calcolato tramite una mappa principale, funzione di regime di rotazione del motore e temperatura dell'olio di attuazione, e viene successivamente corretto per sopperire alla non linearità della risposta del sistema dovuta alle caratteristiche del cassetto distributore. È inoltre prevista la gestione dello switch da bassa ad alta pressione e viceversa della pompa dell'olio, e un'intensificazione del duty cycle nel caso in cui la pompa si trovi in bassa pressione e quindi l'attuazione risulti meno pronta;

- Kd_DC: contributo derivativo del controllore PID. Come nel caso del contributo proporzionale l'output *pid_kd* viene calcolato tramite una mappa principale, funzione di regime di rotazione e temperatura dell'olio, ma non è presente la correzione per la non linearità del sistema; è invece presente la possibilità di escludere l'azione derivativa calibrando opportune soglie, così come rimane identica al blocco proporzionale la parte relativa alla gestione del transitorio della pompa dell'olio;

- Ki_DC: contributo integrale del controllore PID. A differenza dei contributi proporzionale e derivativo la parte che calcola il contributo integrale complessivo *pid_ki* è decisamente più complessa, essendo costituita da tre contributi posti in serie e schematizzabili come segue:

  – *Fast Integrator*: è il primo blocco che si individua seguendo il flusso logico dei dati. Questo riceve l'output della mappa principale e rappresenta il contributo 'veloce' destinato a prevalere nel caso in cui sia richiesto un transitorio di angolo VVT target. Non appena il sistema di controllo riconosce uno stazionario il Fast Integrator viene escluso;

  – *Precontrollo*: è il secondo contributo che entra in gioco nel calcolo di *pid_ki* e dipende da due curve funzione della pressione dell'olio. Il precontrollo assume un ruolo molto importante: data la caratteristica del cassetto di attuazione, in corrispondenza di un segnale PWM pari al 48% la portata di olio destinata alle camere di anticipo o ritardo è nulla, mentre spostandosi leggermente a destra o a sinistra di tale valore la portata aumenta molto rapidamente. Il principio di funzionamento del precontrollo è strettamente legato a questo comportamento in quanto posizionando il cassetto nel punto di portata nulla e assumendo tale posizione come riferimento è possibile raggiungere facilmente tutte le altre configurazioni;

  – *Slow Integrator*: è l'ultimo blocco che agisce sul contributo integrale complessivo, ed interviene sull'output calcolato dal Fast Integrator e dal precontrollo. Il suo ruolo è quello di correggere in modo fine l'errore tra l'angolo VVT target e quello misurato, e il suo contributo è chiaramente predominante negli stazionari: a titolo di esempio, il valore di default della costante di tempo dell'integrale lento è pari a 93 secondi, a fronte di una costante di tempo pari a 1 secondo per l'integrale veloce. Si noti che lo Slow Integrator, trovandosi dopo il precontrollo, è anche chiamato a correggere eventuali errori dovuti all'invecchiamento dei componenti;

- SYM_CORRECTION: questo blocco è deputato alla correzione del comportamento asimmetrico del sistema;

- CAMSHAFT_LOCK: questo blocco comanda l'attuazione per il bloccaggio e lo sbloccaggio dell'albero a camme dalla posizione di riferimento, per esempio all'avviamento del motore.

Oltre ai blocchi appena descritti si fa notare, in basso a destra nella figura 3.6, la presenza di un'ulteriore correzione applicata al duty cycle calcolato dal PID: si tratta di un

fattore correttivo dipendente dalla tensione della batteria. Dato che il solenoide del cassetto che comanda i variatori viene comandato in tensione risulta di fondamentale importanza correggere opportunamente il duty cycle in base alla tensione di alimentazione, dato che la potenza trasmessa al solenoide dipende da essa; pertanto se la batteria risulta essere troppo carica o troppo scarica, a parità di duty cycle imposto, si avrà rispettivamente una maggiore o minore potenza trasmessa e, di conseguenza, un maggiore o minore spostamento del cassetto rispetto a quanto effettivamente necessario; di qui la necessità di tenere conto dello stato di carica della batteria.

3.3 La procedura di calibrazione

Le curve e le mappe principali che devono essere calibrate vengono di seguito proposte e brevemente illustrate per esigenze di chiarezza:

- VBATCOMP: curva per la correzione dell'attuazione in funzione della tensione batteria;
- PRECTL (HP e LP): curve di precontrollo per il duty cycle, funzione della temperatura olio. In input richiedono la temperatura olio nella testa motore stimata dalla ECU *tOil*; in output restituiscono la correzione di precontrollo;
- KPMAP: mappa principale per il guadagno proporzionale. In input richiede il regime di rotazione del motore *v_eng* e la temperatura dell'olio *tOil*; in output restituisce la correzione proporzionale attuata dal PID;
- KDMAP: mappa principale per il guadagno derivativo. In input richiede il regime di rotazione del motore *v_eng* e la temperatura dell'olio *tOil*; in output restituisce la correzione derivativa attuata dal PID;
- ADVCOMP e RETCOMP: parametri scalari per l'eventuale correzione del comportamento asimmetrico del sistema. Si fa notare che i due parametri agiscono sui soli contributi proporzionale e derivativo, e il risultato viene sommato al contributo integrale;
- KIMAP: mappa principale per il guadagno integrale. In input richiede il regime di rotazione del motore *v_eng* e la temperatura dell'olio *tOil*; in output restituisce la correzione integrale attuata dal PID.

La pratica e l'esperienza suggeriscono una procedura abbastanza consolidata per quanto riguarda l'ordine di calibrazione delle varie mappe che compongono le due funzioni

VVT1 e VVT2, dalla quale prenderemo ampiamente spunto nel seguito e che verrà seguita anche per lo sviluppo del tool di automazione, lasciando però allo sviluppatore la validazione delle procedure di calibrazione per ogni singola mappa; per questo verranno adottate e descritte nel seguito delle metodologie di volta in volta diverse, costruite 'su misura' dall'autore per ogni mappa.

Si vuole sottolineare che tutti i valori dei parametri riportati nel seguito sono stati individuati dopo un'accurata analisi delle acquisizioni effettuate al banco.

L'autore vuole ricordare che parte del lavoro descritto in questa tesi era già stato affrontato in passato dal professor Cavina, e la documentazione relativa a tale attività è stata la base su cui si è deciso di impostare il lavoro, nonché un ottimo strumento a supporto della realizzazione degli algoritmi per l'analisi dei dati che verranno descritti nel seguito.

3.3.1 VBATCOMP

La prima curva da calibrare è un vettore costituito da breakpoints di tensione batteria; una possibile procedura proposta è la seguente:

1. spianare VBATCOMP a 1;
2. per ogni breakpoint di tensione individuare due regimi di rotazione a cui effettuare le prove (è consigliabile scegliere due regimi sufficientemente diversi, ad esempio 1500 rpm e 5000 rpm);
3. portare la batteria in corrispondenza del primo breakpoint intervenendo, se necessario, sul duty cycle dell'alternatore;
4. acquisire in stazionario la variabile *valve_dc*, il duty cycle calcolato dal PID, per un tempo almeno pari a 30 secondi per entrambi i regimi scelti;
5. calcolare il valore medio di entrambi i segnali acquisiti, e quindi la media dei due valori ottenuti;
6. ripetere i punti 4 e 5 per tutti i breakpoints della curva e interpolare i valori ottenuti con una polinomiale di secondo grado;
7. scegliere un valore nominale di tensione e determinare a quale valore di *valve_dc* corrisponde ricavandolo tramite la curva interpolante determinata nel punto precedente;
8. i valori da calibrare in VBATCOMP si ottengono calcolando il rapporto tra il valore di *valve_dc* corrispondente alla tensione nominale e i valori di *valve_dc* in corrispondenza dei breakpoint di tensione.

Un procedimento alternativo potrebbe essere quello di non calibrare VBATCOMP, annullandone l'effetto durante l'intero processo di calibrazione del sistema di controllo, avendo cura però di tenere la tensione batteria costante e pari al valore nominale scelto; la calibrazione di VBATCOMP dovrà comunque essere affrontata alla fine della calibrazione delle altre mappe. Tale metodo, per quanto interessante, è stato scartato per l'impossibilità di controllare precisamente la tensione batteria nella sala prove a disposizione, dovendo intervenire continuamente sul ciclo di lavoro dell'alternatore; una sala prove dotata di generatore esterno dedicato in luogo della tradizionale batteria ne consentirebbe l'applicazione.

Si vuole sottolineare che si è deciso di effettuare delle prove a due regimi di rotazione per ogni breakpoint di tensione batteria al fine di adattare la calibrazione a un maggior numero di punti motore; è chiaro che aumentare il numero di punti motore analizzati a parità di tensione batteria significa ottenere risultati più realistici, ma aumenta anche il tempo necessario alla calibrazione. La metodologia proposta è sembrata essere un buon compromesso tra precisione e tempo richiesto per l'acquisizione dei dati.

3.3.2 PRECTL

La seconda curva da calibrare è un vettore costituito da breakpoints di temperatura olio; come già accennato in precedenza si tratta in realtà di due curve. Si consiglia di effettuare la calibrazione per entrambe le mappe forzando la pompa a lavorare di volta in volta in alta o in bassa pressione; il metodo di seguito proposto è comunque applicabile a entrambe le mappe:

1. verificare che VBATCOMP corrisponda alla curva calibrata precedentemente;
2. impostare i seguenti parametri ai valori suggeriti:
   - costante di tempo dell'integrale lento = 14.8 secondi;
   - saturazione superiore dell'output dell'integrale lento = 30;
   - saturazione inferiore dell'output dell'integrale lento = -30;
3. spianare PRECTL_LP e PRECTL_HP a 0;
4. individuare quattro regimi di rotazione (consigliati: 1000 rpm, 2000 rpm, 3000 rpm e 4000 rpm) e due angoli target di VVT (consigliati: -15 °CA e 20 °CA) per ogni breakpoint di temperatura olio;
5. portare il motore sul primo breakpoint di temperatura olio e attendere la regimazione del sistema di controllo osservando le variabili *vvt_target* e *vvt_measured*;

6. acquisire in stazionario la variabile *pid_ki* per un tempo almeno pari a 30 secondi e ripetere l'acquisizione per tutti i valori di angoli VVT e per tutti i regimi di rotazione scelti, a pari temperatura olio; calcolare quindi il valore medio di tutti i segnali ottenuti, avendo cura di escludere la parte iniziale qualora si presenti troppo instabile;
7. calcolare la media delle medie ottenute al punto precedente;
8. ripetere i punti 5, 6 e 7 per ogni breakpoint di temperatura olio;
9. interpolare con una retta le medie ottenute di volta in volta al punto 7;
10. i valori da calibrare in centralina sono i valori assunti dalla retta interpolante in corrispondenza dei breakpoint di curva.

Utilizzando i valori suggeriti si ottengono in totale otto acquisizioni, e quindi otto valori medi, per ogni breakpoint di temperatura olio; per verificare che le prove siano effettivamente valide si può calcolarne il coefficiente di variazione definito come segue:

$$COV = \frac{\sigma}{\mu}$$

dove σ e μ sono rispettivamente la deviazione standard e il valore medio del set di prove relativo a una temperatura olio. Impostando una soglia per il COV è possibile stabilire se le acquisizioni relative a un breakpoint debbano essere ripetute o meno; dai risultati ottenuti si può considerare plausibile un valore di COV pari a 0.05 come limite per l'accettabilità dei risultati.

3.3.3 KPMAP

La terza mappa da calibrare è una matrice definita da breakpoints per la temperatura dell'olio e altrettanti breakpoints per il regime di rotazione del motore.

La procedura proposta, prendendo spunto dal metodo di Ziegler-Nichols, prevede l'esecuzione di varie prove al variare del guadagno proporzionale per ogni breakpoint di mappa, imponendo dei gradini di angolo VVT e verificando il comportamento del PID nell'inseguire il target imposto. Più in dettaglio, partendo da un valore base di guadagno si incrementa tale valore a step di 0.1 fino a quando l'angolo VVT misurato non presenta delle oscillazioni chiaramente visibili; il valore da calibrare in mappa corrisponde al primo guadagno che dà origine ad oscillazioni diviso per 2. Il procedimento di calibrazione viene illustrato nel suo complesso nell'elenco che segue:

1. verificare che VBATCOMP e le due curve di PRECTL corrispondano ai valori calibrati precedentemente;

2. impostare i due parametri ADVCOMP e RETCOMP a 1, annullando la correzione dell'asimmetria del sistema;

3. spianare le curve di correzione delle non-linearità a 1;

4. spianare la mappa del guadagno derivativo KDMAP a 0;

5. spianare la mappa del guadagno integrale KIMAP a 0.8, valore di default scelto;

6. impostare i parametri necessari all'esecuzione dei jumps come segue:

   - limite superiore del jump $=$ *vvt_target*, settato al valore di angolo VVT previsto dal sistema di controllo per quel determinato punto motore;
   - limite inferiore del jump $=$ limite superiore $\pm\ 30$, a seconda del range operativo dei variatori;
   - semiperiodo del gradino $= 7$ secondi.

7. spianare la mappa di guadagno proporzionale KPMAP al valore di partenza pari a 1.5;

8. attivare l'esecuzione dei jumps;

9. acquisire l'angolo target *vvt_target* e l'angolo effettivo misurato *vvt_measured* per un tempo corrispondente ad almeno due transizioni in salita e due transizioni in discesa, quindi disattivare l'esecuzione dei jumps;

10. effettuare l'analisi in frequenza del segnale misurato e verificare la presenza di oscillazioni; *[si consiglia di imporre una soglia per la valutazione delle oscillazioni; a livello qualitativo si consiglia di considerare l'ampiezza massima dello spettro ottenuto dalla trasformata di Fourier, e la soglia consigliata è pari a 0.5]*

11. se l'analisi in frequenza conferma la presenza di oscillazioni, dimezzare il valore attuale di guadagno e calibrarlo in mappa, quindi passare al breakpoint successivo ripartendo dal punto 7; in caso contrario incrementare il guadagno proporzionale di 0.1 e ripetere i punti 8, 9 e 10.

A livello teorico, per ottenere una mappa più precisa e affidabile sarebbe opportuno effettuare la calibrazione forzando la pompa dell'olio a rimanere in bassa pressione e poi ripetere l'intero processo forzando la pompa a lavorare in alta pressione; è altresì vero che la centralina stessa, pur consentendo di modificare le soglie per lo switch della pompa, sopra determinati valori di regime e carico del motore porta automaticamente la pompa in alta pressione, indipendentemente dalle richieste del calibratore. Nella pratica quindi si possono seguire due strade:

- si calibra la mappa senza modificare le soglie di switch della pompa, disinteressandosi del fatto che questa lavori in bassa pressione in corrispondenza dei regimi inferiori e passi in alta pressione ai regimi più elevati;

- si calibra la mappa forzando la pompa a lavorare sempre in alta pressione, e poi si ripete la calibrazione solo per quei regimi di rotazione per i quali la centralina consente alla pompa di operare in bassa pressione, mediando i risultati ottenuti in precedenza con la pompa in alta pressione.

Tra le due alternative proposte, la seconda permette di raggiungere dei risultati forse più attendibili, mentre la prima è di più facile e immediata realizzazione in quanto non prevede l'intervento dell'utente sulle mappe che governano il funzionamento della pompa, ed è anche il procedimento che verrà seguito nella realizzazione del tool di automazione.

3.3.4 KDMAP

Dopo la calibrazione del guadagno proporzionale è necessario calibrare la mappa relativa al guadagno derivativo; anche questa è definita da breakpoints per la temperatura dell'olio e altrettanti breakpoints per il regime di rotazione.

Per determinare i valori da calibrare in mappa si è deciso di confrontare la risposta che il sistema di attuazione presenta per ogni guadagno con quella ottenuta in condizioni di riferimento, vale a dire con la mappa KDMAP spianata a 0; in particolare si è scelto di utilizzare, come parametri di confronto, overshoot e T_{0-99}. Prima di procedere con la calibrazione sarà dunque necessario effettuare una serie di acquisizioni mantenendo il guadagno derivativo nullo per determinare le prestazioni di riferimento; il confronto tra le prove con derivativo non nullo e il riferimento verrà effettuato considerando un indice di prestazione che tenga conto dei parametri di prestazione scelti in precedenza e definito come segue:

$$IP = OS \cdot c_1 \cdot c_2 \cdot c_3$$

dove OS è l'overshoot misurato in quella determinata prova, calcolato come differenza tra il valore massimo di *vvt_measured* e il valore massimo del segnale target, mentre c_1, c_2 e c_3 sono coefficienti di vincolo definiti come segue:

- c_1: tiene conto del limite massimo sull'overshoot. È pari a 1 finché l'overshoot misurato è inferiore a una certa soglia (valore consigliato: 4 °CA), altrimenti cresce linearmente con l'overshoot; in ogni caso è saturato a 2;

- c_2: tiene conto del limite minimo sull'overshoot. È pari a 1 finché l'overshoot misurato è superiore a una certa soglia (valore consigliato: 1 °CA), altrimenti cresce linearmente al diminuire dell'overshoot; in ogni caso è saturato a 5;

- c_3: tiene conto del T_{0-99}, ovvero del tempo necessario al segnale misurato a raggiungere il 99% del target. Il coefficiente è pari a 1 finché il T_{0-99} relativo a un determinato guadagno non è maggiore di una certa soglia (valore consigliato: 5%) rispetto al T_{0-99} di riferimento, altrimenti cresce linearmente al peggiorare del T_{0-99}; in ogni caso è saturato a 3.

La procedura scelta è stata impostata in modo da effettuare l'acquisizione di riferimento spianando la mappa a 0 e incrementare di volta in volta tale valore di 0.4, fino ad arrivare a un guadagno pari a 8. La procedura riportata di seguito prevede in ogni caso il calcolo dell'indice di prestazione in tempo reale, e pertanto il calibratore può implementare a sua discrezione una strategia per valutare l'interruzione della calibrazione qualora si sia rilevato un trend crescente di IP all'aumentare del guadagno:

1. verificare che VBATCOMP, PRECTL_LP, PRECTL_HP e KPMAP corrispondano ai valori calibrati precedentemente;

2. annullare la correzione dell'asimmetria del sistema;

3. annullare la correzione della non-linearità del sistema;

4. spianare la mappa del guadagno integrale KIMAP a 0.8 per tutte le variazioni di guadagno derivativo;

5. impostare i parametri necessari all'esecuzione dei jumps come segue:

   - limite superiore del jump $=$ *vvt_target*;
   - limite inferiore del jump $=$ limite superiore $\pm\ 30$;
   - semiperiodo del gradino $= 7$ secondi.

6. spianare la mappa di guadagno derivativo KDMAP a 0;

7. attivare l'esecuzione dei jumps;

8. acquisire *vvt_target* e *vvt_measured* per un tempo corrispondente ad almeno cinque transizioni in salita e altrettante in discesa, quindi disattivare l'esecuzione dei jumps;

9. ripetere il punto precedente per almeno due volte, o finché non si ottiene una percentuale di prove valide pari o superiore al 66% del totale; *[il criterio per l'accettabilità delle prove di riferimento è riportato nel seguito]*

10. terminata l'acquisizione delle prove di riferimento spianare la mappa KDMAP al primo valore non nullo, attivare l'esecuzione dei jumps quindi acquisire *vvt_target* e *vvt_measured* per un tempo corrispondente ad almeno due transizioni in salita e altrettante in discesa, infine disattivare l'esecuzione dei jumps;

11. ripetere il punto 10 finché non si arriva al valore finale ottimizzato, calcolando di volta in volta gli indici di prestazione per i jumps up e per i jumps down. I punti dal 6 all'11 devono essere ripetuti per tutti i breakpoints di mappa;

12. il valore da calibrare in mappa è quello che rende minimo l'indice di prestazione, ovvero quello che consente le migliori prestazioni del sistema di attuazione.

È necessario chiarire che l'indice di prestazione deve essere calcolato sia per i jumps up, ovvero le transizioni in salita, sia per i jumps down, ovvero le transizioni in discesa del segnale target; in questo modo è possibile scegliere per ogni breakpoint di mappa se favorire le prestazioni del sistema in caso di transitorio a salire oppure a scendere. Generalmente, data la presenza di una molla di richiamo nel cassetto distributore, l'attuazione sarà più efficace in una sola direzione, che può essere determinata effettuando dei gradini di segnale target prima di effettuare la calibrazione del controllore PID; una metodologia di lavoro efficace, peraltro suggerita dallo stesso professore, potrebbe essere quella di stabilire a priori se privilegiare i jumps up oppure i jumps down, calibrare KDMAP considerando solo l'indice di prestazione relativo alla direzione scelta e poi riequilibrare il sistema agendo sui parametri ADVCOMP e RETCOMP.

Un'ulteriore considerazione riguarda le prove di riferimento: dato che su di esse si basa gran parte del processo di calibrazione e da esse dipenderanno poi i valori calibrati in mappa è necessario che tali prove siano affidabili. Per questo la procedura illustrata in precedenza suggerisce un numero minimo di acquisizioni da effettuare; è chiaro che aumentare il numero di prove condotte è sicuramente un buon metodo per ridurre la dispersione dei risultati, dato che l'overshoot e il T_{0-99} di riferimento vanno calcolati mediando i valori ottenuti in tutte le prove con derivativo nullo, e tuttavia come al solito è necessario trovare un compromesso tra affidabilità e tempo necessario alla calibrazione. In ogni caso è consigliabile verificare, per ogni singola prova, che i valori di overshoot e T_{0-99} calcolati per ogni transizione, distinguendo jumps up e jumps down, siano compresi entro una certa soglia rispetto alla mediana calcolata per quella prova; in particolare, le soglie suggerite sono pari a $\pm 10\%$ per l'overshoot e $\pm 2.5\%$ per il T_{0-99}. I jumps che non rispettano tale criterio devono essere scartati; se il numero di jumps scartati risulta superiore all'80% l'acquisizione deve essere ripetuta.

3.3.5 KIMAP

L'ultima mappa da calibrare è quella del guadagno integrale: esattamente come le due mappe KPMAP e KDMAP, anche KIMAP è definita da breakpoints di temperatura dell'olio e altrettanti breakpoints di regime di rotazione del motore.

Il metodo proposto per la calibrazione è abbastanza simile a quanto già illustrato e consiste nell'effettuare una prima prova per determinare le prestazioni di riferimento e poi di aumentare via via il guadagno integrale finché non si trova un valore in grado di rendere minimo l'indice di prestazione. Si vuole sottolineare che in questo caso si è assunto 0.1 come valore di partenza per il calcolo del riferimento, e non 0 come accadeva per il guadagno derivativo.

L'indice di prestazione per il guadagno integrale è definito in modo leggermente diverso, e per la prima volta si tengono in considerazione sia il cosiddetto tempo stazionario T_s (definito come il tempo impiegato dall'angolo VVT effettivo, a partire dall'inizio della transizione del segnale target, a rientrare in una banda di tolleranza e non uscirvi più) sia l'errore medio a regime (definito come il valor medio dell'errore tra angolo effettivo ed angolo misurato, calcolato a partire da un tempo T_s dall'inizio della transizione):

$$IP = T_s \cdot c_1 \cdot c_2$$

dove c_1 e c_2 sono coefficienti di vincolo definiti come segue:

- c_1: tiene conto dell'errore a regime. È pari a 1 finché l'errore a regime è inferiore a una certa soglia (valore consigliato: 0.75 °CA), altrimenti cresce linearmente con l'errore;

- c_2: tiene conto dell'overshoot. È pari a 1 finché l'overshoot relativo a un determinato guadagno è inferiore a una certa soglia (valore consigliato: 5%) rispetto all'overshoot di riferimento, altrimenti cresce linearmente con l'overshoot; in ogni caso è saturato a 5.

Si vuole precisare che prima di procedere al calcolo dei vari parametri di prestazione (escluso l'overshoot) il segnale *vvt_measured* acquisito è stato filtrato con media mobile a 40 campioni dato che presentava un rumore significativo; questa considerazione verrà ripresa ed adeguatamente giustificata nel paragrafo seguente dove si analizzeranno i codici per il trattamento dei dati acquisiti. Di seguito si riporta la procedura di calibrazione proposta:

1. verificare che VBATCOMP, PRECTL_LP, PRECTL_HP, KPMAP e KDMAP corrispondano ai valori calibrati precedentemente;

2. annullare la correzione dell'asimmetria del sistema;

3. annullare la correzione della non-linearità del sistema;

4. impostare i parametri necessari all'esecuzione dei jumps come segue:

   - limite superiore del jump $=$ *vvt_target*;
   - limite inferiore del jump $=$ limite superiore $\pm$ 30;
   - semiperiodo del gradino $=$ 7 secondi.

5. spianare la mappa del guadagno integrale KIMAP a 0.1 per la prova di riferimento;

6. attivare l'esecuzione dei jumps;

7. acquisire *vvt_target* e *vvt_measured* per un tempo corrispondente ad almeno cinque transizioni in salita e altrettante in discesa, quindi disattivare l'esecuzione dei jumps;

8. ripetere il punto precedente per almeno due volte, o finché non si ottiene una percentuale di prove valide pari o superiore al 66% del totale; *[il criterio per l'accettabilità delle prove di riferimento è identico a quello adottato per il contributo derivativo]*

9. terminata l'acquisizione delle prove di riferimento spianare la mappa KIMAP al valore di guadagno successivo, attivare l'esecuzione dei jumps quindi acquisire *vvt_target* e *vvt_measured* per un tempo corrispondente ad almeno due transizioni in salita e altrettante in discesa, infine disattivare l'esecuzione dei jumps;

10. ripetere il punto 9 finché non si arriva al valore calibrato, calcolando di volta in volta gli indici di prestazione per i jumps up e per i jumps down. I punti dal 5 al 10 devono essere ripetuti per tutti i breakpoints di mappa;

11. il valore da calibrare in mappa è quello che rende minimo l'indice di prestazione.

3.4 L'analisi delle acquisizioni

Prima di iniziare lo sviluppo in LabVIEW del tool di calibrazione vero e proprio è stato necessario mettere a punto gli algoritmi di trattamento dei dati, in parte esposti nel paragrafo precedente; a tal fine sono stati condotti alcuni test in sala prove, simulando già la procedura definitiva, e le acquisizioni risultanti sono state elaborate tramite codici MATLAB che, se validati, dovranno servire come base per l'implementazione delle procedure di calibrazione in LabVIEW.

Questa fase preliminare è stata di notevole aiuto per la programmazione del tool in LabVIEW poiché tutta la parte di trattamento dei dati acquisiti era già stata messa a punto; si sarebbe trattato semplicemente di tradurre gli script MATLAB in codice a blocchi di LabVIEW, eventualmente ottimizzando gli algoritmi in modo da poter essere eseguiti in modo più efficiente su un PC realtime.

Nel seguito verranno illustrati, per ogni curva e per ogni mappa descritta nel paragrafo precedente, i codici elaborati, i risultati ottenuti e il confronto con lo storico di calibrazioni messo a disposizione da Maserati; ampio spazio verrà dedicato inoltre all'analisi e al commento delle prove eseguite dato che questa fase ha richiesto notevole tempo prima di poter essere dichiarata conclusa.

3.4.1 La conversione da *dat* a *mat*

Prima di procedere sembra opportuno ricordare che i file contenenti i segnali acquisiti durante le prove vengono esportati da INCA in formato *MDF* (*Measure Data File*) e salvati in un file con estensione *dat*; per poterli trattare è necessario convertirli in un formato leggibile da MATLAB. A tal fine si è impiegato il tool *MDFimport* che ha consentito di selezionare le sole variabili che interessavano ai fini dell'analisi dati e di importarle direttamente nel workspace di MATLAB; si è poi pensato di salvare tali variabili in un file di tipo *csv*, il quale sarebbe poi stato riutilizzato sia da MATLAB che dal tool finale scritto in LabVIEW per verificare il buon funzionamento del software.

Tale metodologia ha consentito di ridurre notevolmente il tempo necessario alla lettura dei dati dall'hard disk in quanto i file *csv* così ottenuti, essendo costituiti solo da poche variabili, presentano una dimensione su disco molto inferiore rispetto ai file *dat* originali, che invece contengono tutte le measures presenti in centralina. Inoltre i file *csv*, a differenza dei *dat*, possono essere aperti e modificati anche da un software comune quale Excel, mentre le acquisizioni originali richiedono necessariamente il tool MDA per poter essere elaborate.

3.4.2 VBATCOMP

La calibrazione della curva di compensazione della tensione batteria è decisamente meno impegnativa rispetto alle altre mappe, e anche la parte di analisi dei dati prevede semplicemente il calcolo di una media e l'interpolazione (o estrapolazione) dei valori in corrispondenza dei breakpoints di curva. Il piano prove utilizzato per la messa a punto dei codici è riportato nella tabella 3.1.

Si fa notare che la tensione inferiore che è stato possibile raggiungere in sala è 11.5 V, ben superiore al minimo breakpoint previsto in ECU; infatti nonostante si sia intervenuto

sulle soglie di lavoro dell'alternatore la centralina, non appena rilevava una tensione batteria inferiore a 11 V, riprendeva automaticamente il controllo dell'alternatore in modo da ricaricare la batteria.

	11.5 V	12 V	13 V	14.5 V
1500 rpm	X	X	X	X
5000 rpm	X	X	X	X

Tabella 3.1: Piano prove per la calibrazione di VBATCOMP

In appendice, sezione A.1, è riportato il codice di analisi dati: data la struttura del piano prove, alla fine dei test e dopo la conversione in *csv* si avevano a disposizione otto file. Le prime righe del codice servono a leggere tali file e calcolare il valore medio di *valvedc*, alla riga 21, memorizzato per ognuno dei file nel vettore `valvedcMEAN`; nelle righe successive, dalla 27 alla 47, si determinano le due polinomiali di secondo grado che interpolano i valori di *valvedc*, con il seguente significato dei simboli:

- `x` è il vettore contenente i breakpoints di tensione coperti durante la prova;
- `y` contiene di volta in volta in valori di `valvedcMEAN`, prima quelli relativi alla prova a 1500 rpm (riga 29) e poi quelli relativi a 5000 rpm (riga 33);
- `p` contiene i coefficienti del polinomio interpolante;
- `xI` è il vettore uniformemente spaziato che rappresenta lo spazio in cui è definita la curva interpolante;
- `yI_1500` e `yI_5000` sono le due curve interpolanti calcolate per ogni valore di `xI`;
- `yMEAN` è la curva interpolante ottenuta come media tra `yI_1500` e `yI_5000` , impiegata per il calcolo dei valori da calibrare in centralina.

Alla riga 49 ha inizio il processo di calibrazione: si definisce il valore nominale di tensione, quindi si passa all'estrapolazione di *valvedc* in corrispondenza del valore scelto (riga 52) e infine si calcolano i valori di VBATCOMP come rapporto tra i valori di `yMEAN` e `valvedcNOM` . È chiaro che, non coincidendo i breakpoints di tensione scelti per la prova con quelli di default in ECU, è stato necessario interpolare nuovamente la curva `VBATCOMP` per determinare i valori precisi da riportare in centralina, salvati poi nel vettore `tNOMbp` (righe $54 - 56$).

vbat	8 V	12 V	13 V	14.5 V
Test	1.45	1.08	1.00	0.89

Tabella 3.2: Risultati ottenuti

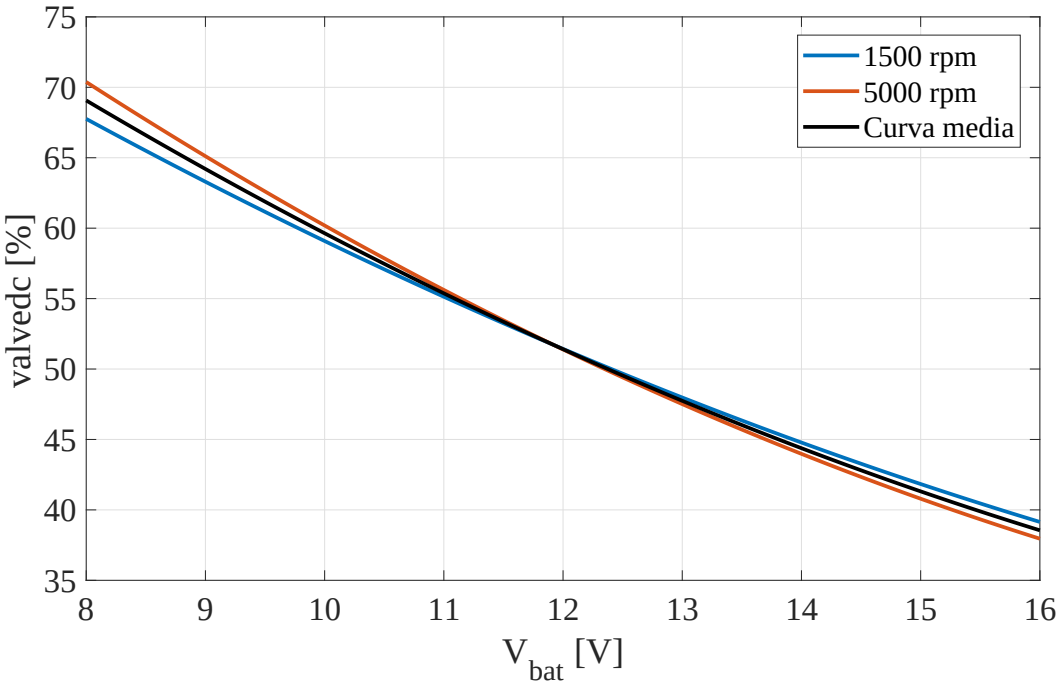


Figura 3.7: Curve di *valvedc* a 1500 rpm, 5000 rpm e media tra le due

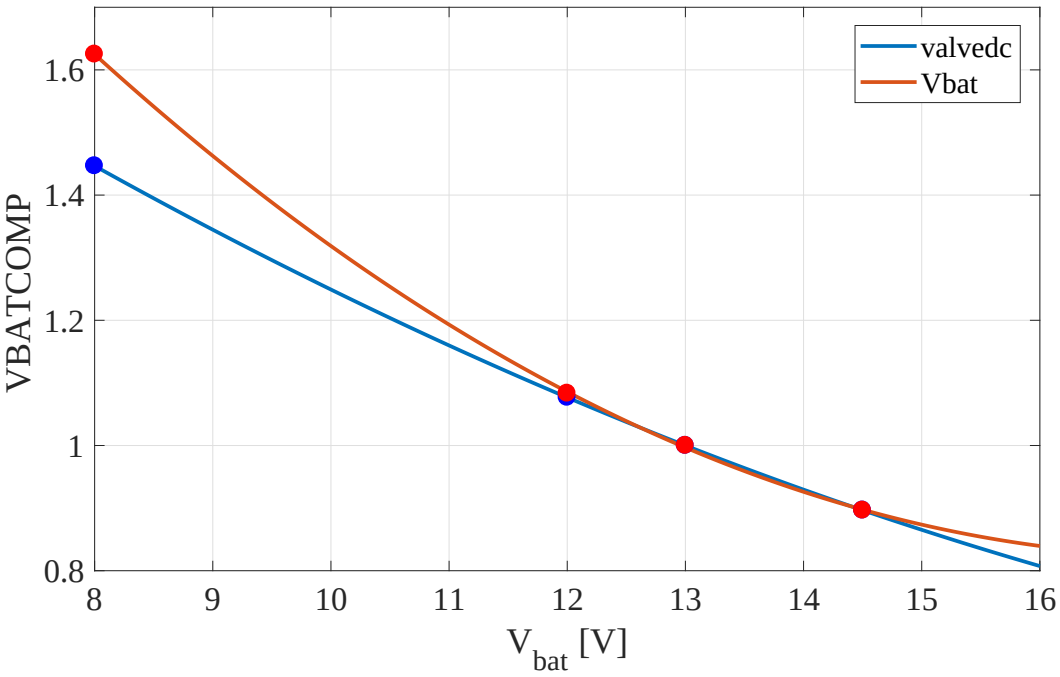


Figura 3.8: Confronto tra i due metodi di calibrazione

Nella tabella 3.2 si riportano i risultati ottenuti; confrontati con lo storico calibrazioni, messo a disposizione dall'Alfa Romeo e qui non riportato per esigenze di riservatezza, tali valori risultano leggermente differenti probabilmente a causa di una diversa scelta della tensione nominale. Seguendo tuttavia il metodo di calibrazione piuttosto grezzo proposto dal produttore, che consiste semplicemente nell'effettuare il rapporto tra i breakpoint di tensione e la tensione nominale, si ottengono valori decisamente più vicini a quelli riportati in tabella.

3.4.3 PRECTL

Come già accennato in precedenza la messa a punto del precontrollo prevede la calibrazione di due curve relative al funzionamento in alta pressione e in bassa pressione della pompa dell'olio. Dato che il metodo descritto nel paragrafo precedente è applicabile ad entrambe, di seguito si tratterà solo della curva ad alta pressione, anche se durante la fase di acquisizione dati sono state condotte alcune prove a bassa pressione; tutte le considerazioni di seguito riportate sono applicabili anche all'altra curva. La tabella 3.3 illustra il piano prove seguito per l'acquisizione dei dati:

	1000 rpm	2000 rpm	3000 rpm	4000 rpm
$T_{olio} = 40$ °C	X (Hi + Lo)	X (Hi + Lo)	X (Hi)	X (Hi)
$T_{olio} = 90$ °C	X (Hi + Lo)	X (Hi + Lo)	X (Hi)	X (Hi)
$T_{olio} = 105$ °C	X (Hi + Lo)	X (Hi + Lo)	X (Hi)	X (Hi)

Tabella 3.3: Piano prove per la calibrazione di PRECTL

Per ognuno dei breakpoints indicati in tabella sono stati esaminati due angoli di VVT target, -15 °CA e 20 °CA, per un totale di 24 acquisizioni in alta pressione (Hi).

È doverosa una precisazione: la sala prove non prevede dispositivi per il condizionamento della temperatura dell'olio e pertanto tutte le prove per PRECTL al variare di tale parametro sono state effettuate agendo sul carico motore e controllando manualmente la temperatura stessa.

Si osservi ad esempio la figura 3.9, nella quale in blu è rappresentato *pid_ki*, in rosso *tOil* (temperatura dell'olio nella testa del motore, stimata dalla ECU e input di mappa per PRECTL) e in arancione *tOilC* (temperatura dell'olio nella coppa, misurata da una termocoppia): si nota infatti che la temperatura ha un andamento crescente per tutta la durata dell'acquisizione, e soprattutto *pid_ki* presenta un transitorio iniziale fortemente instabile che dovrà essere scartato nella fase di analisi al fine di ottenere un valore medio più preciso.

Osservando invece la figura 3.10, relativa allo stesso punto motore ma ad una temperatura olio maggiore, si può vedere come *tOil* sia perfettamente costante, e lo stesso si

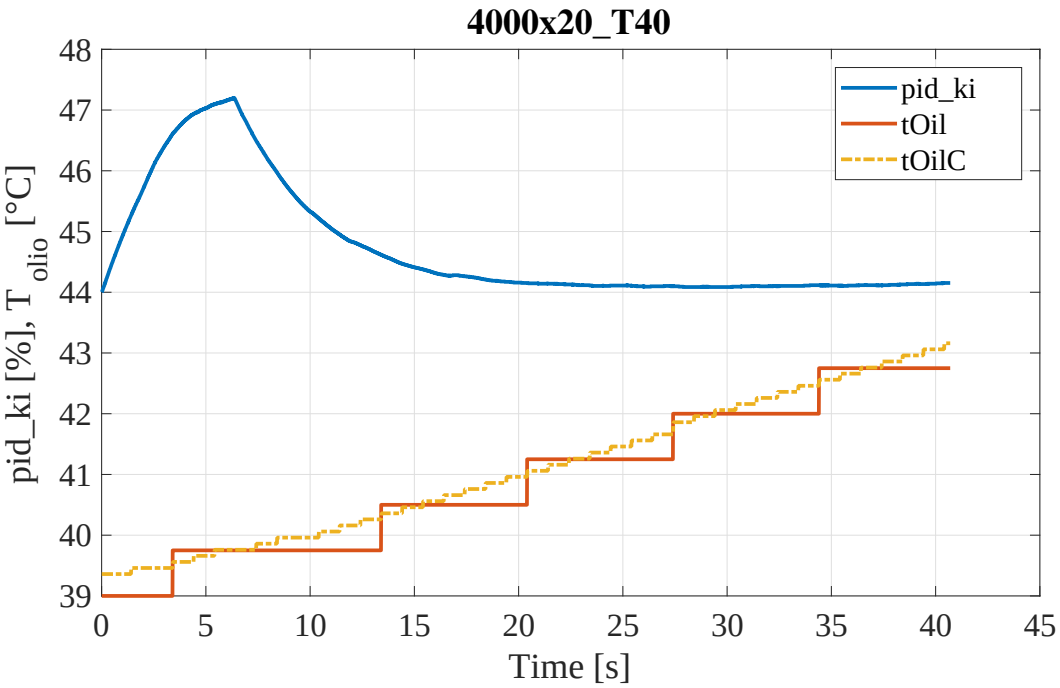


Figura 3.9: Prova a $T_{olio} = 40$ °C, 4000 rpm e 20 °CA

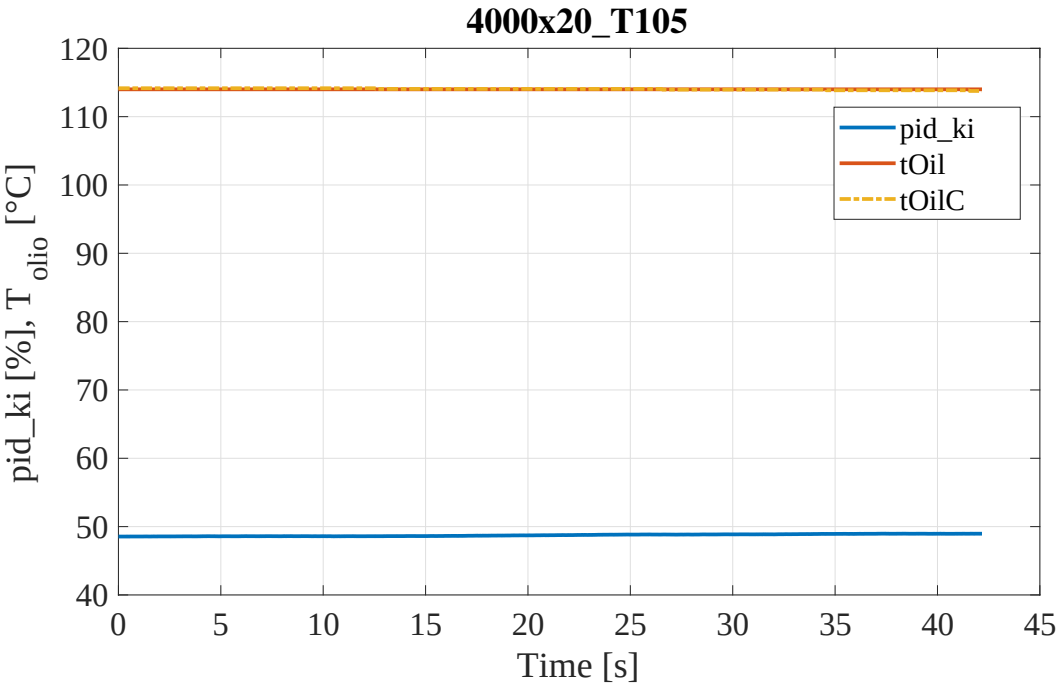


Figura 3.10: Prova a $T_{olio} = 105$ °C, 4000 rpm e 20 °CA

può dire di *pid_ki*; l'unico problema è rappresentato dal fatto che la prova doveva essere condotta a 105 °C come previsto dal piano prove, mentre il controllo 'manuale' della temperatura non ha permesso di ottenere tale valore, superandolo e assestandosi a 114 °C.

Entrambi i problemi appena descritti sono facilmente superabili installando in sala un circuito di condizionamento della temperatura dell'olio motore controllabile tramite PID, risolvendo così sia l'aumento di temperatura dovuto al transitorio di riscaldamento del motore, riscontrato nelle prove a bassa temperatura, sia l'impossibilità di mantenere un valore preimpostato, riscontrata nelle prove ad alta temperatura.

In appendice, sezione A.2, è riportato il codice scritto per l'analisi e il trattamento delle acquisizioni; tale codice è stato pensato per trattare acquisizioni relative a una determinata temperatura olio per volta e quindi, visto il piano prove prima riportato, lo script dovrà essere eseguito per tre volte. Si ricorda che il valore di PRECTL da calibrare in centralina si ottiene effettuando semplicemente la media dei valori di contributo integrale *pid_ki* calcolato dal PID (riga 16) nei vari breakpoints esaminati, a parità di temperatura dell'olio. Prima è necessario calcolare la media degli otto segnali acquisiti, pertanto bisogna anche controllare che il segnale di cui si calcola la media sia effettivamente stabile per tutta la durata nell'acquisizione: qualora ciò non si verificasse è necessario eliminarne la parte instabile, e per questo nelle righe $27 - 35$ si controlla che il massimo e il minimo del segnale siano compresi entro una certa soglia, definita alla riga 7, eliminando tutti quei valori che non vi rientrano.

Dalla riga 102 in poi il codice riconosce le prove relative a uno stesso regime di rotazione (si ricorda che sono due, una con angolo VVT pari a -15 °CA e l'altra con angolo VVT pari a 20 °CA) e ne stima la dispersione calcolando il coefficiente di variazione COV (riga 24); dopodiché, dalla riga 124 si controllano se i valori di COV ottenuti sono inferiori alla soglia definita alla riga 121. Se ciò non accade si eliminano dal vettore `PRECTL` le prove i cui valori risultano troppo dispersi, e si calcola infine il valore da calibrare `PRECTL_cal` effettuando la media dei valori rimasti (riga 156).

Nella tabella 3.4 si possono vedere i valori medi di *pid_ki* e, in ultima riga, la media delle medie; si fa notare che nella prova a $\mathrm{T_{olio}} = 40$ °C le acquisizioni sono tutte accettabili, mentre nelle altre due prove le acquisizioni a 1000 rpm, scritte in corsivo, sono da scartare poiché i COV calcolati, e riportati in corsivo nella tabella 3.5, sono superiori alla soglia impostata, pari a 0.02.

A differenza di quanto descritto in precedenza per VBATCOMP la parte di interpolazione dei valori per determinare la curva definitiva da calibrare in centralina non è stata implementata nello stesso script di analisi delle acquisizioni, in quanto dovendo questo girare una volta per ogni temperatura olio esaminata risultava scomodo inserire una parte di codice che avrebbe dovuto essere invece eseguita solo una volta al termine di tutte

	$T_{olio} = 40$ °C	$T_{olio} = 90$ °C	$T_{olio} = 105$ °C
1000 rpm X −15 °CA	44.18	*47.30*	*48.40*
1000 rpm X 20 °CA	44.12	*49.35*	*49.97*
2000 rpm X −15 °CA	44.96	47.59	48.69
2000 rpm X 20 °CA	43.77	47.71	49.83
3000 rpm X −15 °CA	44.63	48.24	48.56
3000 rpm X 20 °CA	44.12	47.35	48.11
4000 rpm X −15 °CA	45.29	48.83	48.94
4000 rpm X 20 °CA	44.97	47.59	48.10
Valore medio	44.50	47.89	48.75

Tabella 3.4: Risultati ottenuti per la calibrazione di PRECTL – Valori medi di *pid_ki*

	$T_{olio} = 40$ °C	$T_{olio} = 90$ °C	$T_{olio} = 105$ °C
1000 rpm	0.001	*0.029*	*0.023*
2000 rpm	0.019	0.002	0.016
3000 rpm	0.008	0.013	0.007
4000 rpm	0.005	0.018	0.012

Tabella 3.5: Risultati ottenuti per la calibrazione di PRECTL – COV

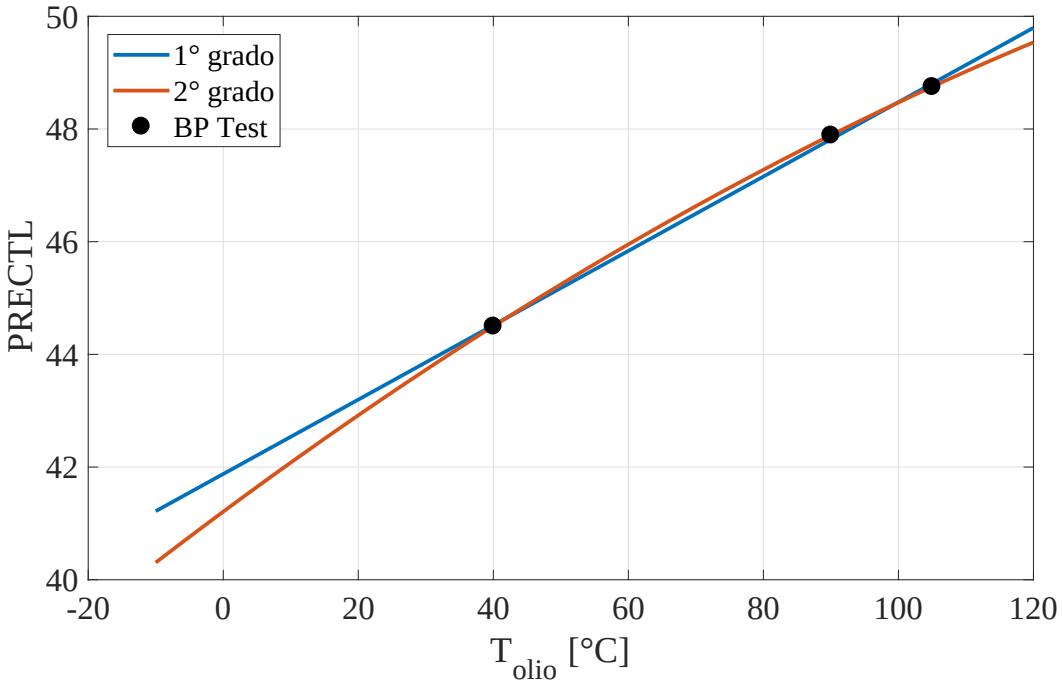


Figura 3.11: Confronto tra polinomio interpolante di 1 ° e 2 ° grado

le procedure intermedie. Si è pertanto pensato di salvare i risultati intermedi in un file *mat*, da caricare ed elaborare successivamente tramite lo script riportato in appendice di seguito a quello principale.

