



TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

**UNIVERSITÀ DEGLI STUDI DI ROMA
TOR VERGATA**

FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA DELL'AUTOMAZIONE

A.A. 2020/2021

Tesi di Laurea

Sviluppo di algoritmi di controllo delle correnti nelle bobine poloidali di macchine per la fusione Tokamak,
con riguardo al design sistematico per la cooperazione tra sistemi embedded di attuazione, misurazione e controllo.

RELATORE

Daniele Carnevale

CANDIDATO

Emanuele Alfano

CORRELATORI

Marco Passeri

*Questa tesi è dedicata ai miei cari nonni, che sempre hanno saputo esserci per me
e alla mia dolce professoressa Beniamina, non c'è giorno che non pensi a te*

A voi dedico questa tesi e questo traguardo, e mi affido per il traguardo di domani.

Indice

Ringraziamenti	1
Introduzione	2
Fusione Termonucleare	2
Struttura di un Tokamak	4
Obiettivi della Tesi	5
1 Elementi Costitutivi	6
1.1 Trasformatore, un Modello di Tokamak	7
1.1.1 Richiami di elettronica	8
1.1.2 Modellazione Fisica	12
1.1.3 Semplificazione Primario - Secondario	13
1.1.4 Dal circuito alla dinamica	14
1.1.5 Funzione di Trasferimento	14
1.1.6 Misura del campo Elettrico del Plasma come indice per la Corrente	16
1.2 Trasduttore di Corrente	18
1.2.1 Sensore scelto	18
1.2.2 Criticità	19
1.2.3 Funzionamento Interno	20
1.2.4 Connessione elettrica	22

1.2.5	Misura	22
1.3	Driver di Corrente - IBT-2	24
1.3.1	Schema Elettrico	25
1.3.2	Connessione di Controllo	25
1.3.3	Benchmark del Driver	27
2	Architettura di Sistema	29
2.1	Architettura ad alto livello	29
2.1	EMP - Libreria di Comunicazione Seriale	31
2.1	Protocollo - COBS	32
2.1.1	Metodo di codifica	32
2.1.2	Caratteristiche chiave	33
2.1.3	Integrità dei pacchetti	35
2.1.4	Struttura del codice	36
2.1.5	Logica di Comunicazione	37
2.1.6	Code Flow	38
2.1.7	Test di Codifica/Decodifica su ogni Device	39
2.2	Online Sampling	40
2.2.1	Interconnessione μ Controllore \Leftrightarrow Companion	41
2.2.2	Storage su file delle informazioni	42
2.3	Post Elaborazione con Matlab	43
2.3.1	Conversioni Dati	43
2.3.2	Filtraggio senza distorsioni di fase	43
3	Stima del modello del sistema	44
3.1	Metodo di stima automatico	45

3.2	Tuning dei coefficienti	47
3.3	Benchmark	48
4	Modello teorico di Controllo	51
4.1	Obiettivo di controllo	51
4.2	Teorema del valore iniziale e del valore finale	53
4.3	Controllo a errore nullo	54
4.3.1	Design di controllore adottato	56
4.4	Simulazione <i>Qualitativa</i> su Simulink	57
4.4.1	Design Base di controllo	58
4.4.2	Design Avanzato di controllo	59
4.5	Conclusioni per il design Avanzato	64
5	Sviluppo Controllo reale	65
5.1	Discretizzazione Zero-Order Hold (Z.O.H.)	67
5.2	Codifica del controllore	69
5.3	Esperimenti	69
5.3.1	Inseguimento singolo	70
5.3.2	Inseguimento Triangolare	71
5.3.3	Cambio riferimento in corsa	72
6	Conclusioni e sviluppi futuri	73
Appendice A - Codice Arduino		75
7.1	Set-up Registri	75
7.1.1	Tic Timer	75
7.1.2	Frequenza PWM	75

7.2	Generatore di Segnale	77
7.2.1	Segnali Base	77
7.2.2	Segnali Composti	78
7.3	Codici Controllore	79
7.3.1	Classe Controllore	80
Appendice B - Codice EMP		83
8.4	Pacchetti in EMP	83
8.4.1	Definizione di 2 Pacchetti	83
8.4.2	Pacchetti Multipli	84
Appendice C - Matlab Post Elaboration		85
9.5	Importazione dei Dati	85
9.5.1	Parsing Tabelle	85
9.5.2	Conversione dei Dati	86
9.6	Stima Parametri	87
9.6.1	Stima parametri automatica	87
9.6.2	Stima parametri manuale	87
9.6.3	Plot dell'esperimento	88
Elenco delle figure		89
Elenco delle tabelle		90
Elenco dei codici		91
Bibliografia		93

Ringraziamenti

Questa tesi è stata resa possibile dal contributo nella mia vita di tante persone, che giorno per giorno mi hanno sempre dato il loro sostegno, a voi dedico questa mia Tesi.

Un ringraziamento speciale alla mia famiglia, in particolare a mia **Madre** e mio **Padre**: è grazie al vostro sostegno e incoraggiamento se oggi sono riuscito a raggiungere questo traguardo.

La forza di arrivare qui, oggi, però non è dovuta solo a loro, devo per forza ringraziare dell'affetto e il sostegno speciale da parte dei miei cari amici, che ogni giorno hanno condiviso con me gioie, sacrifici e successi, senza voltarmi mai le spalle, mi hanno dato la forza di arrivare a questo prezioso traguardo. **Filippo, Gabriele, Marta**, grazie di TUTTO.

Un pensiero in particolare vola verso la mia dolce **Nicoleta**, è sicuramente grazie all'affetto e le attenzioni che mi hai donato che sono riuscito a tenere dritto il timone ed arrivare qui oggi. Per terminare voglio ringraziare tutti i professori che negli ultimi 18 anni hanno guidato il mio cammino, loro che hanno sempre creduto in me e nelle mie capacità meritano tutta la mia gratitudine.

Un ringraziamento speciale va però alla mia professoressa e mentore **Beniamina Rauch** che fu la prima a vedere il mio potenziale e coltivarlo, so che ancora vegli su di me, non troverò mai le parole per ringraziarti abbastanza.

Oltre a lei ringrazio il mio relatore **Daniele Carnevale** che in questi anni universitari, da quando mi ha conosciuto ad oggi, ha sempre creduto in me e mi ha permesso di fare esperienze che mai avevo immaginato.

Un sentito grazie a tutti voi e buona lettura.

Introduzione

Il presente lavoro di tesi si colloca nel contesto delle tecniche di controllo per impianti Tokamak come FTU (Frascati Tokamak Upgrade), attualmente in smaltimento, Proto-Sphera o ITER (International Thermonuclear Experimental Reactor).

La tesi è stata realizzata in collaborazione con il centro ricerche ENEA (Ente Nazionale per l'Energia e l'Ambiente) di Frascati è incentrata sullo sviluppo e creazione del prototipo di controllo delle bobine magnetiche che permettono il controllo e confinamento magnetico del plasma all'interno dell'impianto Tokamak.

Fusione Termonucleare

In fisica nucleare la fusione è il processo mediante il quale i nuclei di due o più atomi vengono compressi tanto da far prevalere l'interazione forte sulla repulsione coulombiana, unendosi tra loro e andando così a formare un nucleo di massa minore della massa dei reagenti con la conseguente liberazione di un'elevata quantità di energia che conferisce al processo caratteristiche fortemente esotermiche.

La massa mancante viene trasformata in energia in accordo con l'equazione di Einstein:

$$E = (m_r - m)c^2$$

Dove m_r è la massa dei reagenti e m è la massa risultante.

Il processo di Fusione Nucleare avviene naturalmente all'interno delle stelle, e trasforma l'idrogeno di cui sono composte in elio.

L'energia nucleare di questo prodotta da questa reazione è elevatissima e di forte interesse, anche ambientale. Tra i maggiori vantaggi troviamo:

1. Il combustibile (idrogeno e deuterio) è praticamente inesauribile ed è a disposizione di tutte le nazioni che abbiano uno sbocco sul mare. Il deuterio può essere estratto dall'acqua, anche se con costi energetici non indifferenti; per fare un esempio, un ditale pieno di deuterio equivale a 20 tonnellate di carbone in termini di energia. Un lago di medie dimensioni contiene deuterio sufficiente a rifornire una nazione di energia per secoli utilizzando la fusione nucleare (ovviamente supponendo di sfruttarlo tutto).
2. Nessuna possibilità di incidenti come quelli di Černobyl' o di Three Mile Island in quanto il reattore non contiene sostanze radioattive come l'uranio o le scorie di fissione. Inoltre, possibili incidenti, come fughe di trizio o perdite di liquido refrigerante, avrebbero un impatto ambientale e radiativo molto più contenuto e temporaneo.
3. Nessun prodotto chimico da combustione (anidride carbonica ad esempio) come residuo immesso nell'atmosfera e quasi nessun contributo al riscaldamento del pianeta.
4. Impossibilità di utilizzo dei reattori per la produzione di materiale per scopi bellici o terroristici
5. Basso livello di radioattività residua e produzione di sostanze con breve vita media (tempo in cui la radioattività si riduce rapidamente).

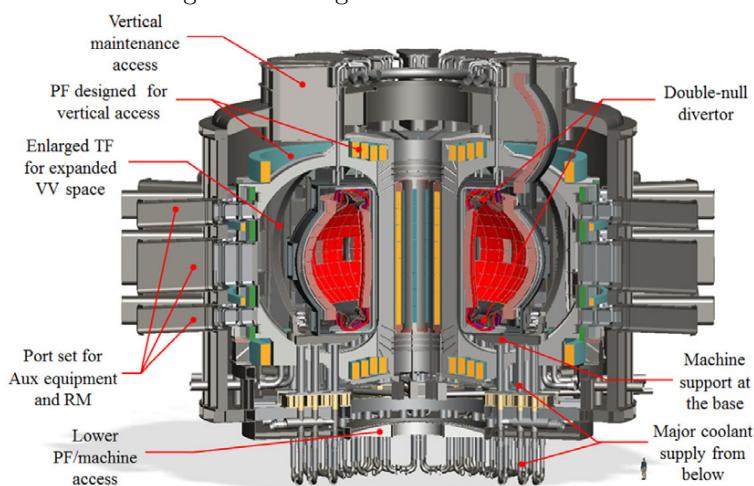
Tutti questi vantaggi rendono lo sviluppo di Centrali termonucleari a **Fusione Nucleare** di grande interesse e tecnologie chiave per preservare l'ambiente del nostro pianeta.

In questa tesi faremo riferimento a impianti sperimentali di tipo Tokamak, che però, va specificato essere solo uno dei design attualmente in sviluppo nelle ricerche sulla Fusione Nucleare controllata.

Struttura di un Tokamak

Un **Tokamak** (acronimo russo per "camera toroidale magnetica") è una macchina di forma toroidale (a ciambella) in cui un gas (solitamente idrogeno) viene portato nello stato di plasma e mantenuto coeso e lontano dalle pareti interne grazie ad un campo magnetico creato da elettromagneti esterni alla camera.

Figura 1: Collegamento dal Datasheet



La forma del Tokamak a toro (ciambella) è studiata per permettere alle particelle del plasma di muoversi all'interno del campo magnetico, creato all'esterno delle pareti (*Esterno Vessel*) in un moto circolare.

Questo movimento avviene poiché le particelle del plasma sono per definizione cariche, e in quanto tali, se immerse in un campo magnetico esse tendono a muoversi seguendo una traiettoria elicoidale (detta anche moto di ciclotrone) attorno alle linee del campo magnetico, che in questo caso sono chiuse e contenute all'interno della sezione del Tokamak (*Interno Vessel*).

L'uso di una confinazione magnetica per questo plasma è dovuta all'impossibilità, per qualunque materiale, di resistere alle enormi temperature raggiunte dal plasma durante la fusione in un contatto diretto con esse.

Grazie all'equazione di Larmor, che definisce il raggio di Larmor:

$$\rho = \frac{mv_{\perp}}{ZeB}$$

v_{\perp} è la velocità della particella perpendicolare al campo magnetico.
 m è la sua massa.
 B è l'intensità del campo magnetico.
 Ze è la carica del portatore.

Abbiamo che questo "**Tubo**" di plasma, contenuto all'interno del Vessel, non si può espandere più di ρ dalla linea di campo.

Ciò rende un campo magnetico il mezzo ottimo per confinare in modo efficiente un plasma, e i Tokamak sono impianti progettati per creare questo confinamento in maniera efficace e sicura, cercando al tempo stesso di compattare il plasma su se stesso (aumentando il campo magnetico B), e conseguentemente avvicinando gli atomi tra loro (riduzione di ρ), aumentando così la probabilità di ottenere una reazione di fusione.

Obiettivi della Tesi

L'obiettivo della tesi è sviluppare il prototipo della scheda di controllo delle bobine, in grado di ricevere comandi attraverso la rete di interconnessione tra i dispositivi di controllo dell'impianto Tokamak, qui pensata per Inter-operare con il Framework MARTE2, e attuare un controllo rapido sulla corrente della bobina seguendo tali riferimenti.

Capitolo 1

Elementi Costitutivi

In questo capitolo si vogliono descrivere e caratterizzare i 3 elementi salienti dell'esperimento:

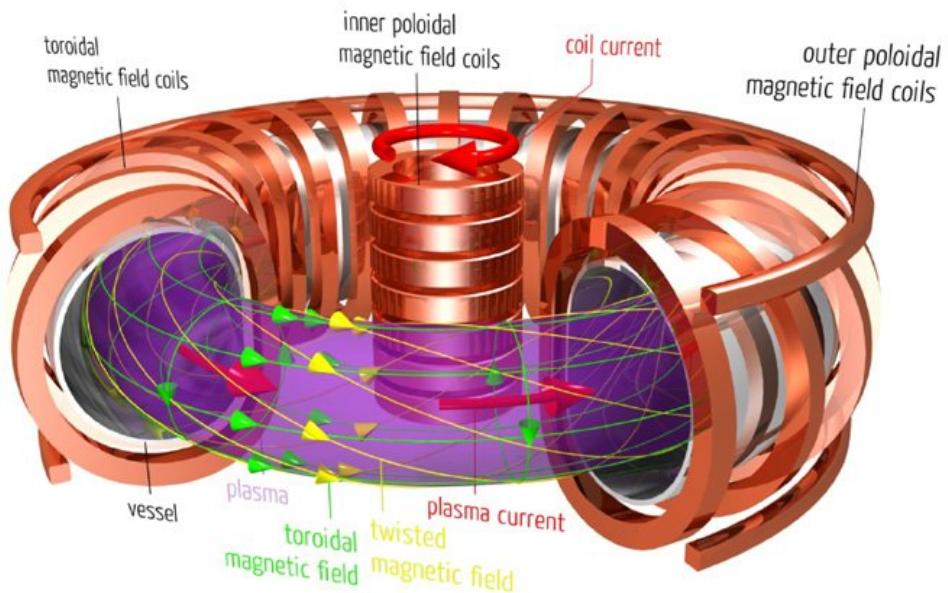
1. *Trasformatore, un Modello di Tokamak*
2. *Trasduttore di Corrente*
3. *Driver di Corrente - IBT-2*

Verranno analizzate le loro caratteristiche chiavi per mettere in luce il perché della scelta, e si evidenzieranno eventuali problemi che affliggono in componenti, problemi di cui si è tenuto conto nello sviluppo del progetto per poterli annullare.

1.1 Trasformatore, un Modello di Tokamak

Come visto nell'introduzione, la tesi ha come obiettivo la prototipazione del sistema di controllo per le bobine poloidali presenti in impianti tokamak.

Figura 1.1: Interno Tokamak



Le bobine (poloidali e toroidali) servono a controllare il plasma presente nel *Vessel* dell'impianto e confinare il Plasma all'interno di un flusso compresso dentro la camera, le alte temperature e la forte compressione a cui è sottoposto il Plasma, permette di realizzare eventi di **Fusione Nucleare** tra gli atomi di idrogeno a base del plasma.

L'interazione tra le **Bobine e Plasma**, ha un modello matematico non dissimile da quello di **Trasformatore Elettrico** nella relazione *Primario-Secondario* (con i dovuti paragoni e le ovvie *Non-Linearità* presenti nel caso di un impianto Tokamak reale qui semplificate a costanti).

Grazie a questa similitudine è stato possibile replicare in sicurezza la fisica presente all'interno di un Tokamak, nell'ambiente controllato del laboratorio.

1.1.1 Richiami di elettronica

Prima di modellare ed analizzare l'esperimento della tesi, è necessario richiamare qualche proprietà/concetto di elettronica per poter comprendere i passaggi matematici e fisici:

- Prima legge di Kirchhoff (legge dei nodi)
- Seconda legge di Kirchhoff (legge delle maglie)
- Induttore Ideale
- Induttanza
- Trasformatore Ideale Monofase

Teorema 1.1.1 (Prima legge di Kirchhoff (legge dei nodi)).

La somma algebrica delle intensità di corrente nei rami facenti capo allo stesso nodo è nulla.

$$\sum I_k = 0 \quad (1.1.1)$$

■

Teorema 1.1.2 (Seconda legge di Kirchhoff (legge delle maglie)).

La somma algebrica delle f.e.m. agenti lungo i rami di una maglia è uguale alla somma algebrica dei prodotti delle intensità di corrente di ramo per le rispettive resistenze (del ramo).

$$\sum_{\forall k} V_k = \sum_{\forall k} f_{emk} \quad (1.1.2)$$

■

Oltre ai teoremi di Kirchhoff, che ci serviranno per ricavare le equazioni della dinamica, enunciamo ora le proprietà degli induttori, indispensabili per poter definire la loro relazione Corrente/Tensione.

Definizione 1.1.1 (Induttore Ideale).

Un Induttore Ideale si oppone solo alle variazioni di corrente, variando la tensione ai suoi capi di conseguenza, non presenta nessuna resistenza elettrica in caso di correnti costanti ai suoi capi.

Il suo valore è dato dal **coefficiente di autoinduzione**, tipicamente espresso con il simbolo L , la cui unità di è l'Henry [H].

Un Induttore accumula energia all'interno di un campo magnetico, e questa relazione è descritta dall'equazione:

$$\Phi_B = Li \quad (1.1.3)$$

$\Phi_B :=$ Flusso magnetico; $L :=$ Coefficiente di autoinduzione

Dalla legge di Faraday(ignorando momentaneamente la conservazione dell'energia, ovvero la legge di Lenz), applicata alla circuitazione del circuito costituito dalla induttanza stessa, si ha:

$$\frac{d\Phi_B}{dt} = V \quad (1.1.4)$$

Dove V è il potenziale indotto ai morsetti del circuito in questione. Perciò, derivando l'equazione 1.1.3 ad entrambi i membri rispetto al tempo, si ottiene:

$$\frac{d\Phi_B}{dt} = L \frac{di}{dt} + i \frac{dL}{dt} \quad (1.1.5)$$

In molti casi fisici, però, l'induttanza può essere considerata costante rispetto al tempo (o tempo-invariante), da cui:

$$\frac{d\Phi_B}{dt} = L \frac{di}{dt} \quad (1.1.6)$$

Combinando le equazioni precedenti si ha:

$$V(t) = L \frac{di(t)}{dt} \Leftrightarrow i(t) = \frac{1}{L} \int V(t) dt \quad (1.1.7)$$



Nel calcolo della 1.1.3, abbiamo omesso la legge di Lenz, poichè per parametrizzare l'induttore il segno

"- " complica inutilmente i calcoli essendo assegnato dalla polarizzazione del circuito esaminato.

Essa al contrario, prende un importanza notevole in presenza di fenomeni di **Induttanza**:

Definizione 1.1.2 (Induttanza).