È stata tentata anche un'interpolazione con un polinomio di secondo grado: il con-

fronto tra la curva di primo grado e quella di secondo grado è riportato in figura 3.11. Ma poiché il coefficiente di correlazione R^2 era praticamente uguale – pari a 1 per il polinomio interpolante di primo grado, e pari a 0.99 per quello di secondo grado – si è deciso di mantenere l'interpolazione lineare.

3.4.4 KPMAP

La calibrazione di KPMAP prevede l'esecuzione di transizioni di angolo VVT e la stima dell'eventuale presenza di oscillazioni. Dato che le oscillazioni in questione sono un fenomeno periodico, quanto meno entro la durata del jump, si è pensato di sfruttare l'analisi in frequenza applicando la trasformata di Fourier al segnale acquisito; dato che la procedura di calibrazione suggerita dal costruttore parla semplicemente di 'oscillazioni chiaramente visibili' è stato necessario procedere per tentativi per identificare delle soglie opportune al caso trattato, in modo che l'algoritmo di identificazione delle oscillazioni fosse in grado di riconoscere adeguatamente ed univocamente tutti i jumps che presentano instabilità dopo l'overshoot.

	1500 rpm	3000 rpm	4000 rpm
$\mathrm{T_{olio}} = 90$ °C	X (Hi + Lo)	X (Hi)	X (Hi)
$\mathrm{T_{olio}} = 105$ °C	X (Hi + Lo)	X (Hi)	X (Hi)

Tabella 3.6: Piano prove per la calibrazione di KPMAP

La tabella 3.6 presenta il piano prove stilato per le acquisizioni di prova; si noti che, come per PRECTL, anche in questo caso sono state effettuate prove con la pompa dell'olio in alta pressione (Hi) e in bassa pressione (Lo) per verificare che il metodo di calibrazione proposto fosse corretto, ma per la messa a punto del codice di trattamento dei dati la distinzione è irrilevante, tant'è che sono state impiegate solo le acquisizioni relative alle prove in alta pressione.

Prima di illustrare il codice MATLAB è doverosa una premessa: per esigenze del calibratore, diversamente dalla procedura di calibrazione classica, le prove seguenti sono state eseguite mantenendo costanti sia VBATCOMP che PRECTL; da questo discenderà, come ovvia conseguenza, che i risultati ottenuti non saranno quelli ottimali prima di tutto perché l'effetto del precontrollo sulla velocità di raggiungimento del target è evidente, e soprattutto perché l'azione correttiva attuata da VBATCOMP agisce sul duty cycle in output. Se VBATCOMP è spianata a 1 è chiaro che il duty cycle non verrà amplificato o ridotto in funzione dello stato di carica della batteria: qualora vi fossero delle oscillazioni, nemmeno queste risentirebbero della correzione di VBATCOMP, con la diretta conseguenza che il codice potrebbe rilevarne la presenza laddove invece non dovrebbero esserci, e viceversa.

In appendice, sezione A.3, è riportato il codice scritto per l'analisi e il trattamento delle acquisizioni; come per PRECTL, tale codice è stato pensato per trattare acquisizioni relative a una determinata temperatura olio per volta e quindi, visto il piano prove prima riportato, lo script dovrà essere eseguito per due volte.

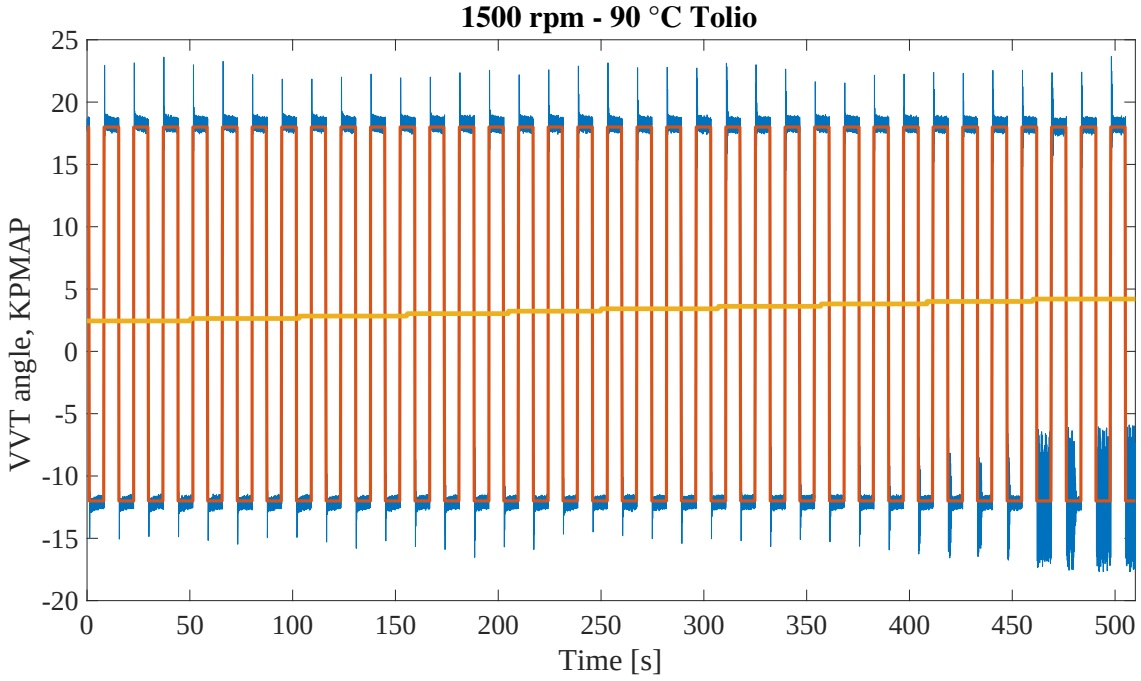


Figura 3.12: Contenuto di un file *csv*

Al solito, le righe iniziali contengono tutti i comandi necessari all'apertura e alla lettura dei file *csv* contenenti i segnali acquisiti, nonché al caricamento di tali segnali nel workspace di MATLAB sotto forma di vettori; dato che nel seguito si parlerà di finestratura manuale su base tempo dei segnali, torna utile ricordare qui che tutte le prove sono state effettuate con un raster di acquisizione pari a 10 ms, ovvero le variabili di interesse sono state campionate con una frequenza di 100 Hz. Si precisa inoltre che per necessità tutti i valori a cui è stata spianata KPMAP, corrispondenti ai guadagni esaminati, sono stati salvati nello stesso file *dat*, ognuno dei quali è relativo a un breakpoint di mappa, pertanto sarà necessario riconoscere durante il trattamento dei dati quale guadagno si sta analizzando; in figura 3.12 si vedono i dati grezzi relativi alla prova a 1500 rpm e $\mathrm{T_{olio}} = 90\,°\mathrm{C}$, in particolare in blu è rappresentato il segnale VVT misurato, in rosso il segnale VVT target e in giallo il valore di KPMAP impostato. Tutto questo è stato implementato nelle righe $34 - 46$ tramite l'utilizzo del comando `diff`: in pratica, avendo acquisito anche il valore di KPMAP corrispondente al breakpoint in esame, è stato possibile riconoscere le transizioni di tale segnale e finestrare così il segnale in funzione dei diversi valori di guadagno.

Dalla riga 48 ha inizio il ciclo `for` necessario ad analizzare ogni singolo guadagno; prima di procedere con il calcolo della trasformata di Fourier è necessario identificare i vari jumps effettuati, sia in up che in down. A tal fine, ancora una volta, si fa uso del comando `diff`: dopo aver finestrato il vettore tempo, l'angolo VVT misurato e l'angolo VVT target (righe $52 - 58$), si individua l'entità del jump (righe $60 - 61$), dopodiché si ricercano tutti gli indici che corrispondono a una transizione del segnale target (righe $65 - 68$). Si fa notare che tali indici sono già suddivisi in base alla direzione, in salita (`iStartUP`) o in discesa (`iStartDOWN`); degno di nota, in questo senso, è stato il processo iterativo seguito per determinare il valore migliore della soglia `tollJump` utilizzata per riconoscere una transizione, infine impostata a un valore pari ad un quarto dell'entità del jump (riga 64). Alle righe $72 - 82$ si costruiscono le due matrici `jumpsUP` e `jumpsDOWN`, contenenti gli indici di inizio e di fine di tutti i jumps individuati nel segnale target e posti in ordine crescente; nelle righe seguenti si valuta la necessità di ridimensionare tali vettori in modo che il numero di jump da analizzare sia uguale in up e in down, per coerenza.

A questo proposito è opportuno ricordare che tutte le prove esaminate in questo capitolo sono state effettuate manualmente, ovvero tutti i parametri di centralina sono stati modificati dall'operatore tramite INCA, e pertanto non è raro ritrovare una certa variabilità nel numero di jumps effettuati per ogni guadagno; di conseguenza i codici qui proposti sono stati elaborati con l'ulteriore obiettivo di eliminare questa variabilità tra le prove, considerando solo i jumps più stabili o eliminando quelli registrati accidentalmente. È chiaro che la logica seguita per lo sviluppo del tool di automazione, descritta nel capitolo seguente, sarà leggermente diversa dato che in quel caso sarà compito del software stesso impostare i parametri in centralina, avviare ed arrestare la registrazione, con precisione sicuramente maggiore rispetto a quanto è possibile effettuare per via manuale.

Dalla riga 98 ha inizio l'analisi in frequenza del segnale, ripetuta per ogni jump up e per ogni jump down; prima di passare alla descrizione tecnica sembra opportuno illustrare a grandi linee il funzionamento del codice. Dato che si considerano sempre due jumps up e due jumps down per guadagno, l'idea seguita è quella di ottenere, tramite l'analisi in frequenza, due vettori booleani così costruiti:

- il primo vettore booleano è relativo ai due jumps up considerati. Il primo elemento del vettore vale 0 se il primo jump non presenta oscillazioni, 1 altrimenti; il secondo elemento del vettore vale 0 se il secondo jump non presenta oscillazioni, 1 altrimenti;

- il secondo vettore booleano è relativo ai due jumps down considerati. Il primo elemento del vettore vale 0 se il primo jump non presenta oscillazioni, 1 altrimenti; il secondo elemento del vettore vale 0 se il secondo jump non presenta oscillazioni, 1 altrimenti;

Sommando gli elementi di entrambi i vettori si ottengono due valori scalari, indicati con `oscUP` e `oscDOWN` nel codice, che, divisi per 2, permettono di valutare se un determinato guadagno è quello critico o meno. In particolare il criterio adottato è il seguente: se almeno il 50% dei jumps up o almeno il 50% dei jumps down presenta oscillazioni, allora il guadagno attuale è quello da calibrare in centralina.

Tornando al codice, si procede con l'applicazione della trasformata di Fourier a ogni jump up per determinato guadagno; la procedura relativa ai jumps down è esattamente identica, pertanto qui ci si limiterà ad esaminare i jumps up. Alle righe $102 - 107$ si esegue la finestratura dei vari segnali per isolare i vari jumps; è interessante sottolineare che alla riga 103 viene effettuata un'ulteriore finestratura sull'angolo VVT target, a partire da 0.5 secondi dopo l'inizio del jump e per una durata di 2 secondi, come rappresentato in figura 3.13. Questo perché il contributo integrale del PID potrebbe smorzare l'entità delle oscillazioni fino ad annullarla del tutto; escludendo la parte iniziale del segnale dall'analisi in frequenza il codice rivelerà solo quelle oscillazioni effettivamente persistenti, escludendo eventuali instabilità immediatamente seguenti l'overshoot e insignificanti ai fini della presente trattazione. Le righe $108 - 118$ contengono l'implementazione della trasformata di Fourier, tramite il comando `fft`; tutti gli ulteriori accorgimenti servono ad eliminare le frequenze specchiate e a restringere l'analisi a valori di frequenza compresi nell'intervallo $[3; 10]$ Hz. Tale scelta è stata fatta in seguito alla constatazione che la frequenza tipica delle oscillazioni da noi ricercate era circa pari a 6 Hz, come si può ben vedere in figura 3.14.

La presenza di oscillazioni in ogni jump viene valutata alle righe $119 - 121$: in pratica si verifica se il massimo dello spettro `semiAmpiezze1` è maggiore della soglia `sAmpLimite` definita alla riga 7, e qualora ciò accada si incrementa lo scalare `oscUP` di un'unità. Infine, alla riga 123, si divide il valore così ottenuto per 2 (utilizzando l'operatore '`/`' di divisione intera), che è il numero di jumps considerati, e lo si memorizza nel vettore `oscillazioniUP` . Al termine dell'analisi di tutti i guadagni relativi a un determinato breakpoint di mappa `oscillazioniUP` sarà un vettore costituito da soli 0 e 1 (il vettore booleano di cui si è parlato in precedenza); cercando il primo elemento pari a 1 (righe $153 - 156$) si può risalire al valore critico di guadagno proporzionale. Le righe seguenti $161 - 170$ sono state inserite in modo da presentare il risultato dell'analisi dei dati come output grafico: si ricorda che il valore di calibrazione è pari al guadagno critico diviso per 2. Ecco quindi che si possono distinguere vari casi:

1. non sono state trovate oscillazioni su nessun jump per nessun guadagno (riga 161): il guadagno critico è l'ultimo disponibile nell'acquisizione, ovvero il maggiore;

2. sono state trovate oscillazioni solo nei jumps down o solo nei jumps up (righe 163 e 165): il guadagno critico è quello relativo al primo jump su cui sono state

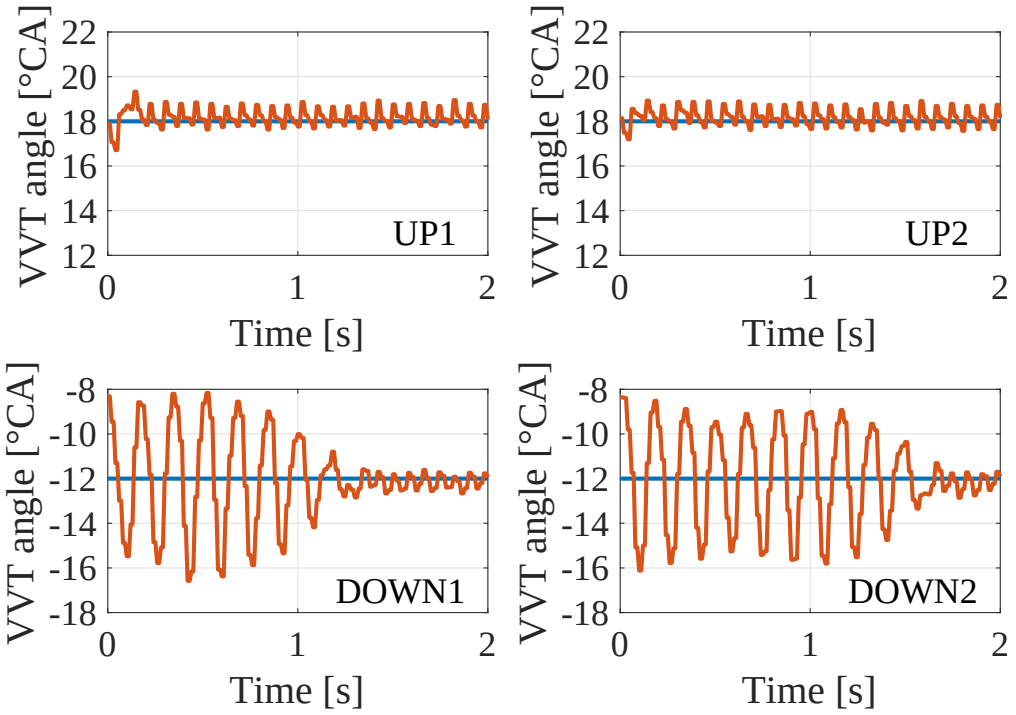


Figura 3.13: Prova a $T_{olio} = 90$ °C e 1500 rpm – Angolo VVT target (in blu) e misurato (in rosso)

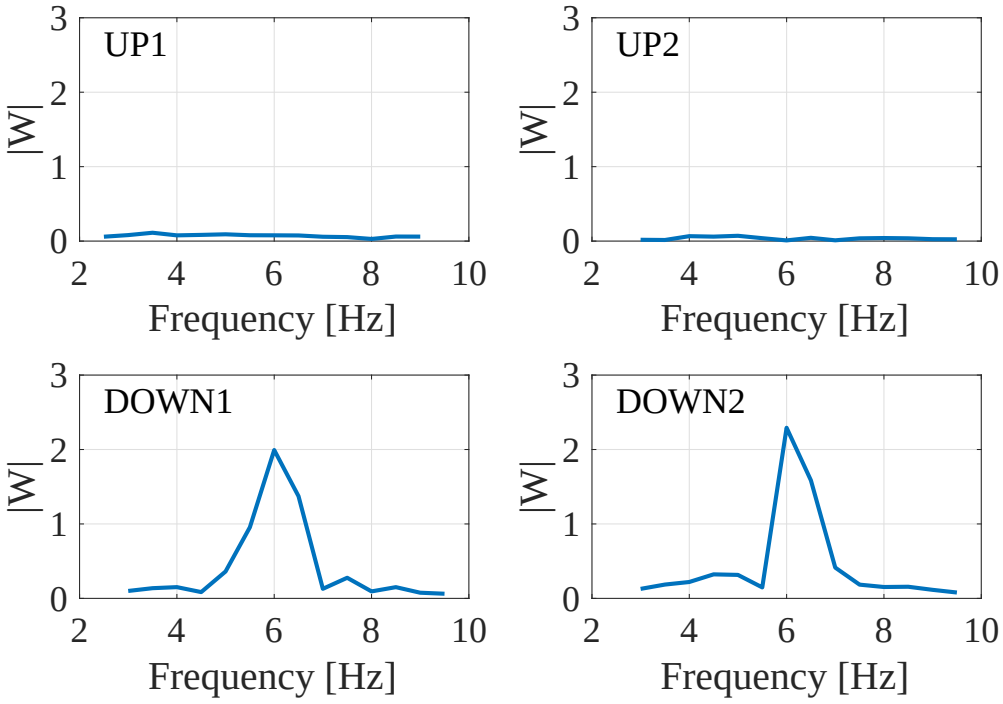


Figura 3.14: Prova a $T_{olio} = 90$ °C e 1500 rpm – Spettro del segnale misurato

individuate le oscillazioni;

3. sono state trovate oscillazioni sia nei jumps up che nei jumps down (riga 167): il guadagno critico è il minore tra i due guadagni, uno relativo ai jumps up e l'altro relativo ai jumps down, che hanno causato oscillazioni.

Un'ultima considerazione sulle figure 3.13 e 3.14: osservando il primo jump up si nota che la finestratura manuale a partire da 0.5 secondi dopo l'inizio del jump è servita ad eliminare una leggera instabilità immediatamente seguente l'overshoot, tant'è vero che lo spettro derivante dall'analisi in frequenza è assolutamente privo di picchi, esattamente identico allo spettro del secondo jump up che non presenta oscillazioni degne di nota. Tutt'altro discorso va fatto per i due jumps down: si vede chiaramente che il primo jump down, e ancor più il secondo, presenta delle oscillazioni chiaramente visibili anche in seguito alla finestratura manuale, e questo viene ampiamente confermato dagli spettri visibili in figura 3.14. In particolare, come già accennato in precedenza, si nota che il picco in entrambi gli spettri è localizzato in corrispondenza di una frequenza pari a 6 Hz; si può anche dedurre che le soglie impostate e la finestratura effettuata hanno permesso di condurre un'analisi particolarmente efficiente dato che la trasformata di Fourier ha individuato perfettamente le oscillazioni principali da noi ricercate, mentre le oscillazioni secondarie di minor ampiezza, dovute essenzialmente al fatto che il sistema di controllo è per la maggior parte scalibrato, non presentano evidenza alcuna sullo spettro.

Nella tabella 3.7 sono riportati i valori di calibrazione ottenuti dalle acquisizioni effettuate; confrontando tali risultati con i valori dello storico calibrazioni, che per riservatezza non viene riportato, si nota subito che entrambe le mappe presentano un andamento leggermente crescente con la temperatura dell'olio, e tuttavia già il trend decrescente all'aumentare del regime di rotazione riscontrabile nella mappa dello storico non è più confermato dai risultati ottenuti dal nostro test. Bisogna comunque sottolineare che i punti motore esaminati nelle prove da noi effettuate non sono sufficienti per individuare qualche trend significativo.

	1500 rpm	3000 rpm	4000 rpm
90 °C	2.00	1.90	2.29
105 °C	2.19	2.00	2.49

Tabella 3.7: Risultati ottenuti per la calibrazione di KPMAP

3.4.5 KDMAP

La calibrazione della mappa KDMAP prevede l'esecuzione di una prima prova di riferimento e poi delle prove standard con mappa spianata a valori non nulli e di volta in volta

crescenti. Per comodità, in fase di acquisizione, si è pensato di suddividere le prove di riferimento da quelle standard, per ogni breakpoint; si avranno pertanto a disposizione dodici file *csv* al termine della prova, escludendo le prove in bassa pressione che non verranno qui descritte, stante il piano prove riportato nella tabella 3.16.

	1500 rpm	3000 rpm	4000 rpm
$T_{olio} = 90$ °C	X (Hi + Lo)	X (Hi)	X (Hi)
$T_{olio} = 105$ °C	X (Hi + Lo)	X (Hi)	X (Hi)

Tabella 3.8: Piano prove per la calibrazione di KDMAP

Nelle figure 3.15 e 3.16 si possono vedere i segnali acquisiti e salvati in ogni file *csv*; si fa notare che il numero di jumps effettuati nella prova di riferimento è decisamente superiore al numero di jumps per singolo guadagno effettuati nelle prove standard, in accordo con la procedura di calibrazione illustrata nella sezione precedente.

Come sarà spiegato nel seguito non tutti i jumps di riferimento verranno considerati nel calcolo, ma quelli ritenuti non accettabili verranno scartati; in questo senso il valore di guadagno, che è pari a 0, non viene utilizzato se non per riconoscere ed eventualmente confermare che la prova in esame è di riferimento o meno. Nelle prove standard invece il valore di KDMAP verrà impiegato per distinguere le varie prove al variare del guadagno derivativo, dato che come nel caso del guadagno proporzionale si è scelto di effettuare un'unica acquisizione mentre si aumentava di volta in volta il guadagno derivativo, per esigenze di tempo e comodità.

In appendice, sezione A.4, è riportato il codice scritto per l'analisi e il trattamento delle acquisizioni; come per KPMAP, tale codice è stato pensato per analizzare acquisizioni relative a una determinata temperatura olio per volta e quindi, visto il piano prove prima riportato, lo script dovrà essere eseguito per due volte. Dovendo tuttavia analizzare anche le prove di riferimento si è pensato di integrare tale analisi nello stesso script, in modo che prima vengano elaborate le prove standard con guadagno diverso da 0 e poi quella di riferimento; segue infine il calcolo del coefficiente di prestazione. Sono stati previsti dei plot intermedi per mostrare l'andamento dei due parametri di prestazione calcolati.

Passando alla descrizione del codice, è facile riconoscere che le righe iniziali fino alla 112, contenenti l'apertura dei file di acquisizione, la finestratura dei guadagni e la finestratura dei jumps, sono esattamente identiche a quelle già descritte nel paragrafo relativo a KPMAP; la parte specifica per KDMAP si trova a partire dalla riga 113. Da qui in avanti si calcolano, per ogni jump e per ogni guadagno, i due parametri necessari al calcolo dell'indice di prestazione. Il calcolo viene ripetuto due volte, prima per i jumps up e poi per i jumps down, ma concettualmente il procedimento è identico: l'overshoot viene determinato alle righe 116 e 127 come differenza tra il massimo (o il minimo, per i

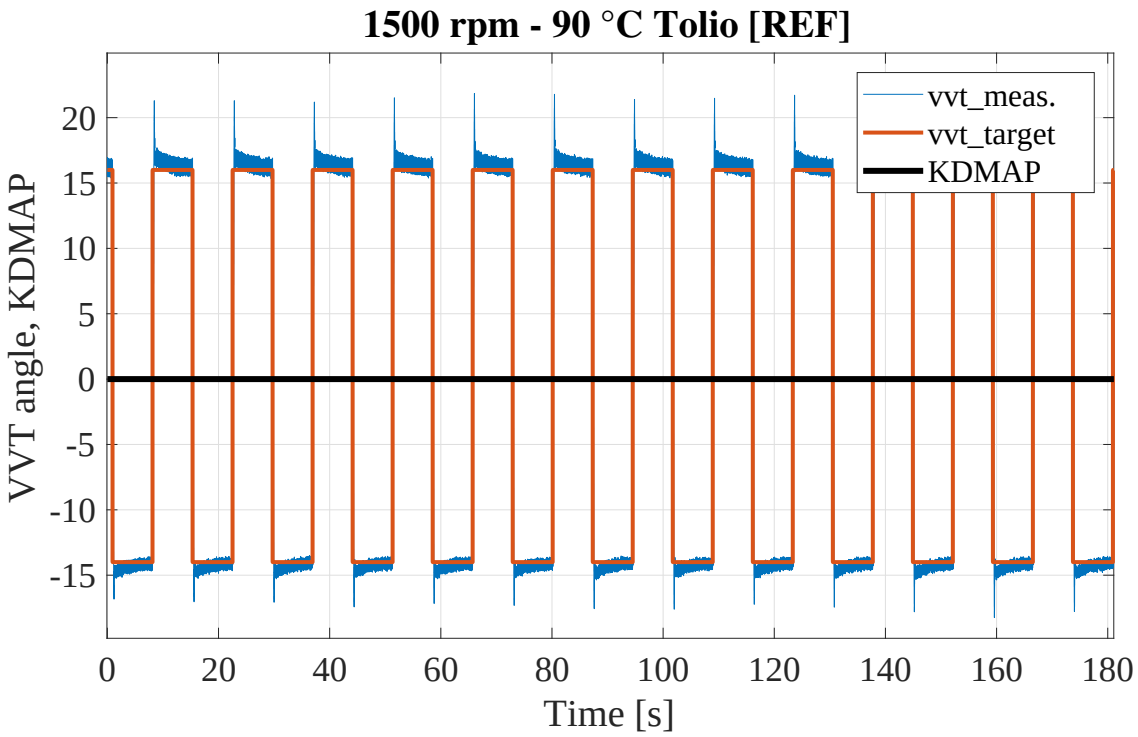


Figura 3.15: Contenuto di un file *csv* – Prova di riferimento

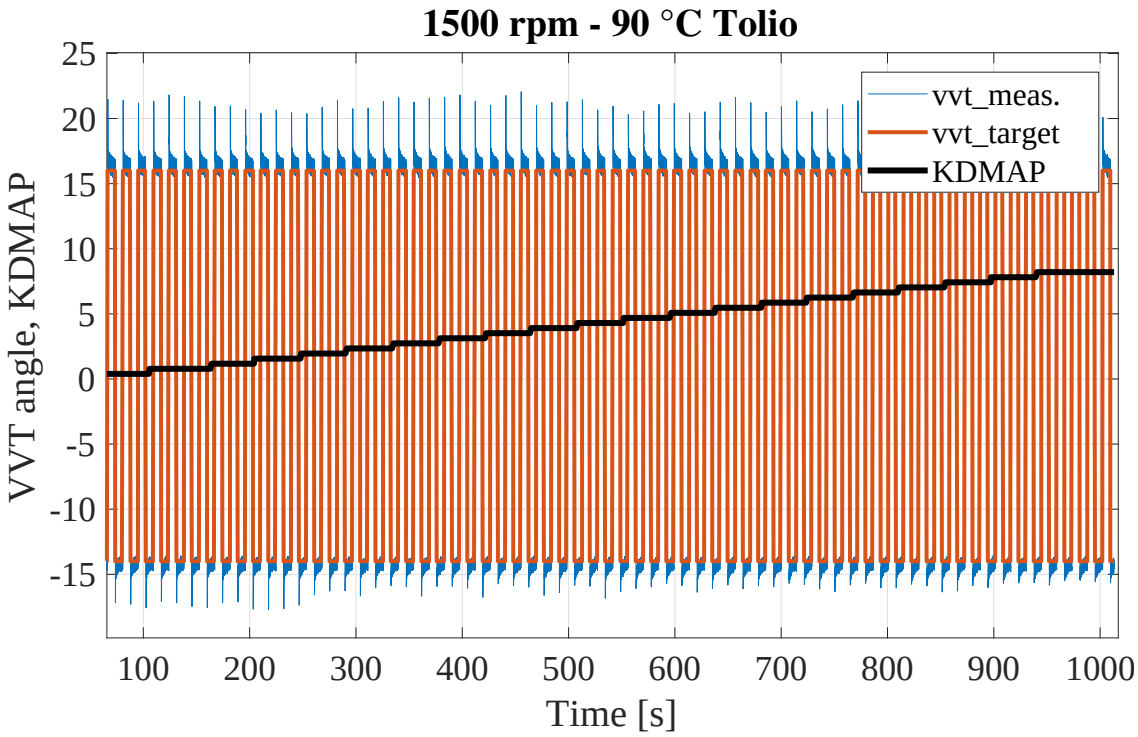


Figura 3.16: Contenuto di un file *csv*

jumps down) dell'angolo misurato *vvt_measured* e il massimo (o il minimo) dell'angolo target *vvt_target*, mentre il T_{0-99} viene determinato alle righe $140 - 151$ per i jumps up e $155 - 166$ per i jumps down. È rilevante sottolineare che il calcolo di T_{0-99} deriva dall'individuazione dell'istante temporale in corrispondenza del quale *vvt_measured* raggiunge il 99% di *vvt_target*: questo viene trovato tramite il comando `diff`, specificando l'opzione `first` in modo da ottenere solo il primo indice trovato, e memorizzato volta per volta nella variabile `i99`. In modo analogo viene determinato l'istante iniziale `i0` per il calcolo di T_{0-99}. Sia per quanto riguarda l'overshoot che per il T_{0-99} i valori relativi a ogni jump vengono memorizzati nei vettori omonimi, distinguendo i jumps up dai jumps down; alle righe 122 e 133 per l'overshoot, 151 e 166 per il T_{0-99}) vengono poi calcolati i valori medi di tali vettori, a loro volta salvati nella struttura `set` in modo tale da poter disporre di un'unica variabile all'interno della quale sono presenti tutti i risultati di una determinata prova, in modo ordinato e facilmente identificabile.

Le righe $174 - 190$ sono state inserite per determinare le rette che interpolano i valori di overshoot e T_{0-99} calcolati nelle varie prove standard; nelle figure 3.17 e 3.19, che verranno dettagliatamente spiegate nel seguito della trattazione, sono state rappresentate con linea tratteggiata. La parte di analisi delle prove standard si conclude con le righe $193 - 216$, nelle quali si rappresentano graficamente tutte le grandezze calcolate e relative alla prova analizzata.

Dalla riga 227 ha inizio la parte di analisi delle acquisizioni di riferimento, che è in larga parte simile al codice relativo alle prove standard appena descritto, a parte l'identificazione dei vari guadagni che in questo caso è inutile dato che nelle prove di riferimento la mappa KDMAP viene sempre mantenuta spianata a 0. Leggermente diverse sono le righe 302 e seguenti, dove vengono calcolati i parametri di prestazione in modo analogo alle prove standard, e tuttavia in questo caso è stato necessario inserire un'ulteriore procedura che eliminasse le prove di riferimento non accettabili. A tal fine sembra opportuno ricordare che la procedura di calibrazione proposta in precedenza per KDMAP consiglia di effettuare almeno tre prove di riferimento ognuna delle quali costituita da almeno cinque transizioni di angolo VVT target, e al termine delle quali scartare le prove i cui jumps non soddisfavano un determinato criterio. Nel nostro caso, come accennato in precedenza, le acquisizioni di prova sono state riunite in un unico file *csv* per ogni breakpoint esaminato, pertanto è stato possibile esaminare tutti i jumps relativi a un determinato punto motore e scartare quelli che risultavano troppo dispersi rispetto alla mediana. Alle righe 311 e 326 vengono calcolate la mediana di tutti i valori di overshoot up e la mediana di tutti i valori di overshoot down, che verranno poi richiamate rispettivamente nelle variabili `medianaUP` e `medianaDOWN` per determinare la validità del singolo jump. In particolare si fa notare che la soglia di accettabilità per l'overshoot è pari al $\pm 10\%$ della mediana,

mentre l'eliminazione dei valori non accettabili viene effettuata alle righe $333 - 341$ per i jumps up e alle righe $344 - 352$ per i jumps down; i valori che soddisfano il criterio si trovano nei vettori `OS_up` e `OS_down` .

Un ragionamento analogo va ripetuto per il T_{0-99}: alle righe 375 e 395 vengono calcolate le mediane di tutti i valori di T_{0-99} per i jumps up e down, rispettivamente, mentre alle righe $402 - 410$ e $413 - 421$ si trovano le due procedure per l'eliminazione dei valori non accettabili. Si noti che in questo caso le soglie di accettabilità sono notevolmente più ristrette rispetto a quelle dell'overshoot: si è scelta una tolleranza del $\pm 2.5\%$ rispetto alla mediana, a fronte del $\pm 10\%$ impiegato per l'overshoot. Un tale valore si è reso necessario dal momento che, come si vede nella figura 3.20, i valori di T_{0-99} si sono rivelati molto meno dispersi rispetto agli overshoot, sia nelle prove a 90 °C che nelle prove a 105 °C di temperatura dell'olio (le prove riportate presentano un T_{0-99} addirittura costante e indipendente dal regime) tanto da ipotizzare che i valori di KDMAP impostati non siano stati sufficientemente elevati da individuare una qualche dipendenza del T_{0-99}. Si ricorda ancora una volta, a tal proposito, che durante tutte le prove qui analizzate il sistema di controllo risultava parzialmente scalibrato dato che VBATCOMP, PRECTL e KPMAP sono state lasciate ai valori di default presenti in centralina.

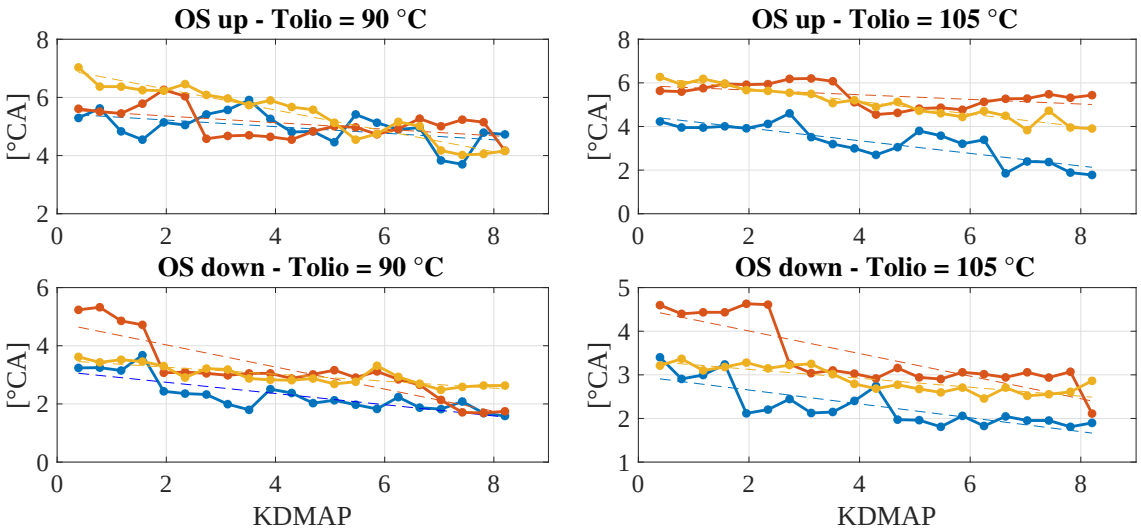


Figura 3.17: Risultati delle prove – Overshoot, prove standard

	1500 rpm	3000 rpm	4000 rpm
90 °C	$5.50 \uparrow 3.45 \downarrow$	$5.30 \uparrow 4.88 \downarrow$	$6.96 \uparrow 3.54 \downarrow$
105 °C	$4.54 \uparrow 3.66 \downarrow$	$5.20 \uparrow 4.15 \downarrow$	$6.47 \uparrow 3.43 \downarrow$

Tabella 3.9: Valori medi degli overshoot di riferimento, [° CA]

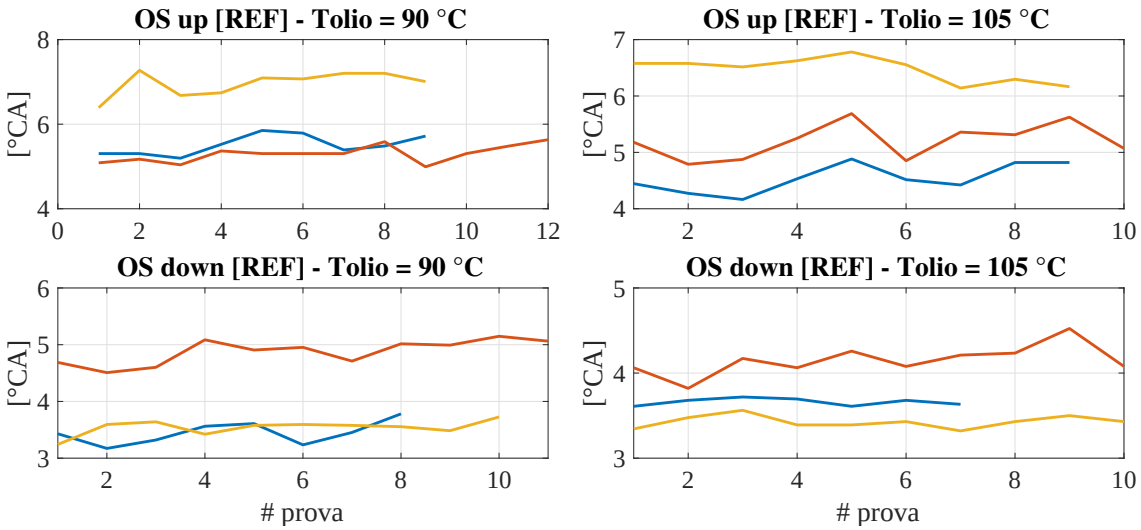


Figura 3.18: Risultati delle prove – Overshoot, prova di riferimento

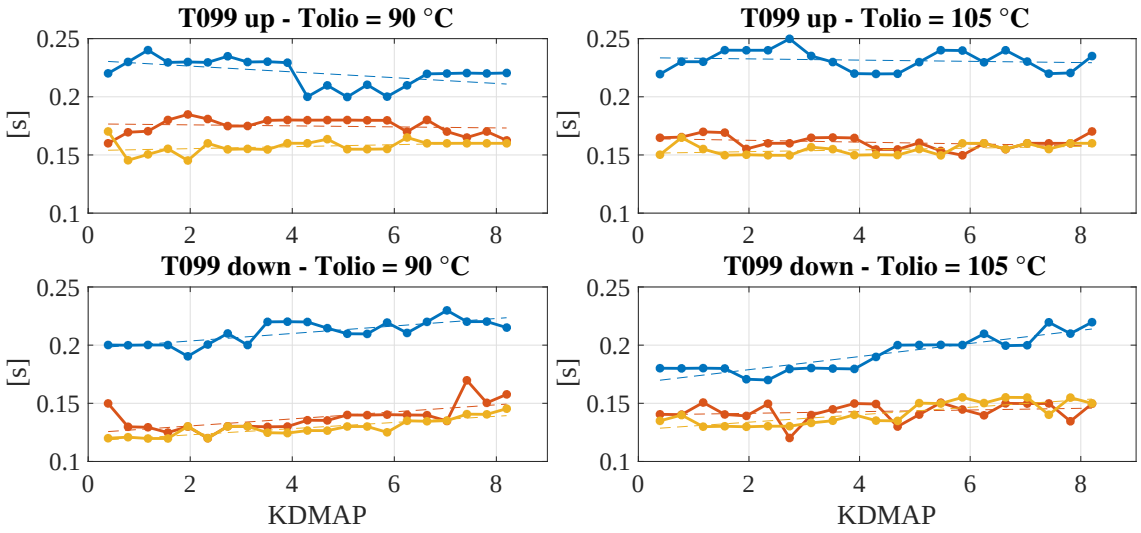


Figura 3.19: Risultati delle prove – T_{0-99}, prove standard

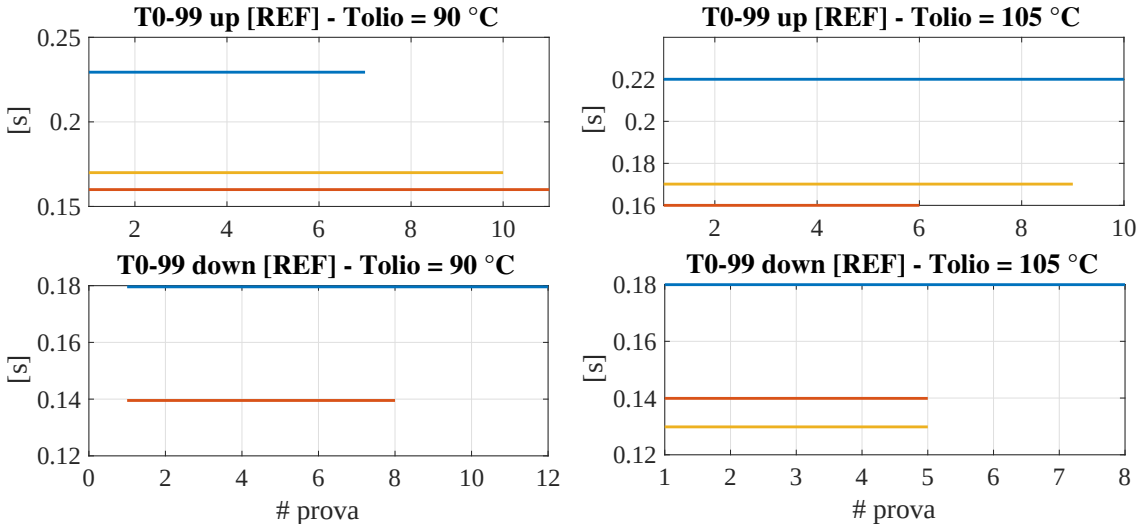


Figura 3.20: Risultati delle prove – T_{0-99}, prova di riferimento

	1500 rpm	3000 rpm	4000 rpm
90 °C	0.23 ↑ 0.18 ↓	0.16 ↑ 0.14 ↓	0.17 ↑ 0.14 ↓
105 °C	0.22 ↑ 0.18 ↓	0.16 ↑ 0.14 ↓	0.17 ↑ 0.13 ↓

Tabella 3.10: Valori medi di T_{0-99} di riferimento, [s]

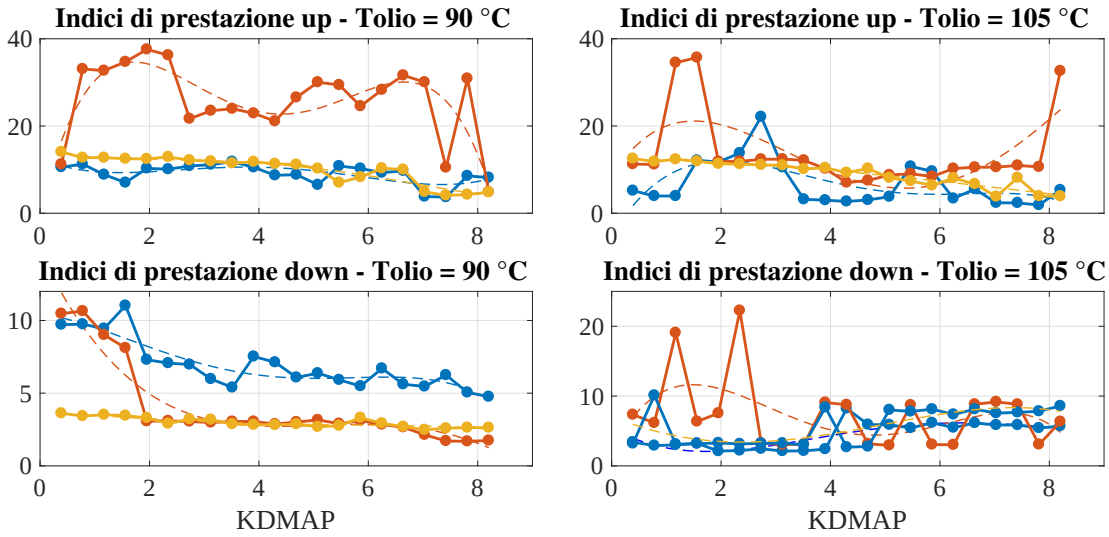


Figura 3.21: Indici di prestazione

Nelle figure precedenti sono riportati tutti i risultati ottenuti dall'analisi delle prove effettuate: in tutti i grafici le curve in blu sono relative alle prove a 1500 rpm, le curve in rosso sono relative alle prove a 3000 rpm mentre le curve in arancione sono relative alle prove a 4000 rpm.

A prima vista le figure 3.17 e 3.19 sembrano confermare il comportamento tipico di un PID: all'aumentare del guadagno derivativo l'overshoot si riduce mentre il T_{0-99} aumenta, e tuttavia i due parametri mostrano delle variazioni molto deboli, soprattutto per quanto riguarda il T_{0-99}, rimanendo in alcuni casi addirittura costanti.

Osservando invece le figure 3.18 e 3.20 si nota che sia per quanto riguarda l'overshoot sia per quanto riguarda il T_{0-99} è stato necessario eliminare delle prove; addirittura, nella prova a 3000 rpm nessuna prova è stata ritenuta accettabile. In casi come questo il codice determina il valore di riferimento facendo la media sui valori di tutte le prove; è chiaro che sarebbe opportuno implementare una strategia un po' più complessa ed efficiente per la gestione delle prove scartate, ed è quello che verrà fatto nel tool di automazione.

	1500 rpm	3000 rpm	4000 rpm
90 °C	7.42	7.03	7.03
105 °C	7.81	5.08	4.29

Tabella 3.11: Risultati ottenuti per la calibrazione di KDMAP

Nella tabella 3.11 si riportano i valori di KDMAP determinati dal codice. Dopo averli confrontati con lo storico calibrazioni a disposizione da un lato si è rilevato, come era già stato fatto per KPMAP, che i breakpoints esaminati nelle acquisizioni di prova sono troppo pochi per poter individuare un trend o per poter stabilire che i valori ottenuti siano giusti o sbagliati, anche a fronte del fatto che non siamo a conoscenza delle modifiche

presenti nel motore oggetto della calibrazione rispetto al valore delle calibrazioni dello storico.

3.4.6 KIMAP

La calibrazione dell'ultima mappa KIMAP prevede, in modo analogo a quanto già esposto per KDMAP, l'esecuzione di una prima prova di riferimento e poi delle prove standard con mappa spianata a valori non nulli e di volta in volta crescenti. Tuttavia in questo caso, per mancanza di tempo, non è stato possibile differenziare le acquisizioni di riferimento da quelle standard, e pertanto al termine della fase di prova ed in seguito alla conversione si avranno a disposizione sei file *csv* relativi alle prove in alta pressione secondo il piano prove riportato nella tabella 3.12:

	1500 rpm	3000 rpm	4000 rpm
$T_{olio} = 90$ °C	X (Hi + Lo)	X (Hi)	X (Hi)
$T_{olio} = 105$ °C	X (Hi + Lo)	X (Hi)	X (Hi)

Tabella 3.12: Piano prove per la calibrazione di KIMAP

Come si può vedere nella figura 3.22 la spazzolata di guadagni esaminati è decisamente ampia: si è partiti da un valore di 0.1 e si è arrivati fino a 3 in step di 0.1, dando origine a prove di durata elevata e superiore ai 20 minuti. I file *dat* risultanti erano decisamente pesanti, tanto che in questo caso più che negli altri si è avvertito il beneficio di estrarre solo i segnali di interesse prima di convertire le acquisizioni in file *csv*.

Un'ulteriore considerazione prima di passare all'analisi del codice: come già anticipato durante la descrizione della procedura di calibrazione, il trattamento dei segnali acquisiti per KIMAP prevede l'applicazione di un filtro a media mobile all'angolo VVT misurato. Si tratta di una metodologia già consigliata ed applicata dal relatore di questa tesi nel corso dell'attività precedente a quella qui descritta, e resa necessaria per il calcolo del tempo stazionario, come si vedrà; in figura 3.23 si può vedere il confronto tra il segnale prima e dopo l'applicazione del filtro.

È opportuno sottolineare che la ricerca del numero ottimo di campioni di cui effettuare la media ha richiesto non poco tempo in quanto era necessario che le oscillazioni a regime introdotte dal PID, e ben evidenti nel segnale originale, venissero completamente smorzate per non falsare il calcolo del tempo stazionario, e tuttavia era anche necessario che la media mobile non stravolgesse del tutto il segnale originale dato che questo filtro introduce un ritardo tanto più grande quanto maggiore è il numero di campioni mediati. Dopo opportuni confronti si è scelto un valore pari a 40.

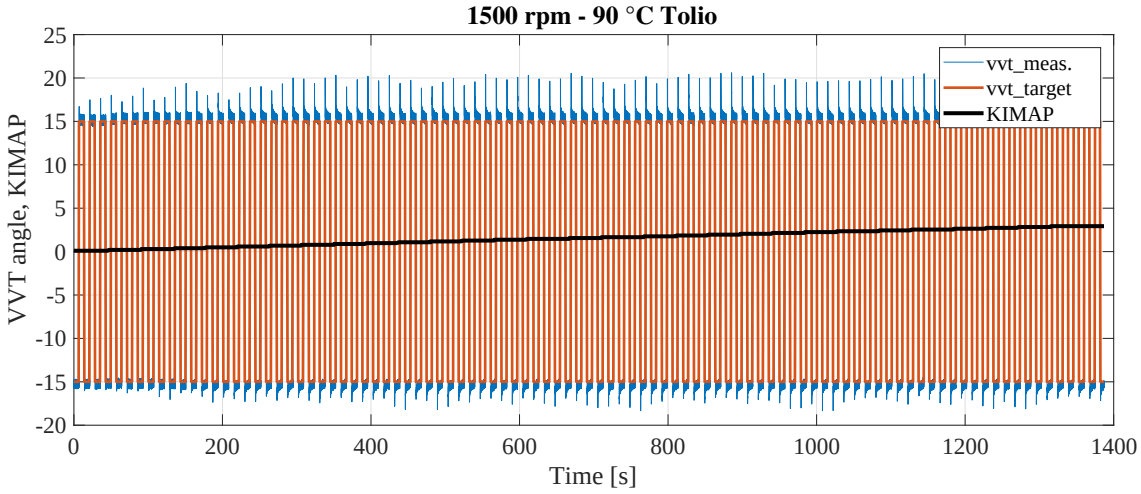


Figura 3.22: Contenuto di un file *csv*

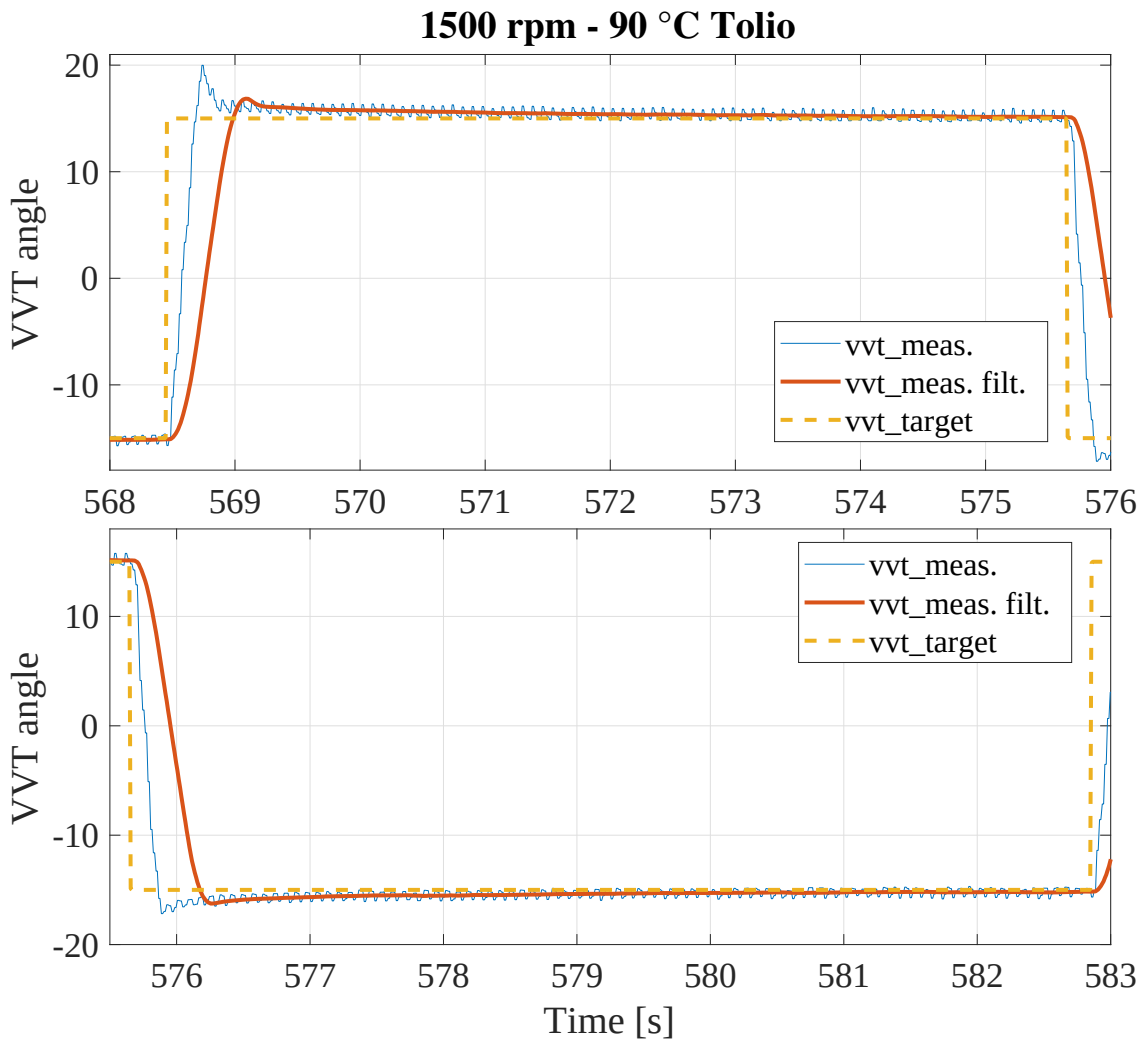


Figura 3.23: Confronto tra segnale originale e segnale filtrato con media mobile

In appendice, sezione A.5, è riportato il codice scritto per l'analisi e il trattamento delle acquisizioni; come per KDMAP, tale codice è stato pensato per analizzare acquisizioni relative a una determinata temperatura olio per volta e quindi, visto il piano prove prima riportato, lo script dovrà essere eseguito per due volte. A differenza del codice per il guadagno derivativo, tuttavia, quello che verrà descritto nel seguito è in grado di riconoscere automaticamente la prova di riferimento in base al valore di guadagno impostato; tale metodologia si è resa necessaria in quanto la prova di riferimento è contenuta nello stesso file delle prove standard, e per il calcolo dell'indice di prestazione risulta chiaramente necessario individuare prima i parametri di riferimento.

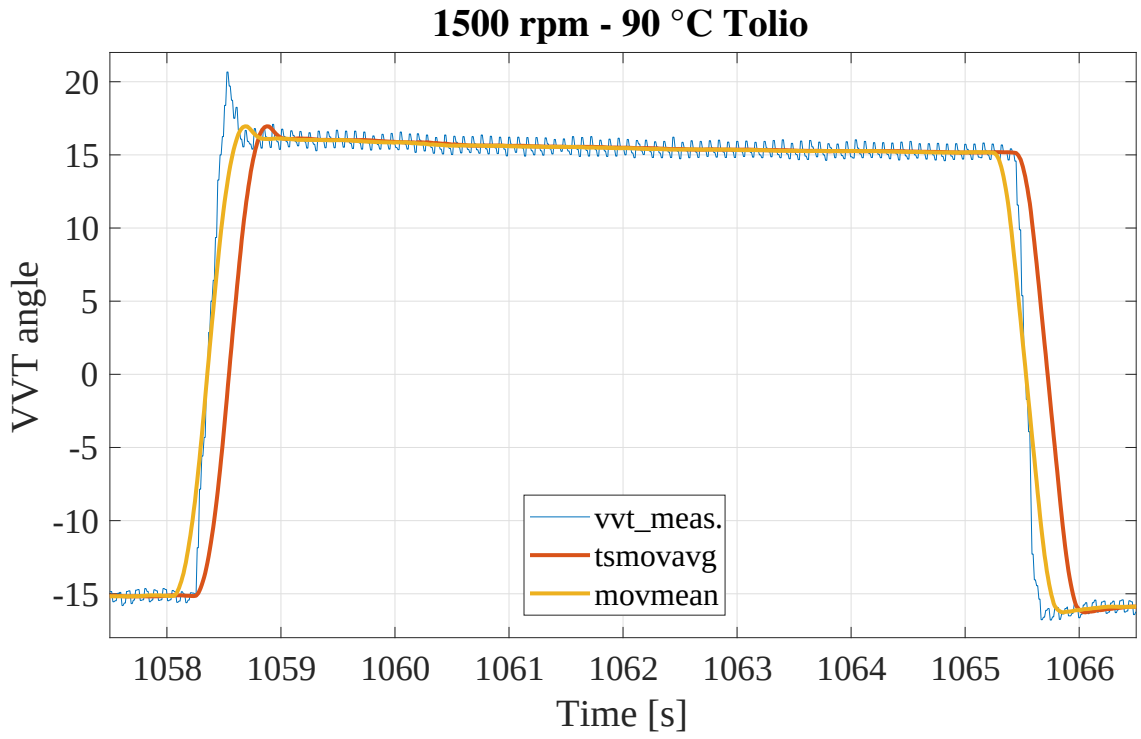


Figura 3.24: Confronto tra `movmean` e `tsmovavg`

Non vale la pena fermarsi sulle righe iniziali che, come al solito, contengono i comandi necessari all'apertura delle acquisizioni e alla lettura dei dati in esse contenuti. Le prime righe interessanti si trovano a partire dalla 37, dove viene applicato il filtro a media mobile sul segnale *vvt_measured*. Si fa notare che MATLAB mette a disposizione due funzioni relative a tale filtro: `movmean` e `tsmovavg`. Guardando le righe 38 e 39 si può vedere come sia stata scelta la seconda, ed osservando la figura 3.24 è subito chiaro il motivo: `movmean` prevede al suo interno un algoritmo per l'eliminazione del ritardo, e infatti si può vedere che il segnale giallo in figura è anticipato rispetto a quello rosso. Tuttavia si vede anche come il jump del segnale giallo sia anticipato anche rispetto al segnale fisico misurato, rappresentato in blu, e questo non è assolutamente accettabile ai fini

della presente analisi in quanto il tempo stazionario non corrisponderebbe più alla realtà; confrontando infatti i segnali giallo e blu agli istanti 1058 e 1066 si vede che quello giallo anticipa sia la transizione in up che quella in down. Ecco quindi che si è deciso di utilizzare la funzione `tsmovavg` in quanto, pur ritardando notevolmente il segnale, quanto meno non modificava l'istante di inizio delle transizioni in up e in down. Vale la pena precisare che il segnale filtrato è stato utilizzato ai fini del calcolo del tempo stazionario e dell'errore a regime; l'overshoot viene calcolato sul segnale originale, come è lecito aspettarsi.

Dalla riga 46 in poi seguono i vari comandi per l'identificazione dei vari guadagni e per la finestratura dei jumps, in modo assolutamente analogo a quanto già visto in precedenza. È interessante invece soffermarsi sulle righe che seguono la 164: qui si trova l'algoritmo di calcolo del tempo stazionario, totalmente diverso da quanto riporta il manuale di calibrazione compilato dal professore nel corso dell'attività precedente. Tale metodologia di calcolo è totalmente nuova ed è stata costruita su misura per il caso di studio attuale, nonostante preveda in ogni caso la definizione di una banda di tolleranza entro la quale si può considerare il segnale stazionario: la riga 165 definisce appunto `toll_staz` pari a 0.2. L'idea che sta alla base del calcolo è abbastanza semplice: in pratica bisogna controllare, per ogni jump, se il singolo campione del segnale filtrato appartiene o meno alla predetta banda (righe $169 - 176$), e in base all'esito del controllo si pone l'elemento m-esimo del vettore `check` pari a 1, se il campione m-esimo rientra nella banda, oppure pari a 0 se il campione m-esimo non vi rientra. Alle righe $177 - 185$ si procede con l'analisi di tale vettore: prima di tutto si verifica che non sia vuoto tramite l'`if` più esterno, quindi tramite il comando `find` si trova l'indice che corrisponde all'ultimo '1' di `check` (riga 178) e si eliminano tutti gli elementi che seguono (riga 179). Dopodiché si inverte `check` tramite la funzione `fliplr` e si ricerca sul vettore così ottenuto l'indice che corrisponde al primo '0', salvandolo nella variabile `startIdx` (riga 180); quindi si sottrae tale valore dalla lunghezza di `check` e si salva il numero così ottenuto nel vettore `startIdxUP`, che alla fine dell'analisi conterrà il numero di campioni, per ogni jump, che appartengono alla banda di tolleranza prima definita. Il calcolo del tempo stazionario viene effettuato alla riga 182 semplicemente moltiplicando il reciproco della frequenza di campionamento `dt`, determinato alla riga 166, per il numero di campioni `startIdxUP`, ottenendo così il tempo cercato. Si fa notare che una simile procedura, che a prima vista può apparire complessa e contorta, si è resa necessaria in quanto il tempo stazionario è definito come il tempo che impiega il segnale misurato a rientrare nella banda di stazionarietà senza più uscirvi; proprio quest'ultima parte della definizione ha obbligato ad effettuare più ricerche sul vettore `check` tramite la funzione `find`, invertendo anche il vettore in modo da ricercare volta per volta l'istante preciso in cui il segnale entrava nella banda di tolleranza e non ne

usciva più. Si vuole infine sottolineare che qualora il vettore `check` risultasse vuoto (riga 183) il tempo stazionario relativo al jump in esame viene posto uguale a 0 (riga 184). Il tempo determinato viene in ogni caso salvato nella variabile `Tstaz_up_medio`, la quale a sua volta verrà salvata nella struttura `set` che contiene tutti i risultati del set di prove. Quanto appena descritto è relativo al calcolo del tempo stazionario per i jumps up, ma è perfettamente applicabile anche ai jumps down, alle righe 191 e seguenti.

Diverso è invece il procedimento seguito per il calcolo dell'errore medio a regime, definito come valore medio della differenza in valore assoluto tra angolo misurato ed angolo target, entrambi considerati a partire dall'istante `Tstaz` in cui *vvt_measured* può considerarsi stazionario. È chiaro che una simile definizione prescinde dal calcolo del tempo stazionario visto prima, ed anche per questo motivo si è cercato di mettere a punto un algoritmo di calcolo per il tempo stazionario che fosse efficiente, preciso ed a prova di errore. In questo senso il calcolo dell'errore medio a regime comporta molte meno difficoltà, come si può vedere alle righe $219 - 223$: si tratta semplicemente di calcolare il valore medio del segnale ottenuto come differenza tra `vvt_measured_mobile`, considerato nella sua fase di stazionario, e `vvt_target`. L'errore medio complessivo relativo a un determinato guadagno si ottiene poi effettuando la media degli errori calcolati per i vari jumps (riga 234). In modo del tutto simile si calcola l'errore medio a regime per i jumps down, alle righe $227 - 232$.

Si vuole sottolineare che, come già accennato prima, il codice deve essere in grado di riconoscere la prova di riferimento da quelle standard: dato che per quanto riguarda KIMAP il guadagno di riferimento è quello inferiore tra tutti i guadagni applicati, quindi presumibilmente il primo, alle righe $238 - 249$ si controlla l'indice `j` del guadagno attuale analizzato. Se `j` è pari a 1 i valori di overshoot, tempo stazionario ed errore medio a regime appena calcolati sono quelli di riferimento, e vengono salvati in variabili apposite.

Alla riga 275 ha inizio il calcolo dell'indice di prestazione con la definizione delle due soglie per l'errore medio a regime e per l'overshoot. Si ricorda che per il guadagno integrale l'indice di prestazione prevede il confronto sia tra overshoot standard e overshoot di riferimento, sia tra errore a regime standard ed errore a regime di riferimento; l'indice complessivo si ottiene moltiplicando i due coefficienti di vincolo c_1 e c_2 per il tempo stazionario. Il primo coefficiente di vincolo c_1 viene calcolato alle righe $280 - 285$ per i jumps up, e alle righe $307 - 312$ per i jumps down, mentre il secondo coefficiente c_2 viene calcolato alle righe $286 - 295$ per i jumps up, e alle righe $313 - 322$ per i jumps down; si noti che in entrambi i casi viene utilizzata la funzione integrata `interp1` con l'opzione `extrap` per interpolare linearmente i valori dei due coefficienti qualora cadano all'interno delle due saturazioni inferiore e superiore. Alle righe 296 e 323 vengono calcolati gli indici di prestazione, rispettivamente per i jumps up e per i jumps down; si è

scelto di calcolare anche l'indice di prestazione della prova di riferimento, che però verrà chiaramente escluso quando si tratterà di individuare l'indice minore. Questo avviene alle righe $298 - 302$ per i jumps up e alle righe $325 - 329$ per i jumps down; in entrambi i casi i due valori trovati vengono salvati in due variabili, rispettivamente `ip.up(k)` e `ip.down(k)`. Dalla riga 333 alla riga 343 viene individuato il minore tra i due indici salvati: questo controllo viene fatto semplicemente per dare un'informazione grafica all'utente se l'indice che è stato scelto sia relativo ai jumps up oppure ai jumps down. Le righe che seguono sono state introdotte per interpolare con delle polinomiali l'andamento degli indici di prestazione così ottenuti e per rappresentarle infine graficamente.

	1500 rpm	3000 rpm	4000 rpm
90 °C	2.57 ↑ 0.95 ↓	4.11 ↑ 2.78 ↓	3.66 ↑ 1.13 ↓
105 °C	3.10 ↑ 1.69 ↓	2.66 ↑ 2.20 ↓	3.50 ↑ 1.68 ↓

Tabella 3.13: Valori medi degli overshoot di riferimento, [° CA]

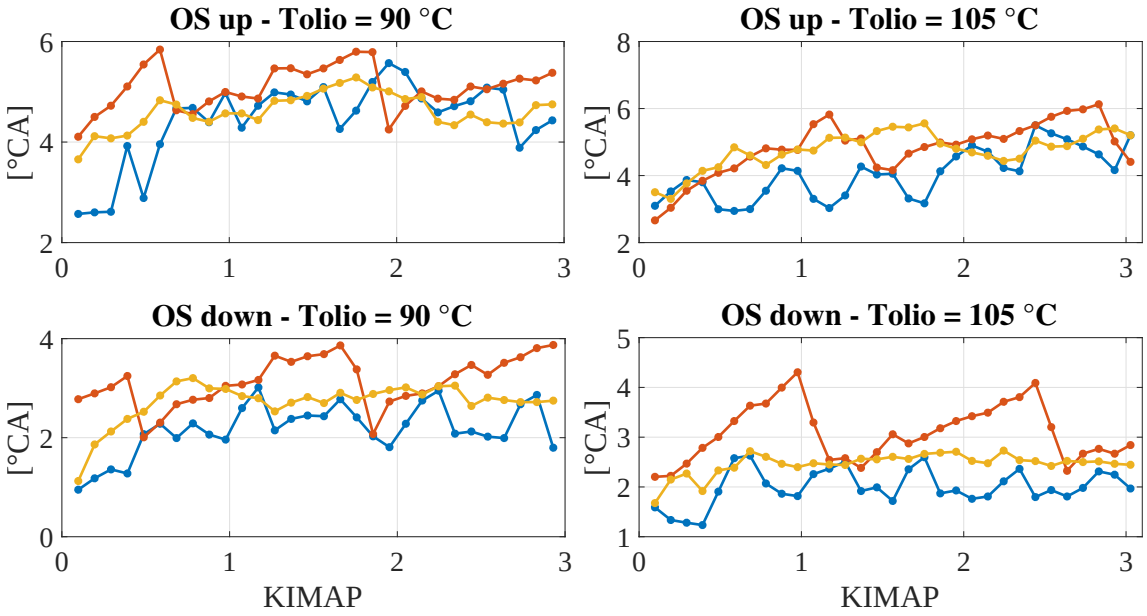


Figura 3.25: Risultati delle prove – Overshoot

	1500 rpm	3000 rpm	4000 rpm
90 °C	0.72 ↑ 6.20 ↓	0.63 ↑ 0.59 ↓	0.50 ↑ 0.49 ↓
105 °C	0.64 ↑ 0.75 ↓	7.12 ↑ 6.45 ↓	5.50 ↑ 0.48 ↓

Tabella 3.14: Valori medi dei tempi stazionari di riferimento, [s]

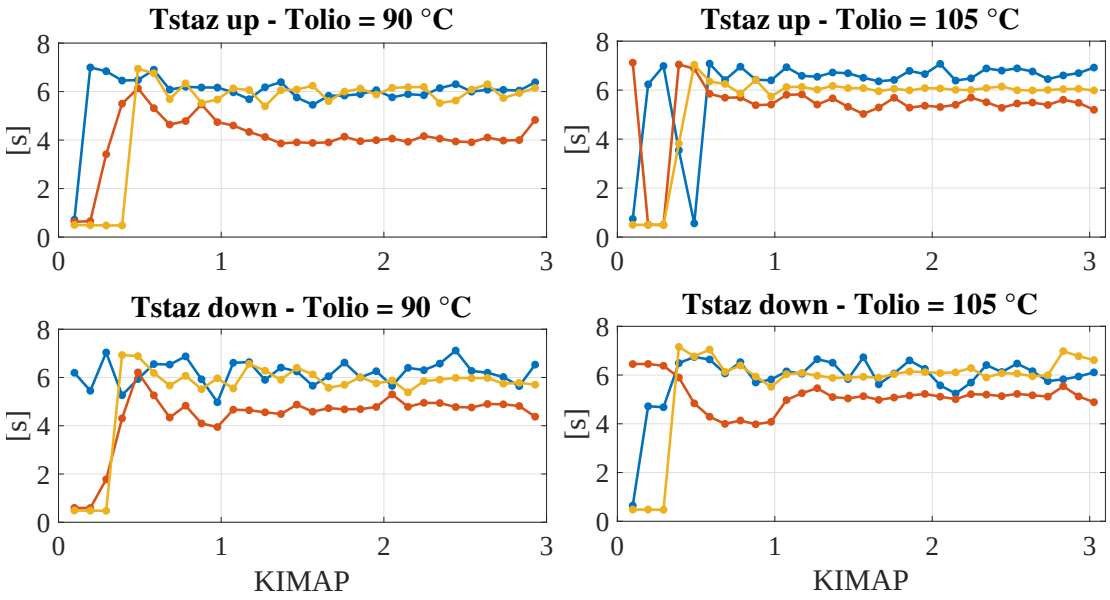


Figura 3.26: Risultati delle prove – Tempo stazionario

	1500 rpm	3000 rpm	4000 rpm
90 °C	0.05 ↑ 0.19 ↓	0.04 ↑ 0.05 ↓	0.49 ↑ 0.63 ↓
105 °C	0.11 ↑ 0.10 ↓	0.23 ↑ 0.25 ↓	0.45 ↑ 0.54 ↓

Tabella 3.15: Valori medi degli errori a regime di riferimento, [° CA]

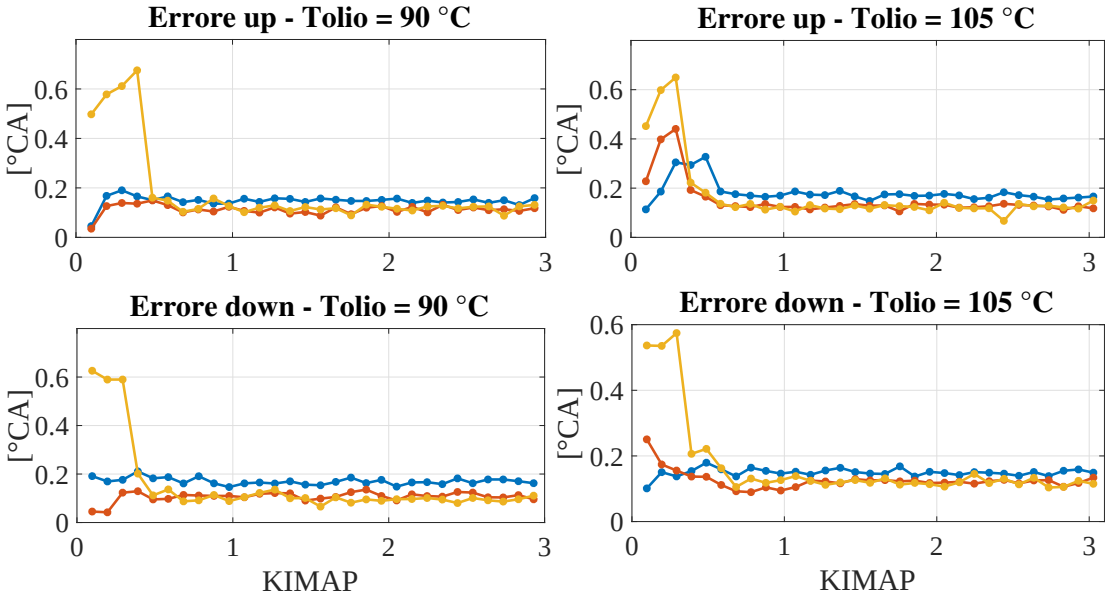


Figura 3.27: Risultati delle prove – Errore medio a regime

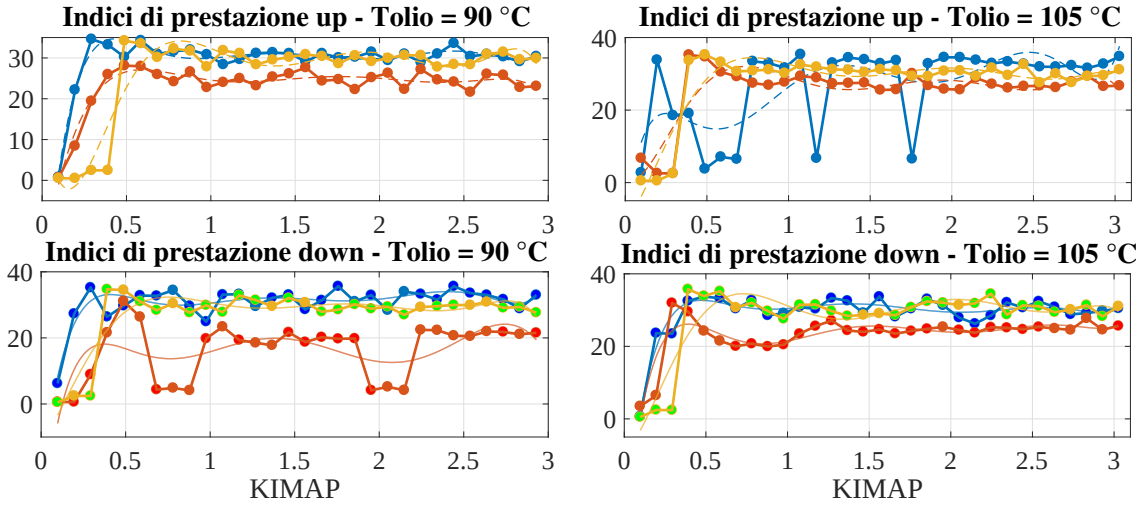


Figura 3.28: Indici di prestazione

Nelle figure precedenti, dalla 3.25 alla 3.28, e nelle tabelle allegate, dalla 3.13 alla 3.15, sono riportati i risultati numerici ottenuti dall'analisi dati; come valeva per i grafici relativi a KDMAP, anche in questo caso il colore blu indica le prove a 1500 rpm, il colore rosso quelle a 3000 rpm e il colore arancione quelle a 4000 rpm. Si noterà che, a differenza di quanto fatto per il guadagno derivativo, in questo caso non si è proceduto con l'interpolazione degli andamenti di overshoot, tempo stazionario ed errore medio a regime, ed osservando i grafici il motivo è subito chiaro: si noterà infatti che gli andamenti rilevati sono pressoché costanti al variare del guadagno integrale. Una leggera variabilità si può riscontrare per quanto riguarda l'overshoot, che mostra un trend crescente al crescere del guadagno impostato, mentre tempo stazionario ed errore medio a regime presentano un andamento evidentemente indipendente da KIMAP. In realtà un'attenta osservazione dei grafici della figura 3.26 ha permesso di trarre alcune interessanti considerazioni:

- per valori di guadagno maggiori di 0.5 il tempo stazionario risulta costante e indipendente dal guadagno integrale applicato;
- a parte rare eccezioni, il tempo stazionario medio di riferimento è inferiore a 1 e si attesta intorno a valori compresi tra 0.5 secondi e 0.7 secondi;
- per valori di guadagno compresi tra 0.4 e 0.6 il tempo stazionario mostra in tutte le prove esaminate un andamento fortemente crescente, ciò che lascia supporre che andrebbe infittito il range di guadagni esaminati tra 0.1 e 1, dato che con elevata probabilità il valore finale di KIMAP, come si vedrà anche dallo storico, ricadrà in questo intervallo.

In merito a quest'ultimo punto è necessaria una precisazione: se da un lato comporta un evidente risparmio di tempo, in quanto è possibile trascurare tutti i valori di guadagno

maggiori di 1, dall'altro apre una serie di problematiche relative al software della ECU: controllando su INCA i valori impostabili per KIMAP ci si è accorti che la risoluzione massima per questa *calibration* è pari a 0.09, cosa che impedisce di infittire l'intervallo come era stato ipotizzato. Sarebbe interessante sapere dal costruttore se è possibile aggirare questo limite; allo stato attuale, nonostante numerose prove, non ci è stato consentito.

Un discorso a parte meritano gli indici di prestazione: come è facile notare, in tutte le prove esaminate, sia in up che in down, gli indici di prestazione presentano un andamento molto simile a quello del tempo stazionario, ovvero crescente per i primi valori della spazzolata di KIMAP e poi costante. Un tale andamento lascia pochi dubbi: i valori ottimi di guadagno da calibrare sono sicuramente quelli minori, tuttavia occorre anche ricordare che il sistema di controllo non è in grado di lavorare con la massima efficienza in quanto le mappe di guadagno proporzionale e derivativo non sono ancora state calibrate, come riportato anche in precedenza. I risultati ottenuti quindi, se da un lato possono essere visti come buoni indicatori dell'efficienza e della validità del codice per l'analisi dei dati, dall'altro devono essere considerati con la dovuta cautela; soltanto la prova definitiva con tutte le mappe calibrate potrà essere utilizzata per le valutazioni finali.

Considerazioni analoghe valgono anche per i valori presenti nella tabella riportata di seguito; quanto detto per la mappa KDMAP vale anche per KIMAP e, soprattutto per quanto riguarda lo storico, in misura ancora maggiore, se possibile: i valori da noi trovati sono abbastanza simili a quelli dello storico, e tuttavia sono troppo pochi i punti esaminati per poter presentare un confronto esaustivo, per il quale sarà necessario condurre l'intero processo di calibrazione dall'inizio alla fine con tutte le mappe e le curve opportunamente calibrate.

	1500 rpm	3000 rpm	4000 rpm
90 °C	0.19	0.19	0.29
105 °C	0.49	0.29	0.29

Tabella 3.16: Risultati ottenuti per la calibrazione di KIMAP

Capitolo 4

Tool per la calibrazione del controllo VVT

4.1 La struttura di base

Partendo dall'esperienza acquisita durante lo sviluppo del tool per la calibrazione del controllo LAMBDA si è deciso di progettare in modo accurato il nuovo tool per la calibrazione del controllo VVT; in questo senso, come si vedrà anche nel capitolo 5, si è scelto di abbandonare l'utilizzo di tanti cicli *while* separati in favore di una più efficiente e comoda struttura basata su una *macchina a stati*. Una macchina a stati è costituita da un ciclo *while* (o da un *timed loop*), da una *case structure* contenuta al suo interno e da un *enum control* che comanda la *case structure*; una tale struttura è particolarmente indicata qualora si debbano realizzare dei software in grado di compiere numerose operazioni in sequenza, operazioni destinate a non venire mai eseguite in contemporanea ma solo seguendo un preciso ordine stabilito dal programmatore o, in alcuni casi, dipendente dalle scelte dell'utente. Così descritta sembra proprio che una macchina a stati si adatti perfettamente alle nostre esigenze, dato che la calibrazione del controllo VVT prevede il rispetto di un preciso ordine per quanto riguarda le mappe, e anche le varie procedure descritte per ogni mappa sono costituite da diversi comandi da eseguire secondo la sequenza proposta.

È chiaro che un simile approccio consente di ottenere numerosi vantaggi, sia dal punto di vista prettamente logico che dal punto di vista organico: innanzitutto è evidente che in questo modo l'intera procedura di calibrazione sarà contenuta all'interno di un'unica *case structure* e quindi all'interno di un unico *timed loop*, indicato con 1 nella figura 4.1, destinato alla gestione dell'automazione nel suo complesso. Un altro vantaggio è la possibilità di estendere verso il basso tale struttura nidificando più macchine a stati all'interno di quella principale, deputando così una macchina a stati per ogni mappa da

calibrare in modo che diventi praticamente impossibile commettere errori nell'ordine di calibrazione, dato che l'*enum* che comanda le macchine a stati più interne viene impostato automaticamente dal codice e non richiede interventi esterni da parte dell'utente; ancora, includendo tutte le operazioni necessarie alla calibrazione all'interno dello stesso loop si ha la certezza che la frequenza di esecuzione sia la stessa per tutte le istruzioni, cosa di non poco conto quando si ha a che fare con applicazioni real time come in questo caso. Infine, ma non per questo di secondaria importanza, l'impiego esteso delle macchine a stati ha permesso di ottenere un ambiente di lavoro estremamente pulito, ordinato e di facile gestione, aspetto non scontato soprattutto per quanto riguarda i linguaggi di programmazione grafica.

Si fa notare che il tool dovrà essere eseguito su un PC realtime costantemente connesso con le attrezzature della sala prove, e pertanto sarà necessario implementare un ciclo che sia in grado di mantenere continuamente in esecuzione il software alla frequenza di clock prefissata; un arresto dell'eseguibile è da escludere se non nel momento in cui venga spento il PC realtime. Idealmente l'eseguibile del tool dovrà essere avviato dai tecnici di sala insieme al PC su cui è stato scaricato, e dovrà rimanere in stato di pausa finché non sarà richiamato dai calibratori. Il principio che sta alla base di una logica simile è la possibilità di mantenere separati e sempre in esecuzione i vari tool per la calibrazione delle funzioni motore e di richiamarne uno alla volta quando è richiesto, presumendo ovviamente che le varie funzioni di controllo motore vengano calibrate singolarmente e senza sovrapposizioni.

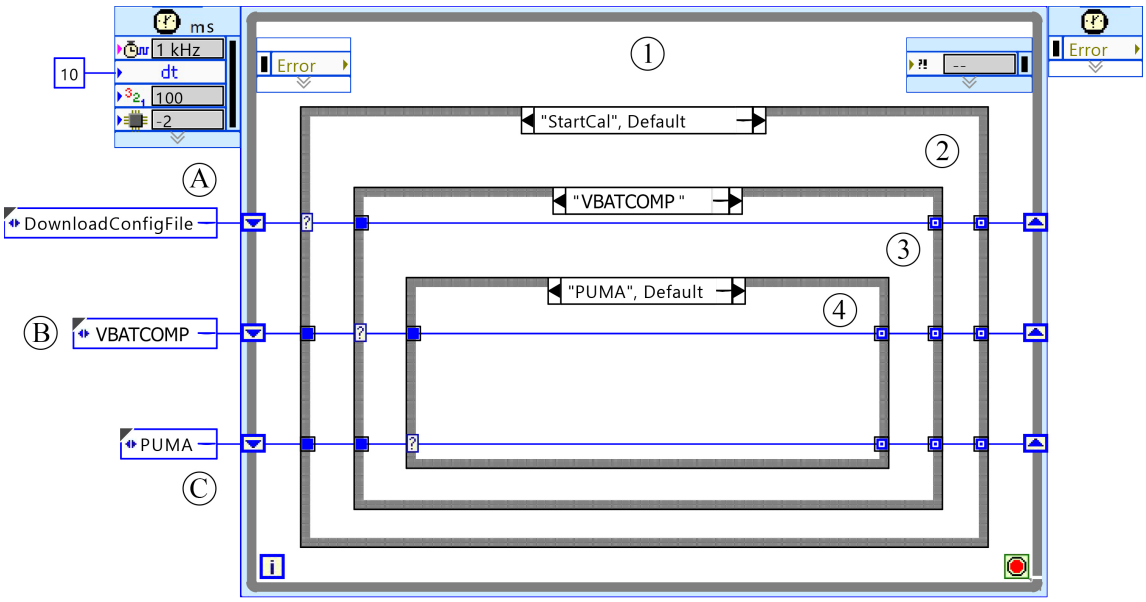


Figura 4.1: Struttura delle macchine a stati presenti nel tool

In figura 4.1 è rappresentato lo schema logico di massima del tool; per ognuno dei

componenti rappresentati in figura si riporta di seguito una breve spiegazione, distinguendo le *case structure*, indicate tramite numeri, dai controlli *enum*, indicati tramite lettere.

- *Timed loop* 1: è il loop principale che mantiene in esecuzione il tool ad una frequenza di 100 Hz; si noti in alto a sinistra la costante 10 che indica il periodo di esecuzione in millisecondi. È opportuno sottolineare che si è scelto un *timed loop* al posto di un più semplice ciclo *while* in quanto il primo viene sempre eseguito alla frequenza impostata mentre il secondo viene eseguito alla massima frequenza consentita dalla CPU in base al carico al quale essa viene sottoposta e al codice presente nel ciclo stesso; un *timed loop* è pertanto in grado di garantire in ogni condizione un determinismo temporale decisamente più accurato;

- Macchina a stati principale 2: è la macchina a stati più esterna deputata alla gestione dei vari processi di automazione; comandata dall'*enum* A, contiene tutto il codice per gestire i vari stati del tool tra cui l'inizializzazione, lo stato di pausa, la calibrazione vera e propria e la generazione del report;

- Macchina a stati secondaria 3: è la macchina a stati intermedia comandata dall'*enum* B e deputata alla gestione della sequenza di calibrazione delle varie mappe; si fa notare che questa *case structure* è presente solo nello stato *StartCal* della macchina a stati 2;

- Macchina a stati interna 4: è la macchina a stati deputata alla gestione dei vari step di calibrazione; comandata dall'*enum* C, questa *case structure* andrà ripetuta dentro ogni caso della macchina a stati 3;

- *Enum* A: controlla la *case structure* 2; è inizializzato allo stato di *Download Configuration File* ed è costituito dai seguenti casi, che vengono eseguiti nell'ordine riportato:

    1. *Download Configuration File*: il tool legge dal disco del PC realtime il file di configurazione contenente tutte le impostazioni e tutti i valori necessari alla calibrazione che verranno proposti all'utente, quindi crea la cartella in cui verranno salvati i dati acquisiti e infine prepara il file di report;
    2. *Backup ECU Default Maps*: prima di avviare il processo di calibrazione il tool ricerca in centralina le curve e le mappe principali e le salva in un apposito cluster nonché nel report di calibrazione. Questo step si rende necessario qualora il calibratore non sia soddisfatto del risultato ottenuto dopo la calibrazione e

voglia ripristinare le mappe precedenti, oppure nel caso in cui non sia possibile esaminare tutti i breakpoints previsti e si voglia sostituire ai breakpoints non calibrati i valori preesistenti in ECU;

3. *Initialization*: il tool verifica che le *measurements* e le *calibrations* necessarie alla calibrazione siano effettivamente state aggiunte dall'utente nell'interfaccia di MCE, in caso contrario il tool non dà il consenso all'avvio della calibrazione;
4. *Pause*: è lo stato di pausa nel quale il software rimane dopo aver eseguito i tre stati iniziali oppure finché l'utente non comanda l'avvio (o la ripresa) del processo di calibrazione;
5. *Start Calibration*: è lo stato previsto per la calibrazione;
6. *Upload Calibration*: in questo stato il tool carica in centralina la mappa (o la curva) appena calibrata; si fa notare che questo stato viene eseguito al termine del processo di calibrazione di ogni curva e mappa;
7. *End of Calibration*: il software reinizializza tutte le variabili al valore iniziale, dopodiché si porta nello stato di pausa in attesa di un'azione da parte dell'utente.

- *Enum* B: controlla la *case structure* 3; è inizializzato allo stato di *None* ed è costituito dai seguenti casi, che vengono eseguiti in base alle scelte dell'utente:

  1. *None*: stato iniziale di check precalibrazione, non può essere impostato dall'utente ma è stato introdotto dal programmatore nel caso si rendano necessarie operazioni di routine prima di avviare la calibrazione;
  2. *VBATCOMP*: stato di calibrazione della curva di correzione in funzione della tensione batteria;
  3. *PRECTL*: stato di calibrazione della curva di precontrollo del PID;
  4. *KPMAP*: stato di calibrazione della mappa di guadagno proporzionale;
  5. *KDMAP_REF*: stato di calibrazione della mappa di guadagno derivativo (prova di riferimento);
  6. *KDMAP*: stato di calibrazione della mappa di guadagno derivativo (prove standard);
  7. *ADVCOMP_RETCOMP*: stato di calibrazione dei parametri di correzione del comportamento asimmetrico del sistema di controllo;
  8. *KIMAP*: stato di calibrazione della mappa di guadagno integrale;

9. *End of Calibration*: stato finale di attesa, non può essere impostato dall'utente ma è stato introdotto dal programmatore per gestire le azioni del calibratore al termine della calibrazione di un parametro e prima del passaggio al parametro successivo;

- *Enum* C: controlla la *case structure* 4; è inizializzato allo stato di *PUMA* ed è costituito dai seguenti casi, che vengono eseguiti nell'ordine riportato:

  1. *PUMA*: in questo stato il tool identifica il breakpoint attuale da calibrare, letto da opportuna curva o mappa come si vedrà nel seguito, e dà il consenso alla calibrazione soltanto se il setpoint motore coincide con il breakpoint letto;
  2. *REC on*: avvio della registrazione dei dati;
  3. *Get VVT angle* (solo per KPMAP, KDMAP, ADVCOMP, RETCOMP e KIMAP): in questo stato il tool si occupa di calcolare i limiti superiore ed inferiore per i jumps da effettuare nelle prove, in funzione del range operativo dei variatori e dell'angolo VVT target imposto dalla mappa in base al punto motore;
  4. *ECU parameters setting*: il tool modifica le *calibrations* di centralina necessarie al processo di calibrazione, in funzione della curva (o della mappa) che si sta calibrando;
  5. *Wait*: stato di attesa durante il quale il tool acquisisce i segnali necessari per l'analisi e la calibrazione, salvandoli sia in una variabile locale che in un file *tdms*; la durata della fase di acquisizione viene calcolata in base al numero di jumps da effettuare (per KPMAP, KDMAP, ADVCOMP, RETCOMP e KIMAP) oppure semplicemente tramite un parametro impostato dall'utente (per VBATCOMP e PRECTL);
  6. *REC off, Save Data File*: il tool svuota completamente il buffer di acquisizione e salva su disco il file di backup *tdms* contenente i dati acquisiti;
  7. *Data analysis*: in questo stato il tool analizza i segnali acquisiti e comanda il passaggio allo stato successivo qualora abbia identificato il valore di calibrazione, altrimenti ritorna a *PUMA* per analizzare il breakpoint successivo;
  8. *Check and Calibration*: qui il tool si occupa di riempire una curva (o una mappa) temporanea con i valori di calibrazione trovati in funzione dei breakpoints scelti dall'utente; se la procedura di trattamento dei dati acquisiti non ha restituito warnings passa allo stato successivo, altrimenti ripete i breakpoints non validi;

9. *Interpolation*: è lo stato finale, eseguito solo al termine del processo di calibrazione, nel quale il tool interpola la curva (o la mappa) determinata secondo i breakpoints utente in modo da riempire la curva (o la mappa) definitiva con i breakpoints di centralina, che dovrà essere caricata in ECU.

È necessario riportare alcune considerazioni prima di procedere con la descrizione del software. Innanzitutto si vuole sottolineare che l'*enum* che comanda la macchina a stati 4 è unico per tutti i parametri da calibrare: da un lato, ciò ha obbligato ad inserire alcuni casi, come *Get VVT angle* o *Interpolation*, che non vengono utilizzati durante la calibrazione di VBATCOMP e PRECTL, ma d'altra parte questa scelta ha permesso un notevole risparmio in termini di controlli, dato che altrimenti sarebbe stato necessario creare un *enum* per ogni mappa da calibrare. Non solo: in questo modo tutte le *case structure* di calibrazione di ogni parametro contengono gli stessi casi e sono ugualmente strutturate, ciò che ha permesso in pratica di realizzare una sola volta la macchina a stati più interna e di copiarla all'occorrenza, rendendosi necessarie solo modifiche secondarie per adattarle di volta in volta ai diversi parametri da calibrare.

Ancora, si fa notare l'uso estensivo dei cosiddetti *shift registers*, rappresentati da coppie di terminali situati ai bordi laterali del *timed loop* 1 in figura 4.1. Gli *shift registers* vengono utilizzati quando è necessario passare i valori dalle iterazioni precedenti alle iterazioni successive in un ciclo: il terminale sul lato destro del loop contiene una freccia verso l'alto e memorizza i dati relativi all'iterazione i, dopodiché LabVIEW trasferisce i dati connessi al lato destro del registro all'iterazione successiva $i+1$ e li rende disponibili tramite il terminale posto sul lato sinistro. È buona norma inizializzare uno *shift register* in modo da definire il tipo di dato che dovrà trasportare. Sempre nella figura 4.1 si vede perciò che tutti e tre gli *enum* che comandano le macchine a stati vengono utilizzati sotto forma di *shift registers* all'interno del *timed loop*, dopo essere stati opportunamente inizializzati tramite le costanti indicate con A, B e C in figura. A proposito di queste costanti è necessario ricordare che in tutti e tre i casi si tratta di *strict type definitions*, indicati dal triangolo nero in alto a sinistra, ovvero di controlli che fanno riferimento a un controllo custom definito in un file apposito di tipo *ctl*; ogni modifica effettuata sul controllo 'padre' ha effetto su tutte le istanze (controlli, indicatori e costanti) ad esso relative. È fortemente raccomandabile che una macchina a stati sia definita tramite un *enum* di tipo *type def* in quanto se si rende necessario intervenire sulla sequenza delle operazioni è sufficiente modificare il controllo di riferimento, con notevole risparmio di tempo. Infatti, come si vedrà nel seguito, il passaggio da uno stato a quello successivo, per ogni macchina a stati, viene gestito automaticamente dal software tramite gli stessi *enum* che vengono utilizzati per inizializzare gli *shift registers*; ecco quindi la necessità di utilizzare dei *type defs*, dato l'elevato numero di costanti presenti.

4.2 Il codice per la gestione dell'automazione

Dopo aver descritto la struttura di base del tool è possibile passare alla definizione delle varie operazioni che il tool esegue dopo essere stato avviato; il diagramma di flusso rappresentato in figura 4.3 è sufficientemente esplicativo, e tuttavia può valere la pena commentarlo brevemente.

Subito dopo l'avvio il tool carica il file di configurazione, controlla che sull'hard disk del PC realtime esistano tutti i percorsi per il salvataggio delle acquisizioni e del report, effettua il backup delle mappe preesistenti in ECU, verifica che tutte le *measurements* e tutte le *calibrations* necessarie alla prova siano state caricate in MCE, dopodiché si porta nello stato di pausa, in attesa di un'azione da parte dell'utente. Tale azione può essere eseguita tramite i due controlli *enum* rappresentati in figura 4.2.

Figura 4.2: Comandi utente per le azioni

L'*enum* 'Stato' serve per la gestione dello stato di esecuzione del tool, e permette di scegliere tra le seguenti azioni:

- Pause: è lo stato di default in cui viene avviato il tool, consente di mettere in pausa il processo di calibrazione attualmente in corso;
- Start: avvia il processo di calibrazione del parametro selezionato dall'utente, oppure dopo aver messo in pausa un processo di calibrazione già in corso permette di riprenderlo ripartendo dal punto in cui ci si era fermati;
- Restart: dopo aver messo in pausa un processo di calibrazione già in corso permette di riprenderlo ripartendo dall'inizio, oppure consente di iniziare la calibrazione di un parametro diverso dal precedente;
- Restore: consente di ripristinare le mappe preesistenti qualora il risultato della calibrazione non fosse soddisfacente;
- Abort: interrompe l'esecuzione del tool; nel normale funzionamento non dovrebbe essere mai selezionato a meno di situazioni di emergenza.

L'*enum* 'Parametro' invece consente all'utente di scegliere quale parametro calibrare tra i seguenti: VBATCOMP, PRECTL, KPMAP, KDMAP, ADVCOMP, RETCOMP, KIMAP.

In definitiva, tramite questi due soli controlli il calibratore può gestire l'intero processo di calibrazione, arrestandolo o riprendendolo a sua discrezione, e scegliendo persino quali parametri calibrare. In merito a questo punto è necessario sottolineare che la disposizione delle voci del menù a tendina che si apre facendo click sull'*enum* 'Parametro' non è assolutamente casuale: è chiaro che la procedura ideale prevede che le mappe vengano calibrate seguendo l'ordine proposto in precedenza, e tuttavia si è voluto lasciare all'utente un certo grado di flessibilità. Sarà pertanto possibile iniziare la calibrazione e completare le prime tre mappe, passare alla calibrazione di un'altra funzione motore e poi riprendere la calibrazione del controllo VVT partendo dalla quarta mappa, ad esempio, o ancora sarà possibile calibrare una sola mappa per volta, eventualmente ripetendo l'operazione più volte finché non si ottengono i risultati voluti. È evidente che in questo modo il calibratore ha la massima libertà possibile, e sicuramente si tratta di una logica più efficiente ed evoluta rispetto a quella che era stata prevista inizialmente e che obbligava l'utente a seguire il processo di calibrazione di tutte le mappe e curve dall'inizio alla fine, senza possibilità di scegliere quali parametri calibrare.

Tornando alla figura 4.3, si nota che la prova relativa a un breakpoint viene ripetuta solo se l'algoritmo di analisi dati restituisce un warning, altrimenti si procede con il breakpoint successivo; dopo aver esaminato tutti i breakpoints definiti dall'utente il tool carica la mappa appena calibrata in ECU, dopodiché si riporta nello stato di pausa in attesa di un'azione da parte dell'utente. È interessante sottolineare che, una volta che l'*enum* 'Stato' si trova in 'Start', per passare alla calibrazione di un altro parametro è sufficiente sceglierlo tramite l'*enum* dedicato, e la calibrazione partirà automaticamente senza dover agire nuovamente sul primo *enum*.

Si ricorda inoltre che i tre stati iniziali che precedono lo stato di pausa vengono eseguiti una sola volta all'avvio dell'eseguibile, e poi mai più fino al successivo riavvio del tool. Per esigenze di chiarezza sono stati introdotti nel front panel tre indicatori booleani che informano l'utente se il tool ha eseguito con successo o meno i tre stati iniziali.