L'induttanza è la proprietà dei circuiti elettrici tale per cui la corrente (intesa variabile nel tempo) che li attraversa induce una **Forza ElettroMotrice** (f.e.m.) Indotta che si oppone alla variazione di corrente, per la legge di Lenz. In questi scenari (vedi Trasformatore Elettrico ad esempio), 2 circuiti sono accoppiati magneticamente tra di loro attraverso 2 induttori, si ottiene quindi che, la variazione del flusso magnetico da parte del primo induttore, crea una **Forza ElettroMotrice** (f.e.m.) indotta contraria:

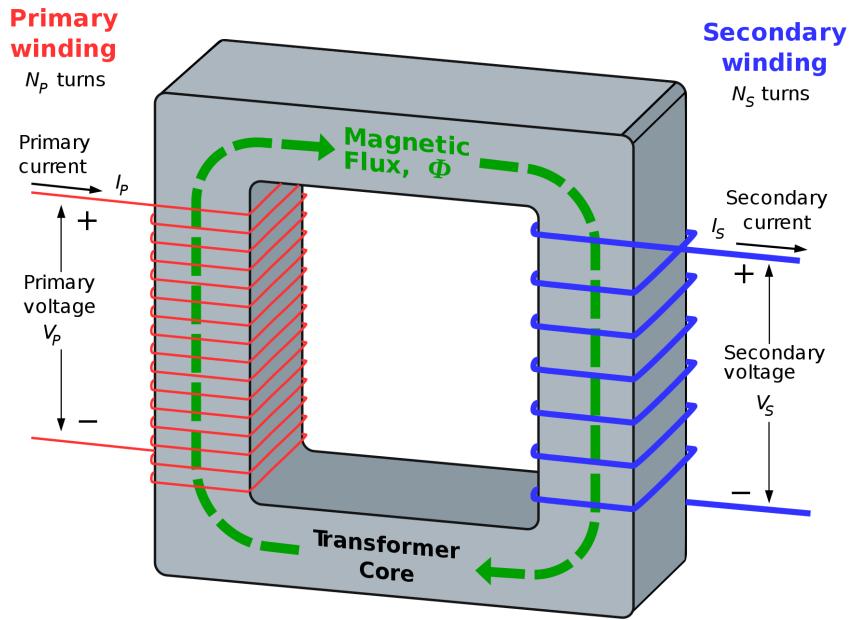
$$-\frac{d\Phi_B}{dt} = \mathcal{E} = V \quad (1.1.8)$$

Dove \mathcal{E} è la **Forza ElettroMotrice** (f.e.m.) indotta.



Per finire, unendo insieme i concetti di **Induttore Ideale** e **Induttanza** è possibile ottenere la descrizione matematica di un Trasformatore Monofase Ideale (*Trasformatore Monofase*¹):

Figura 1.2: Trasformatore Ideale



Definizione 1.1.3 (Trasformatore Ideale Monofase).

Il Trasformatore è una macchina elettrica, basata sul fenomeno dell'induzione elettromagnetica, destinata a trasformare, tra il circuito primario (ingresso) e il circuito secondario (uscita) del trasformatore, i fattori tensione e corrente della potenza elettrica.

Trasferisce quindi energia elettrica da un circuito elettrico a un altro che ha una tensione diversa, accoppiandoli induttivamente, senza che siano a contatto tra loro gli avvolgimenti del trasformatore. Il trasformatore è una macchina reversibile.

Il Trasformatore è costruito affinché il circuito Primario e il Secondario condividano lo stesso campo magnetico, e quindi lo stesso flusso Φ_B .

Nel caso ideale, quindi, si ha che:

$$\Phi_B = \Phi_{B_P} - \Phi_{B_S} \quad (1.1.9)$$

¹Senso di misura del secondario opposto nell'articolo a quello qui presentato

Siccome tale flusso varia nel tempo, induce nei due avvolgimenti le f.e.m.([Induttanza](#)):

$$e_p = N_1 \frac{d\Phi_B}{dt}; e_s = -N_2 \frac{d\Phi_B}{dt} \quad (1.1.10)$$

I segni non sono entrambi concordi a causa del verso del flusso, che nel primario è concorde e nel secondario discorde (e qui torna la forza di Lenz).

Viste dai morsetti del trasformatore, abbiamo che le tensioni istantanee (dovute alle correnti presenti nei 2 rami) sono pari a:

$$\begin{cases} V_P = R_P I_P + L_{PP} \dot{I}_P - L_{SP} \dot{I}_S \\ V_S = R_S I_S - L_{PS} \dot{I}_P + L_{SS} \dot{I}_S \end{cases} \quad (1.1.11)$$

Dove R_P e R_S rappresentano le resistenze equivalenti viste dai morsetti (i carichi).

Con i coefficienti di mutua induttanza che si ricavano dalla Def di [Induttore Ideale](#):

$$\begin{aligned} L_{pp} &= \frac{N_P \Phi_{B_P}}{i_P} & L_{ps} &= \frac{N_S \Phi_{B_P}}{i_P} \\ L_{sp} &= \frac{N_P \Phi_{B_S}}{i_S} & L_{ps} &= \frac{N_S \Phi_{B_S}}{i_S} \end{aligned}$$



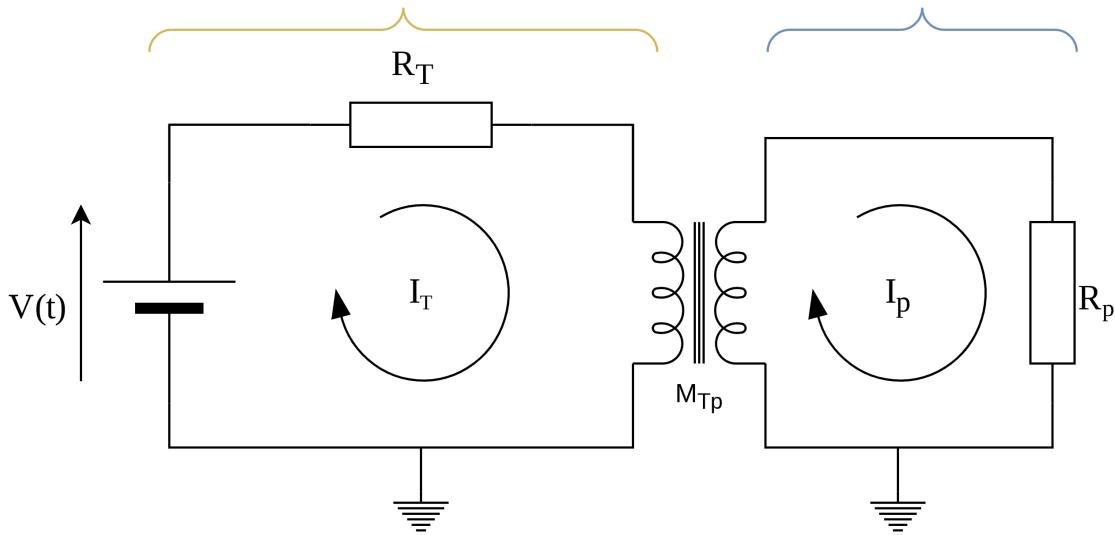
Arrivati a questo punto abbiamo gli strumenti necessari per poter modellare matematicamente l'esperimento.

1.1.2 Modellazione Fisica

Usando i concetti esposti, vediamo ora una buona approssimazione con modello lineare della dinamica tra **Bobina e Plasma**.

Come descritto nell'articolo di Romero et al., «Real time current profile control at JET», è possibile modellare la dinamica **Bobina - Plasma** come fosse il circuito di un **Trasformatore Ideale Monofase**:

Figura 1.3: Circuito Equivalente Bobina/Plasma
 Transformatore/Primario Plasma/Secondario



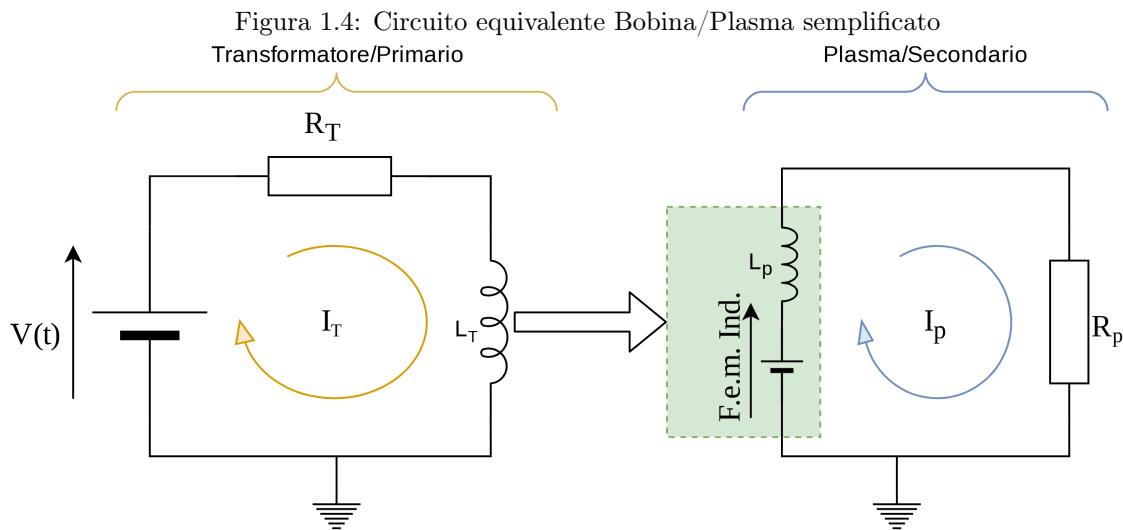
- $V(t)$:= Tensione di controllo della Corrente I_T
 - I_T := Corrente del Trasformatore
 - R_T := Resistenza equivalente Trasformatore
 - I_p := Corrente di Plasma
 - R_p := Resistenza di Plasma
 - M_{Tp} := Coefficiente di Induttanza Trasformatore → Plasma

Difformità dalla realtà Nella realtà R_p e M_{Tp} sono dei parametri che variano in funzione dello stato del plasma (Temperatura, Energia, Evoluzione dell'esperimento, ecc...), ma per ovvie ragioni di difficoltà nel riprodurre in laboratorio simili *Non-Linearità*, noi prenderemo per costanti questi parametri.

1.1.3 Semplificazione Primario - Secondario

Sempre dallo stesso articolo di Romero et al., «Real time current profile control at JET», si evince che è possibile modellare questa relazione tra i 2 circuiti prendendo in considerazione **solo** le forze di induzione dovute alla corrente che il Primario trasferisce sul Secondario, ciò permette di semplificare ulteriormente il circuito trascurando le correnti indotte dal Plasma dentro la Bobina del Primario.

Usando queste osservazioni si ottiene quindi il circuito equivalente:



- $V(t)$:= Tensione di controllo della corrente I_T
- I_T := Corrente del Trasformatore
- R_T := Resistenza equivalente Trasformatore
- I_p := Corrente di Plasma
- R_p := Resistenza di Plasma
- M_{Tp} := Coefficiente di Induttanza Trasformatore → Plasma
 - L_T := Induttanza equivalente Trasformatore
 - L_p := Induttanza equivalente Plasma

Usando questa semplificazione dall'equazione 1.1.11 del trasformatore ignoriamo che la tensione sul Primario è influenzata dal Secondario; in questa condizione il circuito del Primario evolve come sistema indipendente rispetto al Secondario, e la sua evoluzione influenza il Secondario.

1.1.4 Dal circuito alla dinamica

Analizziamo ora la dinamica del circuito 1.1.3.

Dinamica di Plasma

Iniziando dal calcolare la dinamica nella corrente di Plasma usando la [Seconda legge di Kirchhoff \(legge delle maglie\)](#):

$$F_{em} = L_p \dot{I}_p + I_p R_p$$

Sappiamo inoltre che la F_{em} , grazie all'[Induttanza](#) del circuito è pari a:

$$F_{em} = -M_{Tp} \dot{I}_T$$

Da cui otteniamo:

$$-M_{Tp} \dot{I}_T = L_p \dot{I}_p + I_p R_p \quad (1.1.12)$$

Da notare la somiglianza con l'equazione del trasformatore 1.1.11.

Dalla eq 1.1.12 possiamo notare che rendere la corrente di Plasma costante, equivale a fissare una F_{em} di riferimento costante, ed essendo problematico misurare la corrente direttamente nel plasma reale, usiamo la **Tensione di Loop**, che nel caso del trasformatore equivale alla tensione di uscita del Trasformatore.

Dinamica del Trasformatore

Vediamo ora la dinamica della corrente del Trasformatore, in funzione della *Tensione di controllo*:

Sempre grazie alla [Seconda legge di Kirchhoff \(legge delle maglie\)](#) scriviamo:

$$V(t) = L_T \dot{I}_T + I_T R_T \quad (1.1.13)$$

1.1.5 Funzione di Trasferimento

Nella sezione 1.1.4 abbiamo ottenuto la dinamica istantanea dei 2 rami del circuito 1.1.3, ricaveremo ora le funzioni di trasferimento dei 2 rami e troveremo la dinamica del Trasformatore.

Funzione di Trasferimento Plasma

Partendo dall'equazione 1.1.12, supponendo condizioni iniziali nulle, abbiamo:

$$sI_p L_p + I_p R_p = -sI_T M_{Tp} \Rightarrow I_p(sL_p + R_p) = -sI_T M_{Tp} \Rightarrow I_p(s) = \frac{-sM_{Tp}}{(sL_p + R_p)} I_T(s)$$

La cui, funzione di trasferimento risulta essere:

$$\frac{I_p(s)}{I_T(s)} = \frac{-sM_{Tp}}{(sL_p + R_p)} \quad (1.1.14)$$

Funzione di Trasferimento Trasformatore

Similmente, partendo dall'equazione 1.1.13 si ricava:

$$sI_T L_T + I_T R_T = V(s) \Rightarrow I_T(sL_T + R_T) = V(s) \Rightarrow I_T(s) = \frac{1}{(sL_T + R_T)} V(s)$$

La cui, funzione di trasferimento risulta essere:

$$\frac{I_T(s)}{V(s)} = \frac{1}{(sL_T + R_T)} \quad (1.1.15)$$

Osservazione 1.1.1. La funzione di trasferimento che abbiamo ottenuto rispecchia il circuito dell'esperimento, noi però abbiamo un sistema che si controlla in corrente e che ha un uscita in corrente potremmo allora riassegnare l'ingresso come:

$$V(t) = I_{ref}(t) \cdot R_T \Rightarrow I_{ref}(t) = \frac{V(t)}{R_T}$$

Ottenendo così:

$$I_T(s) = \frac{R_T}{(sL_T + R_T)} I_{ref}(s) = \frac{1}{(s\frac{L_T}{R_T} + 1)} I_{ref}(t) \quad (1.1.16)$$

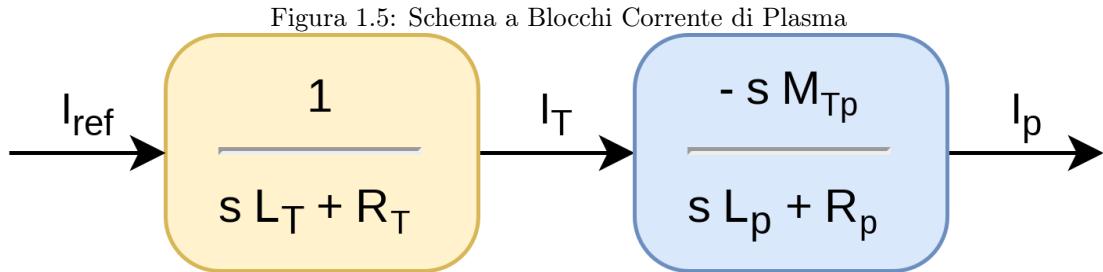
In questa forma possiamo osservare che, per $I_{ref}(t) = I_{ref}$ costante, superato il transitorio, si ottiene $I_T = I_{ref}$, da cui abbiamo che la corrente sul trasformatore viene attuata con un certo ritardo dovuto alla costante di tempo $\tau = \frac{L_T}{R_T}$ \square

Funzione di Trasferimento Complessiva

Connettendo in serie i 2 blocchi e tenendo conto dell'osservazione 1.1.1 si ottiene:

$$P(s) = \frac{I_p(s)}{I_{ref}(s)} = \frac{I_T(s)}{I_{ref}(s)} \cdot \frac{I_p(s)}{I_T(s)} = -\frac{sM_{Tp}}{(sL_p + R_p)(sL_T + R_T)} \quad (1.1.17)$$

Che vista come schema a blocchi diventa:



Essendo in uscita il segno delle correnti inverso dall'ingresso, possiamo eliminare questo fastidioso problema di segno in molti modi, quello che si è usato da qui in avanti nella realizzazione della tesi e degli esperimenti, è stato banalmente invertire i poli del Secondario del trasformatore, ottenendo quindi come funzione di trasferimento:

$$P_{pos}(s) = -P(s) = \frac{sM_{Tp}}{(sL_p + R_p)(sL_T + R_T)} = \frac{-I_p(s)}{I_{ref}(s)} \quad (1.1.18)$$

1.1.6 Misura del campo Elettrico del Plasma come indice per la Corrente

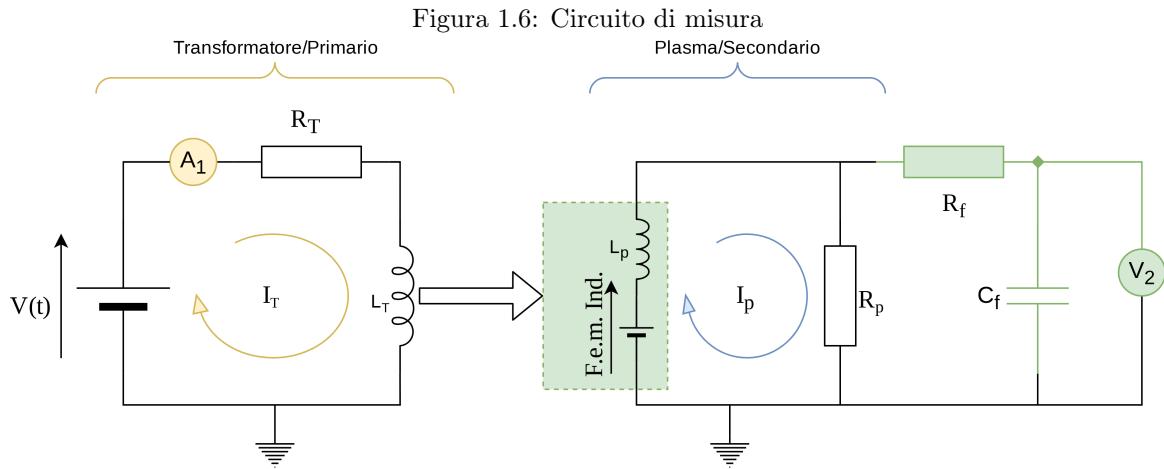
Riprendendo l'equazione della corrente di Plasma 1.1.12, abbiamo che il campo elettrico indotto sul Plasma, ovvero la F_{em} , dipende direttamente dallo stato della corrente di Plasma I_p .

Da essa abbiamo che per I_p costanti, anche la F_{em} è costante.

Nel caso del trasformatore si potrebbe fisicamente misurare la corrente di plasma, simulata dalla corrente sul secondario, ma nel caso di un impianto reale, risulta evidente che è impossibile.

Proprio per questo, come descritto da Cianfarani, «[MAGNETIC DIAGNOSTICS FOR FUSION MACHINES](#)», a pagina 12, si usa come indice di misura Tensione di giro V_{loop} ($\Rightarrow F_{em}$ nel nostro esperimento).

Perciò il circuito reale usato nell'esperimento della tesi, comprensivo di misure è:



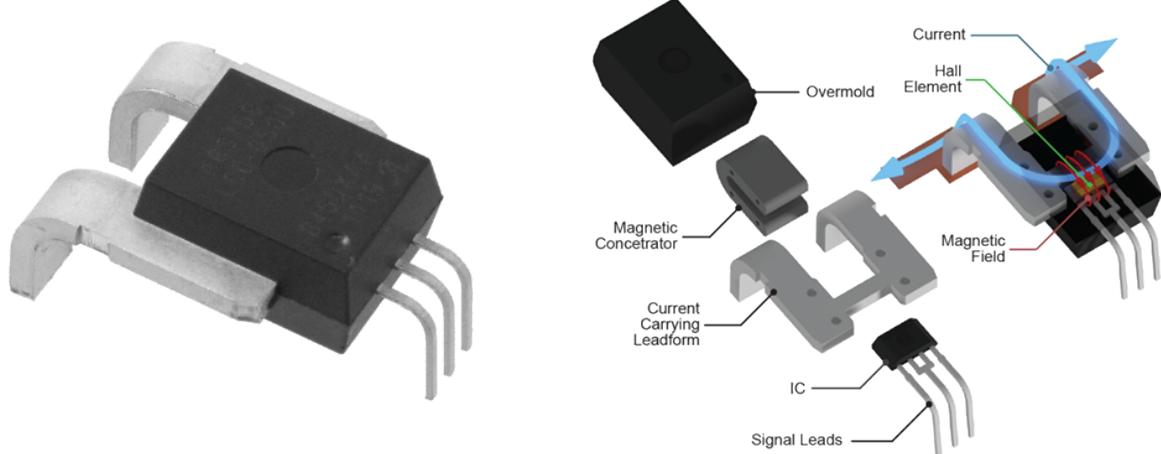
Abbiamo quindi, tra ingresso e uscite i parametri di:

- I_T misurata dal sensore di corrente A_1 [Trasduttore di Corrente](#).
- F_{em} misura dal voltmetro V_2 (che è realizzata da un Pin analogico del [μControllore](#)) e che rappresenta un indice per la corrente I_p quando essa è costante (eq 1.1.12).
- $V(t)$ generata in base alla legge di controllo, e attuata mediante PWM (Redazione, [Nozioni di PWM](#)) dal driver di corrente [Driver di Corrente - IBT-2](#).