In figura 4.4 è rappresentato il codice necessario alla gestione dei vari stati del tool: si noti che si fa uso dei tre indicatori prima definiti per valutare se gli stati di inizializzazione debbano essere eseguiti o meno. Se il *compound* booleano a destra, impostato in modalità AND, riceve almeno un *false* significa che l'inizializzazione non è ancora stata completata e da esso uscirà sempre *false*, pertanto il selettore a due vie in alto lascerà passare l'input inferiore che proviene dallo *shift register* di sinistra; non appena l'inizializzazione è stata terminata il *compound* booleano riceverà tutti valori *true* e pertanto da esso uscirà *true*,

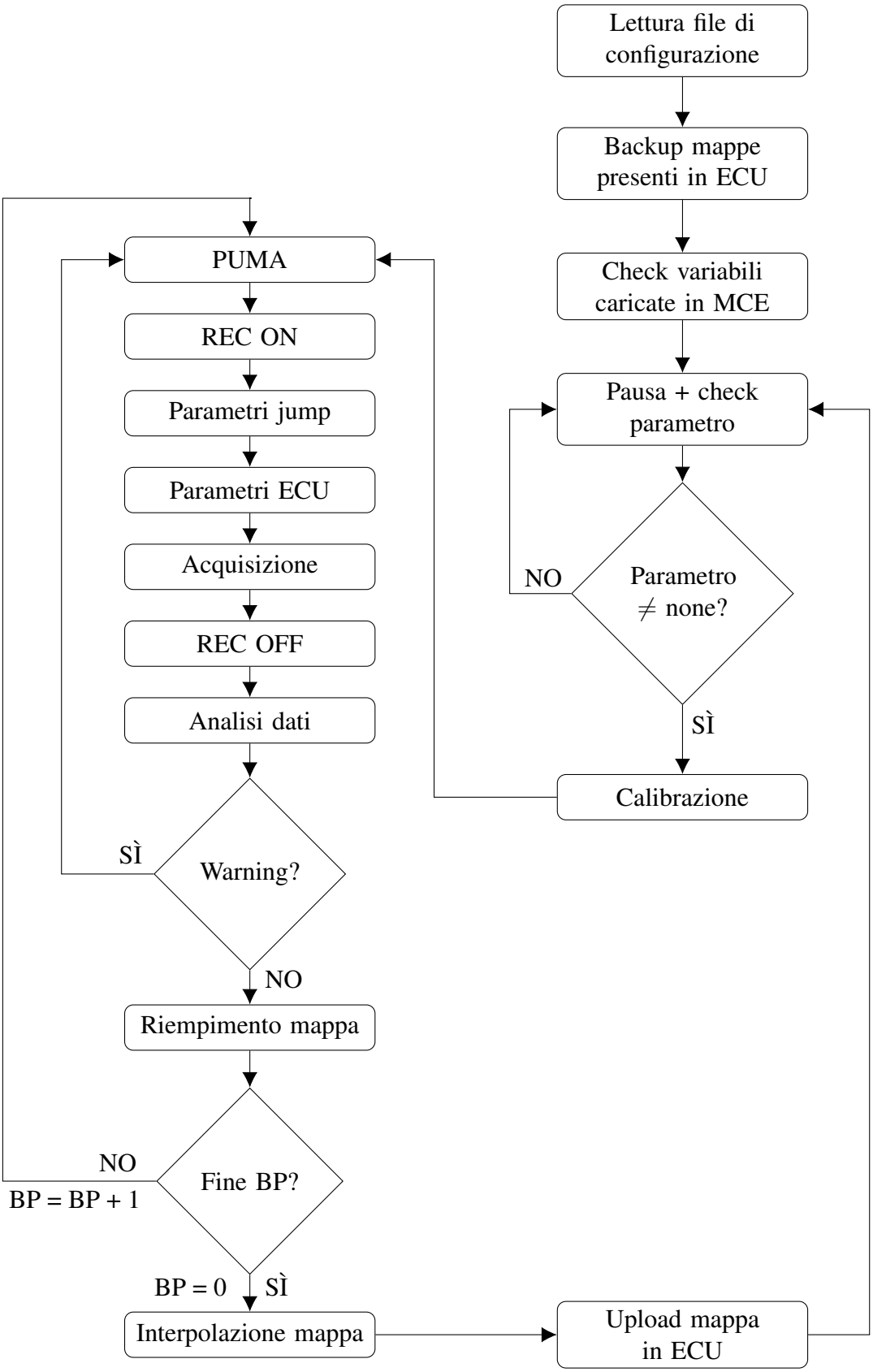


Figura 4.3: Sequenza delle operazioni eseguite dal tool

di conseguenza il selettore a due vie lascerà passare l'input superiore, corrispondente allo stato di pausa. È chiaro che dopo aver completato l'inizializzazione la scelta dell'operazione da eseguire passa all'utente, ed ecco quindi che dentro la *case structure* più esterna, nello stato di pausa, viene effettuato il controllo sull'azione selezionata dall'utente, in basso a destra nella figura.

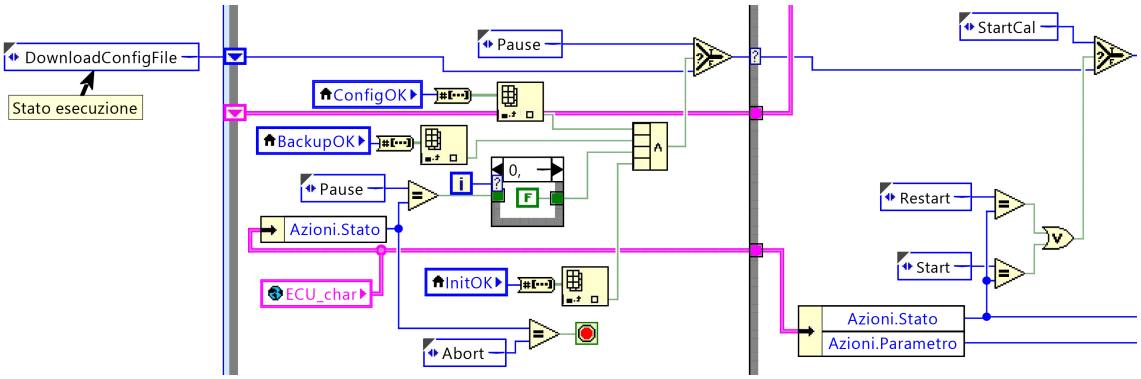


Figura 4.4: Codice per la gestione degli stati del tool

Non appena l'utente sceglie di iniziare la calibrazione il selettore a due vie in alto a destra invierà lo stato 'StartCal' al relativo *shift register* di destra e la macchina a stati principale si porterà in quello stato; la macchina a stati secondaria, responsabile della scelta del parametro da calibrare, viene inizializzata nello stato 'None', all'interno del quale viene controllato il parametro scelto dall'utente tramite un'apposita *case structure*, come si vede in figura 4.5. La costante *type def* presente in ogni pagina della *case structure* invia allo *shift register* che comanda la macchina a stati secondaria il nome del parametro da calibrare scelto dall'utente tramite i comandi rappresentati in figura 4.2, dopodiché all'iterazione successiva la stessa macchina a stati si porta nello stato corrispondente al parametro impostato.

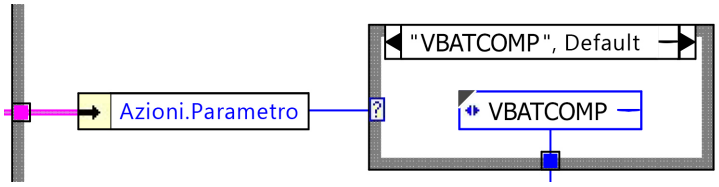


Figura 4.5: Codice per la gestione del parametro da calibrare

Una volta terminata la calibrazione di un parametro il tool si porta nello stato di upload per caricare in ECU la mappa appena ottenuta, dopodiché per procedere è necessario valutare lo stato del processo di calibrazione: se l'ultima mappa calibrata è anche l'ultima della sequenza definita nel capitolo precedente, ovvero KIMAP, allora il tool si porta nello stato 'End of Calibration' della macchina a stati principale, reinizializzando tutte le variabili e preparandosi a un nuovo processo di calibrazione. Altrimenti, come è rappresentato

in figura 4.6, il tool ritorna nello stato ‘StartCal’ e attende la selezione da parte dell’utente di un nuovo parametro da calibrare.

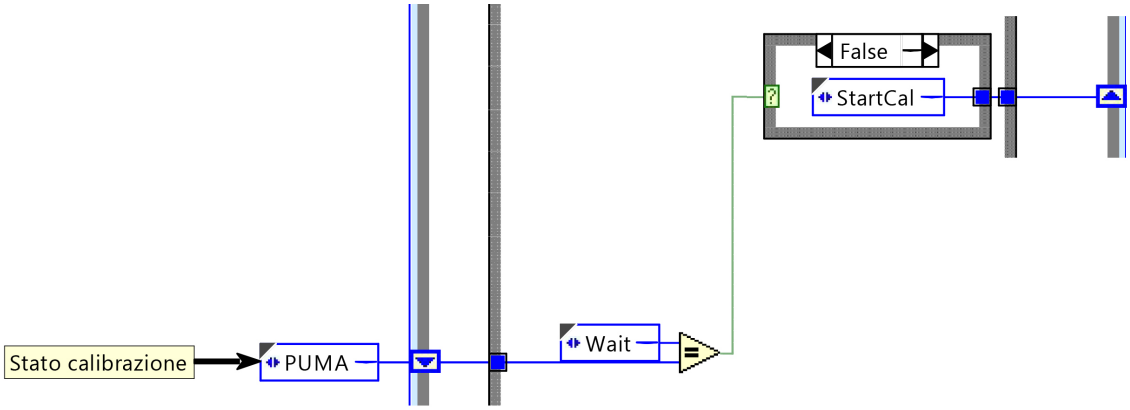


Figura 4.6: Codice per la gestione della fine della calibrazione

Si noti che il controllo sulla mappa viene effettuato tramite lo *shift register* che gestisce gli step di calibrazione, e non tramite lo *shift register* che gestisce i parametri, come sarebbe logico aspettarsi. Questo perché l’utente potrebbe ad esempio scegliere di calibrare singolarmente la mappa di guadagno integrale senza calibrare le altre mappe, nel qual caso il processo di calibrazione non sarebbe terminato dato che le mappe che precedono KIMAP non sono ancora state calibrate, e tuttavia lo *shift register* impiegato per il controllo sarebbe comunque impostato a ‘KIMAP’. Si è pertanto deciso di inserire un check durante la calibrazione di KIMAP che verifichi l’avvenuta calibrazione delle mappe precedenti; in caso affermativo, al termine della calibrazione di KIMAP lo *shift register* che comanda la macchina a stati più interna viene posto a ‘Wait’ e il tool può così riconoscere la fine dell’intera calibrazione, altrimenti viene posto regolarmente a ‘PUMA’ e il tool rimane in attesa di un nuovo parametro da calibrare.

4.3 Il codice per la gestione del processo di calibrazione

Si è visto in precedenza che l’utente può scegliere quale azione eseguire e quale parametro calibrare, tramite gli opportuni comandi sul front panel, avendo così modo di intervenire attivamente nel processo di calibrazione dell’intera funzione. Gli step di calibrazione, invece, sono inaccessibili all’utente per qualunque curva o mappa egli abbia scelto: tramite opportuni indicatori il calibratore può verificare a che punto è arrivata la calibrazione, ma non può in alcun modo modificare le sequenze illustrate nel capitolo 3 dato che queste, una volta definite in fase di progettazione del tool, sono state approvate e congelate e non vi è motivo di apportarvi modifiche di sorta.

Nel seguito si descriveranno le parti di block diagram e di front panel necessarie a gestire il passaggio da uno step a quello successivo, ovvero si descriverà nel dettaglio la macchina a stati prima indicata con il numero 3; per comodità si considererà soltanto una mappa, quella di guadagno integrale, dato che la sequenza di calibrazione si ripete uguale per tutti i parametri considerati.

Nonostante si sia volutamente impedito l'accesso e la modifica da parte dell'utente della procedura di calibrazione delle singole mappe è comunque possibile per il calibratore definire quali breakpoints esaminare durante la calibrazione; in figura 4.7 sono rappresentati i comandi del front panel per il settaggio dei breakpoints, da impostare ovviamente prima di avviare il processo di calibrazione.

È subito evidente che le mappe KPMAP, KDMAP e KIMAP presentano gli stessi controlli: all'utente viene chiesto di inserire i breakpoints di mappa per regime e temperatura dell'olio tramite i due vettori 'RPM_mappa' e 'TOil_mappa', poi i breakpoints che intende esaminare tramite i due vettori 'RPM_user' e 'TOil_user', e infine tramite la matrice 'EnableCal' può scegliere singolarmente quali breakpoints calibrare e quali invece saltare. L'aspetto interessante è che tutti questi controlli vengono letti e aggiornati ad ogni iterazione del *timed loop* principale, in modo che se l'utente decide di inserire o rimuovere un breakpoint durante la calibrazione stessa lo può tranquillamente fare, e il tool si adatta automaticamente al nuovo piano prove.

Le due curve VBATCOMP e PRECTL necessitano ovviamente di impostazioni differenti. VBATCOMP, essendo funzione della sola tensione batteria, richiede l'inserimento dei breakpoints di mappa tramite il vettore 'Vbat_mappa', mentre tramite il vettore 'Vbat_user' il calibratore può scegliere i breakpoints di tensione da esaminare durante la prova; anche qui, il vettore 'EnableCal' consente di scegliere singolarmente quali breakpoints calibrare; si ricorda che la procedura di calibrazione di VBATCOMP prevede di esaminare due diversi regimi a parità di tensione batteria, ed ecco quindi che tramite le quattro matrici, una per ogni breakpoint di tensione, 'BP_Vbat' l'utente può specificare quali regimi impostare. PRECTL, invece, è funzione della sola temperatura dell'olio, e quindi richiederà l'inserimento dei breakpoints di mappa tramite il vettore 'TOil_mappa', nonché dei breakpoints da esaminare durante la prova tramite il vettore 'TOil_user'; discorso analogo vale per il vettore 'EnableCal'. Anche in questo caso si ricorda che il set di prove per la calibrazione della curva di precontrollo prevede di esaminare, a parità di temperatura olio, almeno due angoli VVT target e almeno quattro regimi di rotazione, per i quali sono state previste le cinque matrici 'BP_TOil' (in figura sono rappresentate solo le prime due), tramite le quali l'utente ha la possibilità di specificare i regimi e gli angoli VVT che intende includere nella prova, e questo vale per ogni breakpoint di temperatura olio impostato.

Figura 4.7: Controlli per il settaggio dei breakpoints

Si vede bene che all'utente è lasciato un ampio margine di personalizzazione per quanto riguarda i punti motore da esaminare anche se il tool propone automaticamente, all'avvio, i valori di default consigliati per una corretta calibrazione, essendo l'utente poi libero di modificarli a suo piacimento.

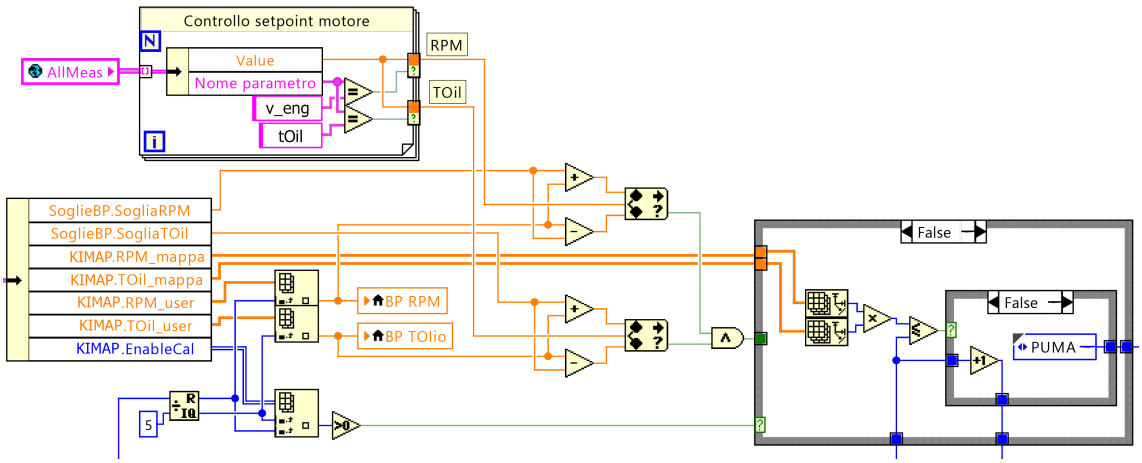
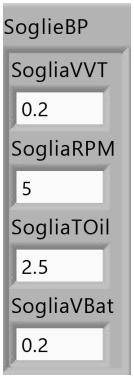


Figura 4.8: Codice per la gestione della calibrazione (PUMA)



Come anticipato in precedenza, la macchina a stati 3 presenta gli stessi casi in ogni parametro da calibrare, e viene inizializzata nello stato 'PUMA', rappresentato in figura 4.8. Questo stato si occupa, nella versione attuale del tool, di controllare semplicemente che il punto motore attuale corrisponda con quello impostato in precedenza dal calibratore; in futuro bisognerà implementare l'integrazione con PUMA in modo che il tool stesso si occupi di portare autonomamente il motore nel punto impostato dall'utente. Per ora il software legge, in funzione di un contatore indicato con '# BP' in figura, i breakpoints inseriti dall'utente tramite i comandi rappresentati in figura 4.8, e controlla tramite la matrice 'EnableCal' se ogni breakpoint deve essere calibrato o meno. Quindi, tramite il ciclo *for* in alto, acquisisce i valori correnti di regime e temperatura olio dalla ECU e verifica che corrispondano con quelli relativi al breakpoint attuale; si fa notare che è concessa una certa tolleranza sui valori di regime e temperatura olio, e tale tolleranza è impostabile dall'utente tramite gli appositi controlli presenti nel front panel e riportati a destra. Se questo è vero, e il contatore è inferiore al numero di elementi della mappa, il tool procede con la calibrazione; altrimenti, se il breakpoint corrente non deve essere calibrato, oppure il setpoint motore non corrisponde al breakpoint impostato il tool rimane nello stato 'PUMA' rispettivamente incrementando di un'unità oppure lasciando invariato il contatore dei breakpoints.

La figura 4.9 è relativa allo stato 'REC on': si tratta di uno stato estremamente semplice il cui unico compito è quello di attivare il loop di registrazione che verrà descritto nella

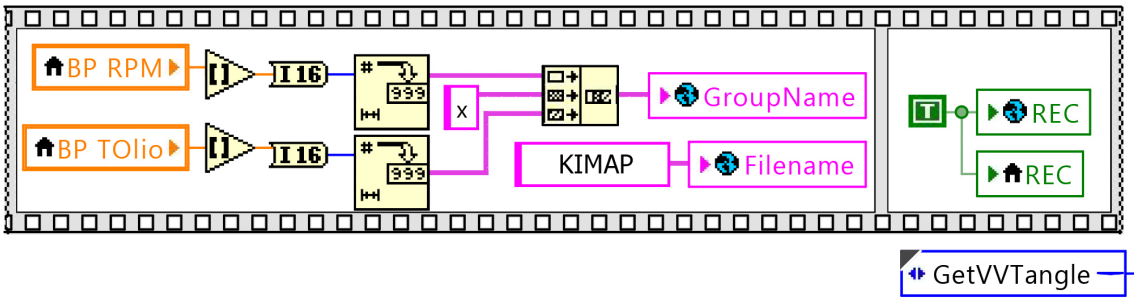


Figura 4.9: Codice per la gestione della calibrazione (REC ON)

prossima sezione. Tale compito viene preceduto dall'impostazione di alcuni parametri relativi al file di salvataggio delle acquisizioni, la cui gestione verrà illustrata anch'essa nel seguito. Si fa infine notare, in basso a destra, l'*enum* necessario a definire lo stato successivo della *state machine*.

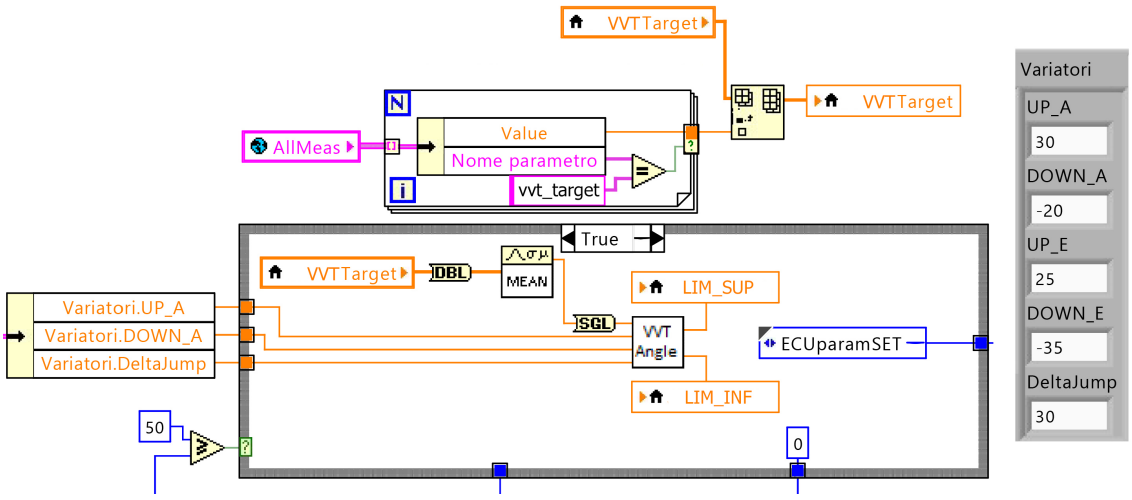


Figura 4.10: Codice per la gestione della calibrazione (Get VVT angle)

In figura 4.10 è rappresentato lo stato 'Get VVT angle' il cui compito è determinare, in funzione dell'angolo VVT target imposto dalla mappa apposita, i limiti superiore e inferiore per i jumps che verranno effettuati nel seguito durante la prova. Il codice legge prima di tutto il range operativo dei variatori impostato dall'utente tramite gli appositi comandi presenti nel front panel e mostrati nella figura a destra, nonché l'entità dei jumps da effettuare tramite il comando apposito presente nello stesso cluster; dopodiché acquisisce la variabile *vvt_target* dalla ECU, che rappresenta proprio l'angolo VVT target sul punto motore attuale, e passa i quattro valori al SubVI 'VVT angle', il quale si occuperà di restituire in uscita i due limiti, che verranno a loro volta salvati nelle variabili locali *LIM_SUP* e *LIM_INF* e mostrati all'utente tramite due indicatori. Si fa notare che l'angolo target *vvt_target* viene acquisito per 50 iterazioni del *timed loop* principale e poi ne viene calco-

lata la media tramite l'apposito blocco prima di passarlo a 'VVT angle', in modo tale da assicurare che il valore utilizzato per il calcolo dei limiti dei jumps sia sufficientemente stabile e appropriato.

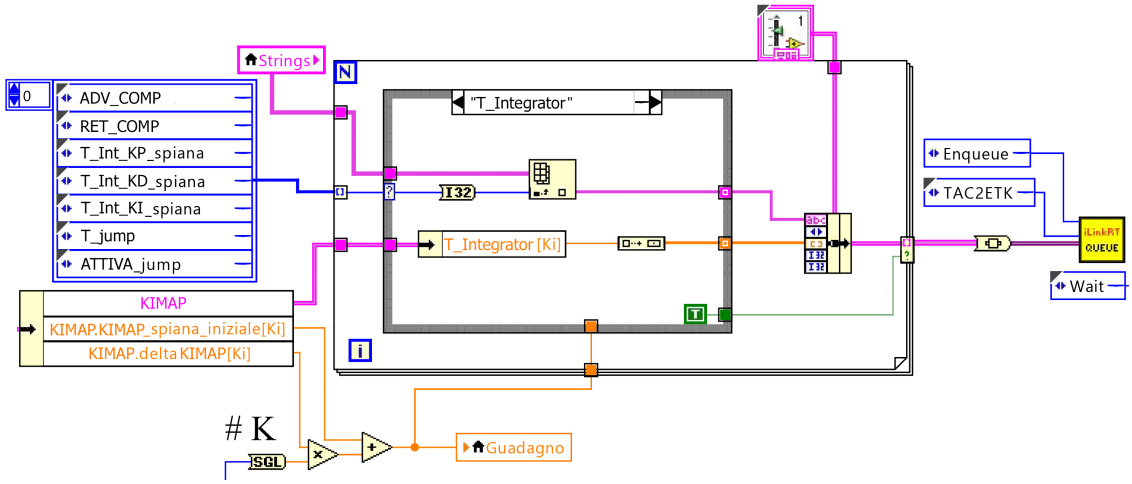


Figura 4.11: Codice per la gestione della calibrazione (ECU parameters setting)

Lo stato seguente 'ECU parameters setting' è rappresentato in figura 4.11. Una descrizione accurata di tale stato richiederebbe di aver prima illustrato il funzionamento del blocco 'iLinkRT Queue' che per esigenze di trattazione verrà trattato più avanti; per ora basti sapere che tale blocco, rappresentato in colore giallo nella figura, consente di modificare il valore di tutte le calibrazioni presenti in centralina se impostato nello stato *TAC2ETK* tramite l'apposito *enum* situato alla sua sinistra. È chiaro che al fine della calibrazione del controllo VVT interessa modificare solo alcuni valori, e sono quelli riportati nell'array di stringhe visibile in figura, peraltro già descritti nel capitolo 3; si fa notare che la scrittura in ECU delle varie *calibrations* viene effettuata tramite un blocco singolo al quale viene passato in input un array di cluster, all'interno del quale sono presenti, tra gli altri, il nome e il valore di tutti i parametri da modificare. La creazione dell'array avviene tramite un ciclo *for*, mentre i valori da settare vengono letti dagli appositi controlli presenti nel front panel e già riportati in figura 4.7. Questo è anche lo stato nel quale viene spianata la mappa di guadagno al valore corrente previsto dal test: in questo senso è stato definito un ulteriore contatore, indicato con '# K' in figura e secondario rispetto a quello già definito per i breakpoints, il cui compito è quello di calcolare il valore di guadagno attuale a cui spianare le varie mappe. Nella pratica il guadagno viene calcolato moltiplicando tale contatore con l'incremento che di volta in volta subirà il guadagno, e il valore così ottenuto viene sommato al guadagno di partenza; sia l'incremento che il guadagno iniziale vengono letti dagli appositi controlli presenti nel front panel, impostabili dall'utente a proprio piacimento. Dato che il contatore viene inizializzato a 0, la prima prova verrà effettuata con il valore iniziale di guadagno, mentre nelle prove successive,

mantenendo costante il contatore dei breakpoint, il contatore di guadagno verrà progressivamente incrementato di un'unità per ogni prova in modo da aumentare il guadagno di mappa secondo l'incremento previsto dall'utente.

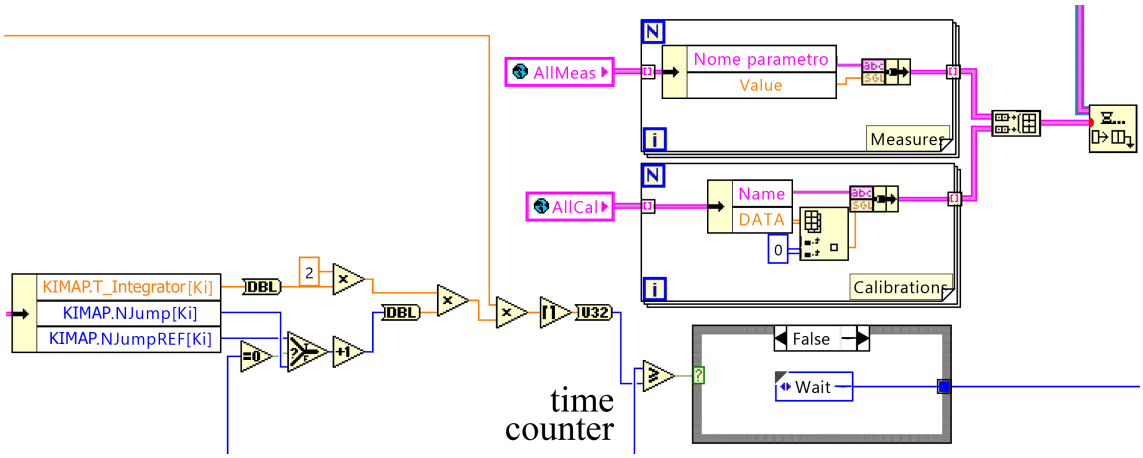


Figura 4.12: Codice per la gestione della calibrazione (Recording)

La figura 4.12 mostra lo stato di registrazione nel quale vengono acquisite tutte le *measurements* (dalla variabile globale *AllMeas*) e tutte le *calibrations* (dalla variabile globale *AllCal*) tramite i due cicli *for* visibili in figura; a loro volta i dati acquisiti vengono inseriti nella *queue* di salvataggio tramite l'apposito blocco 'Enqueue Element', in alto a destra, il cui funzionamento verrà dettagliatamente descritto più avanti nella sezione relativa al salvataggio dei dati. La durata della fase di registrazione, come detto, può essere impostata direttamente dall'utente oppure può dipendere dalla durata e dal numero dei jumps, anch'essi comunque impostati manualmente dal calibratore; nel secondo caso deve essere calcolata moltiplicando semplicemente il numero dei jumps per la relativa durata. Si noti che in figura è stato aggiunto un jump al numero settato dall'utente, in modo che se per qualunque motivo la registrazione dovesse partire in ritardo sarebbe comunque assicurato il numero minimo di jumps richiesti dall'utente. La macchina a stati, in questo senso, richiede la definizione di un nuovo contatore, denominato 'time counter' in figura e inizializzato a 0, il quale viene utilizzato per gestire il tempo di registrazione: dato che il *timed loop* principale viene eseguito a una frequenza di 100 Hz lo stato di recording verrà aggiornato ogni 10 millisecondi, pertanto sarà sufficiente convertire la durata totale della fase di registrazione in millisecondi e incrementare il contatore finché non raggiunge il valore finale. Tale logica consente un controllo estremamente preciso del tempo di registrazione dato che il controllo sulla durata viene effettuato a una frequenza di 100 Hz, e non solo: dato che il numero e la durata dei jumps da effettuare vengono letti alla stessa frequenza, se il calibratore decide di modificare il piano prove in tempo reale può cam-

biare entrambi i parametri e il tool si adatterà automaticamente alle nuove impostazioni.

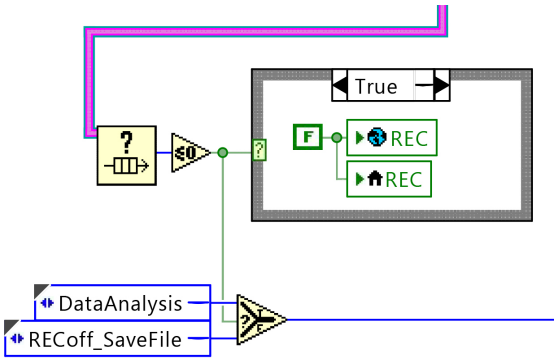


Figura 4.13: Codice per la gestione della calibrazione (Save data file)

La figura 4.13 mostra lo stato di salvataggio dei dati acquisiti: a prima vista può sembrare estremamente semplice dato che contiene solo un blocco che interroga la *queue* di registrazione per determinare il numero di elementi al suo interno, e tuttavia la funzione che tale blocco svolge è molto più delicata e decisamente importante, come verrà illustrato nella sezione seguente. È rilevante sottolineare che sia il passaggio allo stato successivo sia l'arresto del loop di registrazione dipendono proprio dall'output del blocco 'Queue Status' che comanda la *case structure* e il selettore a due vie in basso; l'inserimento di tale blocco si è reso necessario dal momento in cui ci si è accorti che la parte finale delle acquisizioni veniva tagliata dato che la macchina a stati non aspettava lo svuotamento della coda per passare allo stato seguente.

Si tratta, in questo caso, dello stato di analisi dati il cui codice è rappresentato in figura 4.14. È lo stato più importante di tutto il processo di calibrazione dato che da esso dipende il valore di calibrazione da riportare in centralina; anche per questo motivo si è preferito impiegare dei SubVI per il calcolo dei vari parametri come overshoot, tempo stazionario, errore medio a regime ed altri, e per il calcolo degli indici di prestazione, come si può vedere in figura; in tal modo è stato possibile creare i SubVI in modo che eseguissero le stesse operazioni dei codici MATLAB illustrati nel capitolo precedente e soprattutto testarli separatamente rispetto al VI principale che viene qui descritto. È chiaro che dovendo trattare i segnali acquisiti in modo diverso a seconda della mappa o della curva in calibrazione è stato necessario implementare un VI per l'analisi dati per ognuno dei parametri da calibrare, aggiungendo in più altrettanti SubVI per il calcolo dell'indice di prestazione. Inizialmente tali SubVI erano raggruppati in un VI unico per i tre parametri che richiedono il calcolo di un indice di prestazione, cioè KDMAP, ADVCOMP/RETCOMP e KIMAP, ma poi si è pensato di separare i tre casi per maggiore praticità dato che, trattandosi di collegare i vari input allo stesso blocco, si sarebbe creata troppa confusione a

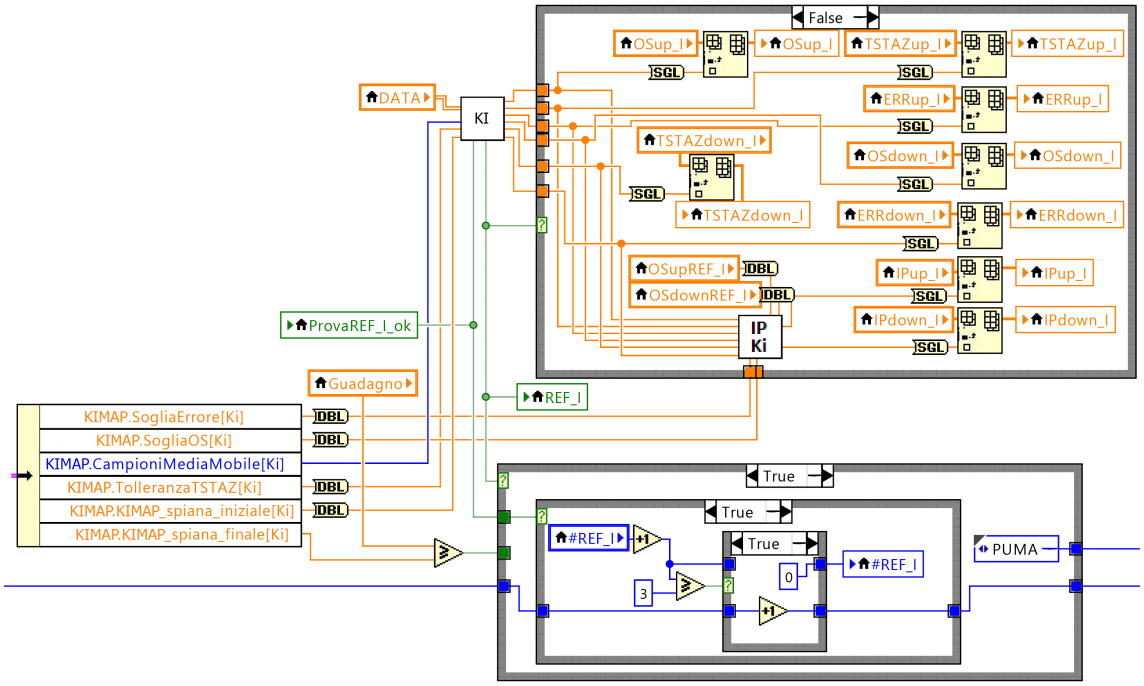


Figura 4.14: Codice per la gestione della calibrazione (Data analysis)

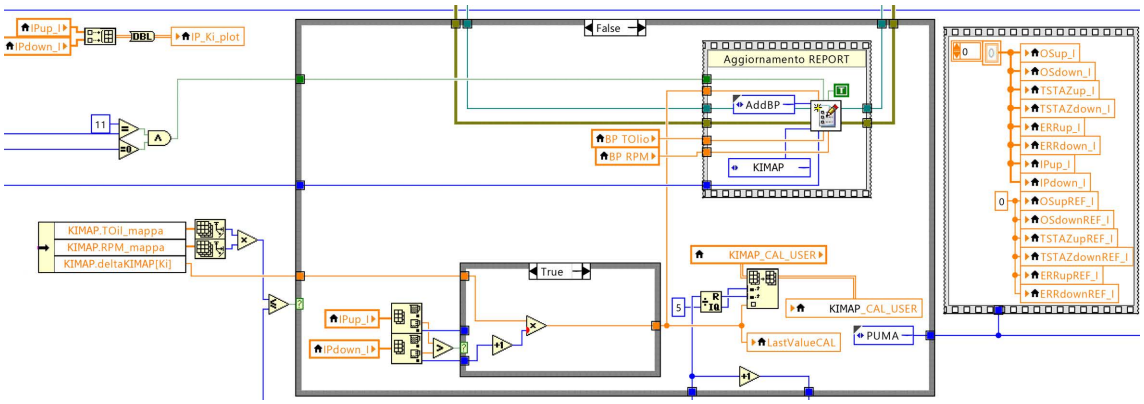


Figura 4.15: Codice per la gestione della calibrazione (Check results)

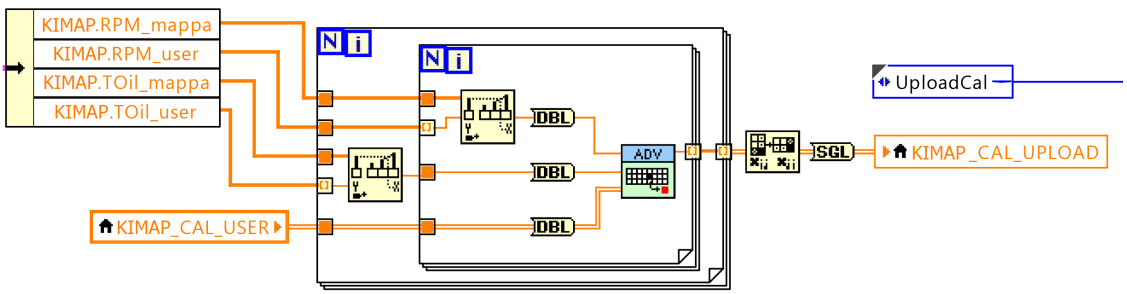


Figura 4.16: Codice per la gestione della calibrazione (Interpolation)

livello di block diagram. Si tralascia la descrizione dei vari SubVI impiegati dato che tutti gli algoritmi di analisi dati sono stati ampiamente descritti in precedenza, e tuttavia vale la pena notare che ogni risultato, anche intermedio, del processo di calibrazione viene salvato in variabili locali e mostrato all'utente tramite i relativi indicatori sul front panel; in ogni caso sia tali risultati sia i settaggi di calibrazione impostati dall'utente tramite il front panel variano a seconda della mappa o della curva che si sta considerando, e mentre degli ultimi si è già parlato in precedenza (vedere figura 4.7), per quanto riguarda i primi si rimanda alle sezioni seguenti dove verranno brevemente illustrati tutti gli indicatori che l'utente ha a disposizione per controllare il processo di calibrazione. Vale infine la pena descrivere brevemente la logica che gestisce il passaggio allo stato successivo implementata tramite le *case structures* nidificate rappresentate in basso a destra: trattandosi della mappa KIMAP, la prima prova da effettuare è quella di riferimento, e tale prova viene ripetuta per tre volte oppure finché non si ottiene una percentuale di accettabilità dei risultati uguale o superiore al 66%, dopodiché si passa alle prove standard. Queste vengono portate avanti incrementando il contatore secondario finché non si raggiunge il valore finale di guadagno impostato dall'utente, dopodiché la macchina a stati riceverà il comando di portarsi nello stato successivo previo azzeramento dello stesso contatore secondario, relativo all'incremento di guadagno.

Lo stato seguente, rappresentato in figura 4.15, ha come compito il riempimento di una mappa (o di una curva) temporanea, funzione dei breakpoints utente, che verrà poi utilizzata per estrapolare i valori da calibrare nella mappa in ECU, funzione di breakpoints diversi. La *case structure* al centro della figura serve a gestire il passaggio allo stato successivo, deputato appunto all'interpolazione dei risultati: quando il breakpoint attuale è l'ultimo di quelli previsti dall'utente la macchina a stati si porta nello stato successivo 'Interpolation', altrimenti ritorna allo stato 'PUMA' incrementando di uno il contatore principale, relativo ai breakpoints. La *case structure* più interna viene utilizzata per determinare il valore di guadagno da calibrare: si notino i due blocchi che determinano il valore minimo di entrambi gli indici di prestazione, salvati nei due vettori *IPup_I* e *IPdown_I*, valori che vengono poi confrontati per determinare quale dei due è inferiore e per valutare se il valore di calibrazione sia riferito ai jumps up oppure ai jumps down. Una volta determinato il guadagno da calibrare per quel punto motore, questo viene inserito nella mappa temporanea *KIMAP_CAL_USER*, dopodiché viene aggiornato il report di calibrazione, in alto a destra. Prima di passare allo stato seguente vengono azzerati tutti gli indicatori relativi alla prova appena effettuata, in modo da rendere il software pronto per la calibrazione del breakpoint successivo o del parametro successivo, se la mappa è già stata completata.

Lo stato finale è quello di interpolazione, o costruzione della mappa da calibrare in

ECU, ed è rappresentato in figura 4.16. Si può vedere che il codice ad esso relativo è abbastanza semplice, trattandosi semplicemente di interpolare una mappa, i cui valori sono stati ottenuti in corrispondenza dei breakpoints utente, per ottenerne un'altra, i cui valori sono funzione dei breakpoints di centralina; tuttavia è stato necessario utilizzare un blocco personalizzato dato che LabVIEW non disponeva di un blocco per l'interpolazione in due dimensioni. La mappa così ottenuta viene salvata nella variabile *KIMAP_CAL_UPLOAD*, pronta per essere scritta in ECU durante la fase seguente di upload.

4.4 Il file di configurazione

Il file di configurazione è un file con estensione *ini* che viene utilizzato per inizializzare ai valori di default tutti i controlli presenti sul front panel, in modo da guidare l'utente all'utilizzo di valori di partenza con i quali tentare la calibrazione. Tale file contiene anche tutte le informazioni necessarie al collegamento via XCP e via iLinkRT, ovvero il numero delle porte utilizzate e l'indirizzo IP del PC realtime su cui viene eseguito il tool.

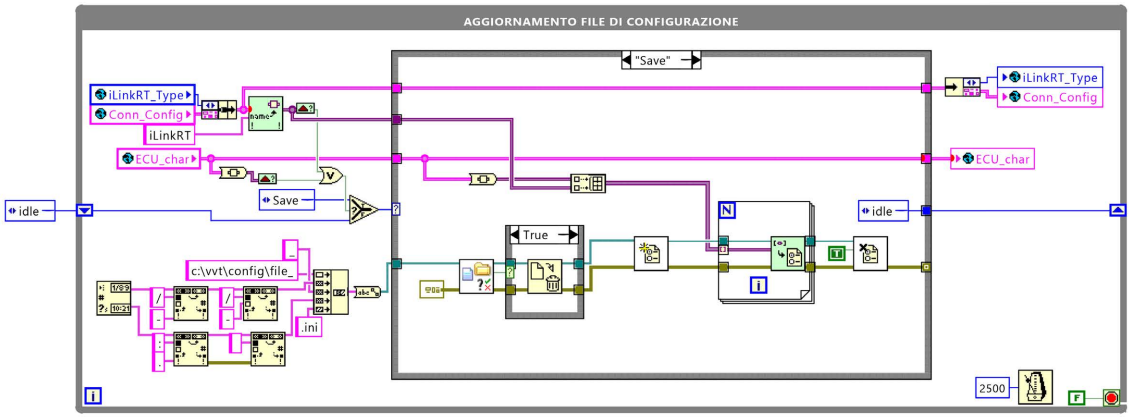


Figura 4.17: Codice per lettura/scrittura del file di configurazione

In figura 4.17 è riportato il loop destinato alla lettura e all'aggiornamento del file di configurazione; la logica impiegata a tale scopo è la seguente:

- in fase di realizzazione del software il programmatore avrà cura di compilare un file di configurazione 'base' che contiene tutti i parametri relativi al processo di calibrazione e che dovrà essere copiato sul PC realtime nel percorso *c:\vvt\config*;

- ad ogni avvio del tool, questo leggerà il file di configurazione base e inizializzerà i vari controlli con i valori in esso contenuti;

- ogni volta che l'utente modifica un controllo sul front panel il loop di aggiornamento del file di configurazione scriverà sull'hard disk del PC realtime, nello stesso percorso del file base, un nuovo file di configurazione identificato con la data e l'ora relative alla modifica effettuata dall'utente;
- se l'utente ritiene che le modifiche da lui effettuate comportino risultati migliori di quelli ottenuti sfruttando il file di configurazione base può rinominare il nuovo file di configurazione e sostituirlo a quello base.

Si fa notare che il loop di aggiornamento del file di configurazione è anch'esso una macchina a stati, comandata dall'*enum* a sinistra impostato nello stato 'idle' e operante a una frequenza decisamente inferiore rispetto al loop principale che gestisce l'automazione; per non appesantire troppo il processore del PC realtime si è scelto infatti di impostare il *while loop* in modo che esegua un'iterazione ogni 2.5 secondi, sicuramente sufficienti per individuare e salvare un'eventuale modifica operata sui controlli del front panel. In pratica ogni 2.5 secondi il loop controlla se l'utente ha modificato i valori dei parametri di calibrazione presenti nel front panel, tramite il SubVI 'Data Changed' messo a disposizione dalla libreria OpenG; se tale SubVI restituisce *true* allora la *state machine* passa nello stato 'save', altrimenti rimane in 'idle'. Il terzo stato, 'load', non viene mai utilizzato all'interno del loop di aggiornamento del file di configurazione, ma viene impiegato nello stato 'Download Configuration File' della macchina a stati principale per inizializzare i controlli all'avvio del tool.

La struttura del file *ini*, di cui si riporta di seguito un estratto, è abbastanza semplice e facilmente comprensibile in modo che qualora si rendesse necessaria qualche modifica ai valori di default non è necessario eseguire il tool e salvare un nuovo file di configurazione, ma è sufficiente aprire il file base con un generico editor di testo e modificare i valori richiesti.

```
[ECU_char]
INCAfirst = "0"
Azioni.Stato = "Pause"
Azioni.Parametro = "VBATCOMP"
VBATCOMP.Vbat_user = "<size(s)=4> 11.50 12.00 13.00 14.50"
VBATCOMP.Vbat_mappa = "<size(s)=4> 8.00 12.00 13.00 14.50"
VBATCOMP.EnableCal = "<size(s)=4> 1 1 1 1"
VBATCOMP.BP_Vbat\5B1\5D = "<size(s)=2> 1500.00 5000.00"
VBATCOMP.BP_Vbat\5B2\5D  = "<size(s)=2> 1500.00 5000.00"
VBATCOMP.BP_Vbat\5B3\5D = "<size(s)=2> 1500.00 5000.00"
VBATCOMP.BP_Vbat\5B4\5D  = "<size(s)=2> 1500.00 5000.00"
VBATCOMP.NOjump\5BF\5D = "33.00"
VBATCOMP.TensioneNominale\5BF\5D = "13.00"
VBATCOMP.DurataAcquisizione\5BF\5D = "0.00"
```

Tra parentesi quadre, nella prima riga, è indicato il nome del cluster principale contenente tutti i controlli modificabili dall'utente; nelle righe seguenti sono indicati invece i cluster secondari e tutti i controlli in essi con, a fianco, i relativi valori. Si noti che per indicare un determinato controllo appartenente a un cluster si usa la notazione '.', mentre il valore che tale controllo possiede è indicato tramite '=' come se fosse una semplice assegnazione.

4.5 L'acquisizione di *measurements* e *calibrations*

Come visto nel capitolo introduttivo l'acquisizione e la trasmissione delle variabili di centralina vengono gestite dal protocollo iLinkRT: l'utente può scegliere quali misure e quali parametri rendere disponibili al software di automazione aggiungendole tramite il tool MCE, dopodiché sarà necessario implementare un VI in grado di ricevere e riorganizzare i dati trasmessi via iLinkRT. Tale VI, nel nostro caso, si chiama 'iLinkRTMaster' e contiene il codice necessario a gestire i dati inviati da MCE; in figura 4.18 è rappresentato il codice per l'acquisizione delle *measurements* mentre in figura 4.19 è rappresentato il codice per l'acquisizione delle *calibrations*.

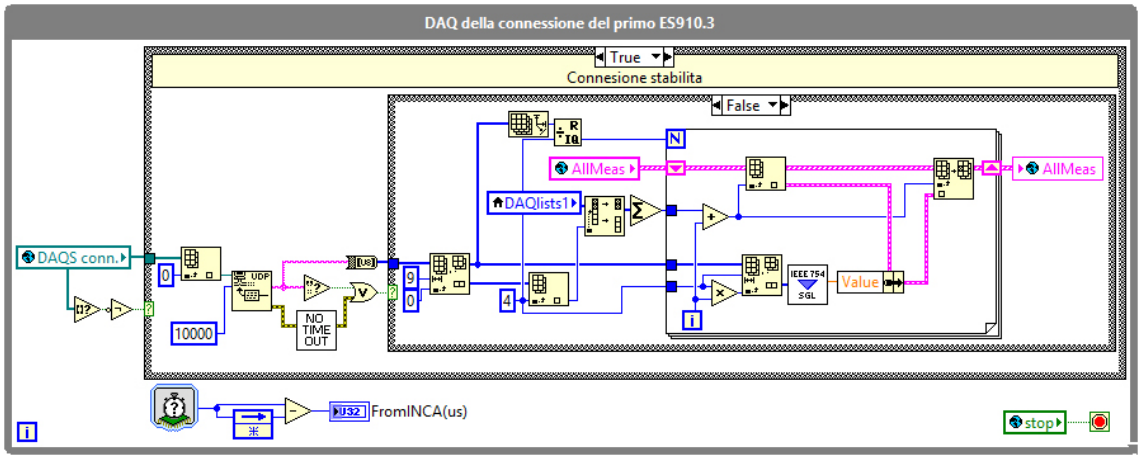


Figura 4.18: Codice per l'acquisizione delle *measurements*

In entrambi i casi si noterà che si tratta di due cicli *while*; il secondo, in particolare, contiene una macchina a stati che ha il compito di inizializzare la connessione via iLinkRT e ottenere tutte le DAQ e tutte le CAL. Tutte le *measurements* verranno utilizzate all'interno del tool grazie al ciclo *while* rappresentato in figura 4.18, che provvede a organizzare ciascuna variabile in un apposito cluster riportandone il nome, il valore, il device da cui è stata inviata e la DAQ list a cui appartiene. È opportuno precisare che all'interno del VI 'iLinkRTMaster' sono presenti due cicli *while* destinati all'organizzazione delle *measurements*, in modo da poter gestire due moduli ES910.3 a cui saranno collegate al-

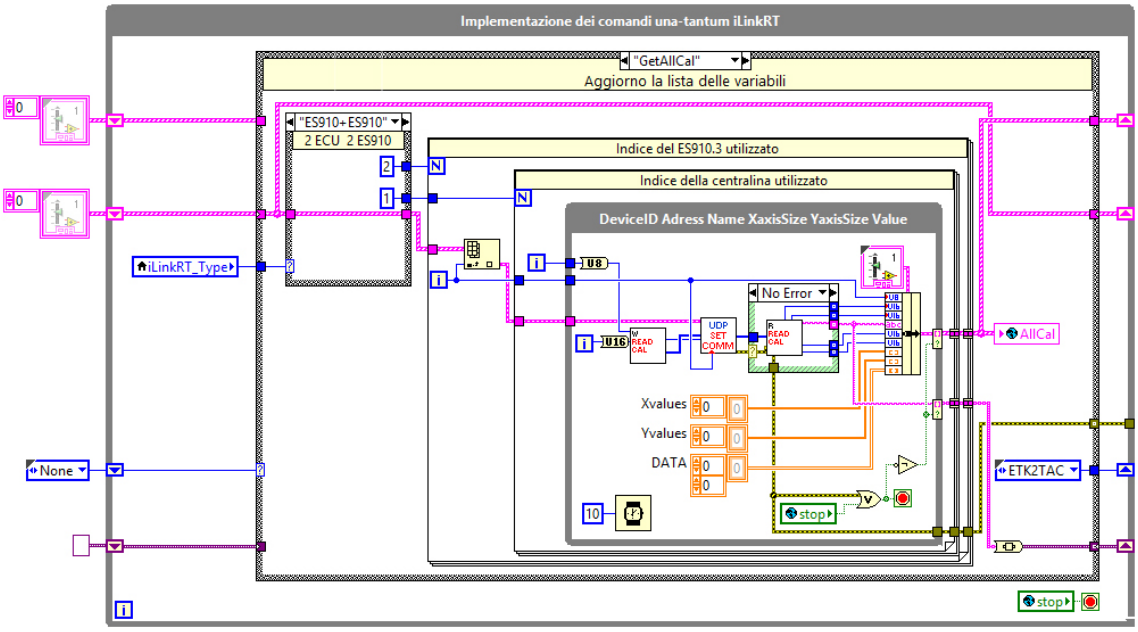


Figura 4.19: Codice per l'acquisizione delle *calibrations*

trettante centraline; si fa notare che è anche possibile utilizzare un solo modulo ES910.3 con due centraline, nel qual caso il VI è in grado di gestire automaticamente i due set di variabili acquisite previa impostazione da parte dell'utente del layout di collegamenti impiegato, il quale può essere scelto tra i seguenti:

- 1 ECU + 1 ES910.3;
- 2 ECU + 1 ES910.3;
- 2 ECU + 2 ES910.3.