1.2 Trasduttore di Corrente

Come detto nell'introduzione, l'obiettivo della tesi è di controllare la corrente sul secondario usando il campo elettrico del secondario, la misura della corrente scorrente sul primario in tale ottica non sarebbe una misura di interesse. Essendo però un lavoro di ricerca, si è preferito poter misurare la corrente effettivamente circolante nel Primario del trasformatore, così da poter meglio interpretare i dati di misura senza ambiguità.

Figura 1.7: Sensore di Corrente



1.2.1 Sensore scelto

Per misurare la corrente del primario, si è posizionato in serie allo stesso il sensore di Corrente Allegro, *High-Precision Linear Current Sensor*.

La famiglia di sensori ha in generale le seguenti caratteristiche:

Tabella 1.1: Riassunto caratteristiche

Bandwidth:	120 kHz
Output rise time :	4.1 μ s
Ultralow power loss:	100 $\mu\Omega$ Resistenza Interna
Single supply operation	4.5 to 5.5 V
Extremely stable output offset voltage	

In oltre, delle tante varianti presenti, si è scelto di usare la ACS770LCB-100B-PFF-T, le cui caratteristiche chiave sono:

Tabella 1.2: ACS770LCB-100B-PFF-T Particolarità

Primary Sampled Current:	± 100 A
Sensitivity Sens (Typ.)	20(mV/A)
Current Directionality	Bidirectional
T_{OP}	-40 to 150 ($^{\circ}\text{C}$)

L'ampio margine di misura, e la robustezza alle variazioni di temperatura rendono rendono il dispositivo perfetto per misurare i nostri esperimenti.

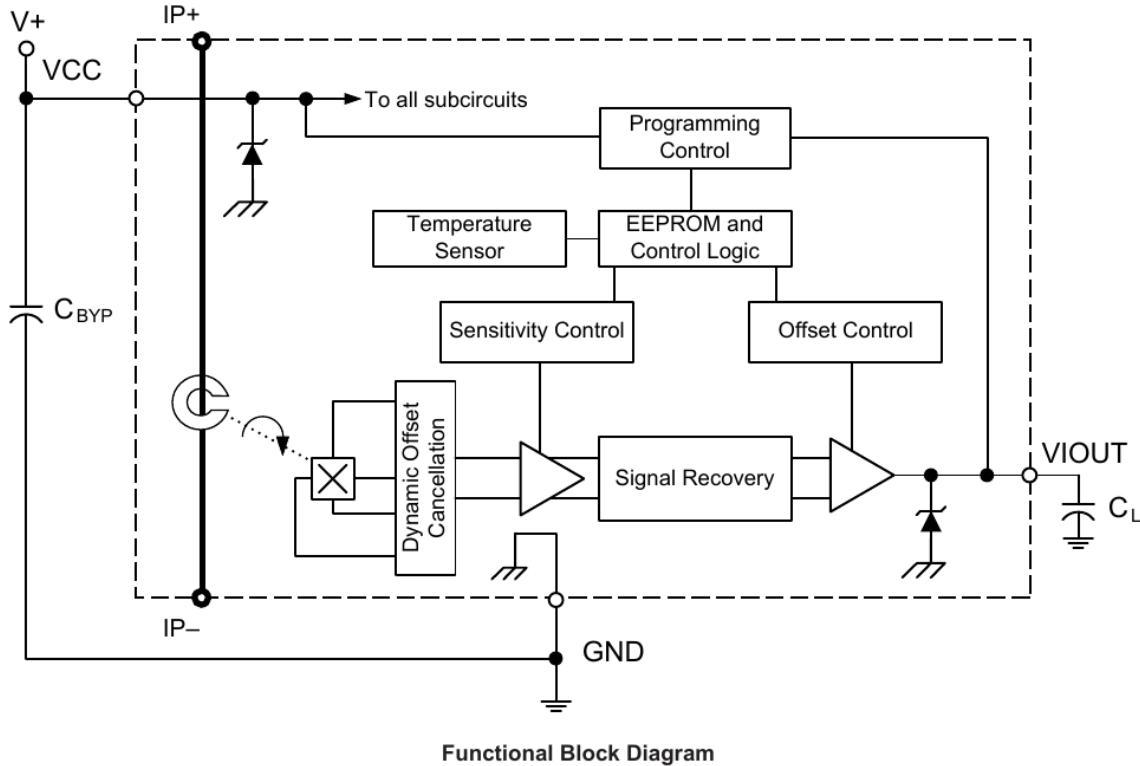
L'ampio margine di misura permette di comprendere tutti i possibili valori di corrente ottenibili in laboratorio, rendendo il prototipo adatto a scopi futuri.

1.2.2 Criticità

Unico punto dolente è il suo principio di funzionamento: essendo il sensore basato su un l'effetto Hall, ovvero una misura diretta del campo magnetico indotto dalla corrente nel conduttore, è importante tenere distante il sensore dal Trasformatore Centrale che nei suoi momenti di massimo flusso di corrente, genera ovviamente un campo magnetico non indifferente.

1.2.3 Funzionamento Interno

Figura 1.8: Schema a Blocchi



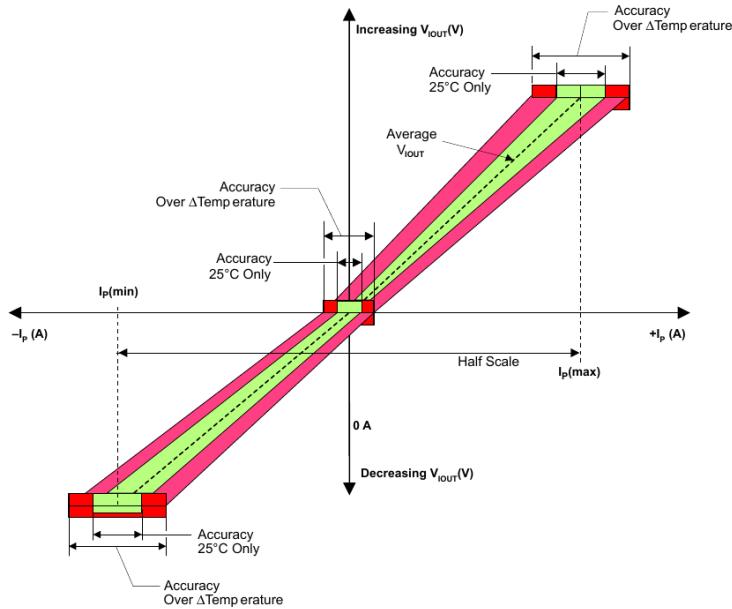
Functional Block Diagram

Tra le caratteristiche chiave dell'Allegro, [High-Precision Linear Current Sensor](#), troviamo il disaccoppiamento fisico tra la corrente da misurare e il circuito di misura.

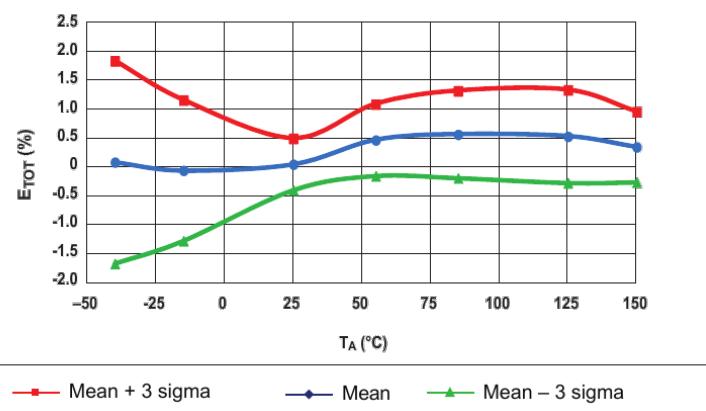
Questa caratteristica chiave, garantisce la salvaguardia del circuito logico a valle, dai possibili eventi catastrofici a monte.

Esso è inoltre fornito di sensori di temperatura e sistemi di *Signal Recovery* che permettono all'hardware stesso di compensare parzialmente *Non-Linearità* termiche e nella misura dell'effetto Hall, ottenendo un output assimilabile a un segnale lineare:

Figura 1.9: Sensibilità



Come si può vedere dal grafico, gli errori sono tanto più marcati quanto maggiore è la corrente da misurare, ma leggendo dal datasheet risulta che il costruttore avverte che l'errore dipende si fortemente dalle temperature di esercizio dell'esperimento, come è possibile vedere nel seguente grafico:

Figura 1.10: Temperatura/NonLinearità
Total Output Error versus Ambient Temperature

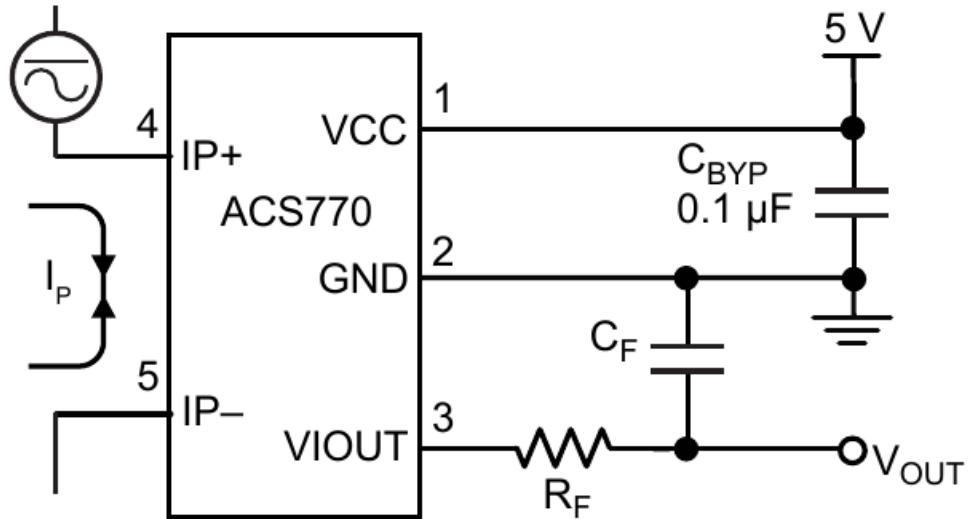
Esso però non è mai, neanche negli esperimenti più sfortunati, superiore al $\pm 2\%$.

Anzi, alle temperature $\approx 25^\circ$, si mantiene contenuto tra $\pm 0.5\%$.

1.2.4 Connessione elettrica

La connessione del sensore è particolarmente semplice, richiedendo esternamente solo un alimentazione stabilizzata e portando subito in uscita la misura.

Figura 1.11: Collegamento dal Datasheet



Rispetto allo schema proposto dal datasheet, però, si è anche deciso di omettere il filtro passabasso sulla **VIOUT**, questa scelta è stata presa per minimizzare il più possibile ritardi di misura della corrente istantanea, poiché le dinamiche del sistema sul secondario, come visto, sono di tipo derivativo, e quindi estremamente rapide.

1.2.5 Misura

Il trasduttore, misura della corrente sotto forma di tensione, la quale varia in base alla **Sensibilità** del modello in uso. Avendo noi il ACS770LCB-100B-PFF-T, il datasheet riporta:

Tabella 1.3: Parametri di **Interesse**

Primary Sampled Current:	$\pm 100 \text{ A}$
Sensitivity Sens (Typ.)	20(mV/A)
Current Directionality	Bidirectional

Ciò implica che la corrente misurata, è calcolabile come:

$$I_{read} = \frac{V_{Read}[V]}{V_{sense}[V/A]}$$

Rimozione Offset Essendo però il device ad alimentazione singola (0–5V), ma la corrente misurabile Bi-direzionale, sorge la necessità di spostare gli 0A a una tensione superiore agli 0V.

Il datasheet riporta che $V_{offset} = \frac{V_{cc}}{2} \approx 2.5V$. Da cui deriva che la vera misura di corrente è:

$$I_{read} = \frac{V_{Read} - V_{offset}}{V_{sense}} \frac{V}{[V/A]} \quad (1.2.1)$$

Al fine di poter misurare l'offset effettivo, durante il set-up viene eseguito a esperimento fermo una misura dell'offset attuale, usando la [Macro per calcolo Offset](#).

Il risultato della computazione, oltre ad essere usato nel controllo è inviato al computer per la post elaborazione dei dati nei grafici.

Analisi Sensibilità Usando nell'esperimento un ADC a 10Bit con tensione di riferimento a 5V, abbiamo che la massima sensibilità del μ C, ovvero il suo bit meno significativo è pari a:

$$V_{step} = \frac{V_{cc}}{2^{10} - 1} = 4,887mV \quad (1.2.2)$$

Il che equivale a una **Sensibilità di Corrente** del μ Controllore pari a:

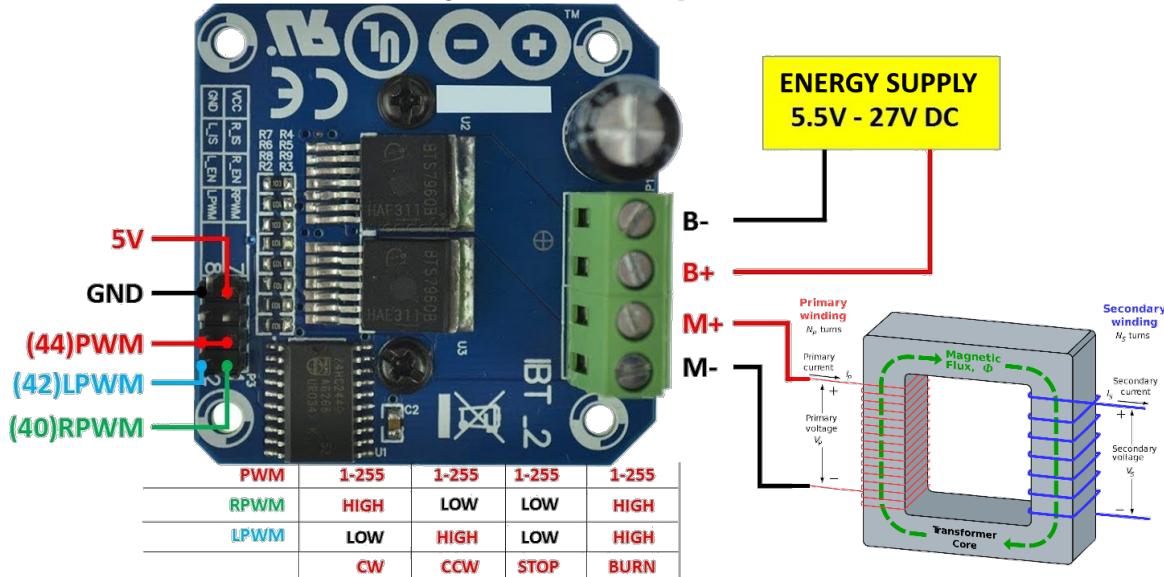
$$I_{step} = \frac{V_{step}}{V_{sense}} = 244,379mA \quad (1.2.3)$$

Il ché rende la misura buona per osservare cosa stia accadendo, ma sicuramente non sufficientemente densa da poterla usare come parametro ingresso di controllo.

1.3 Driver di Corrente - IBT-2

Per l'attuazione del controllo di corrente nella bobina primaria del trasformatore, è stato usato il driver di corrente Handsontec, [BTS7960 High Current 43A H-Bridge Motor Driver](#).

Figura 1.12: IBT-2 TopView.



Esso non è un comune Ponte-H integrato: per poter gestire potenze superiori è stato costruito usando 2 Half-Bridge (Infineon, [BTS7960B High Current PN Half Bridge NovalithIC T](#)) collegati assieme mediante una opportuna logica per ricreare un normale Ponte-H.

Questa scheda in particola ha prestazioni interessanti per gli scopi di questa tesi, i principali sono elencati di seguito:

Tabella 1.4: IBT-2 Specifiche

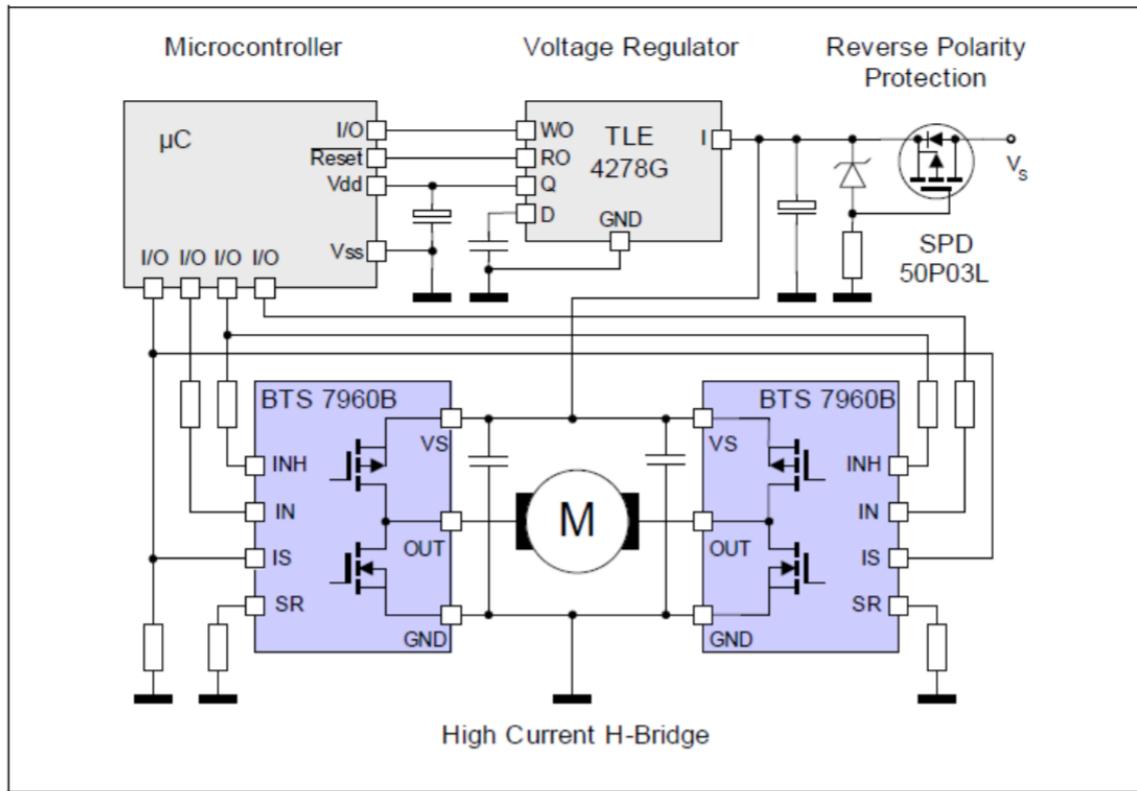
Power Input Voltage:	6 – 27 V
Peak current:	43 A
Massima Frequenza di PWM:	25 kHz
Protezione Sovra Tensioni	
Disaccoppiamento Ingresso di Potenza/Logica di controllo	

Di particolare interesse per l'esperimento è proprio la corrente di picco gestibile: avendo le dinamiche del sistema tempi inferiori ai 5 secondi, poter reggere correnti di picco così elevate rende la scheda perfetta per i nostri scopi.