Sia le *measurements* che le *calibrations* acquisite vengono inserite in due array di cluster, chiamati rispettivamente 'AllMeas' e 'AllCal'; tali cluster sono costituiti dai controlli rappresentati in figura 4.20 e appartengono alla variabile globale 'iLinkRT_GL' in modo che siano subito disponibili per tutti i VI che ne richiedono l'uso. Riprendendo quanto accennato prima è chiaro che l'eventuale utilizzo di due centraline comporterà la creazione di un nuovo array di cluster 'AllMeas2', i cui elementi si differenziano da quelli appartenenti a 'AllMeas' per il valore dell'indicatore 'ConnID', che sarà 0 per la prima centralina e 1 per la seconda.

Vale la pena descrivere brevemente gli indicatori presenti nei due cluster:

- 'AllCal':
  - ConnectionID: indice del modulo ES910.3;
  - DevID: indice della ECU;

- Adress: indirizzo di memoria in ECU della *calibration*;
- Name: nome della *calibration*;
- XaxesSize: dimensione lungo x della curva o della mappa;
- YaxesSize: dimensione lungo y della mappa;
- Xvalues: valori dei breakpoints sull'asse x;
- Yvalues: valori dei breakpoints sull'asse y;
- DATA: valore della *calibration*.

- 'AllMeas':
  - ConnID: indice della ECU;
  - DAQ_ID: indice della DAQ list a cui appartiene la *measurement*;
  - MeasID: indice della *measurement*;
  - Nome parametro: nome della *measurement*;
  - Value: valore della *measurement*.

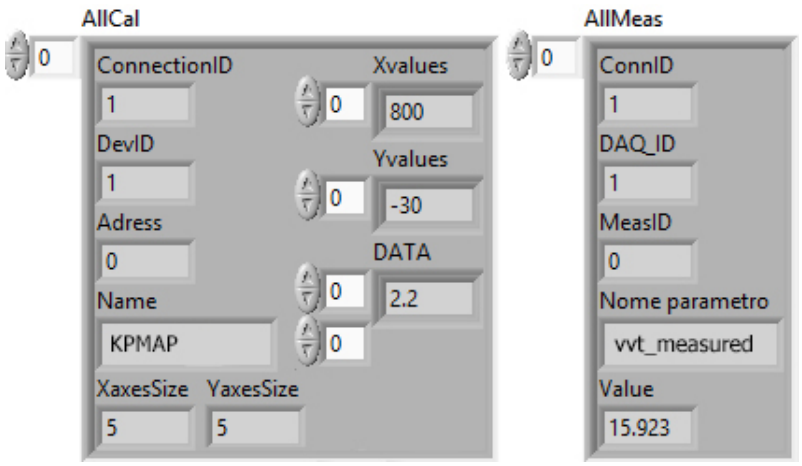


Figura 4.20: Array di cluster 'AllCal' e 'AllMeas'

4.6 La registrazione e il salvataggio dei dati

I dati acquisiti e inviati tramite il protocollo iLinkRT vengono ricevuti da LabVIEW sotto forma di cluster, sia per quanto riguarda le *measurements* sia per quanto riguarda le *calibrations*. Si è visto in figura 4.12 che tutti i dati acquisiti vengono inseriti nella *queue* di salvataggio ma ancora nulla è stato detto su come prelevare gli stessi dati o salvarli

su disco o in variabili locali per consentirne l'analisi e il trattamento. Per prima cosa è necessario definire cosa si intende per *queue*: in LabVIEW una coda viene utilizzata per scambiare dati che vengono generati da una porzione di codice di un VI e vengono utilizzati da un'altra porzione di codice dello stesso VI. La logica con la quale tale coda viene alimentata è di tipo FIFO: nella pratica, se un elemento della coda viene inserito per primo, esso sarà anche il primo a venire prelevato. A differenza di un array standard, pertanto, non è possibile accedere a tutti gli elementi della coda, ma soltanto ad uno per volta; l'unico modo per visualizzare l'intero contenuto della coda è svuotarla completamente. L'aspetto veramente interessante di una coda è però la possibilità di scambiare dati tra cicli *while* o *timed loop* che lavorano a frequenze diverse senza perdere nessun dato e consentendo quindi una perfetta sincronizzazione tra chi alimenta la coda e chi la svuota. Quanto appena illustrato si applica perfettamente al caso in esame; si osservi la figura 4.21: essa rappresenta il loop utilizzato per salvare i dati acquisiti in un file di tipo *tdms*, che rappresenta il formato standard in ambiente LabVIEW per il salvataggio di dati generati da un codice.

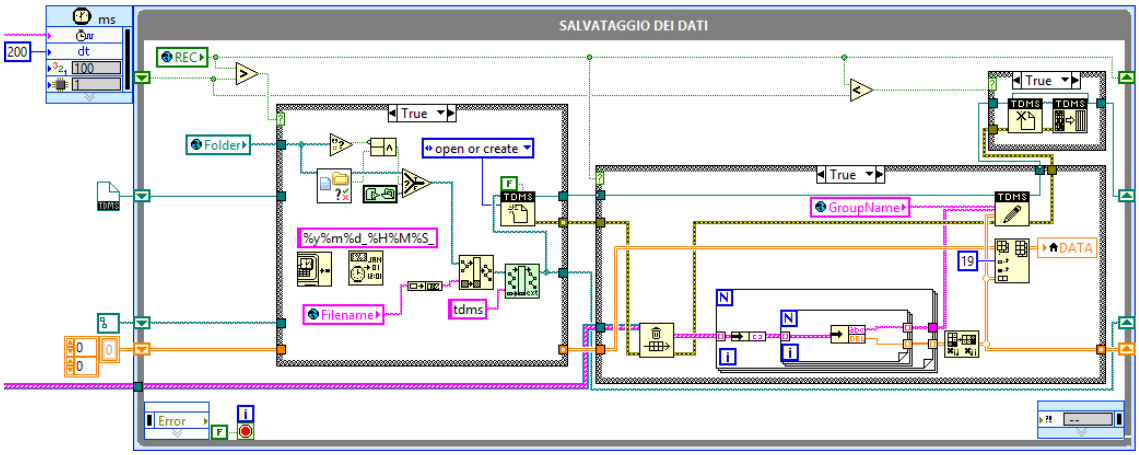


Figura 4.21: Codice per il salvataggio dei dati acquisiti

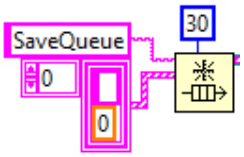
Il ciclo principale è un *timed loop* che viene eseguito ad una frequenza di 5 Hz (si veda la costante 200 in alto a sinistra che indica ogni quanti millisecondi viene eseguita un'iterazione) e contiene al suo interno tre *case structure* comandate, ognuna in modo diverso, dalla variabile globale 'REC', la quale a sua volta viene triggerata dagli stati 'REC on' e 'REC off' appartenenti alla macchina a stati 3 descritta in precedenza:

1. la prima, più a sinistra, si occupa dell'apertura (e della creazione, qualora non sia ancora stato creato) del file *tdms* destinato a ricevere i dati acquisiti, tramite il blocco 'TDMS Open'; si fa notare che il codice contenuto all'interno di questa *case structure* viene eseguito soltanto se la variabile globale passa da *false* a *true*, altri-

menti il contenuto dei tunnel di sinistra viene semplicemente trasferito ai tunnel di destra;

2. la seconda, in basso a destra, si occupa dello svuotamento della coda di registrazione, tramite il blocco 'Flush Queue', e del trasferimento del suo contenuto prima in una variabile locale di tipo matrice, 'DATA', utilizzata per l'analisi dati, e poi nel file vero e proprio tramite il blocco 'TDMS Write'. È interessante notare che tale *case structure* viene comandata direttamente dalla variabile globale 'REC', pertanto il codice al suo interno viene eseguito ogniqualvolta la macchina a stati 3 del loop principale si trova negli stati seguenti a 'REC on'; inoltre la variabile 'DATA' viene alimentata da uno *shift register* di tipo *double*, in modo che quando il *timed loop* svuota la coda e trasferisce i dati acquisiti tale variabile non viene sovrascritta, ma i nuovi valori vengono aggiunti al termine della matrice stessa;

3. la terza, in alto a destra, si occupa della chiusura del file appena scritto, tramite il blocco 'TDMS Close' e della rimozione dei file temporanei di tipo *index*, tramite il blocco 'TDMS Defragment', e tale codice viene eseguito soltanto se la variabile globale che comanda la *case structure* passa da *true* a *false*.



A questo punto è d'obbligo una precisazione: l'utilizzo di un buffer di registrazione si è reso necessario dal momento che il loop principale che gestisce l'automazione e riceve i dati dalla ECU lavora ad una frequenza di 100 Hz, mentre il loop di salvataggio dei dati è stato volutamente impostato ad una frequenza di lavoro pari a 5 Hz per non sovraccaricare il processore del PC realtime e per limitare gli accessi all'hard disk dello stesso PC, rendendo l'intero processo di salvataggio più efficiente e meno oneroso. Tuttavia, essendo le frequenze di lavoro dei due cicli nettamente diverse, è stato necessario rimediare all'asincronia tra i due loop utilizzando un approccio che prevedesse lo scambio di dati tra i due loop per mezzo di una *queue*; tale queue, denominata 'SaveQueue' come si vede nella figura a destra, prima che il software acceda al *timed loop* principale viene inizializzata tramite il blocco 'Obtain Queue' in modo che essa contenga un array di cluster, ognuno dei quali costituito da una stringa, che conterrà il nome della *measurement* o della *calibration*, e da una costante numerica di tipo *double*, che ne conterrà il valore.

Vale la pena soffermarsi sulla dimensione della coda, qui impostata a 30 elementi: dato che il ciclo di salvataggio dei dati svuota la coda cinque volte al secondo, mentre il ciclo principale la riempie cento volte al secondo, si vede bene che ogni iterazione del primo dei due cicli deve gestire una *queue* costituita da 20 elementi, pertanto tale sarà la dimensione minima con la quale dovrà essere inizializzata la coda. Si è scelto tuttavia di

inizializzarla a 30 elementi per mantenere un certo margine di sicurezza, in modo tale che se per una qualunque ragione il ciclo di salvataggio dovesse svuotare la coda in ritardo non ci sarà alcun pericolo di perdere dei dati.

4.7 Il report di calibrazione

La generazione del report di un processo di calibrazione è fondamentale per informare l'utente sull'esito della calibrazione delle varie mappe e curve e anche dei singoli breakpoints, pertanto dopo opportuni test condotti su VI separati si è scelto di generare un report di tipo *xls*, visualizzabile ed eventualmente modificabile con Excel ma anche con un qualsiasi editor di testo. LabVIEW mette a disposizione una libreria per la generazione e la modifica di report di processo, e però tutti i blocchi che ne fanno parte richiedono la dipendenza da controlli *ActiveX* che purtroppo non sono supportati dal PC realtime; pertanto si è deciso di scrivere il contenuto del report come se si trattasse di un semplice file di testo, tramite il blocco 'Write to Text File'.

A tale scopo si è pensato di creare un SubVI polimorfico in grado di gestire completamente tutte le azioni relative al report, di cui in figura 4.22 è riportato il block diagram; tale scelta è stata dettata da diverse ragioni, tra cui la flessibilità di uso, dato che con un SubVI è possibile gestire più operazioni con uno stesso blocco, e la necessità di implementare la generazione di un report anche per il tool di calibrazione del controllo Lambda, per il quale, come si vedrà nel seguito, verrà riutilizzato in toto il SubVI qui descritto previe opportune modifiche.

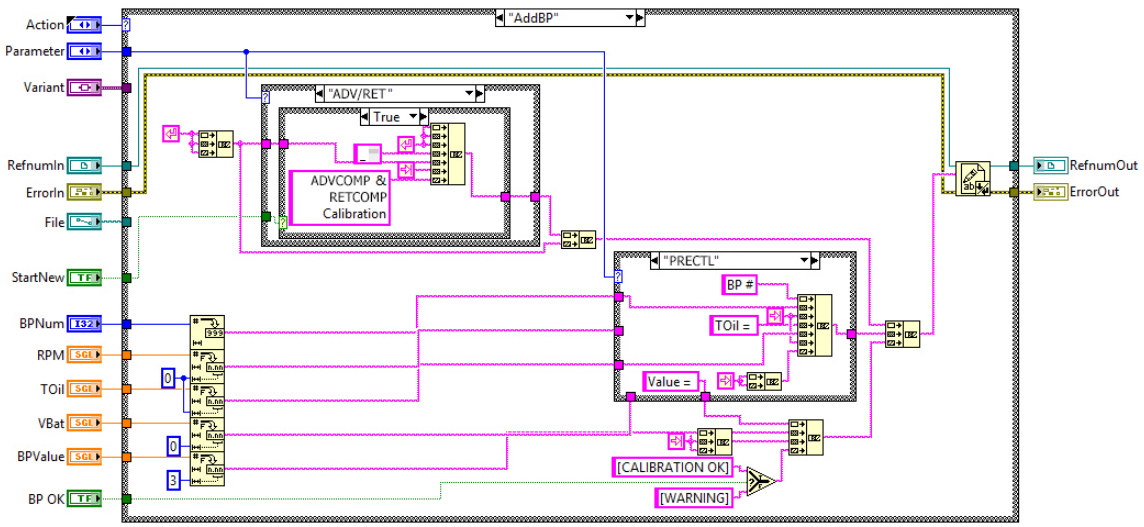


Figura 4.22: SubVI per la gestione del report di calibrazione

Il SubVI 'CreateTextReport' è costituito da una *case structure* principale comandata dall'*enum* 'Action', in alto a destra; gli input del SubVI sono tutti i controlli visibili alla

sinistra della *case structure*, mentre gli output sono tutti gli indicatori posti alla sua destra. I vari input verranno brevemente accennati nel seguito della trattazione, e tuttavia vale la pena identificarne almeno due di fondamentale importanza: 'File', che è una variabile di tipo stringa contenente il percorso di salvataggio e il nome del report generato, e 'RefnumIn', ovvero il puntatore al file stesso; si fa notare, e ciò verrà spiegato più avanti, che 'RefnumIn' viene passato anche come output del SubVI, nel qual caso si chiama ovviamente 'RefnumOut'. Questo SubVI è in grado di gestire da solo tutte le operazioni relative al report di calibrazione, dalla sua creazione, alla sua modifica, alla sua chiusura, e la scelta dell'operazione da compiere viene effettuata impostando l'input 'Action' in uno dei seguenti stati:

- *Create report*: apre il file passato come input ed inserisce il frontespizio, la data e l'ora di creazione; in questo caso l'unico input al SubVI è il percorso di salvataggio del report. Questa azione verrà inserita nel VI principale, precisamente nello stato 'Download Configuration File' della macchina a stati 1;

- *Save base maps*: le mappe di default preesistenti in ECU vengono scritte nel report di calibrazione; gli input al SubVI sono il puntatore al file, il cluster di errore e le mappe base sotto forma di dato *variant*. Questa azione verrà inserita nello stato 'BackupECUDefaultMaps' della macchina a stati 1;

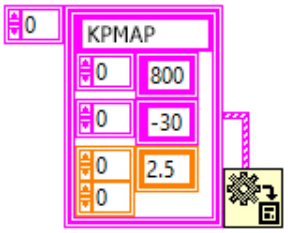
- *Add breakpoint*: questo stato aggiorna il report ogni volta che viene calibrato un breakpoint, aggiungendo al valore di guadagno determinato anche l'eventuale presenza di warning. Gli input al SubVI sono diversi; escludendo i soliti puntatore e cluster di errore, può valere la pena elencare gli altri uno per uno:

  - valore di RPM corrispondente al breakpoint attuale;
  - valore di temperatura dell'olio corrispondente al breakpoint attuale;
  - valore di angolo VVT target corrispondente al breakpoint attuale;
  - valore di tensione batteria corrispondente al breakpoint attuale;
  - valore di guadagno determinato corrispondente al breakpoint attuale;
  - indice del breakpoint appena calibrato: è il contatore principale dei breakpoint presente nella macchina a stati 3;
  - parametro attualmente in calibrazione;
  - booleano che indica se il breakpoint attuale è il primo della curva o della mappa ad essere calibrato;
  - booleano che indica se la calibrazione del breakpoint corrente è andata a buon fine oppure il codice di analisi dati ha restituito un warning.

È chiaro che dei primi quattro input elencati verranno scelti solo quelli relativi al parametro che si sta calibrando; per VBATCOMP, ad esempio, si utilizzerà la tensione batteria e il regime motore, mentre per KIMAP si utilizzerà il regime motore e la temperatura dell'olio. Questa azione andrà inserita in tutti gli stati 'Check results' o 'Interpolation' delle macchine a stati 3 più interne;

- *Compare maps*: confronta, riportandole fianco a fianco, le mappe e le curve preesistenti in centralina prima della calibrazione con le mappe e le curve ottenute dopo il processo di calibrazione; l'unico altro input, oltre ai soliti puntatore e cluster di errore, è costituito da una variabile di tipo *variant* che verrà descritta nel seguito. Questa azione verrà inserita nello stato 'Upload calibration' della macchina a stati 1, avendo cura di eseguirlo però solo al termine dell'intero processo, dopo aver calibrato tutte le mappe e le curve previste;

- *Dispose report*: questa azione riceve in input il puntatore al report e il cluster di errore per chiudere il file di testo, e verrà inserita nello stato 'End of calibration' della macchina a stati 1.



Si è parlato poco sopra di un input di tipo *variant*: in LabVIEW le variabili *variant* rappresentano un tipo di dato che comprende sia un valore (numerico, booleano, stringa o altro) sia i metadati necessari all'interpretazione del valore stesso. In pratica una variabile *variant* può contenere qualunque tipo di dato standard o definito dall'utente, e si adatta molto bene ai VI per i quali gli input variano a seconda della funzione che essi devono compiere, consentendo così al programmatore di realizzare un unico VI in grado di decodificare diversi input invece che tanti VI diversi, con notevole risparmio di spazio e incremento di efficienza. Nel nostro caso la variabile 'Variant' rappresentata in figura 4.22 viene utilizzata, come visto, sia dal caso 'Save base maps' sia dal caso 'Compare maps': per il primo, essa contiene un array monodimensionale di cluster costituiti da una stringa, indicante il nome della mappa, due vettori di stringhe, contenenti i breakpoints, e una matrice, contenente i valori della mappa, come si può vedere nella figura a destra; per il secondo, essa contiene un array bidimensionale degli stessi cluster appena definiti, dato che il caso 'Compare maps' effettua un confronto tra le mappe precedenti e successive alla calibrazione e pertanto si è pensato di aggiungere una dimensione all'array di cluster in modo da consentire l'affiancamento delle varie mappe.

Un'ultima considerazione sulle variabili di tipo *variant*: come accennato, esse richiedono un'interpretazione del loro contenuto prima di poterlo utilizzare nel codice, e pertanto per decodificare una variabile *variant* è necessario impiegare il blocco 'Variant to

Data', visibile anch'esso nella figura in alto a sinistra, al quale è necessario passare in input un oggetto che definisca la struttura della variabile che si sta decifrando. Nel nostro caso, come si può vedere, si è passato proprio l'array di cluster che rappresenta le varie mappe.

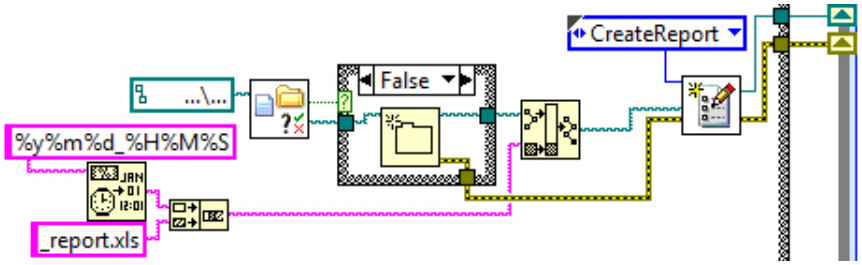


Figura 4.23: Codice per la creazione del report

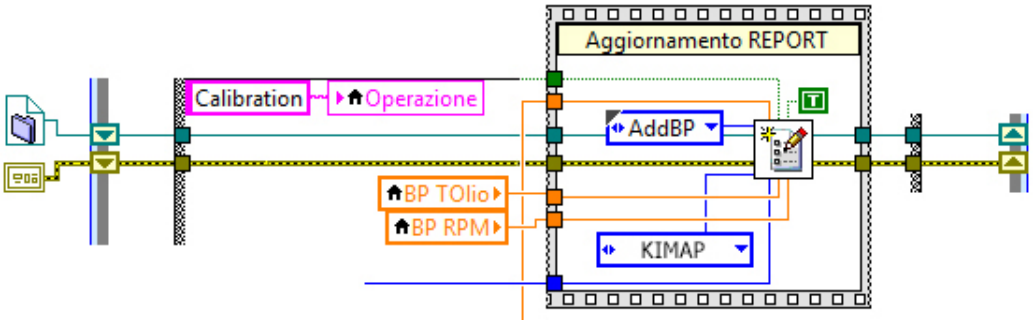


Figura 4.24: Codice per l'aggiunta di un breakpoint calibrato al report

Le figure 4.23 e 4.24 mostrano rispettivamente l'utilizzo del SubVI 'CreateTextReport' per generare un nuovo report e per aggiungere un breakpoint appena calibrato. È interessante notare che in entrambi i casi gli output del SubVI, quello verde in alto corrispondente al puntatore del file e quello giallo in basso corrispondente al cluster di errore, alimentano due *shift registers* del *timed loop* principale, e non solo: nella figura 4.24 si vede che anche due input del SubVI, sempre corrispondenti al puntatore del report e al cluster di errore, provengono dagli stessi *shift registers* di cui sopra, inizializzati con due costanti. Questa logica si è resa necessaria dal momento che, una volta aperto il report, per poter aggiungere in esso ulteriori righe di testo senza sovrascrivere l'intero file bisognava passare in input al SubVI sempre lo stesso puntatore: si capisce quindi che l'utilizzo degli *shift registers* era decisamente appropriato allo scopo, tanto più che così facendo si ha la certezza di lavorare sempre sul report aggiornato dato che il contenuto del file rimane sempre nella memoria del PC e viene aggiornato ad ogni iterazione del *timed loop* principale, ovvero ogni 10 millisecondi.

4.8 Gli indicatori del processo di calibrazione

In figura 4.25 sono rappresentati gli indicatori principali presenti sul front panel che consentono all'utente di seguire e controllare l'avanzamento del processo di calibrazione; tali controlli vengono aggiornati con la stessa frequenza di lavoro del *timed loop* principale e sono definiti come segue:

- *Operazione*: indica lo stato in cui si trova la macchina a stati 1 (stringa);
- *Parametro*: indica lo stato della macchina a stati 2, ovvero il parametro attualmente in calibrazione (stringa);
- *Stato calibrazione*: indica lo stato della macchina a stati 3 (stringa);
- *ConfigOK*: indica se la lettura del file di configurazione base è andata a buon fine o meno (valore booleano);
- *BackupOK*: indica se il backup delle mappe di base preesistenti in centralina è andato a buon fine o meno (valore booleano);
- *LIM_SUP* e *LIM_INF*: indicano rispettivamente l'estremo superiore e l'estremo inferiore dei jumps che vengono effettuati durante la prova;
- *BP Vbat* e *Vbat_INCA*: indicano rispettivamente il breakpoint attuale e il setpoint corrente di tensione batteria;
- *BP RPM* e *RPM_INCA*: indicano rispettivamente il breakpoint attuale e il setpoint corrente di regime motore;
- *BP TOlio* e *TOlio_INCA*: indicano rispettivamente il breakpoint attuale e il setpoint corrente di temperatura dell'olio;
- *BP VVT* e *VVT_INCA*: indicano rispettivamente il breakpoint attuale e il setpoint corrente di angolo VVT target;
- *# BP [1]*: è l'indice del breakpoint attualmente in calibrazione, ovvero il valore del contatore principale della macchina a stati 3;
- *# BP [2]*: è l'indice del guadagno o della prova attualmente in esecuzione, ovvero il valore del contatore secondario della macchina a stati 3;
- *LastValueCAL*: è l'ultimo valore determinato dall'algoritmo dell'analisi dati e che andrà riportato nella mappa o nella curva temporanea;

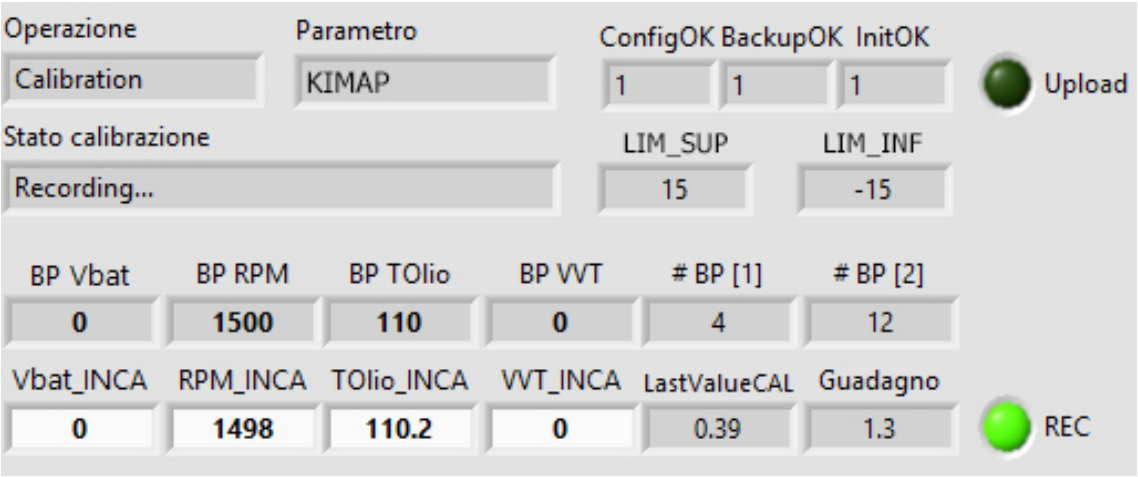


Figura 4.25: Indicatori principali del front panel

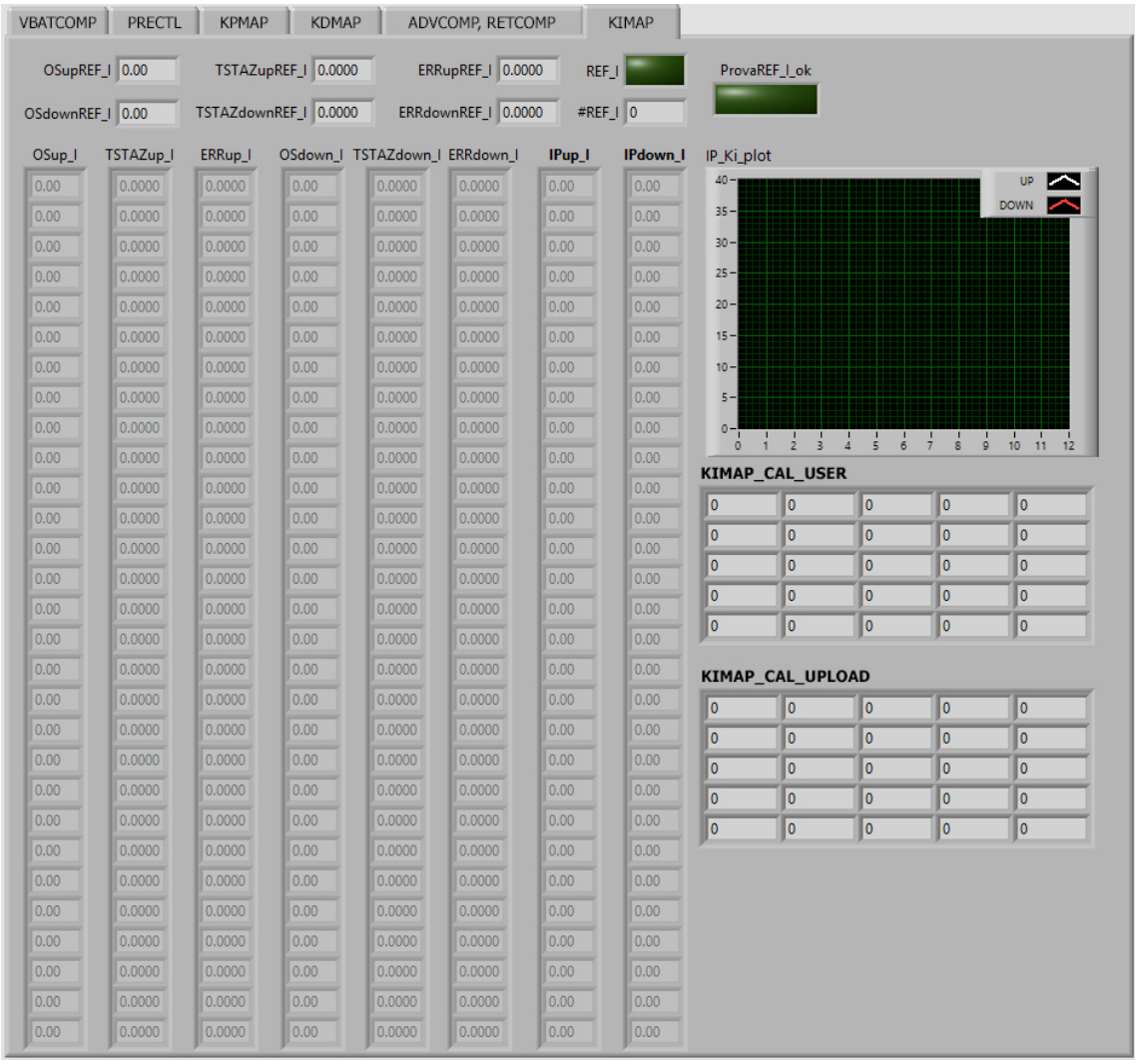


Figura 4.26: *Tab control* con gli indicatori suddivisi per parametro

- *Guadagno*: è il valore di guadagno attualmente impostato e a cui è spianata la mappa in centralina che si sta calibrando;

- *Upload*: indicatore booleano che si attiva durante la fase di upload in centralina delle mappe calibrate;

- *REC*: indicatore booleano che si attiva durante la fase di registrazione.

Gli indicatori a disposizione dell'utente non si limitano chiaramente a quelli appena descritti: si è pensato infatti di inserire un *tab control* con una pagina dedicata a ogni parametro da calibrare, come si può vedere in figura 4.26. Ogni pagina contiene degli indicatori diversi: nell'impossibilità di rappresentarli tutti si è deciso di illustrare solo quelli relativi a KIMAP. In alto si vedono gli indicatori relativi ai parametri di prestazione calcolati per la prova di riferimento, prova che viene notificata tramite l'indicatore booleano in alto a destra; più in basso si vedono i vettori contenenti tutti i parametri di prestazione calcolati per i vari valori di guadagno impostati, mentre a destra si trova un grafico che mostra l'andamento degli indici di prestazione in up e in down in modo che sia subito chiaro qual è il valore di guadagno che li rende minimi. Infine, in basso a destra si trovano due matrici: quella in alto contiene i valori grezzi di calibrazione determinati dai codici per l'analisi dei dati, mentre quella in basso è ottenuta da quella in alto tramite un'opportuna interpolazione.

Anche se non sono rappresentati, gli indicatori relativi alle altre mappe sono simili a quelli appena descritti, e tali che il calibratore può tenere sotto controllo durante il processo tutte le variabili più rilevanti in modo da accorgersi subito se la prova in corso può ritenersi accettabile oppure se è opportuno ripeterla.

4.9 L'interfacciamento con INCA

Per quanto finora descritto il tool prevede il controllo da parte dell'utente utilizzando l'interfaccia di LabVIEW descritta in precedenza, e quindi presuppone un buon livello di conoscenza di LabVIEW da parte dell'utilizzatore, nonché la presenza di un ulteriore PC host in grado di mostrare l'interfaccia utente del tool dato che il PC realtime generalmente non lo consente. È chiaro che sarebbe molto più comodo poter gestire il processo di calibrazione tramite lo stesso PC sul quale è installato INCA, in modo da poter controllare contemporaneamente il software di centralina e l'eseguibile del tool; ancora migliore sarebbe la possibilità di integrare il tool direttamente in INCA, il che risolverebbe tutti i problemi legati alla dipendenza da LabVIEW essendo INCA un software commerciale molto diffuso e praticamente conosciuto dalla quasi totalità del personale di sala.

Si è quindi messo a punto una serie di VI che verranno descritti nel seguito e che consentono di portare l'interfaccia utente del tool di calibrazione direttamente in un *experiment* di INCA, sia per quanto riguarda i controlli modificabili dall'utente sia per quanto riguarda gli indicatori; tali VI consentono nella pratica di aggiungere il PC realtime in INCA come se fosse una centralina standard e di controllare il software in esecuzione su di esso tramite il protocollo XCP.

Si ricorda che l'aggiunta di un hardware su INCA prevede il caricamento dei due file di tipo *a2l* e *hex*, pertanto prima di passare ad analizzare l'*experiment* è opportuno descrivere brevemente come questi due file vengono generati e cosa contengono.

4.9.1 I file *a2l* e *hex* per il tool di calibrazione

Il PC realtime viene riconosciuto da INCA come hardware di tipo *Ethernet System* e anch'esso dovrà mettere a disposizione le *measurements* e le *calibrations* da visualizzare nell'*experiment* proprio come una centralina; nel nostro caso le *measurements* sono rappresentate dagli indicatori del front panel descritti in precedenza, mentre le *calibrations* sono rappresentate dai controlli dello stesso front panel. Ci sono alcune limitazioni in questo senso che hanno obbligato a ridefinire alcuni controlli per consentire ad INCA di riconoscere correttamente tutte le variabili da importare nell'*experiment*:

- tutte le *measurements* devono essere contenute in un unico cluster, denominato nel nostro caso 'ECU_meas';
- tutte le *calibrations* devono essere contenute in un unico cluster, denominato nel nostro caso 'ECU_char';
- il cluster 'ECU_char' deve contenere anche un controllo di tipo numerico a 32 bit che deve essere impostato come primo elemento del cluster;
- 'ECU_meas' e 'ECU_char' devono far parte di una variabile globale contenente altri due controlli, ovvero 'XCParray' e 'XCPcommand';
- tutti gli indicatori presenti nel cluster 'ECU_meas' devono avere regole di accesso in sola lettura da parte di INCA e in sola scrittura da parte del tool;
- tutti i controlli presenti nel cluster 'ECU_char' devono avere regole di accesso in sola scrittura da parte di INCA e in sola lettura da parte del tool (si esclude la fase di inizializzazione del tool durante la quale i controlli vengono impostati ai valori di default);

- non è possibile utilizzare comandi booleani di tipo *latch*; questo ha obbligato a definire come due *enum* i comandi per l'impostazione dell'azione e del parametro da calibrare, rappresentati in figura 4.2;

- non è possibile utilizzare indicatori di tipo booleano, che dovranno essere convertiti in indicatori numerici di tipo *unsigned integer*;

- non è possibile utilizzare indicatori di tipo stringa.

Parte di questi controlli vengono effettuati automaticamente tramite appositi SubVI durante la generazione dei file *a2l* e *hex*, che avviene per mezzo del VI 'GetA2LFromTemplate'; qualora la generazione non andasse a buon fine viene mostrato un messaggio di errore, altrimenti il VI genera i due file richiesti.

È interessante illustrare brevemente il file *a2l*: esso contiene le informazioni necessarie a definire il tipo di hardware che rappresenta ed inoltre contiene tutte le *measurements* e le *calibrations* che dovranno essere visualizzate nell'*experiment*. Di seguito si riportano due estratti del file *a2l* per il tool di calibrazione che definiscono rispettivamente una *calibration*, che però viene chiamata *characteristic*, e una *measure*:

```
/begin CHARACTERISTIC SogliaOSmax[Kd] ""
          VALUE 0x395 __FLOAT32_IEEE_S 0 DEFAULT_CM -1e+026 1e+026
          BYTE_ORDER MSB_FIRST
          FORMAT "%15.5"
          PHYS_UNIT ""
/end CHARACTERISTIC

/begin MEASUREMENT RPM "Engine speed"
          FLOAT32_IEEE DEFAULT_CM 0 0 0 15000
          BYTE_ORDER MSB_FIRST
          ECU_ADDRESS 0xA92
          PHYS_UNIT "rpm"
/end MEASUREMENT
```

Si vede che in entrambi i casi la definizione, racchiusa tra le parole chiave `/begin` e `/end`, prevede per prima cosa il nome (`RPM`), seguito da un'eventuale descrizione tra virgolette (`"Engine Speed"`), quindi il tipo di dato (`FLOAT32_IEEE`) e il range di valori che può rappresentare (`0 15000`); ci sono quindi il criterio di ordinamento dei byte (`MSB_FIRST`), il formato di rappresentazione (`"\%15.5"`, per le *calibrations*), l'indirizzo in memoria (`0xA92`, per le *measurements*) e infine l'unità di misura (`"rpm"`).

È rilevante infine riportare l'estratto di file *a2l* nel quale vengono definiti i parametri di connessione per la comunicazione tra INCA e il PC realtime:

```
/begin XCP_ON_UDP_IP
          0x0100
          0xD903
          ADDRESS "169.254.1.6"
/end XCP_ON_UDP_IP
```

Anche questo blocco di codice è racchiuso tra le parole chiave `/begin` e `/end`, e di particolare interesse sono le ultime due righe: la prima `0xD903` definisce in notazione esadecimale la porta UDP attraverso la quale avviene il collegamento, mentre la seconda `"169.254.1.6"` rappresenta l'indirizzo IP del PC realtime.

4.9.2 L'implementazione dell'XCP

La comunicazione tra il tool e INCA avviene tramite i blocchi 'XCP' e 'iLinkRT Master' che devono essere inizializzati prima di poter essere operativi; entrambe le operazioni di inizializzazione devono essere completate prima che il software acceda al *timed loop* principale, e pertanto sono state inserite in una *flat sequence* esterna al loop. In entrambi i casi si tratta di creare una *queue*: in figura 4.27 si può vedere come viene inizializzata la coda per iLinkRT, che è costituita dall'*enum* 'iLinkRT_Type' e dal cluster 'Conn_Config' contenente i parametri di connessione, mentre in figura 4.28 si può vedere come viene inizializzata la coda per XCP, che è costituita da un *Tick Counter* e dal cluster 'ECU_meas'. Per quanto riguarda iLinkRT, una volta inizializzata la coda viene gestita interamente dal blocco 'iLinkRT Master', rappresentato in azzurro in figura 4.28, mentre per quanto riguarda XCP tale coda deve essere riempita manualmente tramite il solito blocco 'Enqueue Element' rappresentato in basso a destra sempre in figura 4.28: si noti che tale blocco è posizionato nel *timed loop* principale, fuori dalle varie macchine a stati, e pertanto l'aggiornamento della coda viene eseguito con frequenza pari a 100 Hz. Questa operazione è di cruciale importanza dato che è la responsabile dell'aggiornamento di tutte le *measurements* visibili in INCA.

Il blocco XCP non si occupa solo della trasmissione delle *measurements* ma deve aggiornare anche le *calibrations* contenute nel cluster 'ECU_char', ma sia 'ECU_char' che 'ECU_meas' sono definite nel VI principale: pertanto, affinché siano visibili anche da altri VI, entrambi i controlli sono stati spostati in una variabile globale in modo che siano immediatamente disponibili per tutti i VI e SubVI. È interessante notare che, affinché l'utente possa vedere su INCA tutte le *measurements* e tutte le *calibrations* al loro valore più recente, è necessario aggiornare 'ECU_char' e 'ECU_meas' ad ogni iterazione del *timed loop* principale e non solo: trattandosi di variabili globali è di fondamentale importanza verificare che vengano lette (per quanto riguarda 'ECU_char') e soprattutto scritte (per quanto riguarda 'ECU_meas') univocamente dal solo loop principale, pertanto si è

Figura 4.27: Inizializzazione del collegamento via iLinkRT

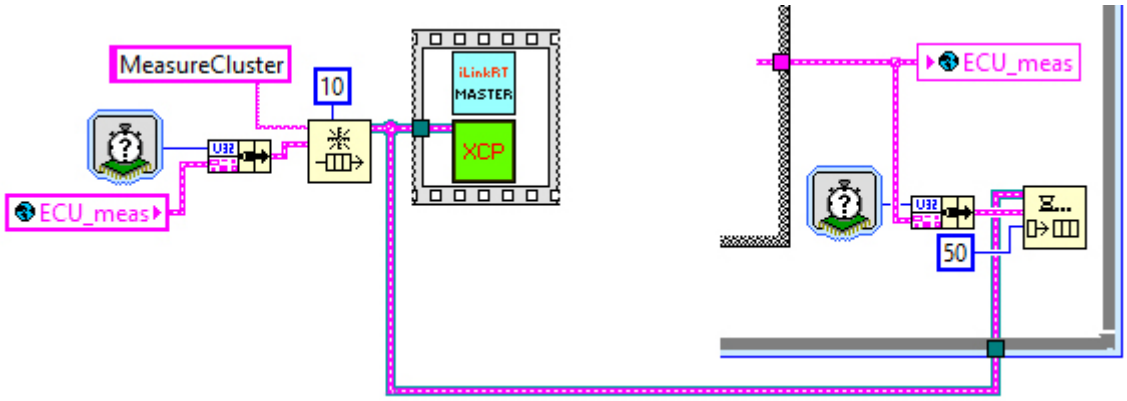


Figura 4.28: Inizializzazione e aggiornamento della coda dell'XCP

reso necessario eliminare ogni istruzione di scrittura su 'ECU_meas' che non appartenesse al *timed loop* principale. La lettura di 'ECU_char' non è critica quanto la scrittura di 'ECU_meas', ma è comunque consigliabile che avvenga una sola volta al di fuori delle varie macchine a stati, utilizzando i blocchi 'Unbundle cluster by name' per recuperare di volta in volta i valori i valori necessari.

4.9.3 L'*experiment* per il controllo del tool

In figura 4.29 è rappresentato l'*experiment* messo a punto per controllare il tool direttamente da INCA; si fa notare che dato il numero relativamente elevato di mappe e curve che è necessario calibrare si è preferito suddividere gli indicatori relativi a ciascun parametro in varie pagine, esattamente come avveniva per l'interfaccia utente di LabVIEW descritta in precedenza. Segue l'elenco e la descrizione delle varie finestre presenti nella pagina relativa a KIMAP:

1. *Stato esecuzione*: permette all'utente di controllare il processo di calibrazione (avvio, pausa, ripresa) e di impostare il parametro da calibrare. Questa finestra è presente in tutte le pagine dell'*experiment*;

2. *Operazione corrente*: contiene degli indicatori booleani che informano sullo stato di esecuzione del tool. Questa finestra è presente in tutte le pagine dell'*experiment*;

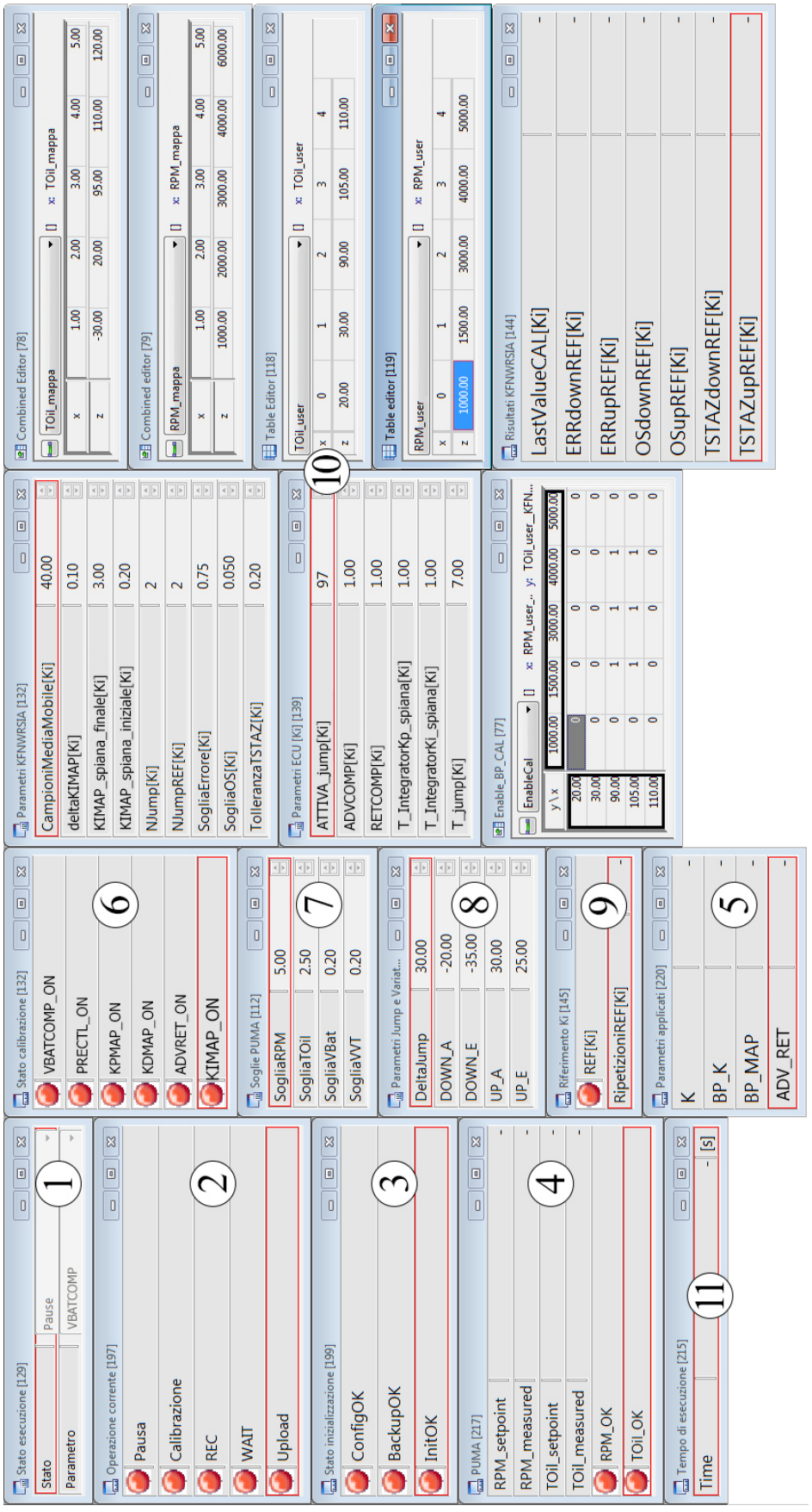


Figura 4.29: *Experiment* per il controllo del tool su INCA

3. *Stato inizializzazione*: contiene degli indicatori booleani che informano sull'avvenuta e corretta inizializzazione del tool, comprendendo la lettura del file di configurazione, il backup delle mappe preesistenti in ECU e il check delle variabili caricate su MCE. Questa finestra è presente in tutte le pagine dell'*experiment*;

4. *PUMA*: indica il breakpoint attualmente in calibrazione, sia come setpoint che come *measurements* della ECU; contiene inoltre due indicatori booleani che informano l'utente sull'avvenuto raggiungimento del setpoint da parte del banco. Questa finestra è presente in tutte le pagine dell'*experiment* ed è stata personalizzata in funzione del parametro (per le mappe vengono visualizzati *RPM* e *TOlio*, mentre per PRECTL si aggiunge *VVT angle* e per VBATCOMP si aggiunge *VBat*);

5. *Parametri applicati*: indica i contatori di breakpoints (principale e secondario) attualmente in calibrazione e il guadagno di mappa (solo per i parametri che lo richiedono) correntemente impostato dal tool. Questa finestra è presente in tutte le pagine dell'*experiment*;

6. *Stato calibrazione*: indica il parametro attualmente in calibrazione. Questa finestra è presente in tutte le pagine dell'*experiment*;

7. *Soglie PUMA*: contiene dei controlli numerici che permettono all'utente di impostare la tolleranza di controllo sul setpoint, in funzione di regime motore, temperatura olio, tensione batteria e angolo VVT target. Questa finestra è presente in tutte le pagine dell'*experiment*;

8. *Parametri jump e Variatori*: contiene dei controlli numerici che permettono all'utente di impostare il range di lavoro dei variatori e l'entità desiderata per i jumps da effettuare durante le prove. Questa finestra è presente in tutte le pagine dell'*experiment*;

9. *Riferimento*: comprende un indicatore booleano, che si attiva in corrispondenza della prova di riferimento, e un contatore numerico che indica il numero della prova di riferimento attualmente in esecuzione. Questa finestra è presente solo nelle pagine relative a KDMAP, ADVCOMP e RETCOMP e KIMAP, dato che prevedono un caso di riferimento;

10. *Parametri KIMAP*, *Parametri ECU*, *Enable_BP_CAL*, *Risultati KIMAP*: contengono tutti i controlli presenti in 'ECU_char' necessari all'utente per l'inserimento dei parametri da impostare in ECU per l'esecuzione delle prove di calibrazione; il nome e il contenuto di tali finestre differiscono in funzione del parametro scelto, ma generalmente comprendono la mappa (o la curva) per l'abilitazione dei breakpoints, gli array per impostare i breakpoints di mappa e i breakpoints utente e una serie

di indicatori che informano l'utente sui risultati intermedi ottenuti tramite l'analisi dati;

11. *Tempo di esecuzione*: contatore del tempo trascorso in secondi dall'avvio del tool; è stato introdotto per dare un feedback visivo all'utente che il tool è effettivamente in esecuzione.