1.3.1 Schema Elettrico

Al suo interno il driver è composto da 2 Half-Bridge Infineon, [BTS7960B High Current PN Half Bridge NovalithIC T](#), connesse secondo lo schema:

Figura 1.13: IBT-2 Schema Elettrico



Il μ C è protetto dal carico connesso all'interno del BTS7960b, lasciando al μ C solo il compito di controllare i segnali.

1.3.2 Connessione di Controllo

Il driver permette 2 modalità di funzionamento:

Doppio PWM Modalità operativa che richiede l'uso di 2 PWM

Ciascun PWM controlla uno dei 2 Half-Bridge, e per evitare di bruciare i driver devono essere controllati singolarmente, il vantaggio di questa configurazione è la possibilità di usare 2 frequenze di controllo diverse.

Singolo PWM Modalità operativa classica di un normale Ponte-H

In questa modalità, la porta nand presente sulla scheda attua la logica di controllo opportuna per governare i 2 Half-Brige come fossero un normale Ponte-H.

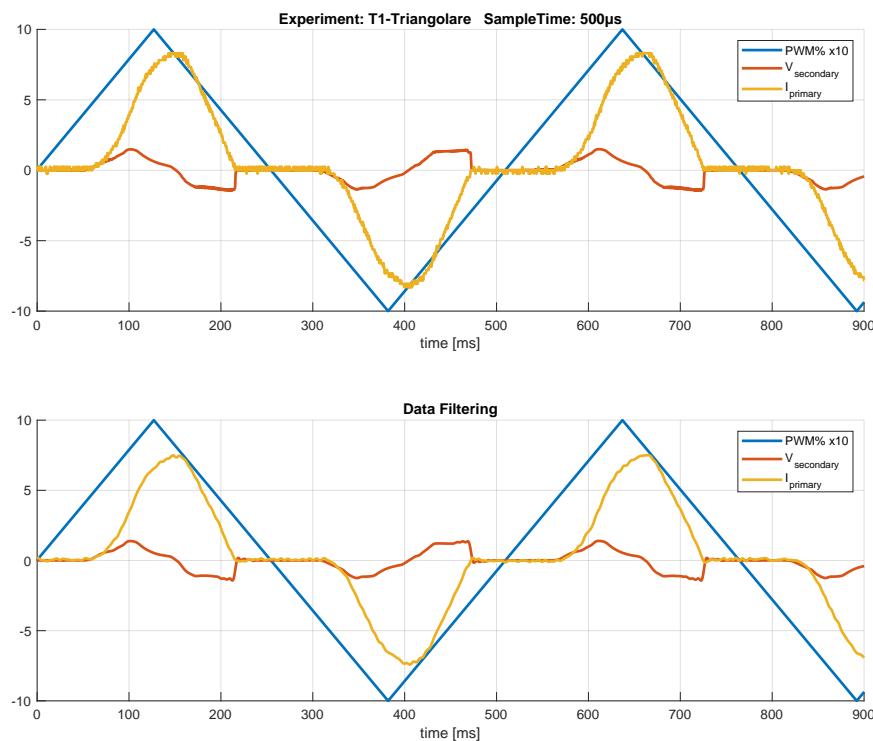
Per il nostro esperimento si è scelto di usare il collegamento **Singolo PWM** così da evitare spiacevoli sorprese e avere il PWM di controllo sempre sincronizzato.

1.3.3 Benchmark del Driver

Il driver sulla carta à buone prestazioni, ma non sono descritte le sue *Non-Linearità*, per farle risaltare si sono effettuati 2 esperimenti usando differenti input di controllo:

1. Onda Triangolare Periodica

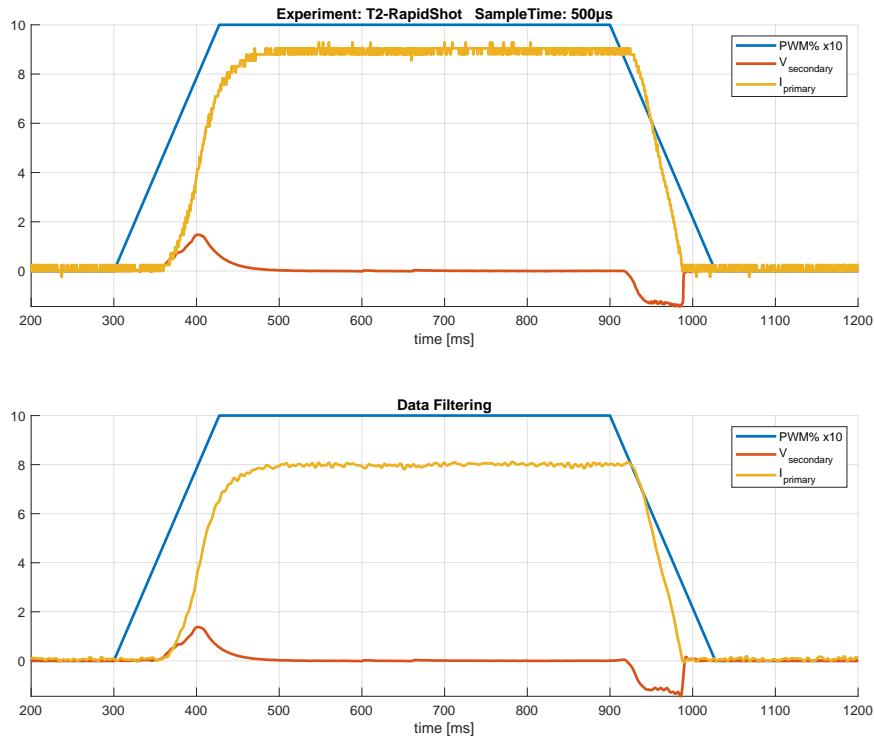
Figura 1.14: Onda Triangolare



Dead-Zone Inferiore L'onda triangolare si presta bene per far risaltare la problematica della Dead-Zone Inferiore, infatti in tutti gli intorni in cui il segnale passa per 0, è possibile vedere come la corrente non vari minimamente, è però possibile notare che i 2 lati non sono simmetrici tra di loro, questo è facilmente spiegabile dal fatto che il primo ha una condizione iniziale $\neq 0$ e di fatto stiamo ancora osservando l'esaurimento del transitorio, la soglia di Dead-Zone Inferiore è quindi calcolata vedendo il primo valore di PWM per cui il sistema risponde a destra degli 0.

2. Onda Trapezoidale Periodica (RapidShot)

Figura 1.15: Onda Trapezoidale



Dead-Zone Superiore Con questo secondo segnale, si vuole mettere in evidenza il ritardo durante la discesa della rampa, pari a circa 20ms (*guarda 900ms*), questo ritardo è in realtà dovuto da una seconda Dead-Zone presente però ai Duty-Cycle alti del PWM.

Sapendo della presenza di queste 2 Dead-Zone, andando a vedere l'inizio della risposta del secondario, essendo il primario troppo poco sensibile come abbiamo avuto modo di analizzare nella sezione "[Trasduttore di Corrente](#)", è possibile risalire ai valori limite della Dead-Zone di questo Ponte-H:

Tabella 1.5: Fasce della Dead-Zone

Dead-Zone Superiore	± 210 PWM
Dead-Zone Inferiore	± 120 PWM

Capitolo 2

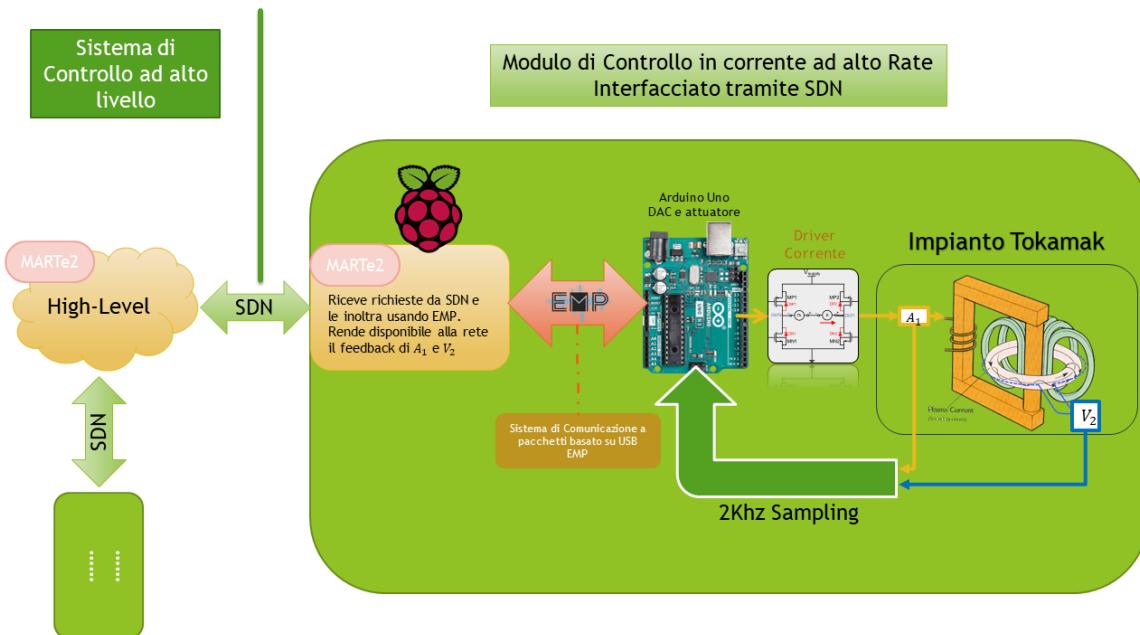
Architettura di Sistema

In questo capitolo verranno affrontati gli aspetti progettuali e i problemi realizzativi per poter campionare l'esperimento e trasferire i dati campionati a un dispositivo esterno collegato alla scheda principale (l'Arduino).

2.1 Architettura ad alto livello

Il progetto finale ha come obiettivo la realizzazione di un architettura di controllo per le bobine poloidali presenti nei reattori tokamak.

Figura 2.1: Architettura di controllo



Lo schema proposto realizza l'obiettivo per una singola bobina, il progetto finale prevederà la ripetizione in serie del medesimo schema per il numero di bobine necessarie.

Dallo schema risulta evidente che tutti i componenti visti nel capitolo "[Elementi Costitutivi](#)" si relazionano con lo stesso μ Controllore: l'**Arduino Uno**.

Figura 2.2: Scheda Arduino Uno



Per riportare i dati fuori e ricevere il riferimento da inseguire nella V_2 , è stata realizzata da me la libreria [Embedded Message Pack\(EMP\)](#), essa è stata scritta in C++ usando classi template, e progettata affinché possa essere Cross-Platform.

Il suo compito specifico, in questo progetto, è di mettere in comunicazione l'**Arduino Uno** con un nodo *MARTE2* installato su di una **Raspberry Pi**.

Quest'ultimo nodo ha il compito di mettere in rete il feedback dell'esperimento, e comunicare all'**Arduino Uno** eventuali cambio di riferimento. Questo ultimo tratto è realizzato mediante il protocollo **SDN**, che viaggia sopra Ethernet e dà garanzie Real-time.

Nella sua forma finale, il progetto prevede la riproduzione in serie di questo schema di controllo per arrivare a controllare tutte le bobine poloidali presenti in un tokamak.

EMP - Libreria di Comunicazione Seriale Embedded Message Pack



Embedded Message Pack(EMP) nasce con l'obiettivo di standardizzare un protocollo e creare una libreria C++ basata su classi Template, che permetta di automatizzare e standardizzare tutto il lavoro di programmazione necessario all'invio/ricezione di dei pacchetti dal formato Pre-Concordati tra 2 Device connessi Peer2Peer (Nessuna pretesa di network-ing). ([Embedded Message Pack\(EMP\)](#))

Il raggiungimento dei suoi obiettivi, si sposa con la possibilità di supportare altre features interessanti:

Multiple-Package Il protocollo di comunicazione che si è deciso di usare per EMP ha permesso di estendere il suo funzionamento e permettere il trasporto, attraverso lo stesso mezzo, di **pacchetti di tipologia e dimensione diversa** all'interno della stessa libreria, evitando al contempo di inviare per ogni pacchetto più byte di quelli strettamente necessario. ⇒ **Alta Efficienza**

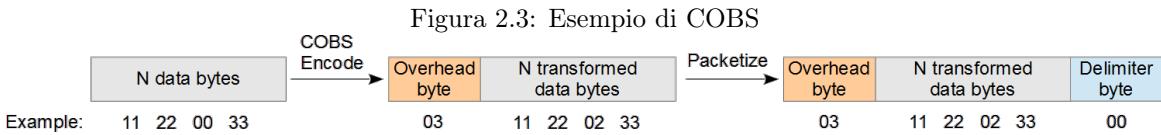
Zero Tempo di negoziazione Sempre grazie al protocollo di comunicazione, EMP è adatto ad un uso ‘Streaming’, questo perché non è necessario alcuna fase di sincronizzazione iniziale o durante la trasmissione in caso di perdita di dati, in aggiunta a ciò, EMP è in grado di scartare pacchetti errati in maniera trasparente all’utilizzatore. Tutto questo grazie al protocollo che **Auto-delimita i singoli pacchetti**. ⇒ **Trasparenza Totale**

Responsabilità Le uniche responsabilità a carico degli utilizzatori sono il riempimento dei pacchetti e la definizione degli stessi tra i 2 estremi della comunicazione.

Consistent Overhead Byte Stuffing (COBS)

Il protocollo di comunicazione che permette l’invio di **pacchetti diversi e senza fasi di negoziazione** alla base della libreria è **COBS**(IEEE, *Consistent Overhead Byte Stuffing (COBS)*).

Si tratta di un algoritmo per la codifica di byte, progettato per essere al tempo stesso efficiente e non ambiguo, che permette la definizione di *data-pack frame* **Auto-delimiti**.



2.1.1 Metodo di codifica

L’algoritmo di COBS trasforma una stringa arbitraria di byte, ciascuno dei quali ha un Range di valori da **[0:255]** in una nuova stringa di byte dove però ogni byte va da **[1:255]**. La dimensione della nuova stringa è sempre pari alla dimensione della precedente + 1.

L’obiettivo di questo metodo di codifica è di eliminare tutti i possibili byte **\0** dal pacchetto in maniera reversibile.

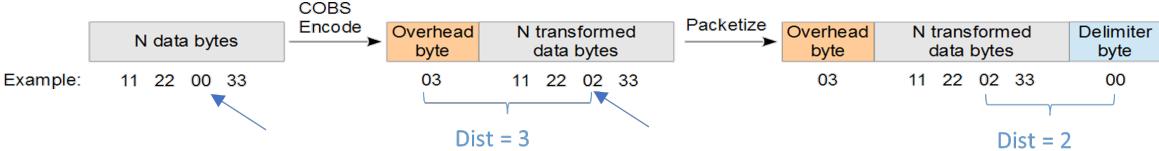
Questo processo rende il carattere **\0** (byte zero) ottimo candidato per essere usato come terminatore di stringa durante l’invio, rendendo un pacchetto COBS-Encoded mai ambiguo e sempre

Auto-delimitato.

L'algoritmo di codifica consiste nel:

1. Inserire un byte '\0' all'inizio del pacchetto
 2. Individuare tutti gli altri byte '\0'
 3. Inserire un byte '\0' alla fine del pacchetto
 4. Sostituire tutti gli '\0' con la distanza dal successivo '\0' nella stringa, ignorando l'ultimo

Figura 2.4: Esempio di COBS con distanza



La versione usata per questo progetto, che comunque non ha l'obiettivo di trasmettere quantità infinite di byte, ha il limite di non poter codificare blocchi di byte che hanno una distanza tra 2 ' $\backslash 0$ ' superiore a 255, questo limite è potenzialmente rimovibile usando un algoritmo più sofisticato.

2.1.2 Caratteristiche chiave

Prima caratteristica vincente della libreria è il suo alto grado di adattabilità, essa infatti per poter funzionare richiede solo **2 informazioni critiche** (e altre di contorno per l'allocazione opportuna dei buffer), esse sono i tipi dei pacchetti in **Input** (*pIn*) e in **Output** (*pOut*) ovvero sia delle strutture in C, con i dati organizzati in base al messaggio da trasferire ([Esempio di definizione Pacchetto in EMP](#)).

In secondo luogo, la codifica Consistent Overhead Byte Stuffing (COBS), che abbiamo appena visto permette di avere pacchetti **Auto-delimitati**, ciò permette quindi di scambiare pacchetti diversi tra loro (sia per tipologia che per lunghezza) lungo lo stesso *Stream* di dati.

L'unica condizione necessaria è che il destinatario sia capace di determinare la tipologia di contenuto

trasportato nel pacchetto semplicemente leggendolo.

Questo può essere facilmente risolto in più modi:

Lunghezza Univoca Ogni possibile pacchetto ha una lunghezza diversa da tutti gli altri, \Rightarrow la lunghezza implica il contenuto

Aggiunta di un campo Tipologia Aggiungendo all'inizio della trasmissione un *type byte*, ovviamente Pre-Concordato, diventa possibile per chiunque sapere come interpretare il contenuto del pacchetto.

Il metodo più universale è sicuramente l'aggiunta di un campo fisso per il tipo, di cui un esempio è visibile nell'appendice al listato 8.12([Definizione Pacchetti Multipli](#)).

In ogni caso, la codifica *Consistent Overhead Byte Stuffing (COBS)* aggiunge 2 byte extra al pacchetto che si vuole inviare, e tanto per la codifica quanto per la decodifica in ricezione, il **costo** è sempre pari a **O(n)**.

Vantaggi:

1. Pacchetti **Self-Delimited**
2. Canale **Multi-Packet ready**
3. Protocollo **Senza Negoziazioni**
4. All'utilizzatore è richiesto solo di Pre-Concordare il formato del pacchetto con l'altro lato dello stream
5. Prerequisiti implementativi minimale (mezzo di comunicazione a bytes di tipo *peer2peer* asincrono)

Svantaggi:

1. Aggiunge 2 byte fissi
2. Richiede O(n) elaborazione sia in codifica che decodifica

2.1.3 Integrità dei pacchetti

Per aumentare ulteriormente i campi d'uso e garantire un layer minimale di **integrità** sui pacchetti in transito, la libreria è stata progettata per includere in maniera trasparente anche un check di errore calcolato usando [*Checksum Calculation 8 Bit \(CRC8\)*](#), aggiunto in trasmissione e rimosso in ricezione.

L'aggiunta e il calcolo del CRC8 viene fatta sul pacchetto non ancora *COBS-Encodato*, ciò garantisce la possibilità di inviare il pacchetto a prescindere da quale sia il risultato del CRC8, e viene quindi verificato dopo aver de-*COBS-Encodato* il pacchetto in ricezione.

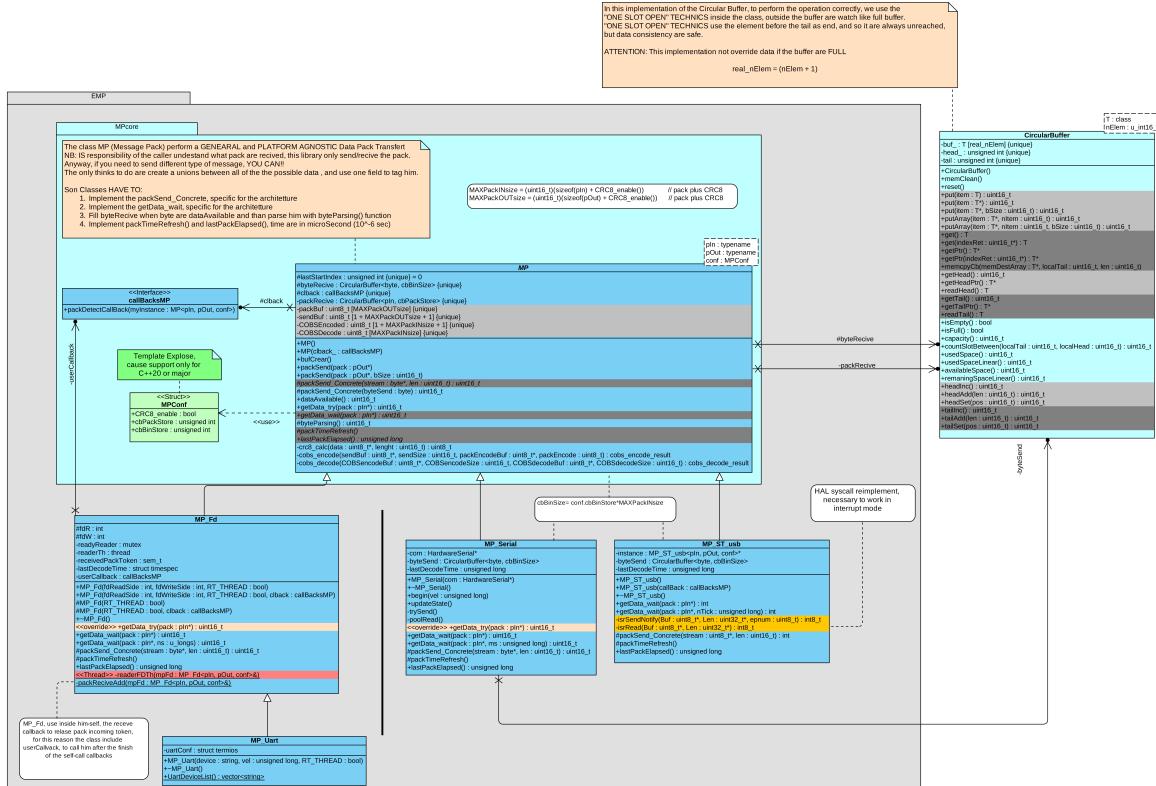
Questa feature deve essere attiva o disattivata da entrambi i lati della libreria (ne consegue che anche lei deve essere pre-concordata tra i 2 estremi).

La libreria, se attiva, è in grado di capire se il pacchetto ha subito degli errori (ovviamente nei limiti del CRC8) e in tal caso scarta il pacchetto ricevuto in maniera totalmente trasparente all'utilizzatore. L'aver usato COBS come sistema di codifica per la trasmissione, garantisce che la decodifica debba avvenire solo nei byte compresi tra 2 zeri, e se questa decodifica presenta un errore, toglie ogni ambiguità sul da farsi poiché il pacchetto viene scartato e si attende un successivo 0 mentre si memorizzano i byte ricevuti nel frattempo.

2.1.4 Struttura del codice

Lo sviluppo del codice è qui riassunto nel **Class Diagram** fatto in UML, del codice:

Figura 2.5: Class Diagram UML



Senza entrare eccessivamente nel dettaglio di Embedded Message Pack(EMP), essendo tutto reperibile nella versione più aggiornata all'interno del repository di *Embedded Message Pack(EMP)*, osserviamo che il codice è diviso in 2 macro blocchi (**MPCore**, **MP**) più una classe di supporto per il buffer circolare.

Questa organizzazione del codice è stata pensata per far compilare su tutte le piattaforme di interesse lo stesso *codice attivo*(codifica e decodifica dei pacchetti + accumulo), e demandare le particolarizzazioni dovute alle varie piattaforme o al mezzo di comunicazione usato nel caso specifico a delle classi figlie.

MPCore Il *Codice attivo* è presente all'interno del package **MPCore**. Tutto il codice contenuto in questo package è scritto in C++11 e per essere compilato necessita di un set minimale di librerie standard, sicuramente presenti in ogni piattaforma di sviluppo.

EMP Il package più generico comprende le classi figlie che concretizzano le operazioni di invio e ricezione, facendo poi elaborare i byte al codice che ereditano dalla classe **MP**.

Le classi sono scritte e pensate per funzionare su una piattaforma specifica, e su essa essere ottimizzate.

Se la piattaforma lo concede come nel caso di Linux, è quindi possibili fattorizzare ulteriormente delle funzionalità comuni e scrivere così, via via, sempre meno codice sicuri che quello in comune, se funziona su una classe, deve funzionare anche per l'altra.

Buffer Circolare La classe *CircularBuffer* è una classe di supporto che implementa un buffer circolare con logica "One Slot Open"(Johnston, *Creating a Circular Buffer in C and C++*) attraverso una classe template.

Il motivo di questa scelta, molto forte e vincolante, essendo principalmente lei la causa per cui tutte le altre sono anch'esse template, è dovuta alla possibilità di istanziare in fase di compilazione, tutta la memoria richiesta per il funzionamento del programma.

Una simile necessità nasce dal dover compilare la classe anche su schede embedded, le quali, notoriamente, non hanno a disposizione un heap in ram spazioso per la memoria dinamica, e beneficiano nelle prestazioni se in fase di compilazione gli indirizzi di memoria sono fissi.

2.1.5 Logica di Comunicazione

La libreria è pensata per automatizzare la trasmissione e la ricezione di 2 tipologie di pacchetti

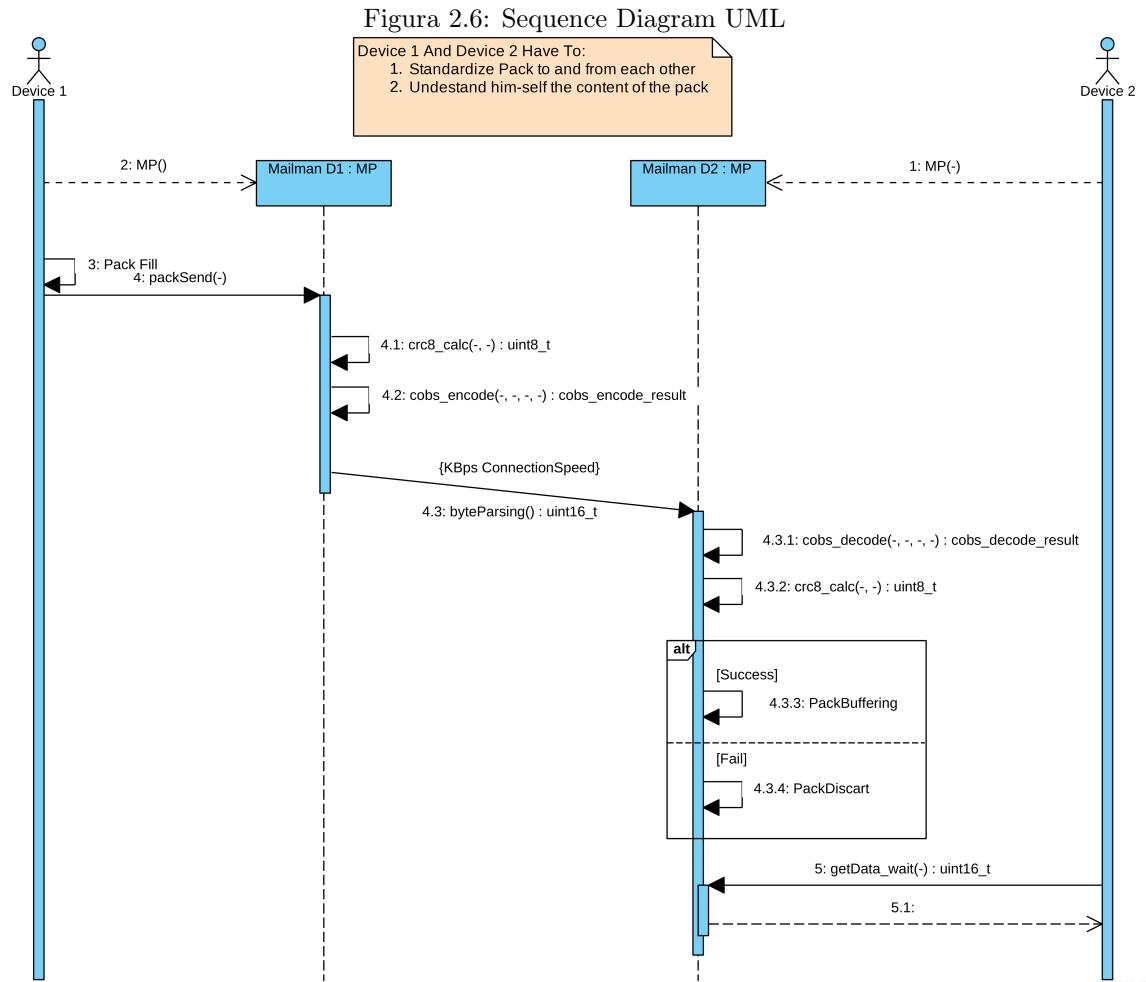
Pre-concordati tra 2 realizzazioni di **MP**, ovviamente non necessariamente sullo stesso dispositivo.

Come riportato in Appendice B ([7.3.1](#)), la definizione dei pacchetti è pensata per essere molto comoda e naturale. Inoltre usando un campo per distinguere il tipo di pacchetto e una union per unire in un'unica area pacchetti differenti, è anche possibile gestire la comunicazione di pacchetti multipli tra i 2 lati della comunicazione, basterà invertire l'ordine dei tipi tra le 2 classi ai 2 estremi della comunicazione.

Il riempimento corretto dei dati nel pacchetto, il calcolo della size utile e il riconoscimento in ricezione del pacchetto trasmesso, è tutto a carico dell'utilizzatore della libreria, essa garantisce la corretta ricetrasmissione e di svegliare un ascoltatore al ricevimento di un pacchetto, qualunque sia la sua lunghezza.

2.1.6 Code Flow

Vediamo ora come la libreria si frappone tra 2 device che vogliono comunicare mediante questo sequence diagram:



In questo esempio generico i 2 device non sono definiti, ovviamente nessuno dei 2 può istanziare concretamente una classe **MP** (essendo una classe virtuale), ma come detto prima, tutti i *Codici attivi* sono racchiusi lì dentro.

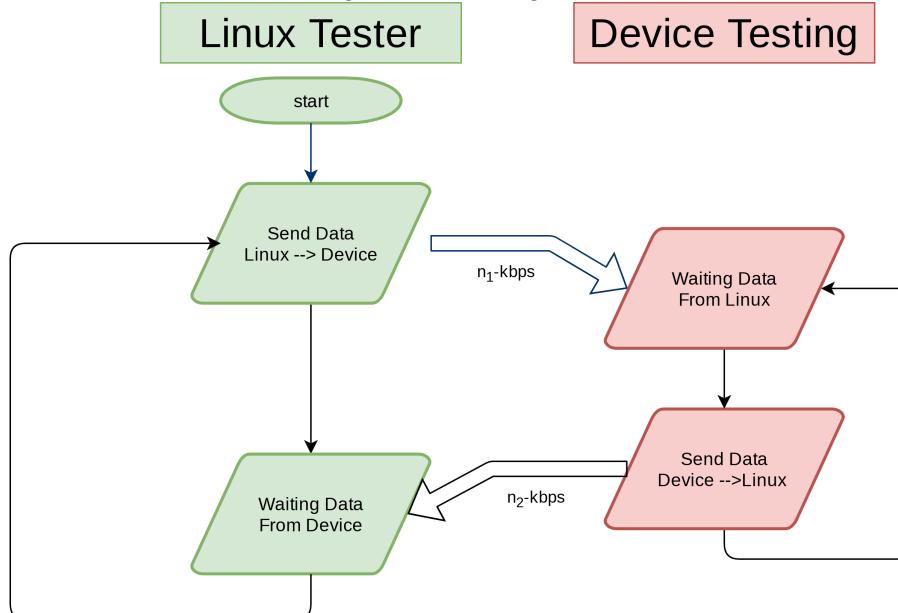
Come si può osservare, eccetto il riempire i dati, inviarli e attendere che arrivi qualcosa, per gli attori il lavoro finisce subito, internamente alla libreria, invece, a parti invertite, abbiamo il calco-

lo del CRC8 ([Checksum Calculation 8 Bit \(CRC8\)](#)) e la codifica usando COBS (IEEE, [Consistent Overhead Byte Stuffing \(COBS\)](#)), e l'omologo dall'altro lato, dopo aver ricevuto i byte, procede alla de-codifica e check per l'integrità.

2.1.7 Test di Codifica/Decodifica su ogni Device

La serie di passi descritta nel [Code Flow](#) è stato testato su i vari dispositivi per cui la libreria è stata sviluppata usando il seguente schema:

Figura 2.7: Testing Flow



I test vengono avviati da **Linux** e puntano a testare la perdita la correttezza della trasmissione usando Embedded Message Pack(EMP), e il funzionamento di Encoding e Decoding delle classi sulle entrambe le architetture.

Ad ora, sulle 3 Piattaforme di sviluppo (Linux, Arduino, STM32) i test sono stati un pieno successo.

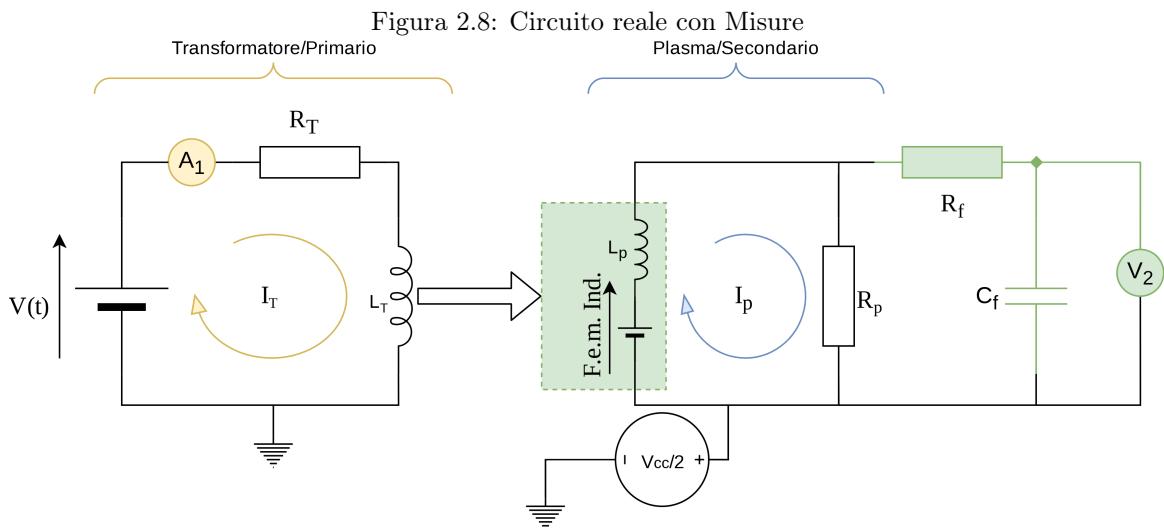
2.2 Online Sampling

Come descritto in figura 2.1, il sistema controlla internamente la corrente, ma comunica con il mondo fuori l'attuale stato della bobina.

Per comunicare i Sample (e ricevere le Reference) è stata infatti sviluppata Embedded Message Pack(EMP)(Sezione 2.1).

Come visto nella sezione "Misura del campo Elettrico del Plasma come indice per la Corrente", il circuito di misura è quello riportato in figura 1.1.6.

E, come per il Trasduttore di Corrente, in realtà anche il voltmetro V_2 possiede un offset a $\frac{V_{cc}}{2}$, aggiunto per poter misurare tanto le correnti positive, quanto quelle negative.



Controllo e misure avvengono alla massima frequenza che l'Arduino è riuscito effettivamente a gestire, ovvero 0.5ms (2Khz), nei quali campiona lo stato del sistema, genera il nuovo valore del PWM per il controllo di $V(t)$, e invia i dati attraverso Embedded Message Pack(EMP).

2.2.1 Interconnessione μ Controllore \Leftrightarrow Companion

Per raggiungere gli obiettivi descritti nella sezione dell'[Architettura ad alto livello](#), i seguenti pacchetti multipli sono stati concordati tra le 2 parti:

```

1 // #####
2 // ##### Companion to Arduino #####
3 // #####
4 struct newRef {
5     int16_t newRef;
6 } __attribute__((packed));
7
8 struct setUpPackAsk {
9     int8_t padding;
10} __attribute__((packed));
11
12 enum LinuxSendType : uint8_t { newRefType, askType };
13
14 struct _packLinux2Ard {
15     LinuxSendType type;
16     union {
17         struct newRef ref;
18         struct setUpPackAsk ask;
19     };
20} __attribute__((packed));
21typedef struct _packLinux2Ard packLinux2Ard;

```

Listing 2.1: Pacchetti Companion \Rightarrow μ Controllore

```

1 // #####
2 // ##### Arduino to Companion #####
3 // #####
4 struct sample {
5     int16_t pwm;
6     int16_t V2_read;
7     int16_t Isense_read;
8     int16_t err;
9 } __attribute__((packed));
10
11 struct setUpPack {
12     int16_t V2_mean;           // Adc read
13     int16_t Isense_mean;      // Adc read
14     int16_t dt;                // Time in us (10^-6)
15 } __attribute__((packed));
16 enum ardSendType : uint8_t { sampleType, setUpPackType };
17
18 struct _packArd2Linux {
19     ardSendType type;
20     union {
21         struct sample read;
22         struct setUpPack setUp;
23     };
24} __attribute__((packed));
25typedef struct _packArd2Linux packArd2Linux;

```

Listing 2.2: Pacchetti μ Controllore \Rightarrow Companion

Essi permettono al Companion di conoscere tutto quello che sta succedendo nel μ C, con un piccolo ritardo dovuto alla trasmissione ed eventuali ritardi interni (nel caso di Linux dovuti allo scheduler).

Le informazioni riguardanti offset e parametri dell'esperimento (`struct setUpPack`), sono raccolti a macchina spenta, all'accensione della scheda nel [Setup del Controllore](#).

Successivamente la scheda evolve per tic di 0.5ms, dove nel tempo morto resta in ascolto di eventuali pacchetti di richiesta da parte del Companion all'interno del [Loop di Controllo](#).

2.2.2 Storage su file delle informazioni

Le informazioni ricevute dal Companion, vengono salvate all'interno di un file di testo contenente 2 tabelle, disposte una sotto l'altra.

La prima delle 2, tabula i dati del pacchetto `struct setUpPack`, il quale risulta utile per poter interpretare i dati successivamente.

Tabella 2.1: Salvataggio di "struct setUpPack"

$V_{2_{mean}}$	$I_{sense_{mean}}$	dt
510	510	500

La seconda invece è lo streaming dei dati `raw`, ottenuti dalla scheda:

Tabella 2.2: Salvataggio di "struct sample" \forall dt

PWM	$V_{2_{read}}$	$I_{sense_{read}}$	e
0	509	510	1
0	509	511	1
...
0	509	510	59
44	510	510	36
48	518	510	28
53	514	510	32
59	513	510	33
...

Essi sono salvati sotto forma di testo ascii normale, separando i campi con un `'tab'`.

2.3 Post Elaborazione con Matlab

I dati salvati su Hard Disk, come visto della sezione [Storage su file delle informazioni](#), vengono presi in ingresso da Matlab e convertiti dal valore del DAC.

2.3.1 Conversioni Dati

Per ottenere la tensione reale letta il cambio è molto semplice:

$$V_{read} = ADC_{Read} \cdot V_{step} = ADC_{Read} \frac{V_{cc}}{2^{10} - 1} = ADC \cdot 4,887mV$$

Partendo da questa equivalenza, per la corrente del primario usiamo l'equazione [1.2.1](#), mentre la tensione sul secondario necessita solo della cancellazione dell'offset:

$$V_2 = V_{read} - V_{2_{offset}} \quad (2.3.1)$$

2.3.2 Filtraggio senza distorsioni di fase

Essendo i dati letti affetti naturalmente da rumore, essi vengono post-filtrati per rendere i dati più puliti e mettere in evidenza le tendenze del sistema. Questo filtraggio è stato realizzato usando un *Filtro digitale a senza distorsione di fase*(Matlab, [Zero-phase digital filtering](#)), come fa intuire il nome, la caratteristica di questo filtro è che non introduce nessun ritardo o anticipazione al segnale filtrato, permettendo di confrontarlo direttamente con i dati puri, eliminando rumori a bassa ed alta frequenza.

Questo filtraggio viene eseguito direttamente nell'importazione della tabella salvata con la struttura vista nella sezione [Storage su file delle informazioni](#), di cui il codice è in appendice [Parsing Tabelle](#).

Capitolo 3

Stima del modello del sistema

In questo capitolo viene affrontato il problema della stima dei parametri del modello reale, in un equivalente lineare matematico, necessario per le simulazioni future e validare i risultati matematici precedenti.

Partendo dall'esperimento con l'[Onda Trapezoidale Periodica \(RapidShot\)](#) per eccitare il sistema senza raggiungere la saturazione della tensione V_2 (ricordiamo che in uscita abbiamo un derivatore filtrato, quindi ogni onda quadra genera automaticamente una forte derivata), si è puntato a ricavare un modello lineare prima della corrente in I_T e successivamente della tensione V_2 .