Capitolo 5

Tool per la calibrazione del controllo LAMBDA

Il tool per l'automazione della calibrazione del controllo LAMBDA è stato realizzato come attività di tesi dall'Ing. Blando; si tratta di un'estensione del TAC esattamente come il tool per la calibrazione del controllo VVT, ma a differenza di quest'ultimo la struttura iniziale del tool LAMBDA è stata definita diversamente in quanto all'epoca della progettazione non era ancora prevista l'integrazione con INCA, e i VI per il collegamento via iLinkRT e l'acquisizione dei dati di centralina erano leggermente diversi da quelli attualmente disponibili e descritti nel capitolo precedente. Non solo: anche l'impostazione dell'intero processo di calibrazione e le operazioni concesse all'utente sono state definite in modo diverso, tanto che sembra opportuno descrivere brevemente il tool come si presentava prima dell'aggiornamento che ha subito. Ad ogni modo, prima di far ciò sarà necessario introdurre brevemente la struttura della funzione di controllo LambdaCTL implementata nella ECU, in modo da aver chiare le mappe e la sequenza di calibrazione.

5.1 La funzione per il controllo LAMBDA

La funzione svolta dal controllo LAMBDA in catena chiusa è quella di correggere il valore di lambda attualmente misurato dalla sonda per riportarlo al target imposto in funzione della condizione di funzionamento del motore; tale correzione viene effettuata, nella pratica, attraverso un fattore che viene moltiplicato alla massa di combustibile iniettata in camera. Data la riservatezza del software di controllo nel seguito si descriverà la generica funzione LambdaCTL, adottando lo stesso modo di procedere già impiegato nel capitolo relativo alla strategia di controllo per il VVT; tale funzione è costituita dalle seguenti sei subroutines:

- 'LambdaCOORDINATOR': ha il compito di coordinare le varie richieste di lambda target e impostare le loro priorità in base alle condizioni di funzionamento del propulsore, tra le quali si ricordano ad esempio la fase di catalyst heating, l'eventuale component protection e lo scavenging. Chiaramente esiste una gerarchia predefinita dal produttore per la gestione di tutte le richieste che giungono al blocco LambdaCOORDINATOR, in uscita dal quale si avrà *lamCylSP*, ovvero il valore di lambda target da impostare in ogni cilindro;

- 'LambdaCHAR': determina i parametri del *plant*, ovvero della sonda lambda a monte del catalizzatore; il sistema fisico viene modellato tramite una funzione di trasferimento del primo ordine avente in input *mFuel*, la quantità di combustibile iniettata, normalizzata rispetto a $\lambda = 1$, mentre in output restituisce *lamSensor*, ovvero il lambda misurato dalla sonda. L'intera dinamica del sistema può essere così definita mediante due parametri, ai quali corrisponderanno altrettante mappe da calibrare: un delay (DLMAP) e un tempo (TAUMAP);

- 'LambdaSET': calcola il setpoint di lambda nel collettore (*lamManSP*) e in corrispondenza della sonda (*lamSensorSP*). Ha come input il target di lambda nel cilindro e i parametri del plant;

- 'LambdaACTIVE': gestisce le condizioni di attivazione e disattivazione del controllo LAMBDA, il quale può essere spento per un malfunzionamento della sonda, in fase di cranking, durante e dopo un cutoff in decelerazione oppure in caso di misfire;

- 'LambdaCORR': è il controller del lambda target, in quanto riceve in input il setpoint *lamSensorSP* e il lambda misurato *lamSensor*, calcola l'errore *lamDelta* tra le due grandezze e calcola la correzione *mFuelCorr* da applicare alla massa di combustibile calcolata in catena aperta. Il valore di *mFuelCorr* dipende a sua volta dalla reattività del controllore espressa tramite la mappa SIGMAP da calibrare;

- 'LambdaDIAG': effettua la diagnosi del sistema di controllo e della sonda.

5.1.1 La procedura di calibrazione: LambdaCHAR

Le mappe da calibrare, come accennato in precedenza, sono due:

- DLMAP: rappresenta il ritardo dovuto al trasporto del pacchetto di gas dal cilindro fino alla sonda lambda. È funzione del regime di rotazione e del carico motore;

- TAUMAP: è la costante di tempo della funzione di trasferimento con la quale viene modellata la dinamica della sonda lambda; anch'essa è funzione del regime di rotazione e del carico motore con gli stessi breakpoints di DLMAP.

La calibrazione di queste mappe prevede l'osservazione della risposta della sonda in seguito a un'eccitazione, ottenuta tramite variazione della quantità di combustibile iniettato; per tutti i test verrà mantenuto il controllo LAMBDA sempre spento, cosa che impone al sistema di controllo di utilizzare solo la catena aperta, senza alcuna possibilità di correzione. In breve, per ogni breakpoints sarà necessario seguire la seguente procedura:

1. disattivazione del controllo LAMBDA;
2. impostazione del lambda target;
3. eccitazione del sistema tramite *fuel steps*;
4. acquisizione e analisi dati: bisogna determinare i due valori di DELAY e TAU per i quali il modello replica al meglio il comportamento del sistema fisico.

È interessante spiegare cosa si intende con *fuel steps*: come già anticipato, l'eccitazione consiste nell'effettuare delle variazioni in salita e in discesa, veri e propri gradini, della quantità di combustibile iniettato; l'ampiezza di tali transizioni dovrà essere compresa tra 0.5 e 1, mentre la relativa durata dipenderà da una mappa funzione del regime di rotazione e del carico che dovrà essere stimata in funzione del tempo necessario alla regimazione del segnale in uscita dalla sonda. L'approccio di calibrazione che si viene delineando è sicuramente di tipo statistico, dato che i valori finali saranno ottenuti tramite una media calcolata su tutti i gradini effettuati: il numero di variazioni deriverà pertanto dal necessario compromesso tra la precisione dei risultati e il tempo richiesto dalle prove.

I codici per l'analisi dei dati acquisiti e la calibrazione di DLMAP e TAUMAP sono stati scritti in MATLAB e implementati nel tool tramite la palette 'MathScript Node' messa a disposizione da LabVIEW, tramite la quale è possibile integrare uno script di MATLAB in un VI di LabVIEW; tali codici non verranno modificati durante l'aggiornamento del tool e pertanto non sembra opportuno descriverli nel dettaglio.

5.1.2 La procedura di calibrazione: SIGMA

Per quanto riguarda il controller vero e proprio, la mappa da calibrare è SIGMAP, relativa alla velocità di intervento del sistema di controllo nell'annullare l'errore tra il lambda target e quello misurato; tale mappa è funzione del regime di rotazione e del carico motore.

La calibrazione del parametro sigma prevede l'esecuzione di gradini di lambda target mantenendo il sistema controllo in funzione per valutarne la risposta in seguito all'eccitazione; l'intero sistema di iniezione sarà quindi controllato normalmente in closed loop, e per ogni breakpoint si seguirà la seguente procedura:

1. attivazione del controllo LAMBDA;
2. abilitazione modalità *lambda steps*;
3. variazione a gradino del lambda target;
4. acquisizione e analisi dati;
5. ripetere i punti precedenti spianando la mappa di sigma a valori compresi nell'intervallo $[0.1; 1]$ in step di 0.1.

Si noti che in questo caso la procedura di calibrazione prevede l'esecuzione di una spazzolata di valori, ed anche il numero di transizioni è sensibilmente ridotto rispetto a quanto visto nella calibrazione di delay e tau, avendo scelto un valore ottimo pari a 6; per quanto riguarda l'ampiezza delle transizioni, invece, il valore consigliato è pari a 0.1.

Le transizioni dovranno essere centrate sul valore di lambda imposto dal sistema di controllo per quel determinato punto motore, anche se per determinati punti è necessario arricchire leggermente la miscela per evitare danni ai sistemi di post-trattamento dei gas di scarico; nella pratica ciò significa che la media tra il valore massimo e il valore minimo dell'onda quadra che eccita il sistema dovrà essere pari all'output di mappa restituito dal controller.

Anche in questo caso il codice di analisi dei segnali acquisiti è stato scritto in MATLAB e implementato nel tool tramite l'apposito VI 'MathScript Node' e non verrà modificato in occasione dell'aggiornamento descritto nel paragrafo seguente, pertanto non sembra rilevante descriverlo nel dettaglio; basti sapere che tale codice ha il compito di determinare il valore ottimo di sigma tra quelli esaminati per ogni breakpoint valutando alcuni parametri che indicano l'efficacia del sistema di controllo nel reagire all'eccitazione imposta.

5.2 La struttura del tool esistente

Il VI principale del tool è stato progettato in modo da avere al suo interno diversi *timed loop*, costantemente in esecuzione a frequenze prestabilite ed ognuno destinato a svolgere funzioni ben precise, a loro volta contenenti delle *case structures* attivate da variabili locali; è chiaro che, essendo il processo di calibrazione una procedura da seguire in modo

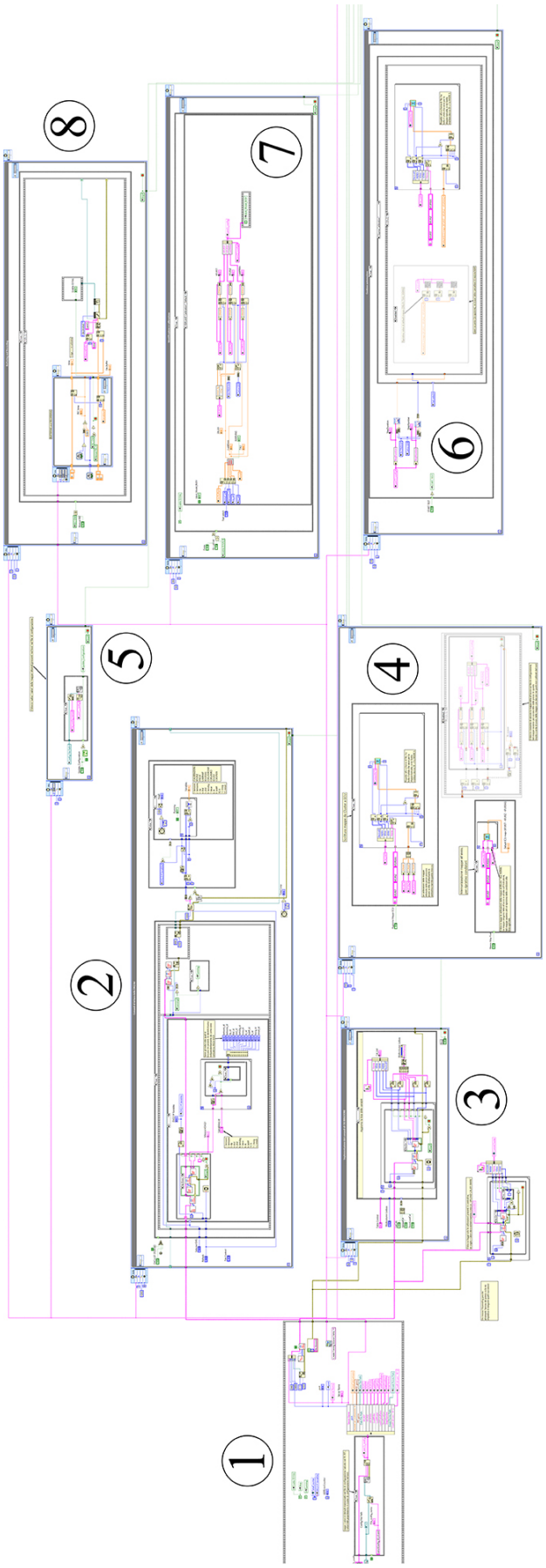


Figura 5.1: Struttura del block diagram

ordinato e con le giuste tempistiche, la sincronizzazione tra i vari loop è di fondamentale importanza, e sarà proprio questo uno dei motivi che ha comportato l'abbandono di una simile struttura a cicli separati nell'ottica dell'aggiornamento futuro. Di seguito si riporta l'elenco dei vari *timed loop* presenti nel VI principale:

- Blocco 1: *flat sequence* per la lettura del file di configurazione e inizializzazione dei controlli del front panel. Viene eseguita automaticamente e una sola volta all'avvio del tool;
- Blocco 2: *timed loop* eseguito alla frequenza di 1000 Hz per la lettura della DAQ list e check delle variabili caricate su MCE. Il codice al suo interno è comandato da un comando booleano manuale del front panel;
- Blocco 3: *timed loop* eseguito alla frequenza di 20 Hz per la lettura dei parametri di calibrazione disponibili. Il codice al suo interno è comandato da un comando booleano manuale del front panel;
- Blocco 4: *timed loop* eseguito alla frequenza di 2 Hz per il backup delle mappe preesistenti in ECU e upload delle mappe al termine della calibrazione. Il codice al suo interno è comandato da un comando booleano manuale del front panel;
- Blocco 5: *timed loop* eseguito alla frequenza di 2 Hz per l'aggiornamento del file di configurazione con i risultati della calibrazione. Il codice al suo interno è comandato da un comando booleano manuale del front panel;
- Blocco 6: *timed loop* eseguito alla frequenza di 10 Hz per la gestione dell'automazione. Il codice al suo interno viene comandato dal comando di avvio del processo di calibrazione presente sul front panel;
- Blocco 7: *timed loop* eseguito alla frequenza di 5 Hz per l'analisi dei dati acquisiti tramite i codici MATLAB, identificazione e calibrazione dei parametri. Il codice al suo interno è comandato automaticamente dal blocco 7 tramite la variabile locale booleana 'Enable ID/VAL';
- Blocco 8: *timed loop* eseguito alla frequenza di 100 Hz per la gestione della registrazione dei dati inviati dalla ECU e salvataggio su disco dei segnali acquisiti. Il codice al suo interno è comandato automaticamente dal blocco 7 tramite la variabile locale booleana 'REC ON'.

Se tale disposizione ha permesso di mantenere ben distinte le varie operazioni svolte dal tool è anche vero che avere dei cicli simultaneamente in esecuzione può comportare problemi di priorità nonché criticità nel momento in cui due cicli tentano di scrivere

contemporaneamente la stessa variabile. Ad ogni modo il tool così come è stato descritto era stato progettato in modo accurato dato che le prove condotte in sala per verificarne il funzionamento hanno restituito un esito positivo. Uno dei principali difetti, se così si può definire, è che alcune operazioni dipendono ancora dall'intervento dell'utente, chiamato ad agire sui comandi del front panel per il backup delle mappe, ad esempio, oppure per il caricamento della DAQ list.

Da parte loro, i controlli del front panel sono stati posizionati in modo intelligente dato che si hanno a disposizione tre clusters nei quali sono raggruppati i comandi principali:

- *General settings*: contiene i comandi per la gestione del tool e del processo di calibrazione, tra cui si ricordano i controlli numerici per il settaggio del punto motore da calibrare, l'*enum* per la scelta del parametro da calibrare e il *command button* per l'avvio della calibrazione;

- *LambdaCHAR calibration*: consente all'utente di impostare i vari parametri necessari alla calibrazione di delay e tau, tra cui l'ampiezza dell'eccitazione e l'offset dell'onda quadra, nonché contiene degli indicatori che permettono di verificare l'esito della calibrazione visualizzando i risultati intermedi ed eventuali warnings;

- *Sigma calibration*: analogamente al cluster precedente, contiene i comandi necessari all'utente per impostare i parametri per l'esecuzione delle prove di *lambda steps* e gli indicatori per visualizzare i risultati intermedi, il valore finale da calibrare ed eventuali warnings.

Il front panel contiene inoltre delle mappe temporanee nelle quali vengono inseriti i valori dei breakpoints calibrati, in modo che l'utente possa continuamente tenere sotto controllo l'andamento del processo di calibrazione, un cluster destinato al file di configurazione, nel caso l'utente voglia aggiornarlo, e dei controlli specifici che consentono di gestire la comunicazione con MCE avviando la lettura della DAQ list e visualizzandone il contenuto.

Volendo ora descrivere il funzionamento del tool si può fare riferimento alla figura 5.2, nella quale sono rappresentate le varie operazioni eseguite dal tool. All'avvio il tool stabilisce la connessione con la centralina, legge i valori di default contenuti nel file di configurazione e li imposta nei controlli del front panel, dopodiché l'utente può decidere di effettuare il backup delle tre mappe DLMAP, TAUMAP e SIGMAP attualmente presenti in ECU tramite il pulsante 'Memo Map First'. Il tool è pronto per iniziare la calibrazione: l'utente può scegliere quale breakpoint calibrare per primo impostando manualmente regime e carico del motore, mentre degli appositi led indicheranno se il setpoint impostato è stato effettivamente raggiunto o meno. Si fa notare che l'impostazione del punto motore è demandata all'utente, il quale dovrà intervenire sul software di controllo del banco per

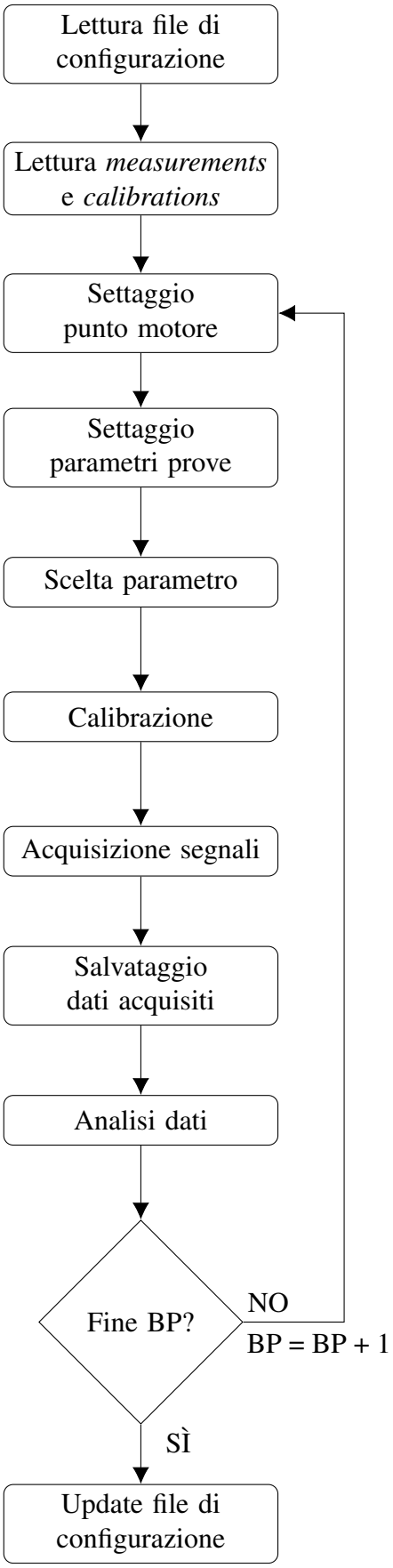


Figura 5.2: Sequenza delle operazioni eseguite dal tool

portare il motore nelle condizioni richieste. Una volta raggiunto e stabilizzato il breakpoint, il calibratore deve scegliere se calibrare LambdaCHAR oppure SIGMA tramite l'*enum* 'Test Select'; dopo aver controllato che tutti i parametri per l'esecuzione delle prove siano corretti, il processo di calibrazione può finalmente essere avviato tramite il pulsante 'Start Test'. Durante la calibrazione il tool imposta automaticamente i vari parametri in centralina, registra i segnali di interesse e li analizza tramite i codici MATLAB; questi restituiscono i valori da calibrare ed eventuali warning che informano l'utente se la prova appena condotta è da ritenersi valida oppure deve essere ripetuta. Al termine della calibrazione, quando tutti i breakpoints sono stati esaminati e le mappe sono complete, l'utente può salvare le mappe appena calibrate nel file di configurazione.

5.3 L'aggiornamento del tool esistente

Nella configurazione appena descritta il tool consente di calibrare la funzione LambdaCTL in modo semiautomatico, per quanto visto in precedenza: l'impostazione dei parametri in centralina è gestita in autonomia dal VI, ma il passaggio da un breakpoint a quello successivo, il backup delle mappe base e l'aggiornamento di quelle calibrate sono tutte operazioni ancora dipendenti dall'utente. Inoltre non è stata ancora prevista la generazione di un report di calibrazione, per tenere traccia delle operazioni compiute durante il processo e di eventuali prove non valide; per di più il tool, allo stato attuale, non può essere integrato in INCA e pertanto il suo impiego è sostanzialmente dipendente dalla presenza di un PC host in sala prove sul quale sia installato LabVIEW limitando il suo utilizzo al personale dotato di una buona esperienza con il software e l'hardware di National Instruments. Si vuole infine sottolineare che questa versione del software non consente all'utente di mettere in pausa, riprendere o arrestare il processo di calibrazione, cosa che ne limita la flessibilità. È chiaro che, anche alla luce di quanto già visto per il tool di calibrazione VVT, quelli appena elencati sono tutti problemi superabili: il quesito che ci si potrebbe porre è relativo alla struttura del tool stesso, ovvero se conviene modificare il progetto esistente oppure ripartire con un progetto totalmente nuovo.

In un primo momento, per prendere confidenza con il processo di calibrazione del controllo LAMBDA e con i codici già implementati, si è pensato di modificare il progetto esistente; tuttavia ci si è resi conto che continuare per questa strada avrebbe allungato parecchio il tempo necessario allo sviluppo, nonché comportato notevoli difficoltà realizzative dal momento che sarebbe stato necessario stravolgere la struttura del software, di per sè già abbastanza complessa. Pertanto si è deciso di congelare il tool già esistente e aprire un nuovo progetto che, nelle ipotesi iniziali, sarebbe dovuto essere molto simile al

software già realizzato per la calibrazione del controllo VVT; nell'elenco che segue sono riassunti i punti fondamentali su cui si è concentrata l'attività di aggiornamento:

- il VI principale deve essere strutturato come macchina a stati, mantenendo un solo *timed loop* con il quale gestire l'intero processo di calibrazione;
- è necessario consentire all'utente di mettere in pausa e riprendere il processo di calibrazione;
- la gestione dei breakpoints deve essere migliorata, in modo da permettere all'utente di scegliere quali punti calibrare, e soprattutto il passaggio da un breakpoint a quello seguente deve essere automatico;
- l'interazione con la ECU deve essere effettuata via iLinkRT sfruttando i nuovi e più efficienti VI descritti nel capitolo relativo al VVT;
- deve essere prevista l'integrazione con INCA tramite XCP, ovvero bisogna suddividere controlli e indicatori nelle variabili globali 'ECU_char' e 'ECU_meas' seguendo tutte le accortezze necessarie alla corretta generazione dei file *a2l* e *hex*;
- al termine della calibrazione il tool deve generare il report del processo;
- il file di configurazione deve essere trasformato da *xml* a *ini*, in modo da consentirne la modifica tramite un qualunque editor di testo, e il suo aggiornamento deve avvenire in automatico non appena l'utente modifica il cluster di configurazione.

5.3.1 La nuova struttura del tool

In figura 5.3 è rappresentata la struttura di base del VI principale: si noti che è esattamente identica alla struttura del tool per la calibrazione del VVT, fatta eccezione per la presenza di un ulteriore *enum*. Essa infatti è costituita da un *timed loop* esterno che contiene tre macchine a stati nidificate comandate dai rispettivi *enum controls*; gli elementi presenti sono i seguenti:

- *Timed loop* 1: è il loop principale contenente tutto il codice del VI destinato alla calibrazione delle varie mappe. Valgono le stesse considerazioni già riportate nel capitolo precedente, compresa la frequenza di esecuzione impostata anche in questo caso a 100 Hz;
- Macchina a stati principale 2: è la macchina a stati più esterna, ha il compito di gestire i vari processi di automazione; comandata dall'*enum* A, contiene tutto il codice per l'esecuzione dei vari stati del tool tra cui l'inizializzazione, la gestione delle azioni dell'utente e la calibrazione;

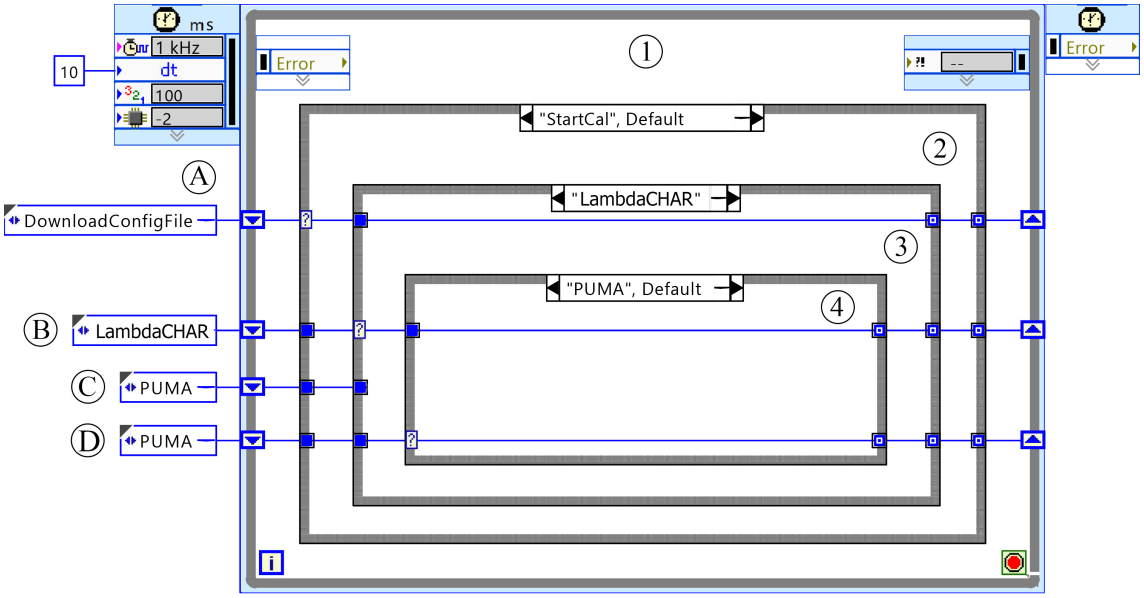


Figura 5.3: Struttura delle macchine a stati presenti nel tool

- Macchina a stati secondaria 3: è la macchina a stati intermedia comandata dall'*enum* B e deputata alla gestione dei parametri da calibrare; si fa notare che questa *case structure* è presente solo nello stato 'StartCal' della macchina a stati 1;

- Macchina a stati interna 4: è la macchina a stati destinata alla gestione dei vari step di calibrazione; comandata dall'*enum* C nel caso si stia calibrando LambdaCHAR, o dall'*enum* D nel caso si stia calibrando SIGMA, questa *case structure* andrà ripetuta dentro i casi 'LambdaCHAR' e 'SIGMA' della macchina a stati 3;

- *Enum* A: controlla la macchina a stati 2; è inizializzato allo stato di 'Download Configuration File' ed è costituito dai seguenti casi, che vengono peraltro eseguiti nell'ordine riportato:

  1. *Download Configuration File*: il tool legge dal disco del PC realtime il file di configurazione in formato *ini*, carica i valori in esso contenuti nei controlli del front panel, quindi crea la cartella in cui verranno salvati i dati acquisiti e prepara il report di calibrazione;
  2. *Backup ECU Default Maps*: il tool ricerca in centralina le mappe da calibrare e le salva in un cluster apposito e nel report di calibrazione, nel caso l'utente voglia ripristinarle al termine del processo di calibrazione;
  3. *Initialization*: il tool verifica che le *measurements* e le *calibrations* necessarie alla calibrazione siano effettivamente state aggiunte dall'utente nell'interfaccia di MCE, in caso contrario il tool non dà il consenso all'avvio della calibrazione;

4. *Pause*: è lo stato nel quale il tool rimane dopo aver eseguito i tre stati iniziali oppure finché l'utente non comanda l'avvio (o la ripresa) del processo di calibrazione;
5. *Start Calibration*: è lo stato destinato alla calibrazione;
6. *Upload Calibration*: in questo stato il tool carica in ECU la mappa appena calibrata, e tale operazione viene effettuata dopo che tutte e tre le mappe DLMAP, TAUMAP e SIGMAP sono state calibrate;
7. *End of Calibration*: in questo stato il software reinizializza tutte le variabili e si riporta nello stato *Pause*, in attesa di un'azione da parte dell'utente.

- *Enum* B: controlla la macchina a stati 3; è inizializzato allo stato di *None* ed è costituito dai seguenti casi, corrispondenti alle due tipologie di calibrazione previste dalla procedura:

  1. *None*: questo stato valuta, al termine della calibrazione di LambdaCHAR e SIGMA, se le mappe sono state calibrate correttamente; in caso affermativo manda il tool nello stato di upload delle mappe in ECU;
  2. *LambdaCHAR*: stato di calibrazione delle mappe DLMAP e TAUMAP;
  3. *SIGMA*: stato di calibrazione della mappa SIGMAP.

- *Enum* C: controlla la macchina a stati 4 durante la calibrazione di LambdaCHAR; è inizializzato allo stato di *PUMA* ed è costituito dai seguenti casi:

  1. *PUMA*: settaggio del punto motore;
  2. *Wait*: stato di attesa necessario alla regimazione dei segnali da acquisire;
  3. *LambdaON*: attivazione del controllo LAMBDA;
  4. *CalcFRmean*: calcolo del fattore di correzione da applicare alla massa di combustibile iniettata;
  5. *LambdaOFF*: disattivazione del controllo LAMBDA;
  6. *CalcLamsoniMean*: calcolo del valore medio misurato dalla sonda lambda;
  7. *CalcLimits*: calcolo iniziale dei limiti di lambda per l'eccitazione del sistema;
  8. *CalcFRpid*: raffinamento dei limiti superiore ed inferiore dell'onda quadra;
  9. *OndaQuadra*: applicazione dell'onda quadra e acquisizione delle *measurements*;
  10. *AnalisiDati*: analisi e trattamento dei segnali acquisiti per l'identificazione dei valori da calibrare in mappa in corrispondenza del breakpoint in esame;

11. *EndCal*: aggiornamento del report di calibrazione, reinizializzazione dei controlli e degli indicatori e preparazione del tool ad effettuare la calibrazione del breakpoint successivo.

- *Enum* D: controlla la macchina a stati 4 durante la calibrazione di SIGMA; è inizializzato allo stato di *PUMA* ed è costituito dai seguenti casi:

    1. *PUMA*: settaggio del punto motore;
    2. *Wait*: stato di attesa necessario alla regimazione dei segnali da acquisire;
    3. *SpianaDelayTau*: in questo stato il tool spiana le due mappe DLMAP e TAUMAP ai valori corrispondenti al breakpoint in esame;
    4. *LambdaON*: attivazione del controllo LAMBDA;
    5. *SpianaSigma*: il tool spiana la mappa SIGMAP al valore corrente di sigma durante la spazzolata;
    6. *LimitiLambda*: calcolo dei limiti superiore e inferiore dell'onda quadra da applicare al lambda target;
    7. *OndaQuadra*: applicazione dell'onda quadra e acquisizione delle *measurements*;
    8. *Ripristino*: ripristino dei parametri ECU ai valori di default precedenti all'esecuzione della prova;
    9. *AnalisiDati*: analisi e trattamento dei segnali acquisiti per l'identificazione dei valori da calibrare in mappa in corrispondenza del breakpoint in esame;
    10. *EndCal*: aggiornamento del report di calibrazione, reinizializzazione dei controlli e degli indicatori e preparazione del tool ad effettuare la calibrazione del breakpoint successivo.

Si sarà notato che la macchina a stati più interna è comandata da due *enum* diversi a seconda che si stia calibrando le due mappe di LambdaCHAR o la mappa di SIGMA, mentre nel tool di calibrazione del VVT la macchina a stati più interna era comandata sempre dallo stesso *enum*, indipendentemente dal parametro in calibrazione. Questa scelta, trasparente all'utilizzatore finale, è stata fatta per poter gestire più comodamente e in modo più flessibile la calibrazione delle tre mappe, mantenendo ben distinto il processo di calibrazione di delay e tau da quello richiesto da sigma. Un confronto con le procedure di calibrazione esposte nel capitolo relativo al VVT non può che confermare questa scelta: si ricorderà infatti che, sebbene le mappe e le curve differissero anche significativamente nella forma, tutte richiedevano di seguire step simili per la calibrazione. Bisogna anche aggiungere che i parametri da calibrare per la funzione VVT1 erano molti

di più rispetto a quelli richiesti dalla funzione LambdaCTL, pertanto nel primo caso era sembrato ragionevole utilizzare un solo *enum*, opportunamente adattato in modo da poter essere utilizzato per tutte le mappe, mentre nel secondo caso è stato possibile utilizzare due *enum*, considerando anche che le procedure di calibrazione da implementare sono solo due.

5.3.2 La nuova gestione dell'automazione

Come anticipato nell'introduzione del capitolo, l'aggiornamento del tool LAMBDA è stato condotto ricalcando e, in parte, copiando quanto già sviluppato per il tool VVT: sebbene le operazioni eseguite dal software siano nella pratica le stesse, processo di calibrazione escluso, può comunque essere utile descriverle brevemente aiutandosi con i diagrammi di flusso riportati nelle figure 5.5 e 5.6.

Passando alla descrizione del diagramma di figura 5.5 si vede che subito dopo l'avvio dell'eseguibile il tool legge il file di configurazione dall'hard disk del PC realtime, quindi effettua il backup delle mappe già esistenti in ECU, controlla che le *measurements* e le *calibrations* richieste per la calibrazione siano state caricate su MCE quindi si porta nello stato di pausa, in attesa di un'azione da parte del calibratore. In questo il tool VVT è identico al tool LAMBDA, e infatti esattamente come nel tool VVT l'utente ha a disposizione due controlli, rappresentati in figura 5.4, per modificare lo stato di esecuzione del software e per scegliere il parametro da calibrare. L'*enum* 'Stato' consente di avviare il processo di calibrazione, metterlo in pausa, riprenderlo o arrestarlo definitivamente, mentre tramite l'*enum* 'Test' l'utente può scegliere se procedere con la calibrazione delle due mappe di LambdaCHAR oppure della mappa di SIGMA. Bisogna anche precisare che tale scelta è stata lasciata all'utente, nonostante la calibrazione di SIGMAP prescinda dalla calibrazione di DLMAP e TAUMAP, per permettere di condurre i due processi di calibrazione con la massima libertà possibile; è chiaro che la procedura ottima prevede la calibrazione di LambdaCHAR e SIGMA in sequenza e la possibilità di derogare a questa regola, pur essendo concessa, rimane tuttavia di responsabilità del calibratore.

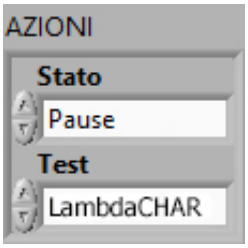


Figura 5.4: Comandi utente per le azioni

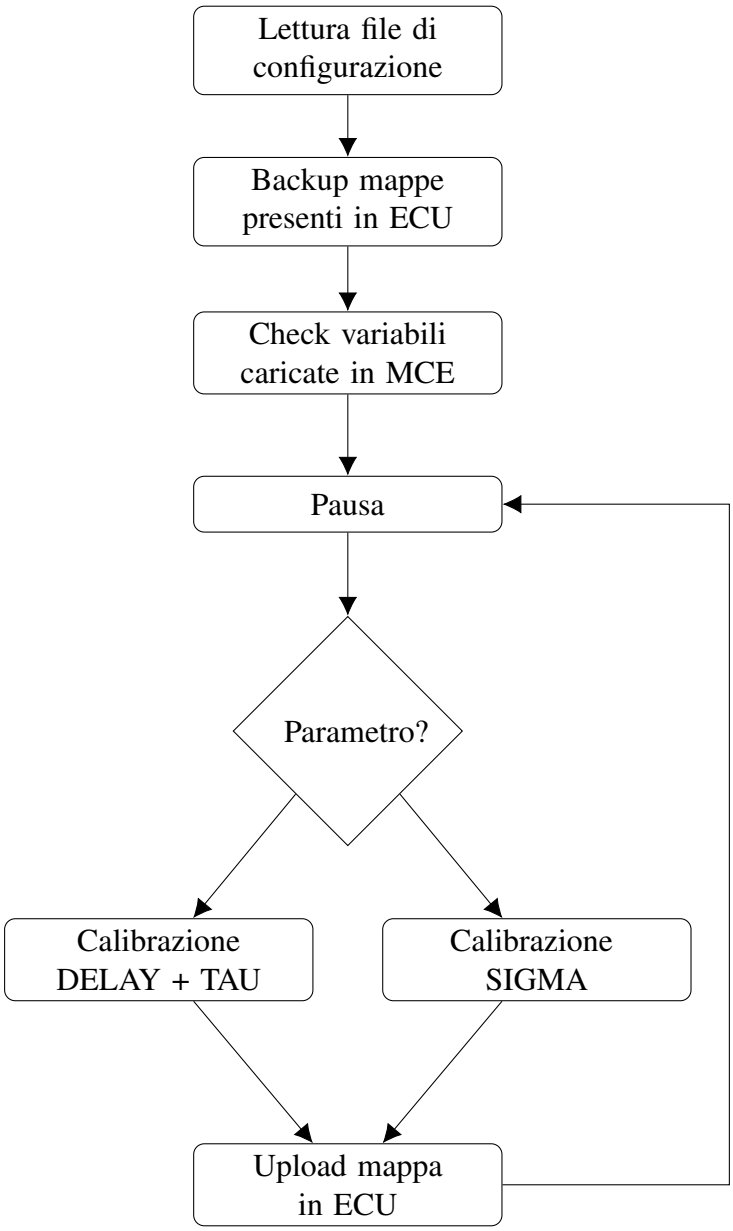


Figura 5.5: Sequenza delle operazioni eseguite dal tool

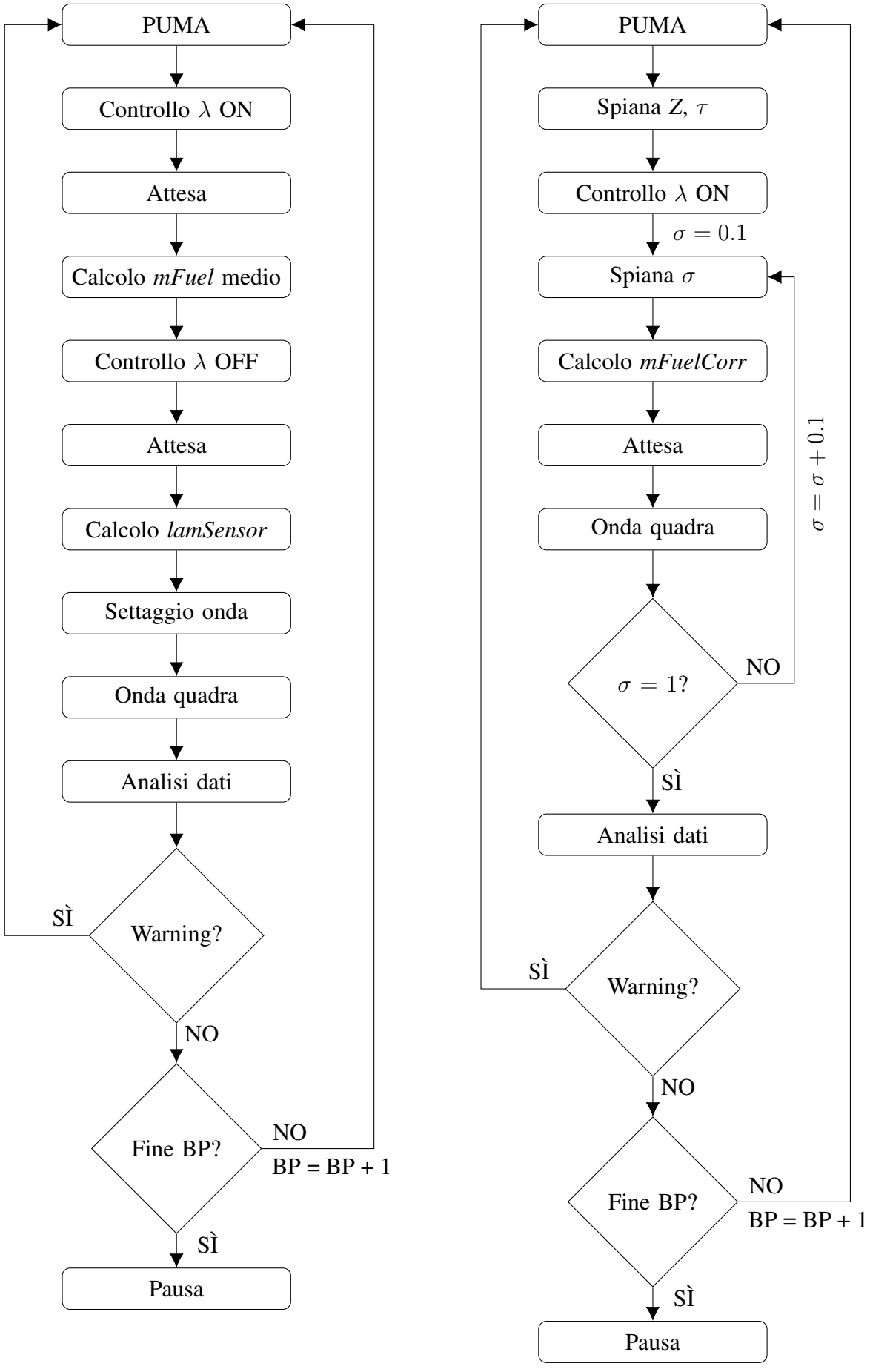


Figura 5.6: Sequenza delle operazioni di calibrazione

Una volta scelto il parametro da calibrare il tool si porta nello stato di 'Start Calibration' e segue due procedure diverse a seconda che si tratti di LambdaCHAR o di SIGMA, rappresentate rispettivamente a sinistra e a destra nel diagramma di flusso di figura 5.6; si dà per scontato che l'esecuzione dei vari step delle due procedure è completamente automatica, in quanto all'utente è richiesto semplicemente il comando di avvio della calibrazione mentre il controllo del tool gli viene restituito solo quando tutti i breakpoints sono stati esaminati e calibrati. In ogni caso, al termine di entrambe le procedure il tool si porta nello stato di pausa, e solo se tutte e tre le mappe sono state calibrate con esito positivo procede con l'upload delle stesse mappe in centralina. Vale la pena evidenziare che, così come accadeva con il tool VVT, anche in questo caso la ripetizione della prova relativa a un breakpoint dipende dalla presenza o meno di warning, e quindi dai valori restituiti dall'analisi dei dati; l'unica differenza è che la calibrazione di LambdaCTL non prevede in nessun caso l'esecuzione di prove di riferimento, pertanto tutto il codice necessario a gestire la validità dei valori di riferimento non è stato implementato. Si tralascia volutamente la descrizione dettaglia del codice LabVIEW, limitandosi a riportare nel seguito alcuni esempi notevoli e le eventuali differenze rispetto al tool VVT.

5.3.3 Il nuovo codice per il processo di calibrazione

In entrambe le procedure di calibrazione sono presenti uno o più stati di 'Attesa': già implementati nella prima versione del tool, sono stati mantenuti perché necessari al fine di consentire la regimazione del sistema in seguito alla variazione di uno dei parametri, generalmente la quantità di combustibile iniettato. Si ricorda infatti che, trovandosi la sonda lambda a monte del catalizzatore, è necessario attendere un certo tempo prima che essa sia in grado di rilevare la variazione imposta. Nella figura seguente è riportato il codice necessario a gestire l'attesa: si noterà che è stato utilizzato un contatore dedicato, che viene incrementato ogni iterazione del *timed loop* principale finché il tempo di attesa non risulta uguale alla variabile 'WaitTime'.

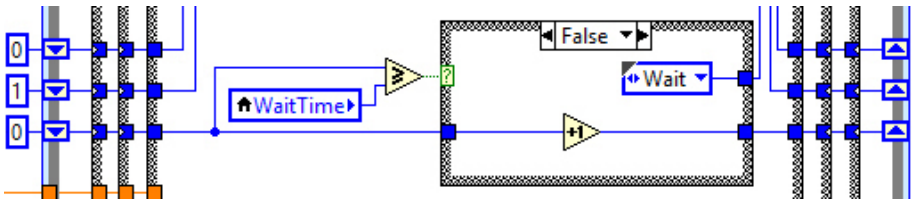


Figura 5.7: Codice per l'implementazione dell'attesa

Osservando il diagramma relativo a SIGMA e ricordando la procedura di calibrazione brevemente esposta nel paragrafo precedente, si noterà che la calibrazione di un breakpoint prevede di effettuare una spazzolata di valori di sigma compresi tra 0.1 e 1; è chiaro

che una simile richiesta è stata facilmente soddisfatta sfruttando la struttura basata su *timed loop* e macchine a stati, in modo analogo a quanto già visto nel capitolo relativo al tool VVT dove era necessario effettuare delle spazzolate di guadagni. Si è pertanto definito un nuovo contatore, dedicato esclusivamente ai valori di sigma, da incrementare di volta in volta tramite *shift register*; l'operazione di incremento è stata inserita nel caso 'SpianaSigma'. Solo quando sono stati esaminati tutti i valori della spazzolata il tool procede con l'analisi dei dati acquisiti per un determinato breakpoint.

Un'altra considerazione: i blocchi 'Onda quadra' si riferiscono all'eccitazione del sistema di controllo tramite l'esecuzione di transizioni in salita e in discesa di massa di combustibile iniettata, per LambdaCHAR, e di lambda target, per SIGMA; dato che i segnali così generati assumono la forma di un'onda quadra si è impiegato tale termine per riferirsi alla fase di eccitazione. Si tratta di un'operazione che può essere implementata in diversi modi, ma in ogni caso si è pensato di sfruttare le opportunità messe a disposizione dalle macchine a stati e di abbandonare il metodo seguito nel vecchio tool, che prevedeva un ciclo *for* con iterazioni opportunamente temporizzate. Il nuovo codice è rappresentato in figura 5.8: sono stati introdotti due *shift registers* dedicati 1 e 2, mentre lo *shift register* 3 è il contatore del tempo di registrazione. Il primo *shift register* viene inizializzato nello stato precedente con un array di due elementi, i quali sono nell'ordine il limite inferiore e superiore delle transizioni da effettuare, mentre il secondo *shift register* viene inizializzato con il limite inferiore, sempre nello stato precedente a 'OndaQuadra', e tale valore è effettivamente quello che assumerà il segnale in uscita.

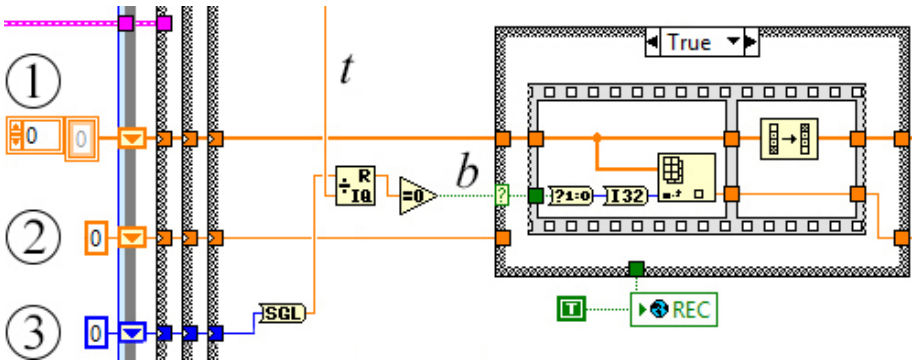


Figura 5.8: Codice per la generazione dell'onda quadra

Generare un segnale a forma d'onda quadra significa nella pratica variare a istanti predefiniti di tempo il valore di un parametro in centralina, e la logica impostata in questo senso è la seguente: nota la durata costante t di ogni transizione, ogni volta che il contatore del tempo raggiunge il valore t o un suo multiplo il parametro dovrà essere modificato, ovvero se prima tale parametro era impostato al limite inferiore dovrà essere sostituito con il limite superiore e viceversa. Per fare ciò si è pensato di invertire l'array di due

elementi trasportato dallo *shift register* 1 ogni volta che il booleano *b* risulta uguale a *true*, e poi impostare come valore dell'onda sempre l'elemento 0 di tale array; in questo modo ad ogni multiplo di *t* l'elemento 0 dell'array bidimensionale sarà alternativamente il limite inferiore o quello superiore, e di conseguenza tale sarà anche il valore dell'onda in output. È chiaro che, essendo la frequenza di aggiornamento del contatore del tempo pari alla frequenza del *timed loop* principale, quindi 100 Hz, le transizioni del segnale così ottenuto saranno estremamente regolari e precise.

Si vuole infine ricordare che tutti i codici per l'analisi dei dati sono stati mantenuti invariati: nonostante l'utilizzo del blocco 'MathScript Node' rallenti leggermente l'esecuzione del codice, tale rallentamento non è determinante o estremamente penalizzante, pertanto si è deciso che un'eventuale trascrizione degli script in linguaggio LabVIEW non avrebbe portato grandi vantaggi. Tuttavia è stato necessario riscrivere la parte di interpolazione dei risultati ottenuti con la spazzolata di sigma, dato che nel vecchio tool veniva eseguita nel codice MATLAB mentre per il nuovo tool, dovendo essere eseguita al termine della spazzolata nello stato 'EndCal', si è ritenuto opportuno utilizzare i blocchi messi a disposizione da LabVIEW, ottenendo il codice rappresentato in figura 5.9.