Essendo il driver di corrente intrinsecamente non lineare, specie nei pressi della Dead-zone, il cui taglio si è visto non essere netto come si spererebbe, si è cercato di ottenere un buon fitting nei transitori del sistema, dove la dinamica è più lineare e "pulita".

Questa stima è ovviamente qualitativa e permette di dimensionare tempi di risposta e sistemi di controllo, le non linearità presenti hanno però dinamiche molto rapide, e grazie a questa proprietà, il controllo pensato lineare che vedremo in seguito, continua ad essere valido anche nel caso reale. Il modello stimato è P_{pos} (eq [1.1.18](#)), per rendere più intuitivi i risultati.

3.1 Metodo di stima automatico

Il metodo di stima automatico passa per l'uso del *System Identification Toolbox* di Matlab, ed è usato allo scopo di ottenere una funzione di trasferimento SISO tra l'ingresso del duty-cycle (PWM) e la tensione sul secondario V2. Avendo a disposizione anche i dati della corrente sul primario, anche se poco sensibili, si è deciso di dividere la stima in 2 fasi: prima si è stimato il modello della corrente nel primario, e a causa della sua scarsa sensibilità di misura (risultato 1.2.3) si è usato il segnale ricostruito come ingresso per la stima della dinamica della tensione sul secondario.

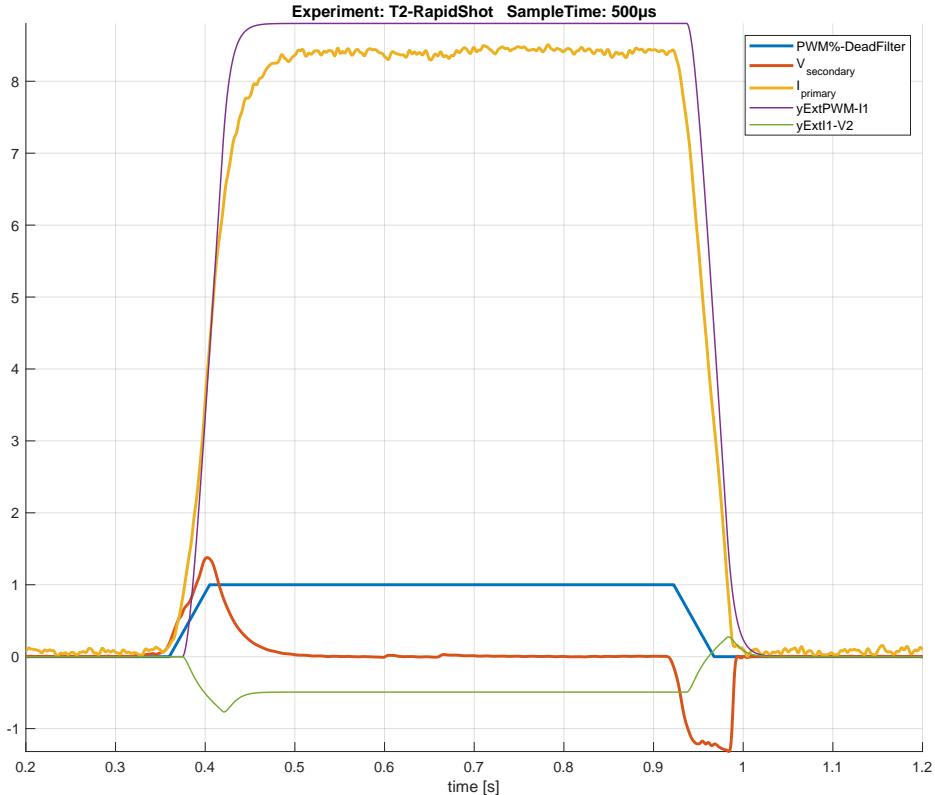
Il numero di *Poli* e *Zeri* di entrambe le funzioni di trasferimento sono state scelte partendo dall'analisi teorica descritta nella sezione [Modellazione Fisica](#), e il codice è presente in appendice a [Stima corrente Primario automatica](#).

Tabella 3.1: Stime parametri con *System Identification Toolbox*

Funzione	Input	Input
$\hat{P}_{p_{wm}I_1}(s) = \frac{970}{s+110}$	PWM	I_1
$\hat{P}_{I_1V_2}(s) = -\frac{0.44 s+11}{s+200}$	I_1	V_2
$\hat{P}_{p_{wm}V_2}(s) = -\frac{430 s+11+4}{s^2+310 s+22+4}$	PWM	V_2

Il risultato è che la stima del primario ha un aspetto simile a quello che ci aspettavamo, ma il secondario già sbaglia completamente aggiungendo un guadagno fisso nel numeratore che non dovrebbe esistere, e un segno "−" che non dovrebbe esistere, stando noi stimando P_{pos} . Di seguito la simulazione delle prime 2 funzioni di trasferimento (essendo la 3° la loro moltiplicazione ha poco senso mostrarla in simulazione sugli stessi dati).

Figura 3.1: Simulazione con stima automatica



Analizzando il grafico abbiamo che i primi 3 segnali (spessi), sono i dati reali dell'esperimento filtrati ([Filtraggio senza distorsioni di fase](#)), mentre le altre 2 linee rappresentano la simulazione usando la stima del Toolbox come modello.

Risulta facile vedere che la corrente del primario è accettabile anche se migliorabile, ma la funzione di trasferimento stimata per il secondario è completamente sbagliata, anche se dimensionalmente è simile.

Usando queste stime come base si sono trovati a mano dei coefficienti che meglio approssimano l'andamento delle curve.

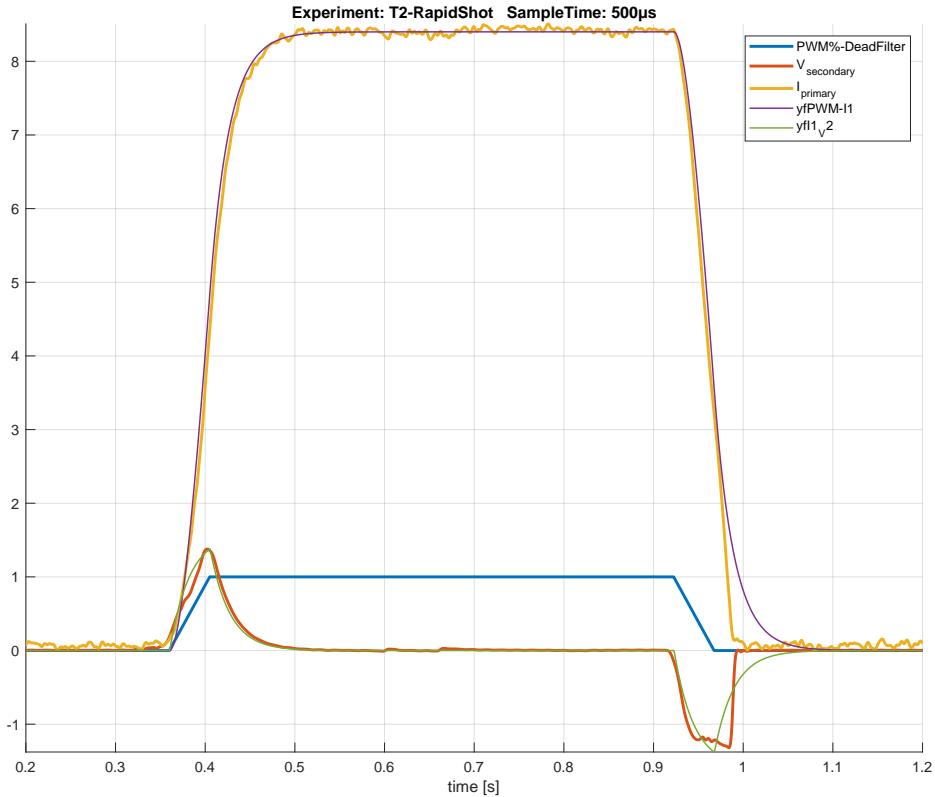
3.2 Tuning dei coefficienti

Partendo dai modelli generati da *System Identification Toolbox* e riportati in tabella 3.1, si sono "limati" i dati a mano per migliorare la stima dei parametri, di seguito è riportata la tabella con i valori migliorati:

Tabella 3.2: Stime parametri tarati a mano

Funzione	Input	Input
$\tilde{P}_{p_{wm}I_1}(s) = \frac{8.4}{0.022s+1}$	PWM	I_1
$\tilde{P}_{I_1V_2}(s) = \frac{8.5 \times 10^{-3}s}{3.6 \times 10^{-4}s+1}$	I_1	V_2
$\tilde{P}_{p_{wm}V_2}(s) = \frac{9.1 \times 10^3 s}{s^2 + 2.8 \times 10^3 s + 1.3 \times 10^5}$	PWM	V_2

Figura 3.2: Simulazione con parametri tarati a mano



In particolare nella Dead-Zone inferiore il modello non risulta accurato, essendo predominanti le dinamiche non lineari, e ciò si vede particolarmente bene sul fronte di discesa del sistema, ma nel complesso la funzione trovata è una buona candidata per uno studio teorico e simulativo del sistema prima di passare alla fase implementativa reale.

3.3 Benchmark

In questa sezione alcune frazioni interessanti di un esperimento lungo con vari segnali e metteremo a confronto il modello trovato con i risultati reali.

Figura 3.3: Esperimento di verifica della stima 1

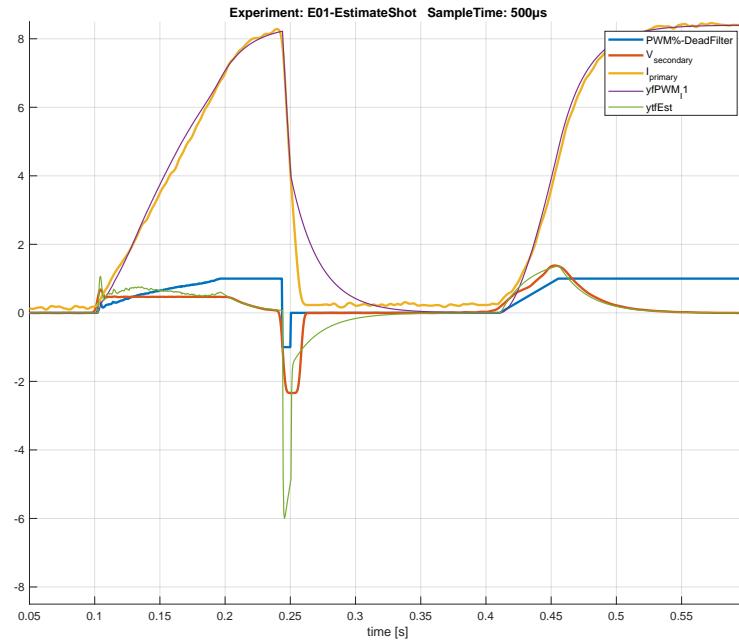


Figura 3.4: Esperimento di verifica della stima 2

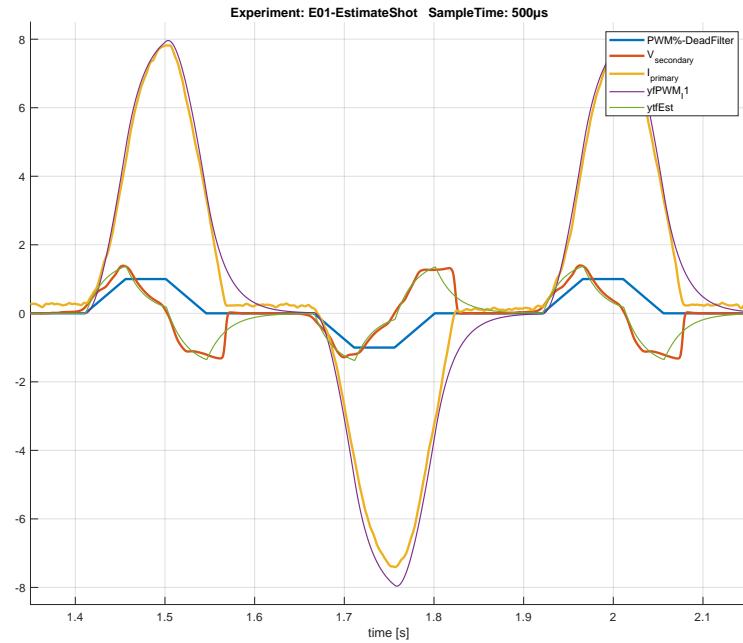
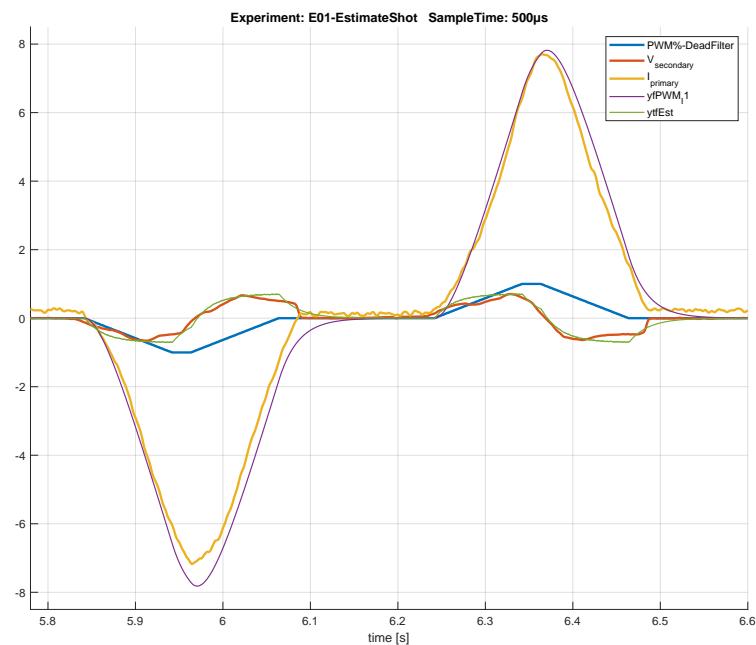


Figura 3.5: Esperimento di verifica della stima 3



Da questi grafici risulta chiaro che, anche se non perfetto, e certamente trascurando delle *Non-Linearità* del sistema originale, la dinamica simulata è rappresentativa dell'andamento reale del sistema complessivo. La funzione di trasferimento :

$$\tilde{P}_{p_{wm}V_2}(s) = \frac{9.1 \times 10^3 s}{s^2 + 2.8 \times 10^3 s + 1.3 \times 10^5} \quad (3.3.1)$$

verrà da ora considerata la funzione stimata del modello e usata nelle simulazioni.

Capitolo 4

Modello teorico di Controllo

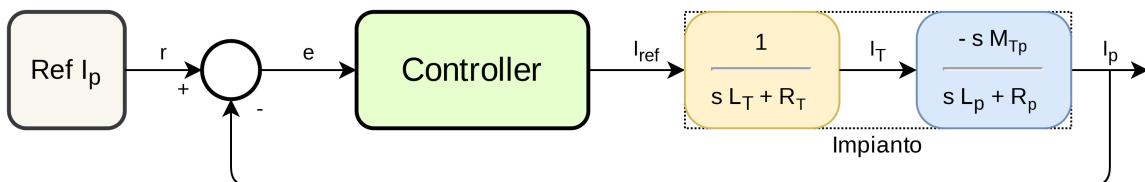
Se lo si desidera, utilizzare questo spazio per inserire un breve riassunto di ciò che verrà detto in questo capitolo. Inserire solo i punti salienti.

4.1 Obiettivo di controllo

Come accennato nella sezione "[Misura del campo Elettrico del Plasma come indice per la Corrente](#)", attraverso il controllo e attuazione della corrente nel Primario del trasformatore, si punta controllare la corrente presente sul primario del trasformatore, e sempre come detto nella sezione "[Misura del campo Elettrico del Plasma come indice per la Corrente](#)", non essendo possibile misurare la corrente di plasma direttamente, si usa la sua misura indiretta che passa per la F_{em} .

In questa tesi l'obiettivo che ci si è ripromessi di raggiungere è la creazione di un controllo ad errore nullo sulla corrente di plasma.

Figura 4.1: Sistema da controllare a blocchi con $P(s)$



Come visto però nella sezione, "[Misura del campo Elettrico del Plasma come indice per la Corrente](#)", la misura a nostra disposizione il voltmetro la V_2 , che equivale alla diagnostica di V_{loop} in un impianto reale.

Dall'equazione della dinamica del plasma 1.1.12 siamo in grado di ricavare che:

$$V_2 = I_p \cdot R_p = F_{em} - L_p \cdot \dot{I}_p$$

La quale, trascurando la dinamica del filtro RC di misura (che comunque è $\geq f_{tic}$ di acquisizione e di conseguenza ha una dinamica estremamente più rapida del segnale da misurare) permette di ricavare, invertendo il segno della tensione come della corrente per avere i segni positivi, la funzione di ingresso-uscita dalla I_{ref} alla V_2 pari a:

$$V_2(s) = I_p(s) \cdot -R_p \Rightarrow V_2(s) = P_{pos}(s) \cdot I_{ref}(s) \cdot R_p \quad (4.1.1)$$

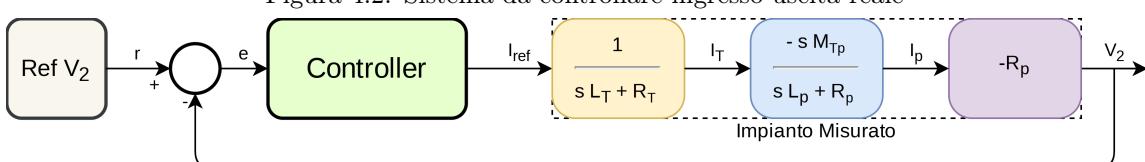
Da cui la funzione di trasferimento complessiva diventa:

$$\frac{V_2(s)}{I_{ref}(s)} = P_{pos}(s) \cdot R_p \quad (4.1.2)$$

E il riferimento di corrente di plasma, diventa un riferimento di tensione secondo l'equazione di conversione:

$$V_{2ref} = I_{p_{ref}} \cdot R_p \quad (4.1.3)$$

Figura 4.2: Sistema da controllare ingresso-uscita reale



4.2 Teorema del valore iniziale e del valore finale

Prima di procedere con il calcolo del controllore per i nostri scopi, è bene ricordare il [Teorema del valore iniziale](#) e il [Teorema del valore finale](#) della Trasformata di Laplace ([«Trasformata di Laplace»](#)):

Teorema 4.2.1 (Teorema del valore iniziale).

Se una funzione reale f ha trasformata razionale $F(s)$ con grado del denominatore maggiore del grado del numeratore (vale comunque sotto ipotesi ancora più larghe , purché F sia non razionale e $f(0^+)$ esista) allora:

$$f(0^+) = \lim_{s \rightarrow \infty} s \cdot F(s) \quad (4.2.1)$$

■

Teorema 4.2.2 (Teorema del valore finale).

Se una funzione reale f ha trasformata razionale $F(s)$ con grado del denominatore maggiore del grado del numeratore e radici del denominatore (poli) nell'origine e o a parte reale negativa, allora:

$$\lim_{t \rightarrow \infty} f(t) = \lim_{s \rightarrow 0} s \cdot F(s) \quad (4.2.2)$$

■

Questi 2 Teoremi chiave, permettono di risolvere comodamente le equazioni differenziali, e quindi calcolare in maniera semplice il controllore che permette di ottenere gli obiettivi di controllo che ci si è prefissati di ottenere.

4.3 Controllo a errore nullo

Definiamo ora $P_m(s)$ come l'impianto Reale misurato tra la corrente di riferimento I_{ref} del primario, e la tensione del campo elettrico V_2 sul secondario.

Ne segue che la funzione di trasferimento complessiva è pari a:

$$P_m(s) = \frac{V_2(s)}{I_{ref}(s)} = \frac{sM_{Tp} \cdot R_p}{(sL_p + R_p)(sL_T + R_T)} = \frac{sM_{Tp} \cdot R_p}{s^2 L_p L_T + s(L_p R_T + L_T R_p) + R_p R_T} \quad (4.3.1)$$

Essendo il nostro obiettivo quello di portare $e \rightarrow 0$ per $t \rightarrow \infty$ con riferimenti $r = cost$, ci calcoliamo la funzione sensitività $W_{er}(s)$ (*sensitivity transfer function*) e con il [Teorema del valore finale](#) progetteremo il controllore $C(s)$ che realizza l'obiettivo:

$$W_{er}(s) = \frac{e(s)}{r(s)} = \frac{1}{1 + P_m(s)C(s)}$$

Sensitivity transfer function
(Carnevale, «Performances and robustness»)

$$r(t) = cost \rightarrow r(s) = \frac{K_r}{s}$$

Trasformata di Laplace del riferimento

Da cui, applicando il [Teorema del valore finale](#) otteniamo:

$$\lim_{t \rightarrow \infty} e(t) = \lim_{s \rightarrow 0} s \cdot W_{er}(s) \cdot r(s) = s \cdot \frac{K_r / s}{1 + P_m(s)C(s)} \quad (4.3.2)$$

Osservazione 4.3.1. Al fine di semplificare il design del controllore, fattorizziamo ora $P_m(s)$ e $C(s)$ nella seguente forma, tralasciando così i dettagli dei termini non utili ai fini del design del controllore.

$$C(s) = K_c \cdot \frac{s^{\rho_{cn}} \cdot \prod_{i=1}^{\#Zeri \neq 0} \left(\frac{s}{z_i} + 1 \right)}{s^{\rho_{cd}} \cdot \prod_{i=1}^{\#Zeri \neq 0} \left(\frac{s}{p_i} + 1 \right)} = \frac{K_c}{s^{\rho_c}} \cdot \frac{C_n(s)}{C_d(s)}$$

- K_c Guadagno statico Controllore.
- C_n Polinomio numeratore senza Zeri in 0.
- C_d Polinomio denominatore senza Poli 0.
- ρ_c è il grado del polo in 0 ($\rho_{cd} - \rho_{cn}$).

$$P(s) = K_p \cdot \frac{s^{\rho_{pn}} \cdot \prod_{i=1}^{\#Zeri \neq 0} \left(\frac{s}{z_i} + 1 \right)}{s^{\rho_{pd}} \cdot \prod_{i=1}^{\#Zeri \neq 0} \left(\frac{s}{p_i} + 1 \right)} = \frac{K_p}{s^{\rho_p}} \cdot \frac{P_n(s)}{P_d(s)}$$

- K_p Guadagno statico Impianto.
- P_n Polinomio numeratore senza Zeri in 0.
- P_d Polinomio denominatore senza Poli 0.
- ρ_p è il grado del polo in 0 ($\rho_{pd} - \rho_{pn}$).

✓

Partendo da questa forma, svolgiamo i calcoli dell'equazione 4.3.2:

$$1 + P_m(s)C(s) = 1 + \frac{K_c}{s^{\rho_c}} \frac{C_n}{C_d} \cdot \frac{K_p}{s^{\rho_p}} \frac{P_n}{P_d} = \frac{s^{\rho_c+\rho_p} C_d P_d + K_c K_p C_n P_n}{s^{\rho_c+\rho_p} C_d P_d} \quad (4.3.3)$$

Da cui otteniamo che:

$$s \cdot W_{er}(s) \cdot r(s) = \cancel{s} \cdot \frac{K_r / \cancel{s}}{1 + P_m(s)C(s)} = K_r \cdot \left(\frac{s^{\rho_c+\rho_p} C_d P_d + K_c K_p C_n P_n}{s^{\rho_c+\rho_p} C_d P_d} \right)^{-1} \Rightarrow$$

$$\lim_{t \rightarrow \infty} e(t) = \lim_{s \rightarrow 0} s \cdot W_{er}(s) \cdot r(s) = \lim_{s \rightarrow 0} K_r \cdot \frac{s^{\rho_c+\rho_p} \cdot C_d P_d}{s^{\rho_c+\rho_p} \cdot C_d P_d + K_c K_p C_n P_n}$$

Tenendo presente che $C_d, P_d, C_n, P_n \rightarrow 1$ per $s \rightarrow 0$, poiché gli Zeri sono stati fattorizzati fuori assieme al guadagno Statico (vedi osservazione 4.3.1), abbiamo che gli esiti per l'errore a *Steady state* (Stato Stazionario) dipendono solo dal grado di $\rho = \rho_c + \rho_p$ con $\rho \in \mathbb{Z}$ e sono pari a:

$$e_\infty = \lim_{t \rightarrow \infty} e(t) = \begin{cases} 0 & \text{se } \rho \geq 1 \\ \frac{K_r}{K_c K_p} & \text{se } \rho = 0 \\ \infty & \text{se } \rho \leq 0 \end{cases}$$

Essendo $\rho_p = -1$ a causa dello Zero in 0, abbiamo che il controllore dovrà necessariamente avere $\rho_c = 2$ per ottenere $\rho = 1$. Il grado minimo di controllore di cui necessitiamo è quindi:

$$C(s) = \frac{1}{s^2} \quad (4.3.4)$$

Questo controllore base è complicabile a piacere aggiungendo altri Poli e Zeri stabili, con l'obiettivo di migliorarne le performance della risposta del sistema.

Osservazione 4.3.2. Il controllore descritto in 4.3.4, anche se stabilizza il sistema e lo rende un inseguitore ad errore nullo, **non è stabile**.

Esso infatti ha 2 poli in 0, rendendolo un polo non semplice e quindi causa di instabilità polinomiale.

Questa conseguenza è inevitabile e bene accetta poiché permette di inseguire con errore nullo, per la durata dell'esperimento, il riferimento.

Bisognerà ovviamente tenerne conto in fase di realizzazione poiché con il procedere del tempo dell'esperimento, i numeri dello stato diventeranno sempre maggiori, e ciò potrà portare a problemi di overflow numerici. ✓

La dinamica e_∞ riportata (4.3) è valida \leftrightarrow il sistema a ciclo chiuso $W_{V_2r}(s)$ risulta Asintoticamente Stabile, analizziamo quindi la sua stabilità:

$$W_{V_2r}(s) = \frac{V_2(s)}{r(s)} = \frac{P_m(s)C(s)}{1 + P_m(s)C(s)}$$

Complementary Sensitivity Transfer Function
(Carnevale, «Performances and robustness»)

E la stabilità di questa funzione è data dai poli del suo denominatore, che abbiamo già calcolato 4.3.3, e la cui formula è:

$$\text{Den}(W_{V_2r}) = s^{\rho_c + \rho_p} C_d P_d = s^{\rho} C_d P_d \quad (C_d \text{ e } P_d \text{ senza poli nell'origine } ^1)$$

Partendo dalla conoscenza che P_d è sicuramente composta da poli stabili nel semipiano \mathbb{C}^- (vedi sezione Trasformatore, un Modello di Tokamak), da cui segue che qualunque polo stabile per il controllore (C_d) non destabilizza il sistema e rende valida la dinamica e_∞ riportata prima (4.3), e anzi, il numeratore del controllore può tranquillamente essere instabile, se dovesse servire.

4.3.1 Design di controllore adottato

In questa tesi la forma di controllore che si è scelto di usare è la seguente:

$$C(s) = \frac{K_2}{s^2} + \frac{K_1}{s} + K_p = \frac{K_2 + sK_1 + s^2K_p}{s^2} \quad (4.3.5)$$

La scelta è dovuta alla sua somiglianza con un più classico PID, che in questo caso non sarebbe bastato a raggiungere le specifiche (vedi sezione Controllo a errore nullo), ma come per lui, tramite un lavoro di *Tuning* dei coefficienti è possibile ottenere delle buone prestazioni per il sistema a ciclo chiuso.

Ulteriore vantaggio di questa struttura è che permettere un semplice controllo switching, variando i coefficienti del numeratore, in base alla fase dell'esperimento.

¹Nota Bene: P_d e C_d sono il risultato della fattorizzazione vista sopra nell'osservazione 4.3.1, non possono quindi esserci poli nell'origine in questi 2 termini **per costruzione**

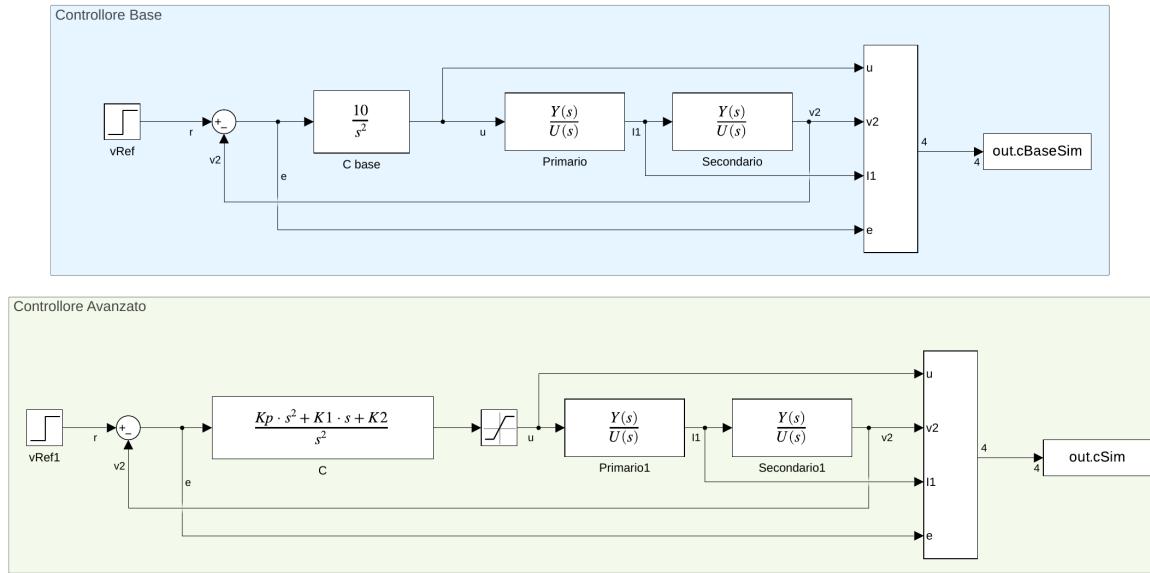
4.4 Simulazione Qualitativa su Simulink

Partendo dal modello stimato che si è ottenuto nel capitolo "Stima del modello del sistema" in cui si è giunti al modello nell'equazione 3.3.1, puntiamo ora a testare il controllo base che abbiamo teorizzato e successivamente la sua evoluzione "PID-style".

Mostreremo prima come si comporta il controllore base 4.3.4, e successivamente modificheremo i parametri (mettendo anche la saturazione per una maggiore aderenza ai dati reali) e mostreremo gli effetti qualitativi sul sistema quando i coefficienti sono presenti o meno e le loro variazioni.

Di seguito i 2 sistemi di controllo usati su Simulink per generare la simulazione:

Figura 4.3: Modelli simulati su Simulink



Il primo ci serve solo per mostrare l'andamento del sistema con un controllo che punta solo a ottenere gli obiettivi di controllo.

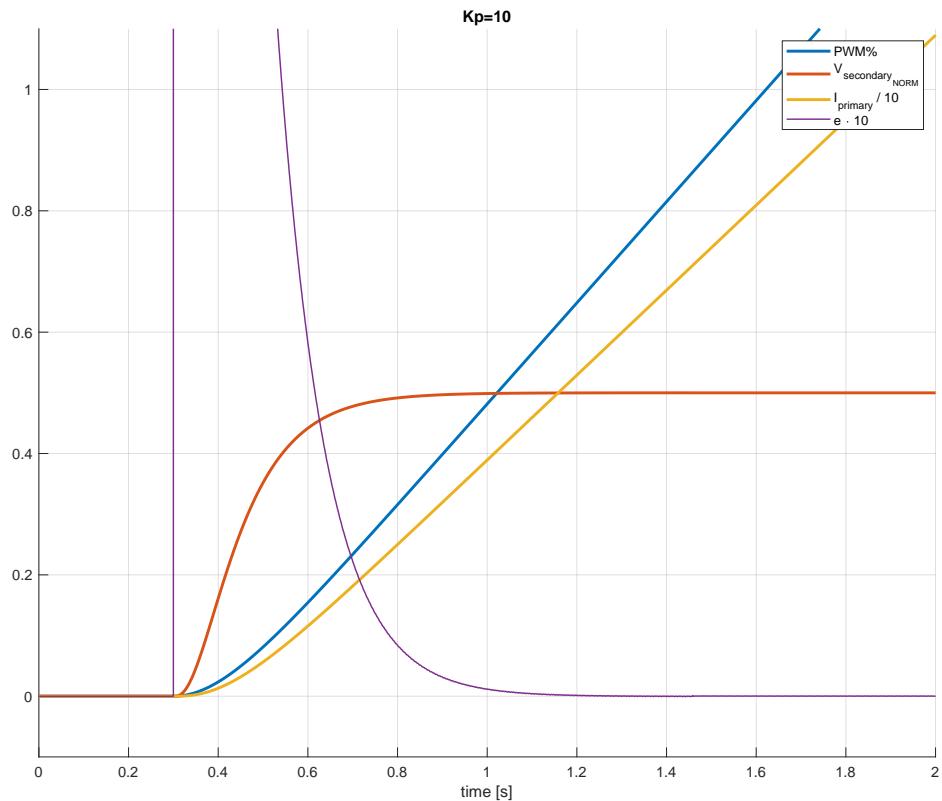
Al contrario il secondo punta ad analizzare la risposta del sistema al variare dei coefficienti con lo scopo preciso di migliorarne le prestazioni. In entrambi i casi, la tensione $V_{2ref} = 0.5V$

NOTA BENE: I grafici riportati hanno il segnale di errore è zoomato di un fattore 10 per meglio mettere in evidenza il comportamento dell'errore

4.4.1 Design Base di controllo

Usando il controllo base e senza saturazione, abbiamo una risposta che conferma i gli esiti calcolati per e_∞ nella sezione 4.3:

Figura 4.4: Modelli simulati su Simulink



Queste performance possono e devono essere ampliamene migliorate poiché raggiunta la saturazione del controllo, l'errore ricomincia subito a crescere ed è impossibile riprenderlo senza resettare l'esperimento.

In questo caso però la saturazione è stata eliminata al fine di mettere maggiormente in evidenza che asintoticamente il controllo proposto realizza le richieste di controllo di avere un inseguitore con errore nullo asintotico.

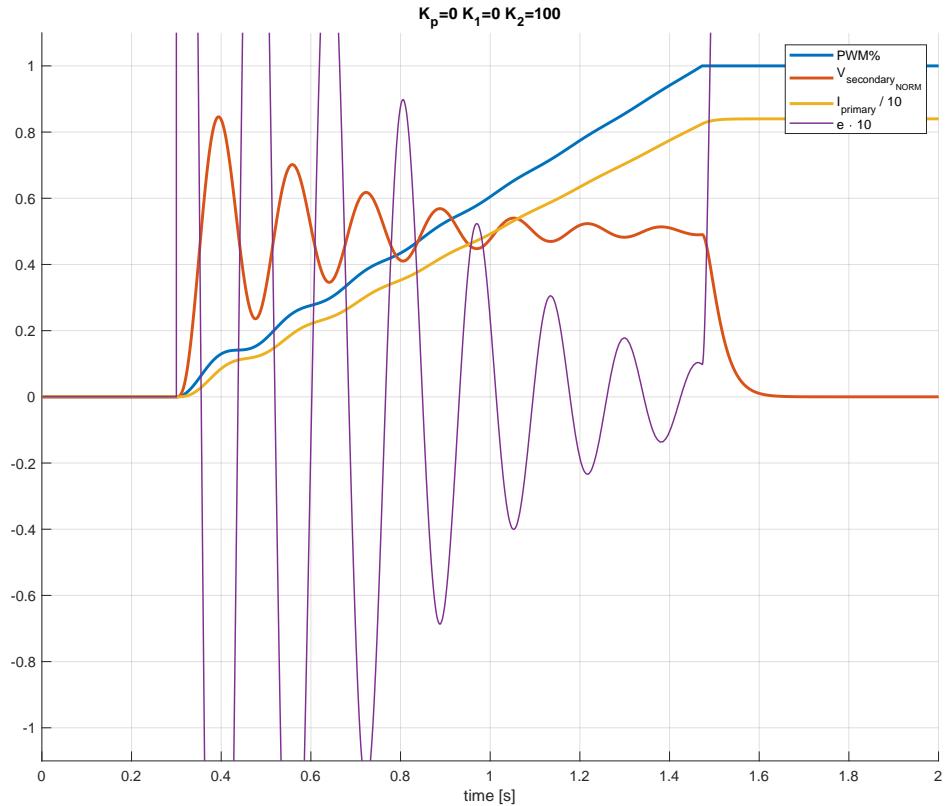
4.4.2 Design Avanzato di controllo

Usando ora il controllore proposto nell'equazione 4.3.5 analizziamo ora gli effetti della presenza dei vari termini, e il loro effetto qualitativo sulla risposta e l'andamento dell'errore.

Ricordiamo nuovamente che il segnale di errore è zoomato di un fattore 10 per meglio mettere in evidenza il comportamento dell'errore.

Analisi controllo Avanzato per $K_p=0$ $K_1=0$ $K_2=100$

Figura 4.5: Controllo Avanzato $K_p=0$ $K_1=0$ $K_2=100$

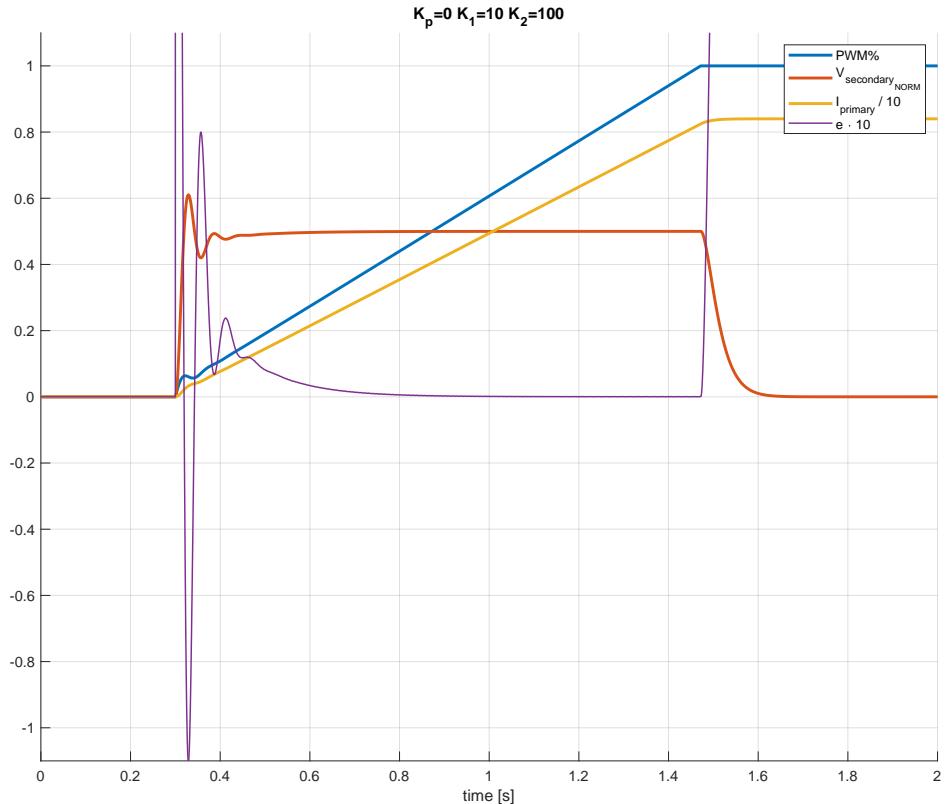


L'aumento del coefficiente per il doppio integratore ha l'effetto di creare risposte più tempestive, ma da atto a dei fenomeni oscillatori smorzati

Analisi controllo Avanzato per $K_p=0 K_1=10 K_2=100$

In questa seconda simulazione si è aggiunto un al controllore il termine integrale:

Figura 4.6: Controllo Avanzato $K_p=0 K_1=10 K_2=100$



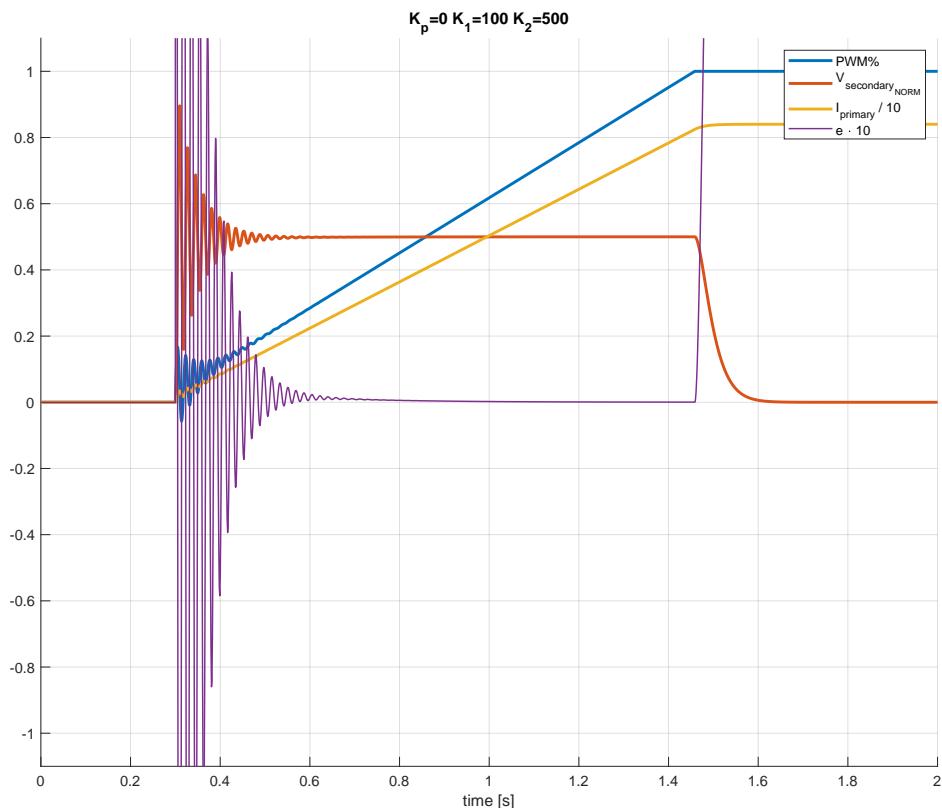
L'effetto del termine integrale è quindi smorzare il transitorio oscillatorio, per poi inseguire il riferimento con un andamento esponenziale.