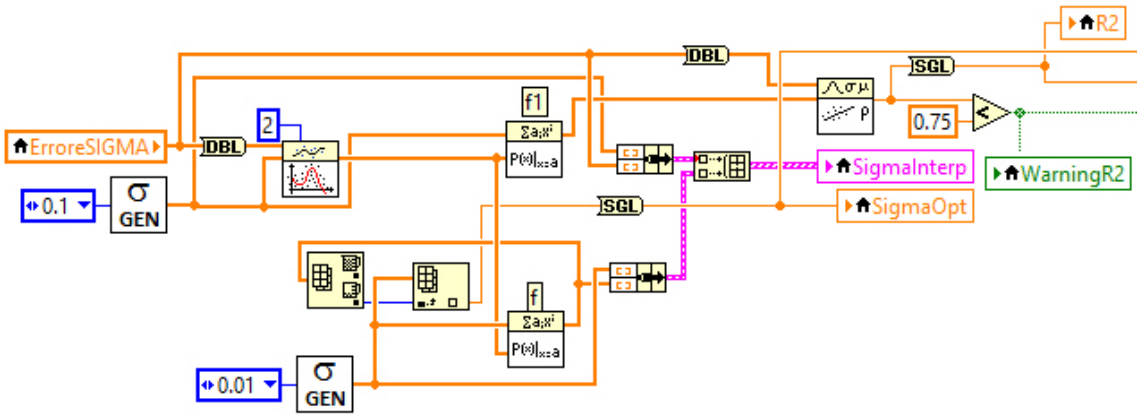


Figura 5.9: Codice per il calcolo della σ ottima

Si ricorda che il valore ottimo di sigma, da ricercare ovviamente all'interno di quelli esaminati nella spazzolata, è quello che permette di minimizzare la variabile *lamDelta*, corrispondente alla differenza tra *lamSensorSP*, valore di lambda target, e *lamSensor*, lambda misurato dalla sonda. Per ogni valore della spazzolata il codice MATLAB restituisce un valore di *lamDelta* che verrà opportunamente salvato nell'array locale 'ErroreSIGMA'; conclusa la spazzolata, i dieci valori contenuti in tale array verranno interpolati con una parabola tramite il blocco 'General Polynomial Fit', dopodiché ne verrà ricercato il minimo e infine individuato il valore di sigma corrispondente a tale minimo. Si fa notare che il blocco 'General Polynomial Fit' restituisce i coefficienti del polinomio interpolante, mentre i valori veri e propri che tale polinomio assume devono essere calcolati

passando in input al blocco ‘Polynomial Evaluation’ i valori della variabile indipendente, ovvero il vettore equispaziato dei valori della spazzolata di sigma; quest’ultimo vettore è stato ottenuto per mezzo del SubVI ‘SigmaGEN’ creato per l’occasione, che permette appunto di ottenere un vettore uniforme di valori compresi tra 0.1 e 1 con spaziatura di 0.1 oppure 0.01 a seconda delle necessità. In fase di test del tool è stato fatto un confronto tra i polinomi interpolanti ottenuti in questo modo e i polinomi ottenuti tramite lo script di MATLAB, ed effettivamente coincidevano; coincidono anche i valori di calibrazione ottenuti per sigma, pertanto si è deciso di mantenere questa parte di codice LabVIEW in sostituzione del codice MATLAB utilizzato nella vecchia versione del tool.

5.3.4 Le nuove funzioni implementate

In precedenza si è accennato al fatto che nel vecchio tool l’utente poteva scegliere il breakpoint da calibrare tramite due comandi appositi, e lo stesso passaggio da un breakpoint appena calibrato a quello successivo era sempre demandato all’utente. La nuova versione è completamente diversa sotto questo aspetto: si è infatti completamente automatizzato il passaggio da un breakpoint a quello successivo, durante il processo di calibrazione, mentre la scelta dei breakpoint da calibrare avviene nello stesso modo già implementato per il tool VVT, come si può vedere in figura 5.10.

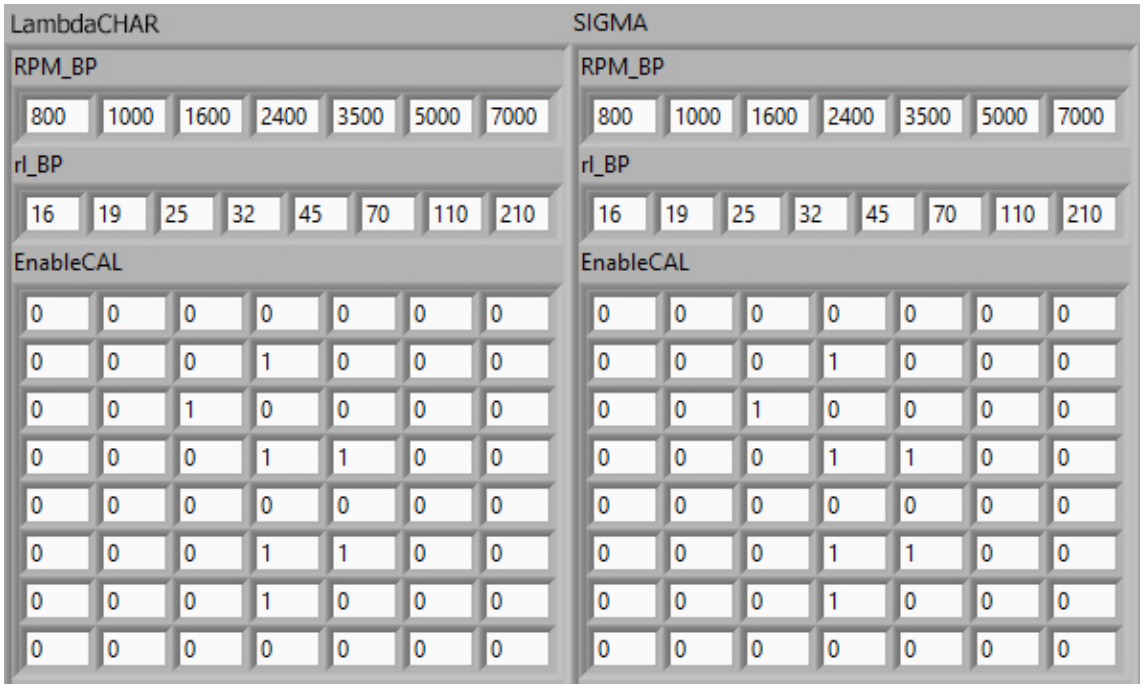


Figura 5.10: Controlli per la gestione dei breakpoint

In questo caso si hanno a disposizione due mappe ‘EnableCAL’ per l’attivazione o la disattivazione di un breakpoint, mentre i breakpoints stessi vengono inseriti tramite

due vettori appositi, ‘RPM_BP’ e ‘rl_BP’; si fa notare che in questo caso i breakpoints utente devono coincidere con quelli di mappa. Il codice che interpreta le due matrici è fondamentalmente identico a quello del tool VVT: i punti a cui corrisponde uno 0 nella matrice verranno saltati, quelli a cui corrisponde un 1 verranno esaminati. Rimane anche il vantaggio che l’utente può modificare tale matrice durante l’esecuzione del software, e le modifiche saranno considerate valide dal primo ciclo utile del *timed loop* principale.

La versione precedente del tool non prevedeva la generazione di un report al termine del processo di calibrazione, pertanto è stato necessario implementarla: si è trattato nella pratica di rivedere il SubVI ‘CreateTextReport’ realizzato per il tool VVT e adattarlo alle esigenze attuali. In figura 5.11 è riportato il block diagram del SubVI: si riconoscerà la struttura già descritta nel capitolo precedente, e pertanto non vale la pena soffermarcisi nuovamente. Si noterà comunque che la *case structure* per la gestione dei parametri presente nel tool VVT è stata sostituita da due *case structures* separate, e ovviamente i controlli per l’impostazione dei breakpoints di temperatura olio, tensione batteria e angolo VVT target sono stati sostituiti dall’unico controllo per l’impostazione del carico motore.

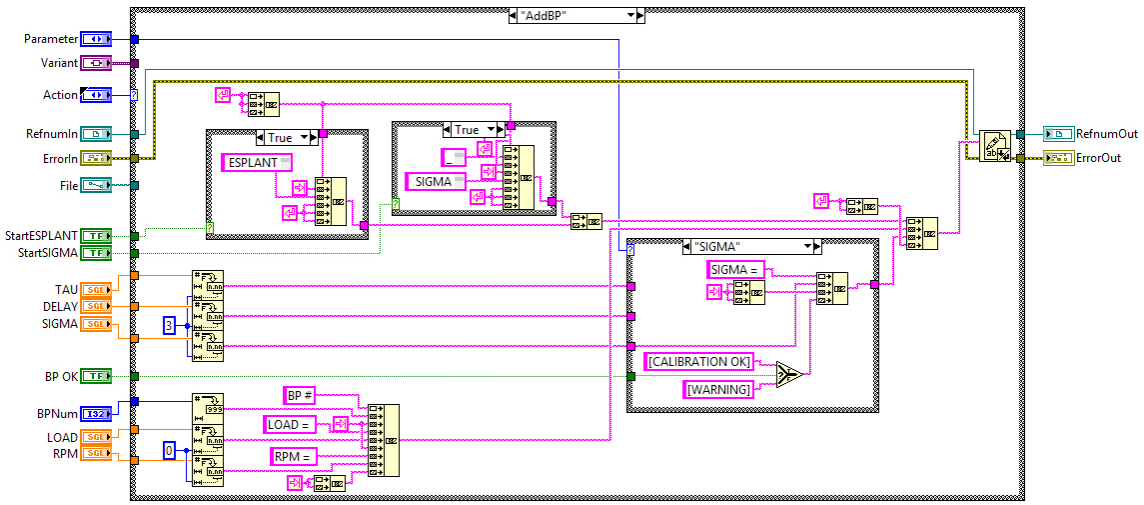


Figura 5.11: Codice per la generazione del report

Per il resto l’utilizzo del SubVI ‘CreateTextReport’ all’interno del VI principale è rimasto invariato: la creazione del report avviene nello stato ‘Download Configuration File’ della macchina a stati principale, il salvataggio delle mappe preesistenti avviene nello stato ‘Backup ECU Maps’, l’aggiunta dei vari breakpoints calibrati avviene nello stato ‘EndCal’ della macchina a stati 3 sia per LambdaCHAR che per SIGMA, mentre il confronto tra le mappe appena calibrate e quelle base avviene nello stato ‘Upload Calibration’ e così anche la chiusura e il salvataggio definitivo del report, in formato *xls* come per il tool VVT.

Discorso analogo vale per la gestione del file di configurazione: il vecchio tool leggeva automaticamente un file di tipo *xml* all'avvio per inizializzare i vari controlli del front panel, ma l'aggiornamento di tale file con i nuovi valori inseriti dall'utente rimaneva un'operazione manuale. Nel nuovo tool si è deciso di modificare la struttura del file di configurazione trasformandolo in un file di tipo *ini* e sfruttando il codice già messo a punto per il tool VVT. L'accortezza di raccogliere tutti i controlli da inizializzare nell'unico cluster 'ECU_char' ha permesso di mantenere invariato il loop per la lettura e il salvataggio del file di configurazione descritto nel capitolo precedente, dato che i blocchi 'Read Section Cluster' e 'Write Section Cluster' richiedono in input il solo cluster di controlli di cui leggere o salvare i valori, loop che pertanto non verrà descritto nuovamente.

5.4 L'interfacciamento con INCA

La versione precedente del tool era stata testata per il funzionamento al banco con esito positivo, e tuttavia richiedeva ancora un PC host per il controllo del software tramite LabVIEW; l'integrazione del tool in INCA, sebbene prevista nel progetto iniziale del tool, di fatto non è stata implementata per mancanza di tempo. Tale caratteristica è invece presente nella versione aggiornata: la comunicazione tra il tool e INCA nella pratica è stata realizzata tramite il VI 'XCP' già descritto nel capitolo 4. A differenza del tool VVT, il cui sviluppo è iniziato senza prevedere i due cluster di indicatori e controlli 'ECU_char' e 'ECU_meas', per quanto riguarda il tool LAMBDA si è pensato fin da subito di realizzare due interfacce distinte: una locale sul front panel, riservata ai programmatori e destinata al debug, e un'altra su una variabile globale, contenente i due cluster 'ECU_char' e 'ECU_meas' e destinata all'interfacciamento con INCA. Questo ha obbligato di fatto a duplicare alcuni controlli e alcuni indicatori, ma ha permesso anche un notevole risparmio di tempo dato che la lettura e la scrittura di 'ECU_char' e 'ECU_meas' è stata prevista fin dall'inizio insieme alla lettura e alla scrittura delle variabili locali.

Per quanto riguarda la generazione dei due file *a2l* e *hex* valgono le stesse considerazioni già riportate nel capitolo precedente per il tool VVT: è stato necessario seguire alcuni accorgimenti affinché i due file potessero essere generati senza problemi, ma l'esperienza acquisita durante lo sviluppo del tool VVT ha consentito di completare questa fase in modo veloce ed efficiente. I due file sono già stati descritti nel capitolo precedente pertanto può avere senso passare subito alla descrizione dell'implementazione del collegamento e dell'*experiment* appositamente creato.

5.4.1 L'implementazione dell'XCP

Esattamente come nel tool VVT, anche qui la comunicazione tra il tool e INCA avviene grazie ai due blocchi 'XCP' e 'iLinkRT Master', i quali richiedono di essere inizializzati. In figura 5.12 si può vedere la creazione e l'inizializzazione della coda per iLinkRT, tramite l'apposito blocco 'iLinkRT Queue', mentre nella figura 5.13 è riportato il codice per la creazione e il riempimento della coda per l'invio dei dati contenuti in 'ECU_meas'. Si noterà che in entrambi i casi è esattamente identico al codice già illustrato per il tool VVT.

Figura 5.12: Inizializzazione del collegamento via iLinkRT

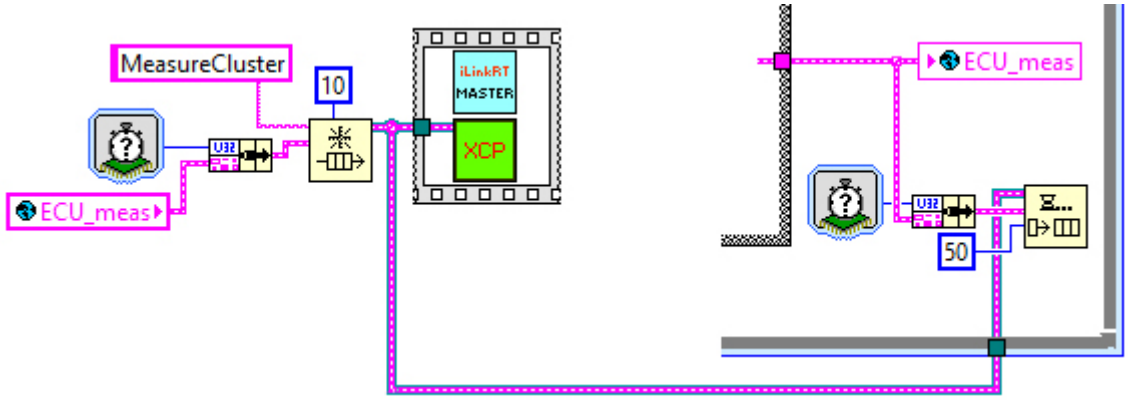


Figura 5.13: Inizializzazione e aggiornamento della coda dell'XCP

5.4.2 L'*experiment* per il controllo del tool

In figura 5.14 è rappresentato l'*experiment* messo a disposizione dell'utente per controllare il tool su INCA; come nell'*experiment* per il tool VVT anche in questo caso si è sfruttata la possibilità di creare delle pagine per suddividere i controlli e gli indicatori relativi LambdaCHAR dai controlli e gli indicatori relativi a SIGMA, mantenendo in ogni caso una serie di comandi comuni a entrambe le pagine, come si vedrà. A titolo d'esempio si descrivono le finestre principali disponibili nella pagina di LambdaCHAR:

1. *Azioni*: permette all'utente di controllare il processo di calibrazione (avvio, pausa, ripresa) e di impostare il parametro da calibrare. Questa finestra è presente in tutte le pagine dell'*experiment*;

2. *Parametri ECU*: consente all'utente di impostare i codeword necessari al processo di calibrazione da modificare in ECU. Questa finestra è presente in tutte le pagine dell'*experiment*; *[è consigliabile mantenere i valori di default caricati in fase di inizializzazione del tool]*

3. *Soglie PUMA e BP*: contiene dei controlli numerici che permettono all'utente di impostare la tolleranza di controllo sul setpoint, in funzione di regime di rotazione e carico motore, nonché il numero di tentativi da effettuare per ogni breakpoint in caso di warning prima di passare al breakpoint successivo. Questa finestra è presente in tutte le pagine dell'*experiment*;

4. *Parametri PID*: contiene dei controlli numerici che consentono all'utente di inserire i guadagni del PID per il settaggio dell'eccitazione;

5. *Parametri LambdaCHAR*: questa finestra consente all'utente di impostare i parametri tipici di LambdaCHAR per l'esecuzione delle varie prove, tra cui il numero dei gradini e l'ampiezza e l'offset dell'eccitazione;

6. *Stato Esecuzione*: contiene degli indicatori booleani che informano l'utente sullo stato di esecuzione del tool. Questa finestra è presente in tutte le pagine dell'*experiment*;

7. *PUMA*: informa l'utente sull'effettivo raggiungimento del breakpoint attuale, per il quale è visualizzato anche l'indice, da parte del banco. Questa finestra è presente in tutte le pagine dell'*experiment*;

8. *Stato Inizializzazione*: contiene degli indicatori booleani che informano sull'avvenuta e corretta inizializzazione del tool, comprendendo la lettura del file di configurazione, il backup delle mappe preesistenti in ECU e il check delle variabili caricate su MCE. Questa finestra è presente in tutte le pagine dell'*experiment*;

9. *Test in corso*: contiene quattro indicatori booleani che indicano quale parametro si sta calibrando e se la calibrazione è stata portata a termine con esito positivo. Questa finestra è presente in tutte le pagine dell'*experiment*;

10. *Stato LambdaCHAR [1]*: fornisce le informazioni restituite dai codici per l'analisi dei dati e i risultati intermedi ottenuti durante la calibrazione;

11. *Stato LambdaCHAR [2]*: fornisce informazioni sul breakpoint attuale, ovvero quante transizioni sono state completate, quante devono esserne eseguite in totale e quanti tentativi di calibrazione sono stati effettuati;

12. *BP RPM*, *BP LOAD* e *EnableCAL*: consentono all'utente di impostare i breakpoints da esaminare ed attivarne o disattivarne la calibrazione;

13. *Tempo di esecuzione*: contatore del tempo trascorso in secondi dall'avvio del tool; è stato introdotto per dare un feedback visivo all'utente che il tool è effettivamente in esecuzione.

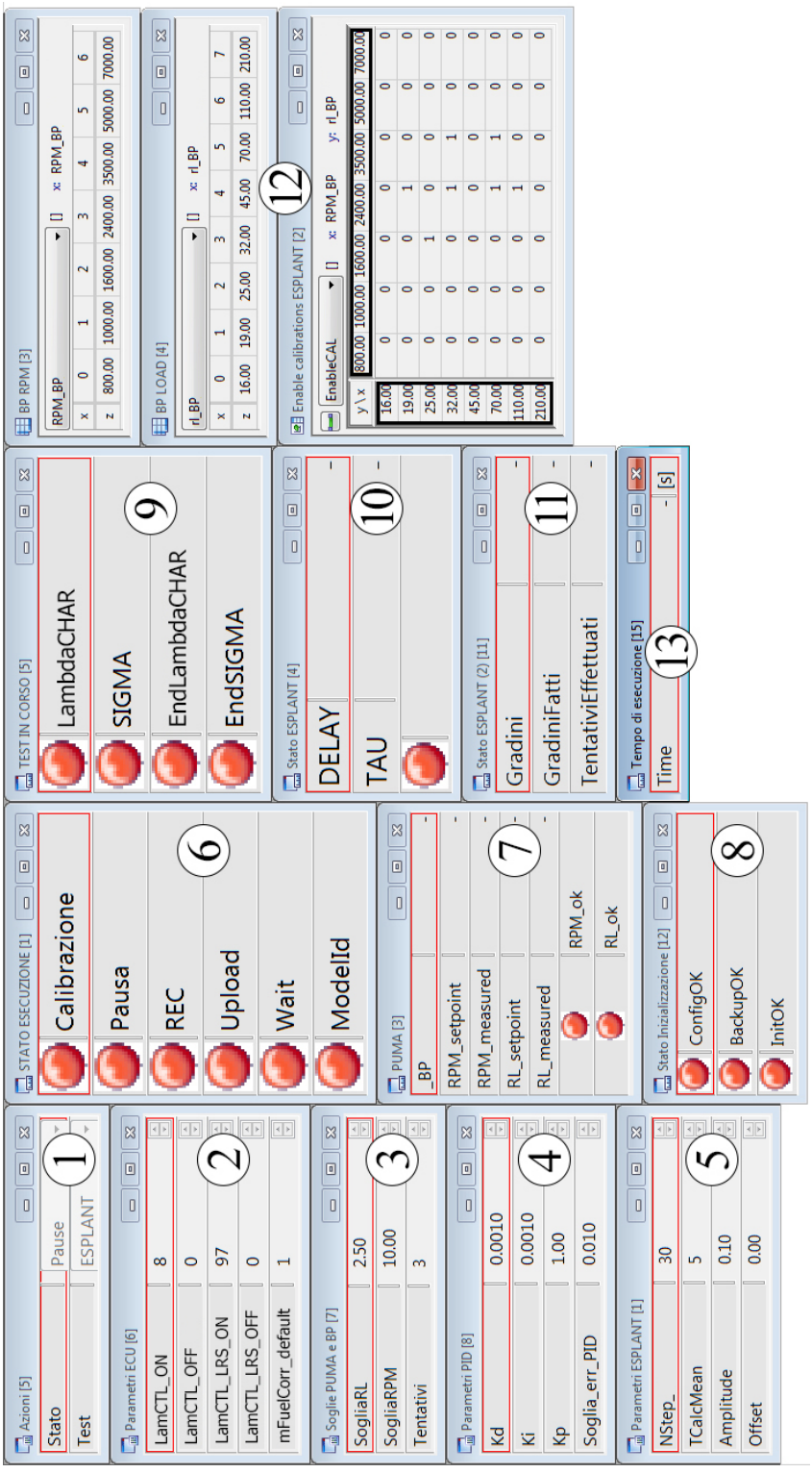


Figura 5.14: *Experiment* per il controllo del tool su INCA

Capitolo 6

Conclusioni e sviluppi futuri

L'attività di tesi descritta in questo elaborato ha rappresentato per l'autore un interessante percorso di crescita, sia a livello tecnico che a livello professionale; il software realizzato e gli obiettivi raggiunti possono essere ritenuti soddisfacenti, sebbene esistano ampi margini per il miglioramento e il potenziamento di entrambi i tool illustrati in precedenza. Nel seguito si cercherà di indicare brevemente le attività che devono essere ancora concluse e di trarre le conclusioni riguardo quelle già portate a termine.

Innanzitutto, per entrambi i tool deve essere implementata l'integrazione con il software di controllo del banco motore, ovvero PUMA: si tratta di un'attività già completata con esito positivo per quanto riguarda il tool per l'esecuzione dei piani quotati, e pertanto c'è motivo di credere che l'aggiunta di questa parte nel tool LAMBDA e nel tool VVT non comporterà grosse difficoltà, anche perché gli stati di settaggio del punto motore in entrambi i tool sono già stati impostati in modo da prevedere l'integrazione con PUMA. Il punto cruciale di questa attività sarà modificare in modo intelligente il codice già realizzato per il tool relativo ai piani quotati in modo che possa essere facilmente integrato negli altri tool, senza richiedere stravolgimenti del codice e senza ridurne l'efficienza.

Inoltre sia il tool VVT che il tool LAMBDA, nella versione aggiornata, devono ancora essere provati al banco: non tanto per quello che concerne la parte di analisi dati, per la quale, come si è detto, i codici sono stati già validati e perfezionati, quanto piuttosto per la parte di gestione dell'automazione, della lettura delle *measurements* e la scrittura delle *calibrations* in ECU e del salvataggio dei segnali acquisiti. In particolare, i codici che svolgono queste operazioni sono già stati testati singolarmente e con successo in sala prove o in locale, e tuttavia non è ancora stato possibile verificarne l'integrazione con i due tool.

Per quanto riguarda il tool VVT le considerazioni da fare sono diverse: innanzitutto si ricorderà che nell'introduzione del capitolo 3 si è detto che le funzioni deputate al controllo dei variatori sono due, VVT1 e VVT2, mentre nel seguito si è parlato solo della prima,

relativa al controllo dei variatori sull'asse a camme di scarico. Il tool è stato realizzato con lo scopo di calibrare questa funzione, mentre non è ancora stata prevista la possibilità di calibrare la funzione VVT2; è chiaro che si tratterà di introdurre nell'*experiment* un ulteriore *enum* in modo che l'utente possa scegliere quale delle due funzioni calibrare. A livello di codice le modifiche non saranno troppo pesanti, in quanto la procedura di calibrazione è esattamente la stessa, mentre cambieranno solo i nomi delle variabili da leggere e dei parametri da modificare in ECU.

Inoltre si sarà notato che, nel capitolo relativo alle procedure di calibrazione dei parametri di VVT1, nulla è stato detto in merito a ADVCOMP e RETCOMP, i due valori per la correzione del comportamento asimmetrico del sistema. In effetti la mancanza di tempo per effettuare le acquisizioni di prova al fine di mettere a punto i codici per l'analisi dei dati ha impedito di definire una vera e propria procedura di calibrazione; in tal senso il punto di partenza sarà necessariamente il lavoro già svolto dal relatore di questa tesi, come del resto è stato anche per tutti gli altri parametri. Bisogna tuttavia aggiungere che la parte di automazione che riguarda il settaggio del punto motore è già stata realizzata in LabVIEW, così come la macchina a stati destinata alla calibrazione, pertanto rimane da stabilire il procedimento di calibrazione ed effettuare alcune acquisizioni per poter definire l'algoritmo di analisi dati. Si vuole sottolineare che la scelta di trascurare ADVCOMP e RETCOMP è stata volutamente fatta in modo da privilegiare gli altri parametri da calibrare, sicuramente più importanti al fine del buon funzionamento del sistema di controllo.

Ancora, è necessario prevedere in entrambi i tool la gestione delle bancate e delle relative centraline: allo stato attuale sia il tool LAMBDA che il tool VVT sono in grado di lavorare su una sola centralina per volta, e qualora il motore da calibrare ne possieda due entrambi i tool non sono in grado di distinguere quale delle due centraline si sta calibrando, nonostante *measurements* e *calibrations* vengano acquisite da entrambi gli hardware (si ricordi il parametro 'DeviceID' di cui si è parlato nel capitolo introduttivo in merito al protocollo iLinkRT). Sarà pertanto necessario prevedere la possibilità da parte dell'utente di scegliere come gestire le due centraline, e di conseguenza le due bancate: si potrebbe ad esempio inserire un ulteriore *enum* in modo da poter decidere su quale delle due bancate lavorare, e poi copiare le calibrazioni così ottenute sulla centralina dell'altra bancata, oppure calibrare singolarmente le due centraline.

Ad ogni modo, al netto di quanto appena discusso, i risultati ottenuti e gli obiettivi raggiunti aprono a diverse prospettive future: le possibilità applicative messe a disposizione dall'hardware National Instruments e dai protocolli XCP e iLinkRT consentiranno in futuro di ampliare la struttura del TAC con nuovi tool per la calibrazione delle varie funzioni motore, permettendo così di ottenere un unico software eseguito sul PC realtime

di sala in grado di controllare in autonomia il funzionamento del banco e la calibrazione del motore, con grande efficienza, ottima precisione e notevole risparmio di tempo e denaro.

Appendice A

Codici MATLAB per l'analisi dei dati

A.1 VBATCOMP

```
close all
clear all
clc

path = uigetdir(default,'Seleziona la cartella con le acquisizioni...');

if path ~= 0
    files=dir(fullfile(path,'*.csv'));
    for i = 1:length(files)
        fid = fopen([path '\' files(i).name],'r');
        sizeS = [1,6];
        count = 1;
        clear DATA;
        while ~feof(fid)
            DATA(count,:) = fscanf(fid,'%f,%f,%f,%f,%f,%f\n',sizeS);
            count = count + 1;
        end
        vvtmeasMEAN(i) = mean(DATA(:,5));
        vvttargetMEAN(i) = mean(DATA(:,6));
        vbatMEAN(i) = mean(DATA(:,4));
        valvedcMEAN(i) = mean(DATA(:,2));
        errore(i) = abs(vvtmeasMEAN(i) - vvttargetMEAN(i));
        maxmin(:,i) = [max(DATA(:,6));min(DATA(:,6))];
        fclose(fid);
    end

    % Interpolazione
    x = [11.5,12.,13.,14.5];
    y = valvedcMEAN(1:4);
    p = polyfit(x,y,2);
    xI = 8:0.01:16;
    yI_1500 = polyval(p,xI);
    y = valvedcMEAN(5:8);
    p = polyfit(x,y,2);
```

```
35      yI_5000 = polyval(p,xI);
36      for k = 1:length(yI_1500)
37          yMEAN(k) = (yI_1500(k) + yI_5000(k))/2;
38      end
39      figure(1)
40      plot(xI,yI_1500,xI,yI_5000,'LineWidth',2);
41      xlabel('V_{bat} [V]')
42      ylabel('valvedc [%]')
43      grid on
44      hold on
45      plot(xI,yMEAN,'color','black','LineWidth',2);
46      legend('1500 rpm','5000 rpm','Curva media')
47      set(0,'DefaultTextInterpreter','Tex')
48
49      % Calibrazione
50      vNOM = 13;                  % TENSIONE NOMINALE SCELTA
51      vNOMbp = [8;12;13;14.5];
52      valvedcNOM = interp1(xI,yMEAN,vNOM,'spline')
53      VBATCOMP = yMEAN/valvedcNOM;
54      for i = 1:length(vNOMbp)
55          tNOMbp(i) = interp1(xI,VBATCOMP,vNOMbp(i),'spline');
56      end
57      VBATCOMP2 = 13./vNOMbp;
58      pp = polyfit(vNOMbp,VBATCOMP2,2);
59      VBATCOMP2_interp = polyval(pp,xI);
60
61      figure(2)
62      plot(xI,VBATCOMP,'LineWidth',2),hold on
63      plot(xI,VBATCOMP2_interp,'LineWidth',2)
64      plot(vNOMbp,tNOMbp,'b.','MarkerSize',30)
65      plot(vNOMbp,VBATCOMP2,'r.','MarkerSize',30)
66      xlabel('V_{bat} [V]')
67      ylabel('VBATCOMP')
68      legend('valvedc','Vbat')
69      grid on
70      set(0,'DefaultTextInterpreter','Tex')
71
72  end
```

A.2 PRECTL

```
close all
clear all
clc

path = uigetdir(default,'Seleziona la cartella con le acquisizioni...');

soglia = 0.1;
app = zeros(4000,10);

if path ~= 0
    files=dir(fullfile(path,'*.csv'));
    for i = 1:length(files)
        fid = fopen([path '\' files(i).name],'r');
        sizeS = [1,5];
        count = 1;
        clear DATA;
        while ~feof(fid)
            DATA(count,:) = fscanf(fid,'%f,%f,%f,%f,%f\n',sizeS);
            count = count + 1;
        end

        pid_ki = DATA(:,2);
        time = DATA(:,1);
        tOil = DATA(:,4);
        tOilSwmp = DATA(:,5);

        % Finestratura segnale (si considera solo la parte stabile)
        count = 0;
        cf = soglia + 1;
        while cf > soglia
            count = count + 1;
            maxT = max(pid_ki(count:end));
            minT = min(pid_ki(count:end));
            cf = maxT - minT;
        end

        % Calcolo PRECTL per la prova i-esima
        PRECTL(i) = mean(pid_ki(count:end));

        % Plot
        k = length(files);
        if k > 12
            aumenta = 2;
            if mod(k,4) == 0
                a = k/4; b = 2;
            elseif mod(k,3) == 0
                a = k/3; b = 3;
            end
        else
```

```
50              aumenta = 1;
51              if mod(k,2) == 0
52                  a = k/2; b = 2;
53              elseif mod(k,3) == 0
54                  a = k/3; b = 3;
55              end
56          end
57          set(0,'defaulttextinterpreter','none')
58          if (i > k/2 && aumenta ~= 1)
59              figure(aumenta + 1)
60              subplot(b,a,i-k/2),plot(time,pid_ki,time,tOil,'LineWidth',2)
61              hold on
62              plot(time,tOilSwmp,'-.','LineWidth',2)
63              %set(gca,'YLim',[40 120])
64              if i==7
65                  figure(1000),plot(time,pid_ki,time,tOil,'LineWidth',2)
66                  hold on
67                  plot(time,tOilSwmp,'-.','LineWidth',2)
68              end
69          else
70              figure(aumenta)
71              subplot(b,a,i),plot(time,pid_ki,time,tOil,'LineWidth',2)
72              hold on
73              plot(time,tOilSwmp,'-.','LineWidth',2)
74              %set(gca,'YLim',[40 120])
75              if i==7
76                  figure(1000),plot(time,pid_ki,time,tOil,'LineWidth',2)
77                  hold on
78                  plot(time,tOilSwmp,'-.','LineWidth',2)
79              end
80          end
81          grid on
82          %legend('pid_ki','tOil','tOilSwmp')
83          t = files(i).name;
84          stI = strfind(t,'POil');
85          t = strrep(t,'.csv','');
86          title(t(stI+5:end))
87          xlabel('Time [s]')
88          set(0,'defaulttextinterpreter','tex')
89          ylabel('pid_ki [%], T_{olio} [°C]')
90          figure(1000)
91          title(t(stI+5:end))
92          xlabel('Time [s]')
93          set(0,'defaulttextinterpreter','tex')
94          ylabel('pid_ki [%], T_{olio} [°C]')
95          legend('pid_ki','tOil','tOilSwmp')
96          fclose(fid);
97      end
98
99      % Valore da inserire in mappa: media di PRECTL calcolati per le varie
100     % prove; si esegue prima un controllo sul COV, calcolato a parità di
```

```
101     % regime, per escludere i set di prove troppo dispersi.
102     proveXrpm = 2;
103     aa = proveXrpm;
104     condition = 1;
105     while condition
106         for z = 1:proveXrpm
107             vec(z) = PRECTL(aa-proveXrpm+z);
108         end
109         COV(aa/proveXrpm) = std(vec)/mean(vec);
110         if aa+proveXrpm <= i
111             condition = 1;
112             aa = aa + proveXrpm;
113         elseif (aa+proveXrpm >= i) & (i-aa ~= 0)
114             condition = 0;
115             vec = PRECTL(aa+1:end);
116             COV(end+1) = std(vec)/mean(vec);
117         else
118             condition = 0;
119         end
120     end
121     sogliaCOV = 2.e-2;
122     PRECTL
123     disp(' Controllo delle prove...')
124     for bb = 1:length(COV)
125         if COV(bb) > sogliaCOV
126             proveEliminate(bb) = 1;
127             z = bb*proveXrpm-proveXrpm+1;
128             count = 1;
129             if z + bb*proveXrpm > length(PRECTL)
130                 PRECTL(z:end) = [];
131             else
132                 while count <= proveXrpm
133                     PRECTL(z) = [];
134                     count = count + 1;
135                 end
136             end
137         else
138             proveEliminate(bb) = 0;
139         end
140     end
141     COV
142     PRECTL
143
144     % Riconosco le prove eliminate
145     nEl = find(proveEliminate == 1);
146     idx = 1;
147     for j = 1:length(nEl)
148         startEl = nEl(j)*proveXrpm-proveXrpm+1;
149         endEl = startEl + proveXrpm - 1;
150         for kz = startEl:endEl
151             nomiProve{idx} = files(kz).name;
```

```
152             idx = idx + 1;
153         end
154     end
155 
156     PRECTL_cal = mean(PRECTL);
157     disp([' Il valore di PRECTL da calibrare è ' num2str(PRECTL_cal)])
158     disp(' ')
159     if isempty(nEl)
160         disp(' Tutte le prove effettuate risultano corrette.')
161     else
162         disp([' Sono state eliminate ' num2str(length(nomiProve)) ' prove:'])
163         disp(' ')
164         for v = 1:length(nomiProve)
165             disp(nomiProve{v})
166         end
167     end
168     disp(' ')
169 
170 end
```

```
close all
clear all
clc

load valUser

bpMAPPA = [-9.75,9.75,39.75,90,120];
bpMAPPA2 = [-30,20,95,110,120];
bpUSER = [40,90,105];
bpUSER2 = -10:0.01:120;

% Interpolazione
p1 = polyfit(bpUSER,valUSER,1);
p12 = polyfit(bpUSER,valUSER,2);
valUSER2 = polyval(p1,bpUSER2);
valMAPPA = polyval(p1,bpMAPPA);
valMAPPA2 = polyval(p1,bpMAPPA2);

% Plot
figure(1),plot(bpUSER2,valUSER2,'--','LineWidth',2)
grid on, hold on
bpSTORICO = bpMAPPA;
valSTORICO = [39.2,43.3,44.,52.,53.];
p2 = polyfit(bpSTORICO,valSTORICO,1);
valSTORICO2 = polyval(p2,bpUSER2);
figure(1),plot(bpUSER2,valSTORICO2,'--','LineWidth',2)
plot(bpUSER,valUSER,'b.','MarkerSize',30)
plot(bpSTORICO,valSTORICO,'r.','MarkerSize',30)
xlabel('T_{olio} [°C]'), ylabel('PRECTL')
legend('Test','Storico','BP Test','BP Storico','Location','NorthWest')

% Confronto R^2: polinomio interpolante di 1° e 2° grado
x1 = polyval(p1,bpUSER);
x2 = polyval(p12,bpUSER);
rsquare(valUSER',x1')
rsquare(valUSER',x2')
figure(2),plot(bpUSER2,valUSER2,'LineWidth',2)
hold on, grid on
valUSER22 = polyval(p12,bpUSER2);
plot(bpUSER2,valUSER22,'LineWidth',2)
plot(bpUSER,valUSER,'k.','MarkerSize',30)
xlabel('T_{olio} [°C]'), ylabel('PRECTL')
legend('1° grado','2° grado','BP Test','Location','NorthWest')
```

A.3 KPMAP

```
close all
clear all
clc

path = uigetdir(default,'Seleziona la cartella con le acquisizioni...');

sAmpLimite = 0.5;        % Soglia per valutazione oscillazioni

if path ~= 0
    files=dir(fullfile(path,'*.csv'));
    for i = 1:length(files)
        fid = fopen([path '\' files(i).name],'r');
        sizeS = [1,4]; count = 1;
        clear DATA;
        while ~feof(fid)
            DATA(count,:) = fscanf(fid,'%f,%f,%f,%f\n',sizeS);
            count = count + 1;
        end
        fclose(fid); clear time KPMAP vvt_target vvt_measured
        time_w = DATA(:,1);
        KPMAP = DATA(:,4);
        vvt_measured = DATA(:,3);
        vvt_target = DATA(:,2);
        figure(i),plot(time_w,vvt_measured),hold on
        plot(time_w,vvt_target,'LineWidth',2),plot(time_w,KPMAP,'LineWidth',3)
        xlabel('Time [s]')
        t = files(i).name;
        stI = strfind(t,'POil');
        t = strrep(t,'_JUMP.csv','');
        title(t(stI+5:end))
        set(gca,'XLim',[0 max(time_w)])
        legend('vvt_measured','vvt_target','KPMAP')

        % Riconosco i guadagni
        clear indiciCambioGuadagno indiciStartEndGuadagno
        indiciCambioGuadagno = find(diff(KPMAP) ~= 0);
        indiciStartEndGuadagno(:,1) = [1;indiciCambioGuadagno(1)];
        for k = 2:length(indiciCambioGuadagno)
            indiciStartEndGuadagno(:,k) = [indiciCambioGuadagno(k-1)+1;...
                                           indiciCambioGuadagno(k)];
        end
        indiciStartEndGuadagno(:,k+1) = [indiciCambioGuadagno(k)+1;...
                                         length(vvt_target)];
        for m = 1:length(indiciStartEndGuadagno)
            KPMAP_user(m) = KPMAP(indiciStartEndGuadagno(2,m)-5);
        end

        for j = 1:length(indiciStartEndGuadagno)

```

```
% Finestratura del segnale per ogni guadagno
clear vvt_target vvt_measured
vvt_target = vvt_target(indiciStartEndGuadagno(1,j)...
                :indiciStartEndGuadagno(2,j));
vvt_measured = vvt_measured(indiciStartEndGuadagno(1,j)...
                :indiciStartEndGuadagno(2,j));
time = time_w(indiciStartEndGuadagno(1,j)...
             :indiciStartEndGuadagno(2,j)) ...
             - time_w(indiciStartEndGuadagno(1,j));

maxTarget = max(vvt_target); minTarget = min(vvt_target);
jump = maxTarget - minTarget;

% Trovo gli indici di inizio per i vari jumps
tollJump = jump/4;
iStartUP = find(diff(vvt_target) > jump-tollJump...
              & diff(vvt_target) < jump+tollJump);
iStartDOWN = find(diff(vvt_target) < -jump+tollJump...
              & diff(vvt_target) > -jump-tollJump);

% Considero tutti i jumps
clear jumpsUP jumpsDOWN
if iStartUP(1) < iStartDOWN(1)
    for abc = 1:min([length(iStartUP),length(iStartDOWN)]) - 1
        jumpsUP(:,abc) = [iStartUP(abc);iStartDOWN(abc)];
        jumpsDOWN(:,abc) = [iStartDOWN(abc);iStartUP(abc+1)];
    end
else
    for abc = 1:min([length(iStartUP),length(iStartDOWN)]) - 1
        jumpsDOWN(:,abc) = [iStartDOWN(abc);iStartUP(abc)];
        jumpsUP(:,abc) = [iStartUP(abc);iStartDOWN(abc+1)];
    end
end

% Sistemo l'ultimo jump
if iStartUP(end) > iStartDOWN(end)
    jumpsDOWN(:,abc+1) = [iStartDOWN(end);iStartUP(end)];
else
    jumpsUP(:,abc+1) = [iStartUP(end);iStartDOWN(end)];
end

% Verifico che il numero dei jumps sia uguale in UP e in DOWN
if length(jumpsUP(1,:)) > length(jumpsDOWN(1,:))
    jumpsUP = jumpsUP(:,1:length(jumpsDOWN(1,:)));
else
    jumpsDOWN = jumpsDOWN(:,1:length(jumpsUP(1,:)));
end

% Calcolo FFT jumps up
oscUp = 0;
for aa = 1:length(jumpsUP(1,:))
```

```
101                 clear ww tt
102                 ww = vvt_measured(jumpsUP(1,aa):jumpsUP(2,aa));
103                 ww = ww(50:250);    % Finestratura manuale
104                 ww = ww - mean(ww);
105                 tt = time(jumpsUP(1,aa):jumpsUP(2,aa))...
106                    - time(jumpsUP(1,aa));
107                 tt = tt(1:201);     % Finestratura manuale
108                 tTot1 = max(tt);
109                 Fb1 = 1/tTot1;
110                 fft1 = abs(fft(ww))*2/length(ww);
111                 freq1 = (0:round((length(fft1)-1)/2))*Fb1;
112                 semiAmpiezze1 = fft1(1:length(freq1));
113                 % Considero le frequenze comprese tra 3 e 10 Hz per la
114                 % valutazione delle oscillazioni
115                 fstart = find(freq1 >= 2.5,1,'first');
116                 fend = find(freq1 <= 9.5,1,'last');
117                 semiAmpiezze1 = semiAmpiezze1(fstart:fend);
118                 freq1 = freq1(fstart:fend);
119                 if max(semiAmpiezze1) > sAmpLimite
120                     oscUp = oscUp + 1;
121                 end
122             end
123             oscillazioniUp(j) = oscUp/2;
124
125             % Calcolo FFT jumps down
126             oscDown = 0;
127             for aa = 1:length(jumpsDOWN(1,:))
128                 clear ww tt
129                 ww = vvt_measured(jumpsDOWN(1,aa):jumpsDOWN(2,aa));
130                 ww = ww(50:250);    % Finestratura manuale
131                 ww = ww - mean(ww);
132                 tt = time(jumpsDOWN(1,aa):jumpsDOWN(2,aa))...
133                    - time(jumpsDOWN(1,aa));
134                 tt = tt(1:201);     % Finestratura manuale
135                 tTot1 = max(tt);
136                 Fb1 = 1/tTot1;
137                 fft1 = abs(fft(ww))*2/length(ww);
138                 freq1 = (0:round((length(fft1)-1)/2))*Fb1;
139                 semiAmpiezze1 = fft1(1:length(freq1));
140                 % Considero le frequenze comprese tra 3 e 10 Hz per la
141                 % valutazione delle oscillazioni
142                 fstart = find(freq1 >= 2.5,1,'first');
143                 fend = find(freq1 <= 9.5,1,'last');
144                 semiAmpiezze1 = semiAmpiezze1(fstart:fend);
145                 freq1 = freq1(fstart:fend);
146                 if max(semiAmpiezze1) > sAmpLimite
147                     oscDown = oscDown + 1;
148                 end
149             end
150             oscillazioniDown(j) = oscDown/2;
151
```

```
152         end
153         upCritico = find(oscillazioniUp >= 0.5,1,'first');
154         if isempty(upCritico)
155             upCritico = NaN;
156         end
157         downCritico = find(oscillazioniDown >= 0.5,1,'first');
158         if isempty(downCritico)
159             downCritico = NaN;
160         end
161         if isnan(upCritico) & isnan(downCritico) % No oscillazioni
162             KPMAP_cal(i) = KPMAP_user(end)/2;
163         elseif isnan(upCritico) % No oscillazioni in up
164             KPMAP_cal(i) = KPMAP_user(downCritico)/2;
165         elseif isnan(downCritico) % No oscillazioni in down
166             KPMAP_cal(i) = KPMAP_user(upCritico)/2;
167         else % Oscillazioni sia in up che in down
168             KPMAP_cal(i) = min([KPMAP_user(upCritico);...
169                                 KPMAP_user(downCritico)])/2;
170         end
171
172     end
173
174 end
```

A.4 KDMAP

```
close all
clear all
clc

%% ANALISI PROVE CON DERIVATIVO NON NULLO

path = uigetdir(default,'Seleziona la cartella con le acquisizioni...');

if path ~= 0
    files=dir(fullfile(path,'*.csv'));
    for i = 1:length(files)
        fid = fopen([path '\' files(i).name],'r');
        sizeS = [1,4]; count = 1; clear DATA;
        while ~feof(fid)
            DATA(count,:) = fscanf(fid,'%f,%f,%f,%f\n',sizeS);
            count = count + 1;
        end
        fclose(fid);
        clear time KDMAP vvt_target vvt_measured
        time_w = DATA(:,1);
        KDMAP = DATA(:,4);
        vvt_measured = DATA(:,3);
        vvt_target = DATA(:,2);
        figure(i),plot(time_w,vvt_measured),hold on
        plot(time_w,vvt_target,'LineWidth',2)
        plot(time_w,KDMAP,'k','LineWidth',3)
        xlabel('Time [s]')
        t = files(i).name;
        stI = strfind(t,'POil');
        t = strrep(t,'_JUMP.csv','');
        title(t(stI-2:end))
        legend('vvt_measured','vvt_target','KDMAP')

        % Riconosco i guadagni
        clear indiciCambioGuadagno indiciStartEndGuadagno
        indiciCambioGuadagno = find(diff(KDMAP) ~= 0);
        indiciStartEndGuadagno(:,1) = [1;indiciCambioGuadagno(1)];
        for k = 2:length(indiciCambioGuadagno)
            indiciStartEndGuadagno(:,k) = [indiciCambioGuadagno(k-1)+1;...
                                           indiciCambioGuadagno(k)];
        end
        indiciStartEndGuadagno(:,k+1) = [indiciCambioGuadagno(k)+1;...
                                         length(KDMAP)];
        for m = 1:length(indiciStartEndGuadagno)
            KDMAP_user(m,i) = KDMAP(indiciStartEndGuadagno(2,m)-5);
        end

        % Inizializzazione vettori
        OS.up = zeros(1,length(indiciStartEndGuadagno));
```

```
50          OS.down = zeros(1,length(indiciStartEndGuadagno));
51          T099.up = zeros(1,length(indiciStartEndGuadagno));
52          T099.down = zeros(1,length(indiciStartEndGuadagno));
53
54          for j = 1:length(indiciStartEndGuadagno)
55
56              % Finestratura del segnale per ogni guadagno fornito
57              clear vvt_target vvt_measured
58              vvt_target = vvt_target(indiciStartEndGuadagno(1,j):...
59                             indiciStartEndGuadagno(2,j));
60              vvt_measured = vvt_measured(indiciStartEndGuadagno(1,j):...
61                               indiciStartEndGuadagno(2,j));
62              time = time_w(indiciStartEndGuadagno(1,j):...
63                           indiciStartEndGuadagno(2,j))...
64                           - time_w(indiciStartEndGuadagno(1,j));
65
66              maxTarget = max(vvt_target); minTarget = min(vvt_target);
67              jump = maxTarget - minTarget;
68
69              % Trovo gli indici di inizio per i vari jumps in UP e DOWN
70              tollJump = jump/4;
71              iStartUP = find(diff(vvt_target) > jump-tollJump ...
72                           & diff(vvt_target) < jump+tollJump);
73              iStartDOWN = find(diff(vvt_target) < -jump+tollJump ...
74                             & diff(vvt_target) > -jump-tollJump);
75
76              % Considero tutti i jumps tranne il primo e l'ultimo
77              clear jumpsUP jumpsDOWN
78              if iStartUP(1) < iStartDOWN(1)
79                  for abc = 1:min([length(iStartUP),length(iStartDOWN)]) - 1
80                      jumpsUP(:,abc) = [iStartUP(abc);iStartDOWN(abc)];
81                      jumpsDOWN(:,abc) = [iStartDOWN(abc);iStartUP(abc+1)];
82                  end
83              else
84                  for abc = 1:min([length(iStartUP),length(iStartDOWN)]) - 1
85                      jumpsDOWN(:,abc) = [iStartDOWN(abc);iStartUP(abc)];
86                      jumpsUP(:,abc) = [iStartUP(abc);iStartDOWN(abc+1)];
87                  end
88              end
89
90              % Sistemo l'ultimo jump
91              if iStartUP(end) > iStartDOWN(end)
92                  jumpsDOWN(:,abc+1) = [iStartDOWN(end);iStartUP(end)];
93              else
94                  jumpsUP(:,abc+1) = [iStartUP(end);iStartDOWN(end)];
95              end
96
97              % Escludo il primo e l'ultimo dei jump non completi
98              if iStartUP(1) < iStartDOWN(1)
99                  jumpsUP(:,1) = [];
100             else
```

```
101                 jumpsDOWN(:,1) = [];
102             end
103
104             % Verifico che il numero dei jumps sia uguale in UP e in DOWN
105             if length(jumpsUP(1,:)) > length(jumpsDOWN(1,:))
106                 jumpsUP = jumpsUP(:,1:length(jumpsDOWN(1,:)));
107             else
108                 jumpsDOWN = jumpsDOWN(:,1:length(jumpsUP(1,:)));
109             end
110
111             jumpsUP = jumpsUP + 1; jumpsDOWN = jumpsDOWN + 1;
112
113             % Calcolo l'OVERSHOOT per i jumps up
114             clear OS_up
115             for n = 1:length(jumpsUP(1,:))
116                 OS_up(n) = max(vvt_measured(jumpsUP(1,n):jumpsUP(2,n))) ...
117                           - maxTarget;
118                 if OS_up(n) < 0
119                    OS_up(n) = 0.;
120                 end
121             end
122             OS_up_medio = mean(OS_up);
123
124             % Calcolo l'OVERSHOOT per i jumps down
125             clear OS_down
126             for n = 1:length(jumpsDOWN(1,:))
127                 OS_down(n) = minTarget - min(vvt_measured(jumpsDOWN(1,n):...
128                                                     jumpsDOWN(2,n)));
129                 if OS_down(n) < 0
130                    OS_down(n) = 0.;
131                 end
132             end
133             OS_down_medio = mean(OS_down);
134
135             % OS in output
136             OS.up(j)= OS_up_medio; OS.down(j)= OS_down_medio;
137
138             % Calcolo T0-99 per i jumps up
139             clear T099_up
140             for n = 1:length(jumpsUP(1,:))
141                  i0 = find(vvt_measured(jumpsUP(1,n):jumpsUP(2,n)) > minTarget
142                                + 1.e-6*(maxTarget-minTarget),1,'first');
143                  i99 = find(vvt_measured(jumpsUP(1,n):jumpsUP(2,n)) > minTarget
144                                + 0.99*(maxTarget-minTarget),1,'first');
145                 if (~isempty(i0)) & (~isempty(i99))
146                      T099_up(n) = time(i99) - time(i0);
147                 else
148                      T099_up(n) = 0;
149                 end
150             end
151             T099_up_medio = mean(T099_up);
```

```
152
153             % Calcolo T0-99 per i jumps down
154             clear T099_down
155             for n = 1:length(jumpsDOWN(1,:))
156                  i0 = find(vvt_measured(jumpsDOWN(1,n):jumpsDOWN(2,n)) < ...
157                      maxTarget - 1.e-6*(maxTarget-minTarget),1,'first');
158                  i99 = find(vvt_measured(jumpsDOWN(1,n):jumpsDOWN(2,n)) < ...
159                      maxTarget - 0.99*(maxTarget-minTarget),1,'first');
160                 if (~isempty(i0)) & (~isempty(i99))
161                   T099_down(n) = time(i99) - time(i0);
162                 else
163                  T099_down(n) = 0;
164                 end
165             end
166             T099_down_medio = mean(T099_down);
167
168             % T0-99 in output
169             T099.up(j)= T099_up_medio; T099.down(j) = T099_down_medio;
170
171         end
172         set.OS{i} = OS; set.T0_99{i} = T099;
173
174         % Interpolazione parametri al variare del guadagno
175         deg = 1;        % grado di interpolazione
176         % OS
177         xGain = min(KDMAP_user(:,i)):0.01:max(KDMAP_user(:,i));
178         p = polyfit(KDMAP_user(:,i)',OS.up,deg);
179         OS_interp.up = polyval(p,xGain);
180         set.OS_interp{i}.up = OS_interp.up;
181         p = polyfit(KDMAP_user(:,i)',OS.down,deg);
182         OS_interp.down = polyval(p,xGain);
183         set.OS_interp{i}.down = OS_interp.down;
184         % T099
185         p = polyfit(KDMAP_user(:,i)',T099.up,deg);
186         T099_interp.up = polyval(p,xGain);
187         set.T099_interp{i}.up = T099_interp.up;
188         p = polyfit(KDMAP_user(:,i)',T099.down,deg);
189         T099_interp.down = polyval(p,xGain);
190         set.T099_interp{i}.down = T099_interp.down;
191
192         % Plot di T099 e OS
193         figure(98)
194         subplot(2,1,1),plot(KDMAP_user(:,i),OS.up,'LineWidth',2,...
195             'color',colore(i)),hold on
196         plot(xGain,OS_interp.up,'--','color',colore(i))
197         title('OS up')
198         grid on
199         subplot(2,1,2),plot(KDMAP_user(:,i),OS.down,'LineWidth',2,...
200             'color',colore(i)),hold on
201         plot(xGain,OS_interp.down,'--','color',colore(i))
202         xlabel('KDMAP')
```

```
203         grid on
204         title('OS down')
205         figure(99)
206         subplot(2,1,1),plot(KDMAP_user(:,i),T099.up,'LineWidth',2,...
207             'color',colore(i)),hold on
208         plot(xGain,T099_interp.up,'--','color',colore(i))
209         title('T099 up')
210         grid on
211         subplot(2,1,2),plot(KDMAP_user(:,i),T099.down,'LineWidth',2,...
212             'color',colore(i)),hold on
213         plot(xGain,T099_interp.down,'--','color',colore(i))
214         xlabel('KDMAP')
215         title('T099 down')
216         grid on
217      end
218 else
219     return
220 end
221
222 %% ANALISI PROVE DI RIFERIMENTO
223
224 clearvars -except set KDMAP_user
225 path = uigetdir(default,'Seleziona la cartella con le acquisizioni...');
226
227 if path ~= 0
228     files = dir(fullfile(path,'*.csv'));
229     for i = 1:length(files)
230         fid = fopen([path '\' files(i).name],'r');
231         sizeS = [1,4]; count = 1; clear DATA;
232         while ~feof(fid)
233             DATA(count,:) = fscanf(fid,'%f,%f,%f,%f\n',sizeS);
234             count = count + 1;
235         end
236         fclose(fid); clear time KDMAP vvt_target vvt_measured
237         time = DATA(:,1);
238         KDMAP = DATA(:,4);
239         vvt_measured = DATA(:,3);
240         vvt_target = DATA(:,2);
241         figure(10*i),plot(time,vvt_measured),hold on
242         plot(time,vvt_target,'LineWidth',2)
243         plot(time,KDMAP,'k','LineWidth',3)
244         xlabel('Time [s]')
245         t = files(i).name;
246         stI = strfind(t,'POil');
247         t = strrep(t,'_JUMP_RIF.csv','_RIF');
248         title(t(stI-2:end))
249         legend('vvt_measured','vvt_target','KDMAP')
250
251         % Finestratura dei vari jumps
252         maxTarget = max(vvt_target); minTarget = min(vvt_target);
253         jump = maxTarget - minTarget;
```

```
254
255         % Trovo gli indici di inizio per i vari jumps in UP e DOWN
256         tollJump = jump/4;
257         clear iStartUP iStartDOWN
258         iStartUP = find(diff(vvt_target) > jump-tollJump ...
259                       & diff(vvt_target) < jump+tollJump);
260         iStartDOWN = find(diff(vvt_target) < -jump+tollJump ...
261                         & diff(vvt_target) > -jump-tollJump);
262
263         % Considero tutti i jumps tranne il primo e l'ultimo
264         clear jumpsUP jumpsDOWN
265         if iStartUP(1) < iStartDOWN(1)
266             for abc = 1:min([length(iStartUP),length(iStartDOWN)]) - 1
267                 jumpsUP(:,abc) = [iStartUP(abc);iStartDOWN(abc)];
268                 jumpsDOWN(:,abc) = [iStartDOWN(abc);iStartUP(abc+1)];
269             end
270         else
271             for abc = 1:min([length(iStartUP),length(iStartDOWN)]) - 1
272                 jumpsDOWN(:,abc) = [iStartDOWN(abc);iStartUP(abc)];
273                 jumpsUP(:,abc) = [iStartUP(abc);iStartDOWN(abc+1)];
274             end
275         end
276
277         % Sistemo l'ultimo jump
278         if iStartUP(end) > iStartDOWN(end)
279             jumpsDOWN(:,abc+1) = [iStartDOWN(end);iStartUP(end)];
280         else
281             jumpsUP(:,abc+1) = [iStartUP(end);iStartDOWN(end)];
282         end
283
284         % Escludo il primo e l'ultimo dei jump non completi
285         if iStartUP(1) < iStartDOWN(1)
286             jumpsUP(:,1) = [];
287         else
288             jumpsDOWN(:,1) = [];
289         end
290
291         % Verifico che il numero dei jumps sia uguale in UP e in DOWN
292         if length(jumpsUP) > length(jumpsDOWN)
293             jumpsUP = jumpsUP(:,1:length(jumpsDOWN));
294         else
295             jumpsDOWN = jumpsDOWN(:,1:length(jumpsUP));
296         end
297
298         jumpsUP = jumpsUP + 1; jumpsDOWN = jumpsDOWN + 1;
299
300         % Calcolo l'OVERSHOOT per i jumps up
301         clear OS_up
302         for n = 1:length(jumpsUP)
303             OS_up(n) = max(vvt_measured(jumpsUP(1,n):jumpsUP(2,n))) - maxTarget;
304             if OS_up(n) < 0
```

```
305                 OS_up(n) = 0.;
306             end
307         end
308
309         % Calcolo di media, mediana, dev standard
310         set.OS_ref_tutti{i}.media.up = mean(OS_up);
311         set.OS_ref_tutti{i}.mediana.up = median(OS_up);
312         set.OS_ref_tutti{i}.devstd.up = std(OS_up);
313
314         % Calcolo l'OVERSHOOT per i jumps down
315         clear OS_down
316         for n = 1:length(jumpsDOWN)
317             OS_down(n) = minTarget - min(vvt_measured(jumpsDOWN(1,n):...
318                                                 jumpsDOWN(2,n)));
319             if OS_down(n) < 0
320                 OS_down(n) = 0.;
321             end
322         end
323
324         % Calcolo di media, mediana, dev standard
325         set.OS_ref_tutti{i}.media.down = mean(OS_down);
326         set.OS_ref_tutti{i}.mediana.down = median(OS_down);
327         set.OS_ref_tutti{i}.devstd.down = std(OS_down);
328
329         % Eliminazione prove errate (valore lontano da mediana)
330         medianaUP = set.OS_ref_tutti{i}.mediana.up;
331         medianaDOWN = set.OS_ref_tutti{i}.mediana.down;
332         hh = 1;
333         while hh <= length(OS_up);
334             checkUP = (OS_up(hh) > medianaUP*0.9 ...
335                       & OS_up(hh) < medianaUP*1.1);
336             if ~checkUP
337                 OS_up(hh) = [];
338             else
339                 hh = hh+1;
340             end
341         end
342
343         hh = 1;
344         while hh <= length(OS_down);
345             checkDOWN = (OS_down(hh) > medianaDOWN*0.9 ...
346                         & OS_down(hh) < medianaDOWN*1.1);
347             if ~checkDOWN
348                 OS_down(hh) = [];
349             else
350                 hh = hh+1;
351             end
352         end
353
354         % OS in output
355         set.OS_ref_tutti{i}.up = OS_up; set.OS_ref_tutti{i}.down = OS_down;
```

```
356         OS.up = mean(OS_up); OS.down = mean(OS_down);
357
358
359         % Calcolo T0-99 per i jumps up
360         clear T099_up
361         for n = 1:length(jumpsUP)
362              i0 = find(vvt_measured(jumpsUP(1,n):jumpsUP(2,n)) ...
363                  > minTarget + 1.e-6*(maxTarget-minTarget),1,'first');
364              i99 = find(vvt_measured(jumpsUP(1,n):jumpsUP(2,n)) ...
365                  > minTarget + 0.99*(maxTarget-minTarget),1,'first');
366             if (~isempty(i0)) & (~isempty(i99))
367                  T099_up(n) = time(i99) - time(i0);
368             else
369                  T099_up(n) = 0;
370             end
371         end
372
373         % Calcolo di media, mediana, dev standard
374         set.T099_ref_tutti{i}.media.up = mean(T099_up);
375         set.T099_ref_tutti{i}.mediana.up = median(T099_up);
376         set.T099_ref_tutti{i}.devstd.up = std(T099_up);
377         set.T099_ref_tutti{i}.data.up = T099_up;
378
379         % Calcolo T0-99 per i jumps down
380         clear T099_down
381         for n = 1:length(jumpsDOWN)
382              i0 = find(vvt_measured(jumpsDOWN(1,n):jumpsDOWN(2,n)) ...
383                  < maxTarget - 1.e-6*(maxTarget-minTarget),1,'first');
384              i99 = find(vvt_measured(jumpsDOWN(1,n):jumpsDOWN(2,n)) ...
385                  < maxTarget - 0.99*(maxTarget-minTarget),1,'first');
386             if (~isempty(i0)) & (~isempty(i99))
387               T099_down(n) = time(i99) - time(i0);
388             else
389              T099_down(n) = 0;
390             end
391         end
392
393         % Calcolo di media, mediana, dev standard
394         set.T099_ref_tutti{i}.media.down = mean(T099_down);
395         set.T099_ref_tutti{i}.mediana.down = median(T099_down);
396         set.T099_ref_tutti{i}.devstd.down = std(T099_down);
397         set.T099_ref_tutti{i}.data.down = T099_down;
398
399         % Eliminazione prove errate (valore lontano da mediana)
400         medianaUP = set.T099_ref_tutti{i}.mediana.up;
401         medianaDOWN = set.T099_ref_tutti{i}.mediana.down;
402         kk = 1;
403         while kk <= length(T099_up);
404             checkUP = (T099_up(kk) > medianaUP*0.975 ...
405                      & T099_up(kk) < medianaUP*1.025);
406             if ~checkUP
```

```
407                 T099_up(kk) = [];
408             else
409                 kk = kk+1;
410             end
411         end
412
413         kk = 1;
414         while kk <= length(T099_down);
415             checkDOWN = (T099_down(kk) > medianaDOWN*0.975 ...
416                         & T099_down(kk) < medianaDOWN*1.025);
417             if ~checkDOWN
418                 T099_down(kk) = [];
419             else
420                 kk = kk+1;
421             end
422         end
423
424         % T0-99 in output
425         set.T099_ref_tutti{i}.up = T099_up;
426         set.T099_ref_tutti{i}.down = T099_down;
427         if ~isempty(T099_up)
428             T099.up = mean(T099_up);
429         else
430             T099.up = mean(set.T099_ref_tutti{i}.data.up);
431         end
432         if ~isempty(T099_down)
433             T099.down = mean(T099_down);
434         else
435             T099.down = mean(set.T099_ref_tutti{i}.data.down );
436         end
437
438         set.OS_ref{i} = OS;
439         set.T0_99_ref{i} = T099;
440     end
441
442     % Plot dei valori ottenuti in condizione di riferimento
443     for j = 1:i
444         t = files(j).name;
445         stI = strfind(t,'POil');
446         t = strrep(t,'_JUMP_RIF.csv','');
447         tl{j} = t(stI-2:end);
448         figure(198)
449         subplot(2,1,1),plot(set.OS_ref_tutti{j}.up,'LineWidth',2),hold on
450         title('Overshoot - Prove di riferimento')
451         ylabel('OS up')
452         xlabel('# prova')
453         grid on
454         subplot(2,1,2),plot(set.OS_ref_tutti{j}.down,'LineWidth',2),hold on
455         ylabel('OS down')
456         grid on
457         xlabel('# prova')
```

```
458         figure(199)
459         subplot(2,1,1),plot(set.T099_ref_tutti{j}.up,'LineWidth',2),hold on
460         title('T0-99 - Prove di riferimento')
461         ylabel('T0-99 up')
462         xlabel('# prova')
463         grid on
464         subplot(2,1,2),plot(set.T099_ref_tutti{j}.down,'LineWidth',2),hold on
465         ylabel('T0-99 down')
466         grid on
467         xlabel('# prova')
468     end
469     figure(198),subplot(2,1,1),legend(tl,'Interpreter','none')
470     figure(198),subplot(2,1,2),legend(tl,'Interpreter','none')
471     figure(199),subplot(2,1,1),legend(tl,'Interpreter','none')
472     figure(199),subplot(2,1,2),legend(tl,'Interpreter','none')
473 else
474     return
475 end
476
477 %% CALCOLO INDICE DI PRESTAZIONE
478
479 % METODO DI CALCOLO DEI COEFFICIENTI DI VINCOLO:
480 %
481 %   c1: si valuta il valore assoluto dell'OS. Se è inferiore alla soglia
482 %   massima, c1 vale 1; se è compreso tra la soglia e la soglia + 25%, c1 è
483 %   interpolato linearmente tra 1 e 2; se è superiore alla soglia + 25%, c1
484 %   è saturato a 2.
485 %
486 %   c2: si valuta il valore assoluto dell'OS. Se è superiore alla soglia
487 %   minima, c1 vale 1; se è compreso tra la soglia e la soglia - 80%, c2 è
488 %   interpolato linearmente tra 1 e 5; se è inferiore alla soglia - 80%, c2
489 %   è saturato a 5.
490 %
491 %   c3: si valuta la prontezza di risposta del sistema (T099). Se la
492 %   differenza rispetto al riferimento è inferiore al 5%, c3 vale 1; se è
493 %   compresa tra il 5% e il 50%, c3 è interpolato linearmente tra 1 e 30;
494 %   se è superiore al 50%, c3 è saturato a 30.
495
496 sogliaOSmax = 4; sogliaOSmin = 1; sogliaT099 = 0.05;
497
498 for k = 1:i
499     % Jumps UP
500     for aa = 1:length(set.OS{k}.up)
501         if abs(set.OS{k}.up(aa)) <= sogliaOSmax
502             c1_up(aa,k) = 1.;
503         elseif abs(set.OS{k}.up(aa)) > sogliaOSmax*1.25
504             c1_up(aa,k) = 2.;
505         else
506             c1_up(aa,k) = interp1([sogliaOSmax, sogliaOSmax*1.25],[1,2],...
507                                   abs(set.OS{k}.up(aa)),'linear','extrap');
508         end
```

```
509         if abs(set.OS{k}.up(aa)) >= sogliaOSmin
510             c2_up(aa,k) = 1.;
511         elseif abs(set.OS{k}.up(aa)) >= sogliaOSmin*0.2
512             c2_up(aa,k) = 5.;
513         else
514             c2_up(aa,k) = interp1([sogliaOSmin*0.2,sogliaOSmin],[5,1],...
515                                 abs(set.OS{k}.up(aa)),'linear','extrap');
516         end
517         if set.T0_99{k}.up(aa) > set.T0_99_ref{k}.up
518             if abs((set.T0_99{k}.up(aa) - set.T0_99_ref{k}.up) ...
519                    /set.T0_99_ref{k}.up) <= sogliaT099
520                 c3_up(aa,k) = 1.;
521             elseif abs((set.T0_99{k}.up(aa) - set.T0_99_ref{k}.up) ...
522                    /set.T0_99_ref{k}.up) > sogliaT099
523                 c3_up(aa,k) = 3.;
524             else
525                 c3_up(aa,k) = interp1([sogliaT099,sogliaT099*2],[1,3],...
526                    abs((set.T0_99{k}.up(aa) - set.T0_99_ref{k}.up)...
527                    /set.T0_99_ref{k}.up),'linear','extrap');
528             end
529         else
530             c3_up(aa,k) = 1.;
531         end
532         i_temp_up(aa,k) = set.OS{k}.up(aa)*c1_up(aa,k)*c2_up(aa,k)*c3_up(aa,k);
533     end
534     if i_temp_up(end,k) == 0
535         [ip_min,idxUP] = min(i_temp_up(1:end-1,k));
536     else
537         [ip_min,idxUP] = min(i_temp_up(:,k));
538     end
539     ip.up(k) = ip_min;
540
541     % Jumps DOWN
542     for bb = 1:length(set.OS{k}.down)
543         if abs(set.OS{k}.down(bb)) <= sogliaOSmax
544             c1_down(bb,k) = 1.;
545         elseif abs(set.OS{k}.down(bb)) > sogliaOSmax*1.25
546             c1_down(bb,k) = 2.;
547         else
548             c1_down(bb,k) = interp1([sogliaOSmax, sogliaOSmax*1.25],[1,2],...
549                 abs(set.OS{k}.down(bb)),'linear','extrap');
550         end
551         if abs(set.OS{k}.down(bb)) >= sogliaOSmin
552             c2_down(bb,k) = 1.;
553         elseif abs(set.OS{k}.down(bb)) >= sogliaOSmin*0.2
554             c2_down(bb,k) = 5.;
555         else
556             c2_down(bb,k) = interp1([sogliaOSmin*0.2,sogliaOSmin],[5,1],...
557                 abs(set.OS{k}.down(bb)),'linear','extrap');
558         end
559         if set.T0_99{k}.down(bb) > set.T0_99_ref{k}.down
```

```
560             if abs((set.T0_99{k}.down(bb) - set.T0_99_ref{k}.down)...
561                     /set.T0_99_ref{k}.down) <= sogliaT099
562                 c3_down(bb,k) = 1.;
563             elseif abs((set.T0_99{k}.down(bb) - set.T0_99_ref{k}.down)...
564                     /set.T0_99_ref{k}.down) > sogliaT099
565                 c3_down(bb,k) = 3.;
566             else
567                 c3_down(bb,k) = interp1([sogliaT099,sogliaT099*2],[1,3],...
568                     abs((set.T0_99{k}.down(bb) - set.T0_99_ref{k}.down)...
569                     /set.T0_99_ref{k}.down),'linear','extrap');
570             end
571         else
572             c3_down(bb,k) = 1.;
573         end
574         i_temp_down(bb,k) = set.OS{k}.down(bb)*c1_down(bb,k)...
575             *c2_down(bb,k)*c3_down(bb,k);
576     end
577     if i_temp_down(end,k) == 0
578         [ip_min,idxDOWN] = min(i_temp_down(1:end-1,k));
579     else
580         [ip_min,idxDOWN] = min(i_temp_down(:,k));
581     end
582     ip.down(k) = ip_min;
583
584     % Scelta dell'indice minore
585     if ip.down(k) < ip.up(k)
586         idx_cal = idxDOWN;
587         ip_cal = ip.down(k);
588         minIP = 'down';
589         disp('Indice di prestazione migliore --> jumps down')
590     else
591         idx_cal = idxUP;
592         ip_cal = ip.up(k);
593         minIP = 'up';
594         disp('Indice di prestazione migliore --> jumps up')
595     end
596
597     % Interpolazione indici (polinomio di 4° grado)
598     deg = 4;
599     xGain = min(KDMAP_user(:,k)):0.001:max(KDMAP_user(:,k));
600     % Indici UP
601     p_up = polyfit(KDMAP_user(:,k),i_temp_up(:,k),deg);
602     i_interp_up(:,k) = polyval(p_up,xGain);
603     % Indici DOWN
604     p_down = polyfit(KDMAP_user(:,k),i_temp_down(:,k),deg);
605     i_interp_down(:,k) = polyval(p_down,xGain);
606
607     % Valore di calibrazione
608     t = files(k).name;
609     stI = strfind(t,'POil');
610     t = strrep(t,'_JUMP_RIF.csv','');
```

```
611     KDMAP_cal(1,k) = KDMAP_user(idx_cal,k);
612     disp(['PROVA                    : ' t(stI-2:end)])
613     disp(['KDMAP (user)           : ' num2str(KDMAP_cal(1,k))])
614
615     % Valore di calibrazione interpolato (ricerco il minimo del polinomio
616     % interpolante vicino al valore di KDMAP_user)
617     if strcmp(minIP,'down')
618         xGainRound = round(xGain/0.001)*0.001;
619         search = round(KDMAP_cal(1,k)/0.001)*0.001;
620         xh = find(xGainRound == search);
621         if isempty(xh)
622             [ip_min_interp,idxDOWN_interp] = min(i_interp_down(:,k));
623             KDMAP_cal(2,k) = xGain(idxDOWN_interp);
624         else
625             add = min([500,length(i_interp_down)-xh(end)])-1;
626             sub = min([500,xh(1)])-1;
627             [ip_min_interp,idxDOWN_interp] = ...
628                 min(i_interp_down(xh(1)-sub:xh(end)+add,k));
629             KDMAP_cal(2,k) = xGain(xh(1) - sub + idxDOWN_interp);
630         end
631     else
632         xGainRound = round(xGain/0.001)*0.001;
633         search = round(KDMAP_cal(1,k)/0.001)*0.001;
634         xh = find(xGainRound == search);
635         if isempty(xh)
636             [ip_min_interp,idxUP_interp] = min(i_interp_up(:,k));
637             KDMAP_cal(2,k) = xGain(idxUP_interp);
638         else
639             add = min([500,length(i_interp_down)-xh(end)])-1;
640             sub = min([500,xh(1)])-1;
641             [ip_min_interp,idxUP_interp] = ...
642                 min(i_interp_up(xh(1)-sub:xh(end)+add,k));
643             KDMAP_cal(2,k) = xGain(xh(1) - sub + idxUP_interp);
644         end
645     end
646     disp(['KDMAP (interpolazione) : ' num2str(KDMAP_cal(2,k))])
647     disp(' ')
648
649 end
```

A.5 KIMAP

```
close all
clear all
clc

set(0,'defaulttextinterpreter','none')

path = uigetdir(default,'Seleziona la cartella con le acquisizioni...');

if path ~= 0
    files=dir(fullfile(path,'*.csv'));
    for i = 1:length(files)
        fid = fopen([path '\' files(i).name],'r');
        sizeS = [1,10];
        count = 1;
        clear DATA;
        while ~feof(fid)
            DATA(count,:) = fscanf(fid,'%f,%f,%f,%f,%f,%f\n',sizeS);
            count = count + 1;
        end
        fclose(fid);
        clear time KIMAP vvt_target vvt_measured
        time_w = DATA(:,1);
        KIMAP = DATA(:,4);
        vvt_measured = DATA(:,3);
        vvt_target = DATA(:,2);
        figure(i),plot(time_w,vvt_measured),hold on
        plot(time_w,vvt_target,'LineWidth',2)
        plot(time_w,KIMAP,'k','LineWidth',3)
        xlabel('Time [s]')
        t = files(i).name;
        stI = strfind(t,'POil');
        t = strrep(t,'_JUMP.csv','');
        title(t(stI-2:end))
        legend('vvt_measured','vvt_target','KIMAP')

        % Filtraggio con media mobile
        campioni_mobile = 40;
        % vvt_measured_mobile = movmean(vvt_measured,campioni_mobile);
        vvt_measured_mobile = tsmovavg(vvt_measured,'s',campioni_mobile,1);
        figure(i*10),plot(time_w,vvt_measured,time_w,vvt_measured_mobile,...
            time_w,vvt_target)
        xlabel('Time [s]')
        legend('Segnale originale','Segnale filtrato','Target')

        % Riconosco i guadagni
        clear indiciCambioGuadagno indiciStartEndGuadagno
        indiciCambioGuadagno = find(diff(KIMAP) ~= 0);
        indiciStartEndGuadagno(:,1) = [1;indiciCambioGuadagno(1)];
        for k = 2:length(indiciCambioGuadagno)
```

```
50              indiciStartEndGuadagno(:,k) = [indiciCambioGuadagno(k-1)+1;...
51                                             indiciCambioGuadagno(k)];
52          end
53          indiciStartEndGuadagno(:,k+1) = [indiciCambioGuadagno(k)+1;...
54                                           length(vvt_target)];
55          for m = 1:length(indiciStartEndGuadagno)
56              KIMAP_user(m,i) = KIMAP(indiciStartEndGuadagno(2,m)-5);
57          end
58
59          % Inizializzazione vettori
60          OS.up = zeros(1,length(indiciStartEndGuadagno));
61          OS.down = zeros(1,length(indiciStartEndGuadagno));
62          T1090.up = zeros(1,length(indiciStartEndGuadagno));
63          T1090.down = zeros(1,length(indiciStartEndGuadagno));
64          T099.up = zeros(1,length(indiciStartEndGuadagno));
65          T099.down = zeros(1,length(indiciStartEndGuadagno));
66          Tstaz.up = zeros(1,length(indiciStartEndGuadagno));
67          Tstaz.down = zeros(1,length(indiciStartEndGuadagno));
68
69          for j = 1:length(indiciStartEndGuadagno)
70
71              % Finestratura del segnale per ogni guadagno fornito
72              clear vvt_target vvt_measured vvt_measured_mobile time
73              vvt_target = vvt_target(indiciStartEndGuadagno(1,j):...
74                              indiciStartEndGuadagno(2,j));
75              vvt_measured = vvt_measured(indiciStartEndGuadagno(1,j):...
76                                indiciStartEndGuadagno(2,j));
77              vvt_measured_mobile = vvt_measured_mobile(indiciStartEndGuadagno(1,j)
78                                                :indiciStartEndGuadagno(2,j))
79              ;time = time_w(indiciStartEndGuadagno(1,j):...
80                            indiciStartEndGuadagno(2,j)) ...
81                   - time_w(indiciStartEndGuadagno(1,j));
82
83              maxTarget = max(vvt_target); minTarget = min(vvt_target);
84              jump = maxTarget - minTarget;
85
86              % Trovo gli indici di inizio per i vari jumps in UP e DOWN
87              tollJump = jump/4;
88              iStartUP = find(diff(vvt_target) > jump-tollJump ...
89                            & diff(vvt_target) < jump+tollJump);
90              iStartDOWN = find(diff(vvt_target) < -jump+tollJump ...
91                              & diff(vvt_target) > -jump-tollJump);
92
93              % Considero tutti i jumps tranne il primo e l'ultimo
94              clear jumpsUP jumpsDOWN
95              if iStartUP(1) < iStartDOWN(1)
96                  for abc = 1:min([length(iStartUP),length(iStartDOWN)]) - 1
97                      jumpsUP(:,abc) = [iStartUP(abc);iStartDOWN(abc)];
98                      jumpsDOWN(:,abc) = [iStartDOWN(abc);iStartUP(abc+1)];
99                  end
100             else
```

```
101                 for abc = 1:min([length(iStartUP),length(iStartDOWN)]) - 1
102                     jumpsDOWN(:,abc) = [iStartDOWN(abc);iStartUP(abc)];
103                     jumpsUP(:,abc) = [iStartUP(abc);iStartDOWN(abc+1)];
104                 end
105             end
106
107             % Sistemo l'ultimo jump
108             if iStartUP(end) > iStartDOWN(end)
109                 jumpsDOWN(:,abc+1) = [iStartDOWN(end);iStartUP(end)];
110             else
111                 jumpsUP(:,abc+1) = [iStartUP(end);iStartDOWN(end)];
112             end
113
114             % Escludo il primo e l'ultimo dei jump non completi
115             if iStartUP(1) < iStartDOWN(1)
116                 jumpsUP(:,1) = [];
117             else
118                 jumpsDOWN(:,1) = [];
119             end
120
121             % Verifico che il numero dei jumps sia uguale in UP e in DOWN
122             if length(jumpsUP(1,:)) > length(jumpsDOWN(1,:))
123                 jumpsUP = jumpsUP(:,1:length(jumpsDOWN(1,:)));
124             else
125                 jumpsDOWN = jumpsDOWN(:,1:length(jumpsUP(1,:)));
126             end
127
128             jumpsUP = jumpsUP + 1; jumpsDOWN = jumpsDOWN + 1;
129
130             % Calcolo l'OVERSHOOT per i jumps up
131             clear OS_up
132             for n = 1:length(jumpsUP(1,:))
133                 OS_up(n) = max(vvt_measured(jumpsUP(1,n):jumpsUP(2,n)))-maxTarget
134                 if OS_up(n) < 0
135                    OS_up(n) = 0.;
136                 end
137             end
138             OS_up_medio = mean(OS_up);
139
140             % Calcolo l'OVERSHOOT per i jumps down
141             clear OS_down
142             for n = 1:length(jumpsDOWN(1,:))
143                 OS_down(n) = minTarget - min(vvt_measured(jumpsDOWN(1,n):...
144                                                     jumpsDOWN(2,n)));
145                 if OS_down(n) < 0
146                    OS_down(n) = 0.;
147                 end
148             end
149             OS_down_medio = mean(OS_down);
150
151             % OS in output
```

```
152             OS.up(j)= OS_up_medio;
153             OS.down(j)= OS_down_medio;
154
155             % Calcolo Tstaz: per trovare il tstaz definiamo il vettore
156             % check, il cui m-esimo elemento vale 1 se il segnale rientra
157             % nella banda di tolleranza, 0 altrimenti. Quindi invertiamo
158             % il vettore ottenuto, eliminiamo gli ultimi elementi, nel caso
159             % siano nulli (a causa del filtraggio), e ricerchiamo il primo
160             % elemento nullo, che indica l'ultimo campione prima che il
161             % segnale entri definitivamente in stazionario.
162
163             % Calcolo Tstaz per i jumps up
164             clear lastOne check startIdx dt tt
165             toll_staz = 0.2;        % tolleranza (ampiezza della semibanda)
166             dt = mean(diff(time(jumpsUP(1,n):jumpsUP(2,n))));
167             for n = 1:length(jumpsUP(1,:))
168                 tt = vvt_measured_mobile(jumpsUP(1,n):jumpsUP(2,n));
169                 for m = 1:length(tt)
170                     if tt(m) > maxTarget - toll_staz ...
171                      & tt(m) < maxTarget + toll_staz
172                         check(m) = 1;
173                     else
174                         check(m) = 0;
175                     end
176                 end
177                 if ~isempty(check)
178                     lastOne = find(check == 1,1,'last');
179                     check = check(1:lastOne);
180                     startIdx = find(fliplr(check) == 0,1,'first') - 1;
181                     startIdxUP(n) = length(check) - startIdx;
182                     Tstaz_up(n) = dt*startIdxUP(n);
183                 else
184                     Tstaz_up(n) = 0;
185                 end
186             end
187             Tstaz_up_medio = mean(Tstaz_up);
188
189             % Calcolo Tstaz per i jumps down (2° METODO)
190             clear lastOne check startIdx dt tt
191             dt = mean(diff(time(jumpsDOWN(1,n):jumpsDOWN(2,n))));
192             for n = 1:length(jumpsDOWN(1,:))
193                 tt = vvt_measured_mobile(jumpsDOWN(1,n):jumpsDOWN(2,n));
194                 for m = 1:length(tt)
195                     if tt(m) > minTarget - toll_staz ...
196                      & tt(m) < minTarget + toll_staz
197                         check(m) = 1;
198                     else
199                         check(m) = 0;
200                     end
201                 end
202                 if ~isempty(check)
```

```
                    lastOne = find(check == 1,1,'last');
                    check = check(1:lastOne);
                    startIdx = find(fliplr(check) == 0,1,'first') - 1;
                    startIdxDOWN(n) = length(check) - startIdx;
                    Tstaz_down(n) = dt*startIdxDOWN(n);
                else
                    Tstaz_down(n) = 0;
                end
            end
            Tstaz_down_medio = mean(Tstaz_down);

            % Tstaz in output
            Tstaz.up(j) = Tstaz_up_medio;
            Tstaz.down(j) = Tstaz_down_medio;

            % Calcolo errore medio a regime per i jumps up
            for n = 1:length(jumpsUP(1,:))
                errore_up(n) = mean(abs(vvt_measured_mobile(jumpsUP(1,n) ...
                    + startIdxUP(n):jumpsUP(2,n)) - vvt_target(jumpsUP(1,n)...
                    + startIdxUP(n):jumpsUP(2,n))));
            end
            errore_up_medio = mean(errore_up);

            % Calcolo errore medio a regime per i jumps down
            for n = 1:length(jumpsDOWN(1,:))
                errore_down(n) = mean(abs(vvt_measured_mobile(jumpsDOWN(1,n)...
              + startIdxDOWN(n):jumpsDOWN(2,n)) - vvt_target(jumpsDOWN(1,n)...
              + startIdxDOWN(n):jumpsDOWN(2,n))));
            end
            errore_down_medio = mean(errore_down);

            % Errore in output
            errore.up(j) = errore_up_medio;
            errore.down(j) = errore_down_medio;

            if j == 1
                set.OS_ref{i}.up = OS.up(j);
                set.OS_ref{i}.down = OS.down(j);
                set.T1090_ref{i}.up = T1090.up(j);
                set.T1090_ref{i}.down = T1090.down(j);
                set.T099_ref{i}.up = T099.up(j);
                set.T099_ref{i}.down = T099.down(j);
                set.Tstaz_ref{i}.up = Tstaz.up(j);
                set.Tstaz_ref{i}.down = Tstaz.down(j);
                set.errore_ref{i}.up = errore.up(j);
                set.errore_ref{i}.down = errore.down(j);
            end

        end
        set.OS{i} = OS;
        set.T1090{i} = T1090;
```

```
254         set.T099{i} = T099;
255         set.Tstaz{i} = Tstaz;
256         set.errore{i} = errore;
257         clear OS Tstaz errore Tstaz_up Tstaz_up_medio Tstaz_down
258         clear Tstaz_down_medio OS_up OS_up_medio OS_down OS_down_medio
259         clear errore_up errore_up_medio errore_down errore_down_medio
260     end
261
262
263     %% CALCOLO INDICE DI PRESTAZIONE
264
265     % METODO DI CALCOLO DEI COEFFICIENTI DI VINCOLO:
266     %
267     %   c1: si valuta l'errore a regime. Se è inferiore a una certa soglia, c1
268     %   vale 1, altrimenti cresce linearmente con l'errore.
269     %
270     %   c2: si valuta l'overshoot. Se non è peggiore oltre una certa soglia
271     %   rispetto all'overshoot calcolato nelle condizioni di riferimento
272     %   (guadagno integrale minimo) c2 vale 1, altrimenti cresce linearmente
273     %   con l'overshoot.
274
275     sogliaErrore = 0.75; sogliaOS = 0.05;
276
277     for k = 1:i
278         % Jumps UP
279         for aa = 1:length(set.errore{k}.up)
280             if abs(set.errore{k}.up(aa)) <= sogliaErrore
281                 c1 = 1.;
282             else
283                 c1 = interp1([0, sogliaErrore],[0,1],...
284                     abs(set.errore{k}.up(aa)),'linear','extrap');
285             end
286             if abs((set.OS{k}.up(aa) - set.OS_ref{k}.up)/set.OS_ref{k}.up)...
287                     <= sogliaOS
288                 c2 = 1.;
289             else
290                 c2 = interp1([sogliaOS,sogliaOS*10],[1,10],...
291                     abs(set.OS{k}.up(aa)),'linear','extrap');
292             end
293             if c2 > 5
294                 c2 = 5;
295             end
296             i_temp_up(aa,k) = set.Tstaz{k}.up(aa)*c1*c2;
297         end
298         if i_temp_up(end,k) == 0
299             [ip_min,idxUP] = min(i_temp_up(2:end-1,k));
300         else
301             [ip_min,idxUP] = min(i_temp_up(2:end,k));
302         end
303         ip.up(k) = ip_min;
304
```

```
305         % Jumps DOWN
306         for bb = 1:length(set.OS{k}.down)
307             if abs(set.errore{k}.down(bb)) <= sogliaErrore
308                 c1 = 1.;
309             else
310                 c1 = interp1([0, sogliaErrore],[0,1],...
311                     abs(set.errore{k}.down(bb)),'linear','extrap');
312             end
313             if abs((set.OS{k}.down(bb) - set.OS_ref{k}.down)...
314                     /set.OS_ref{k}.down) <= sogliaOS
315                 c2 = 1.;
316             else
317                 c2 = interp1([sogliaOS,sogliaOS*10],[1,10],...
318                     abs(set.OS{k}.down(bb)),'linear','extrap');
319             end
320             if c2 > 5
321                 c2 = 5;
322             end
323             i_temp_down(bb,k) = set.Tstaz{k}.down(bb)*c1*c2;
324         end
325         if i_temp_down(end,k) == 0
326             [ip_min,idxDOWN] = min(i_temp_down(2:end-1,k));
327         else
328             [ip_min,idxDOWN] = min(i_temp_down(2:end,k));
329         end
330         ip.down(k) = ip_min;
331
332         % Scelta dell'indice minore
333         if ip.down(k) < ip.up(k)
334             idx_cal = idxDOWN + 1;
335             ip_cal = ip.down(k);
336             minIP = 'down';
337             disp('Indice di prestazione migliore --> jumps down')
338         else
339             idx_cal = idxUP + 1;
340             ip_cal = ip.up(k);
341             minIP = 'up';
342             disp('Indice di prestazione migliore --> jumps up')
343         end
344
345         % Interpolazione indici (polinomio di 7° grado)
346         deg = 7;
347         xGain = min(KIMAP_user(:,k)):0.001:max(KIMAP_user(:,k));
348         % Indici UP
349         p_up = polyfit(KIMAP_user(:,k),i_temp_up(:,k),deg);
350         i_interp_up(:,k) = polyval(p_up,xGain);
351         % Indici DOWN
352         p_down = polyfit(KIMAP_user(:,k),i_temp_down(:,k),deg);
353         i_interp_down(:,k) = polyval(p_down,xGain);
354
355         % Valore di calibrazione
```

```
356         t = files(k).name;
357         stI = strfind(t,'POil');
358         t = strrep(t,'_JUMP.csv','');
359         KIMAP_cal(1,k) = KIMAP_user(idx_cal,k);
360         disp(['PROVA                    : ' t(stI-2:end)])
361         disp(['KIMAP (user)          : ' num2str(KIMAP_cal(1,k))])
362 
363         % Valore di calibrazione interpolato (ricerco il minimo del polinomio
364         % interpolante vicino al valore di KIMAP_user)
365         if strcmp(minIP,'down')
366             xGainRound = round(xGain/0.001)*0.001;
367             search = round(KIMAP_cal(1,k)/0.001)*0.001;
368             xh = find(xGainRound == search);
369             if isempty(xh)
370                 [ip_min_interp,idxDOWN_interp] = min(i_interp_down(:,k));
371                 KIMAP_cal(2,k) = xGain(idxDOWN_interp);
372             else
373                 add = min([500,length(i_interp_down)-xh(end)])-1;
374                 sub = min([500,xh(1)])-1;
375                 [ip_min_interp,idxDOWN_interp] = ...
376                     min(i_interp_down(xh(1)-sub:xh(end)+add,k));
377                 KIMAP_cal(2,k) = xGain(xh(1) - sub + idxDOWN_interp);
378             end
379         else
380             xGainRound = round(xGain/0.001)*0.001;
381             search = round(KIMAP_cal(1,k)/0.001)*0.001;
382             xh = find(xGainRound == search);
383             if isempty(xh)
384                 [ip_min_interp,idxUP_interp] = min(i_interp_up(:,k));
385                 KIMAP_cal(2,k) = xGain(idxUP_interp);
386             else
387                 add = min([500,length(i_interp_down)-xh(end)])-1;
388                 sub = min([500,xh(1)])-1;
389                 [ip_min_interp,idxUP_interp] = ...
390                     min(i_interp_up(xh(1)-sub:xh(end)+add,k));
391                 KIMAP_cal(2,k) = xGain(xh(1) - sub + idxUP_interp);
392             end
393         end
394         disp(['KIMAP (interpolazione) : ' num2str(KIMAP_cal(2,k))])
395         disp(' ')
396 
397     end
398 
399     %% Plot indici di prestazione nelle varie prove
400 
401     colore = 'brgmycbrgmycbrgmyc';
402     figure(998)
403     for kj = 1:size(i_temp_up,2)
404         plot(KIMAP_user,i_temp_up(:,kj),'.-','MarkerSize',...
405             20,'Color',colore(kj)),hold on
406         plot(xGain,i_interp_up(:,kj),'LineWidth',2,'Color',colore(kj))
```

```
407     end
408     grid on
409     xlabel('KIMAP')
410     title('Indici di prestazione (JUMPS UP)')
411     cx = 1;
412     for kj = 1:length(files)
413         t = files(kj).name;
414         stI = strfind(t,'POil');
415         t = strrep(t,'_JUMP.csv','');
416         tl{cx} = t(stI-2:end);
417         ttl{kj} = tl{cx};
418         tl{cx+1} = [tl{cx} ' interpolato'];
419         cx = cx+2;
420     end
421     legend(tl,'Interpreter','none')
422
423     figure(999)
424     for kj = 1:size(i_temp_down,2)
425         plot(KIMAP_user,i_temp_down(:,kj),'.-','MarkerSize',...
426             20,'Color',colore(kj)),hold on
427         plot(xGain,i_interp_down(:,kj),'LineWidth',2,'Color',colore(kj))
428     end
429     grid on
430     xlabel('KIMAP')
431     title('Indici di prestazione (JUMPS DOWN)')
432     cx = 1;
433     for kj = 1:length(files)
434         t = files(kj).name;
435         stI = strfind(t,'POil');
436         t = strrep(t,'_JUMP.csv','');
437         tl{cx} = t(stI-2:end);
438         tl{cx+1} = [tl{cx} ' interpolato'];
439         cx = cx+2;
440     end
441     legend(tl,'Interpreter','none')
442
443
444     %% Plot OS, T099, T1090 nelle varie prove
445
446     for aa = 1:i
447         % Jumps UP
448         figure(198)
449         subplot(2,2,1),plot(KIMAP_user(:,aa),set.OS{aa}.up),hold on
450         xlabel('KIMAP')
451         ylabel('OS up')
452         grid on
453
454         subplot(2,2,2),plot(KIMAP_user(:,aa),set.T099{aa}.up),hold on
455         xlabel('KIMAP')
456         ylabel('T099 up')
457         grid on
```

```
458
459           subplot(2,2,3),plot(KIMAP_user(:,aa),set.Tstaz{aa}.up),hold on
460           xlabel('KIMAP')
461           ylabel('Tstaz up')
462           grid on
463
464           subplot(2,2,4),plot(KIMAP_user(:,aa),set.errore{aa}.up),hold on
465           xlabel('KIMAP')
466           ylabel('Errore stazionario up')
467           grid on
468
469           % Jumps DOWN
470           figure(199)
471           subplot(2,2,1),plot(KIMAP_user(:,aa),set.OS{aa}.down),hold on
472           xlabel('KIMAP')
473           ylabel('OS down')
474           grid on
475
476           subplot(2,2,2),plot(KIMAP_user(:,aa),set.T099{aa}.down),hold on
477           xlabel('KIMAP')
478           ylabel('T099 down')
479           grid on
480
481           subplot(2,2,3),plot(KIMAP_user(:,aa),set.Tstaz{aa}.down),hold on
482           xlabel('KIMAP')
483           ylabel('Tstaz down')
484           grid on
485
486           subplot(2,2,4),plot(KIMAP_user(:,aa),set.errore{aa}.down),hold on
487           xlabel('KIMAP')
488           ylabel('Errore stazionario down')
489           grid on
490       end
491
492       ttl = ttl(1:length(files));
493       figure(198),legend(ttl,'Interpreter','none')
494       figure(199),legend(ttl,'Interpreter','none')
495
496   end
```

Ringraziamenti

Sarebbe un peccato non approfittare di questa occasione per ringraziare tutte le persone che, in un modo o nell'altro, hanno contribuito affinché potessi portare a termine questo lavoro.

Il Professor Nicolò Cavina, le cui lezioni hanno saputo trasmettermi la passione per questo lavoro, e che ringrazio anche per aver creduto in me e per avermi offerto la possibilità di continuare anche in futuro il percorso iniziato con questa tesi.

Marco, per la sua gentilezza e per esser sempre stato disponibile ad aiutarmi nei momenti di bisogno, per la grande abilità dimostrata nel risolvere i problemi e per avermi insegnato in poco tempo molte cose delle quali farò sicuramente tesoro per il mio futuro.

Giovanni, per avermi accompagnato fin dall'inizio in questo percorso e per il grande contributo che ha rappresentato per tutto il lavoro qui descritto; desidero poi ringraziare tutti i ragazzi del team di Alma Automotive per avermi accolto in azienda.

I miei genitori, per esser sempre stati al mio fianco e per avermi dato l'opportunità di iniziare e concludere nel migliore dei modi un percorso universitario decisamente impegnativo.

I miei amici e tutte le persone che ho avuto la fortuna di conoscere in questi cinque anni e che hanno lasciato un segno nella mia vita, per aver condiviso nel bene e nel male tante avventure.

Bibliografia

[1] *Appunti del corso di Motori a Combustione Interna e Propulsori Ibridi M tenuto dal Prof. Nicolò Cavina*.

[2] *Appunti del corso di Sperimentazione e Calibrazione di Motori a Combustione Interna M tenuto dal Prof. Nicolò Cavina*.

[3] G. BLANDO, *Sviluppo di strumenti per la calibrazione automatica di funzioni di controllo motore*, tesi di laurea, Scuola di Ingegneria e Architettura - Università di Bologna, 2016.

[4] G. FERRARI, *Motori a combustione interna*, Edizioni Il Capitello Torino, 1996.

[5] C. LAMBERTI, *Sviluppo di soluzioni per la gestione semiautomatica delle prove su motori a combustione interna, basate sulla comnunicazione tra sistemi di analisi combustione e sistemi di calibrazione motore*, tesi di laurea, Scuola di Ingegneria e Architettura - Università di Bologna, 2015.

[6] P. TARDO, *Analisi e sviluppo di soluzioni per la calibrazione semi-automatica di funzioni di controllo motore*, tesi di laurea, Scuola di Ingegneria e Architettura - Università di Bologna, 2016.