In questo secondo esperimento abbiamo già migliorato molto le performance dal controllo base, avendo visibilmente migliorato il Tempo di assestamento del sistema e lasciando 600ms di esperimento con errore ampiamente trascurabile.

Analisi controllo Avanzato per $K_p=0 K_1=100 K_2=500$

Abbiamo adesso aumentato i coefficienti del doppio e singolo integratore per vedere come migliorano le performance del sistema ad un loro aumento coordinato:

Figura 4.7: Controllo Avanzato $K_p=0 K_1=100 K_2=500$

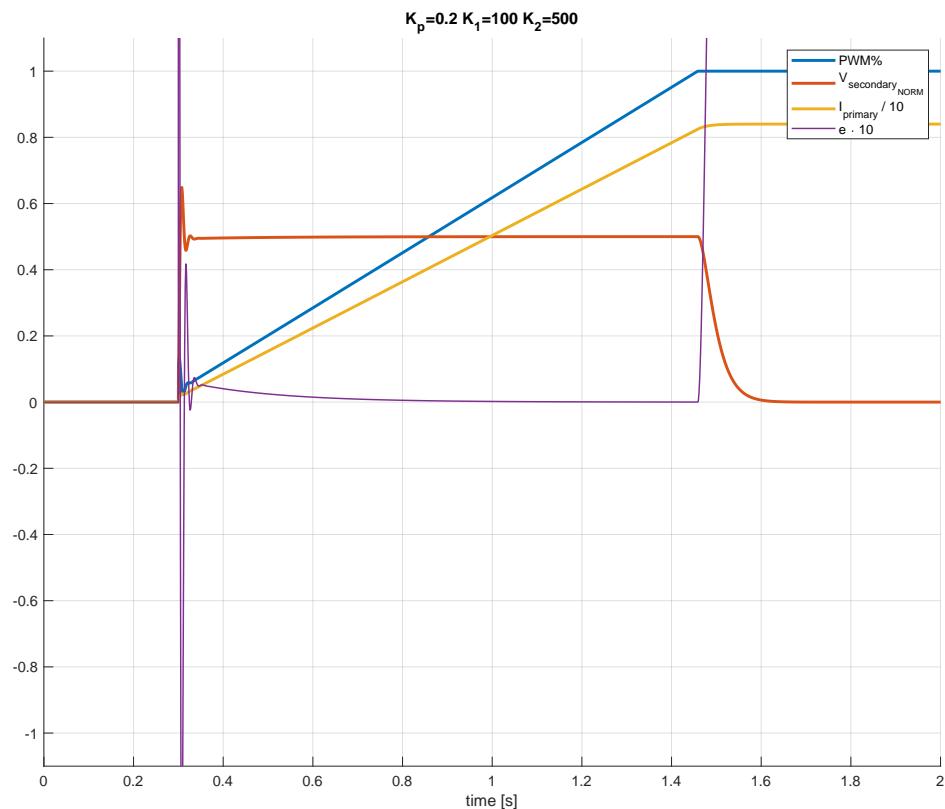


L'effetto è un ulteriore miglioramento sul Tempo di assestamento e una una discesa più pronunciata dell'errore verso lo 0, ma abbiamo pagato questo incremento prestazionale con il ritorno degli effetti oscillatori smorzati, ed esse sono anche a frequenze maggiori, non proprio desiderabile.

Analisi controllo Avanzato per $K_p=0.2$ $K_1=100$ $K_2=500$

Aggiungiamo quindi al caso precedente un termine proporzionale all'errore, così da rispondere subito alle variazioni del riferimento:

Figura 4.8: Controllo Avanzato $K_p=0.2$ $K_1=100$ $K_2=500$

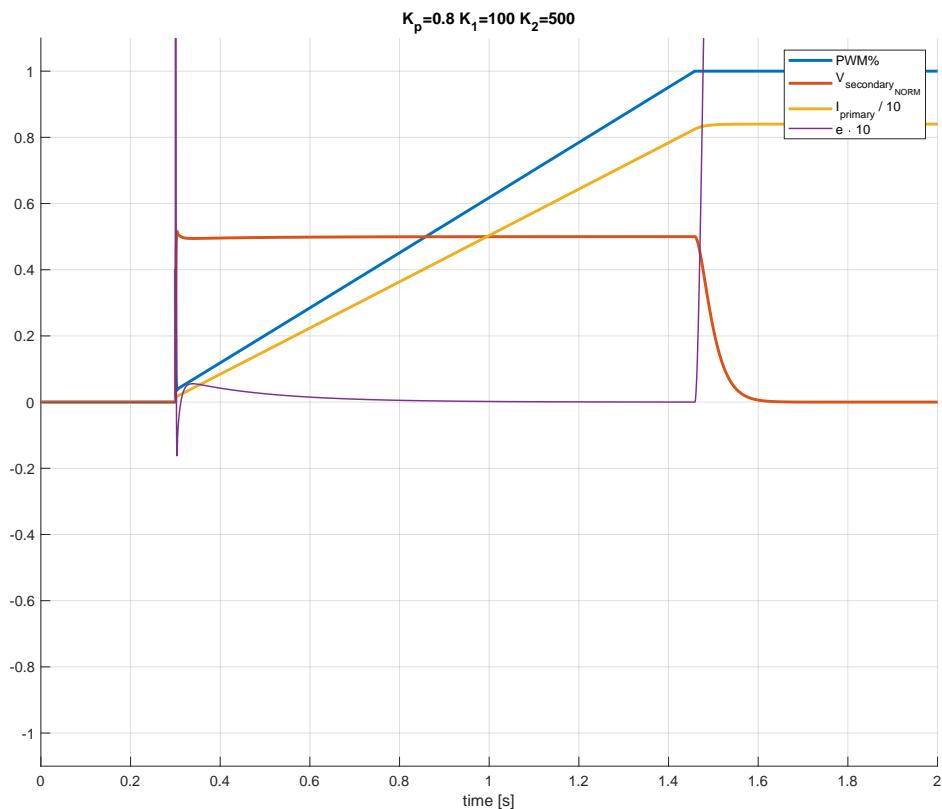


Essendo K_p istantaneo con la variazione del riferimento, abbiamo che l'inseguimento di V_{2ref} inizia prima, riducendo il margine di fase dovuto al caricamento degli integratori, ciò riduce il fenomeno di wind-up presente precedentemente e causa delle oscillazioni attorno al riferimento, e permette di tornare al comportamento del 2° esperimento, con performance nettamente superiori.

Analisi controllo Avanzato per $K_p=0.8$ $K_1=100$ $K_2=500$

In quest'ultimo esperimento analizziamo l'effetto dell'aumento del termine proporzionale sulla dinamica:

Figura 4.9: Controllo Avanzato $K_p=0.8$ $K_1=100$ $K_2=500$



Rispetto al caso precedente, l'errore iniziale si è molto ridotto, passando da un fuori scala in entrambi i segni, a un errore di -0.019 di picco negativo (il picco positivo è per definizione 0.5 essendo il riferimento dato come gradino).

Al contrario, superata questa prima fase, il resto della dinamica è fondamentalmente identica al caso precedente.

4.5 Conclusioni per il design Avanzato

In conclusione possiamo dire che il controllore "PID-style" definito nell'equazione 4.3.5 permette di ottenere gli obiettivi di controllo prefissati, e i suoi coefficienti hanno un effetto sulle performance ottenibili.

La loro variazione ha il seguente effetto qualitativo:

K_2 Permette di raggiungere gli obiettivi di inseguimento con errore nullo, un suo aumento migliora la risposta ma causa delle oscillazioni smorzate a frequenze via via più elevate (Raggiungimento degli obiettivi di controllo).

K_1 Se dimensionato opportunamente rende più marcato lo smorzamento delle oscillazioni dovute a K_2 fino a tornare ad un andamento esponenziale (Migliora Tempo di Assestamento).

K_p Migliorare le performance nei primi istanti di controllo, prima che gli integratori possano arrivare a convergenza (Miglioramento del Tempo di risposta).

I valori calcolati sul modello, non possono essere implementati 1:1 nel caso reale, a causa delle *Non-Linearità* che sono state trascurate nella creazione del modello, e per tutti i problemi dovuti al tempo di campionamento e attuazione del sistema che è pari a $2Khz$.

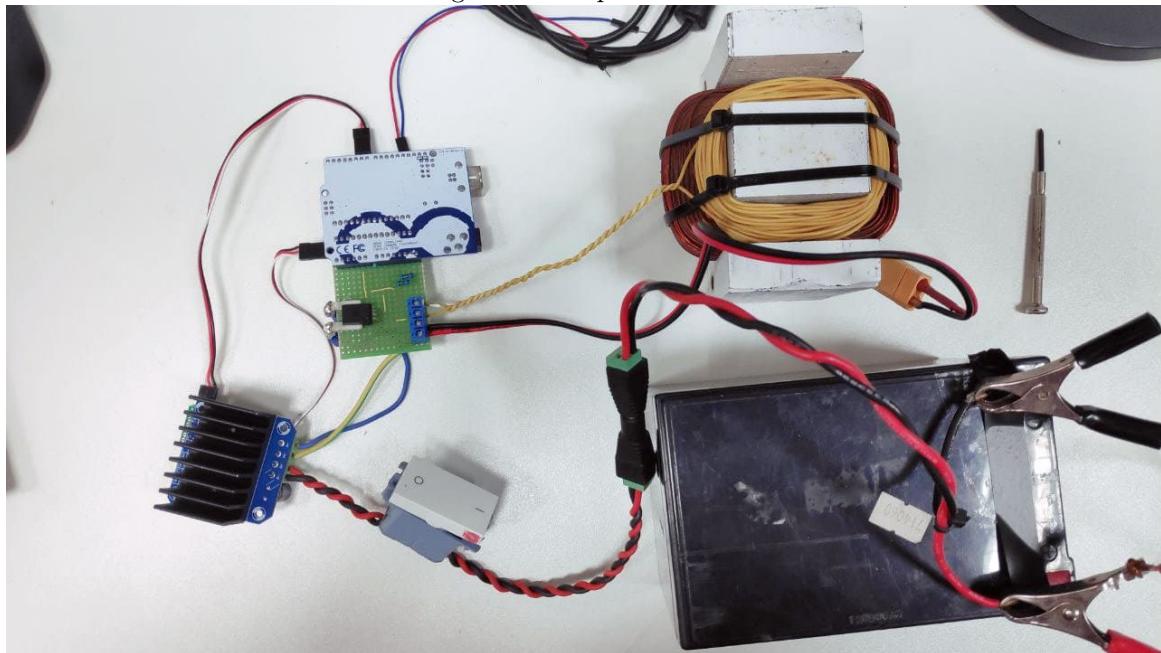
Non di meno, le osservazioni qui riportate continuano ad essere valide anche sul sistema reale, ed essendo le *Non-Linearità* presenti soprattutto nei pressi della *Dead-zone* e della *Saturazione* e assimilabili a del disturbo non troppo ampio, avremo modo di mostrare che il controllore così progettato è anche robusto per errori di attuazione e misura.

Capitolo 5

Sviluppo Controllo reale

In questo capitolo vedremo come è implementato nella realtà il controllore descritto nel capitolo [Modello teorico di Controllo](#) all'interno del **μ Controllore**, tareremo i suoi coefficienti e mostreremo come si comporta in un esperimento reale con tutti i problemi ad esso connessi ([Non-Linearità del BTS7960 High Current 43A H-Bridge Motor Driver](#), [errori di quantizzazione, discretizzazione del controllo, ecc...](#))

Figura 5.1: Impianto Reale



Essendo un **μ Controllore** un computer non eccessivamente potente, è necessario realizzare la funzione di trasferimento del controllo (equazione 4.3.5) mediante un sistema a tempo discreto.

Per semplificarcici l'implementazione, invece di creare un unico sistema del 2° ordine, si è optato per

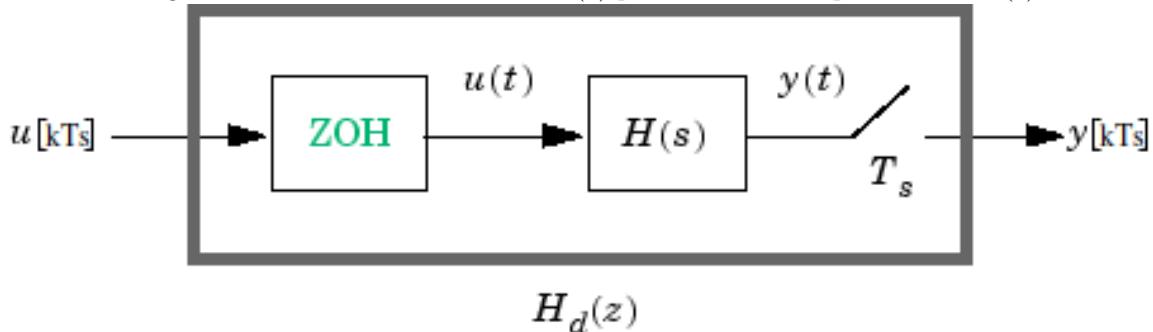
sommare tra loro le tre funzioni di trasferimento semplici che compongono il controllore, così da semplificare il debugging e permettere un più semplice controllo sullo stato dei componenti, utile per saturare gli integratori una volta che l'uscita ha superato la soglia di attuabilità (*Saturazione*).

In fine, essendo l'obiettivo di controllo realizzato da una rampa, dopo un certo tempo in saturazione, per evitare lo spreco di energia da parte della batteria che alimenta il sistema, il codice disattiva il controllo, resetta tutti gli stati e mette il riferimento a 0. Questo stato non è permanente ma persiste fino al raggiungimento di un nuovo input di controllo mediante *Embedded Message Pack(EMP)*.

5.1 Discretizzazione Zero-Order Hold (Z.O.H.)

Un oggetto del tipo **Zero-Order Hold** (Z.O.H.) altro non è che un convertitore Digitale→Analogico che permette di interfacciare dei segnali **Tempo Discreto** $\mathbb{T} = \mathbb{Z}$ che evolvono $\forall \Delta T_s$ ¹ con sistemi dinamici **Tempo Continuo** $\mathbb{T} = \mathbb{R}$.

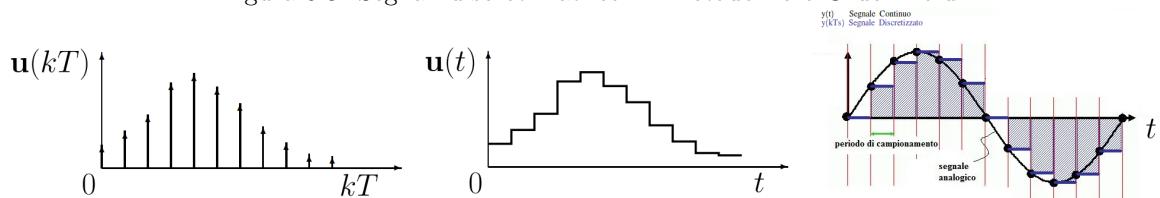
Figura 5.2: Discretizzazione ZOH $H_d(z)$ per il sistema Tempo continuo $H(s)$



Questa interconnessione "congela" l'ultimo segnale discreto $u(kT_s)$ ricevuto e lo ripropone come un segnale costante in ingresso al sistema continuo che evolve in maniera indipendente.

L'uscita $y(kT_s)$ è la discretizzazione di $y(t)$, campionata ogni T_s .

Figura 5.3: Segnali discretizzati con il metodo Zero-Order Hold



Il sistema $H(s)$ durante gli intervalli T_s evolve avendo una costante in ingresso, e al successivo istante di campionamento l'uscita raggiunta viene campionata e mantenuta per ul successivo T_s , e questo procedimento all'infinito.

¹ T_s = Sampling Time

Ora che abbiamo descritto qualitativamente cosa avviene usando una discretizzazione ZOH, vediamo ora come diventa $H_d(s)$ nello spazio di stato in termini matematici. Il procedimento di discretizzazione necessita di passare attraverso lo spazio di stato dei 3 sistemi dinamici che compongono il controllore 4.3.5 e usando le formule del professore Zanasi, «[Discretizzazione di un sistema tempo continuo](#)», si ottengono i risultati seguenti risultati:

Tabella 5.1: Funzioni di trasferimento nello spazio di Stato, da $\mathbb{T} = \mathbb{R} \rightarrow \mathbb{T} = \mathbb{Z}$

$C_{I^2}(s) = \frac{K_2}{s^2}$ $\begin{cases} \dot{x} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}x + \begin{pmatrix} 0 \\ K_2 \end{pmatrix}u \\ y = \begin{pmatrix} 1 & 0 \end{pmatrix}x \end{cases}$	$C_I(s) = \frac{K_1}{s}$ $\begin{cases} \dot{x} = x + K_1 \cdot u \\ y = x \end{cases}$	$C_p(s) = K_p$ $\begin{cases} y = K_p \cdot u \end{cases}$
$\begin{cases} x^+ = \begin{pmatrix} 1 & T_s \\ 0 & 1 \end{pmatrix} \cdot x_k + K_2 \begin{pmatrix} T_s^2/2 \\ T_s \end{pmatrix} u_k \\ y_k = (1 \ 0) \cdot x_k \end{cases}$ $C_{I^2}(z) _{T_s} = K_2 \cdot \frac{T_s}{2} \cdot \frac{z+1}{(z-1)^2}$	$\begin{cases} x^+ = x_k + K_1 T_s \cdot u_k \\ y_k = x_k \end{cases}$ $C_I(z) _{T_s} = \frac{K_1 T_s}{z-1}$	$\begin{cases} y_k = K_p \cdot u_k \end{cases}$ $C_p(z) _{T_s} = K_p$

Si può notare che in tutti e 3 i sistemi si è scelta una realizzazione della funzione di trasferimento nello Spazio di Stato tempo continuo in *Forma Compagna di Osservabilità* (Zanasi, «[Forme Canoniche Spazio di Stato](#)»).

Questa scelta non è stata casuale e ha lo scopo di semplificare in futuro la codifica per un sistema di controllo **Switching nei Coefficienti**, volto a migliorarne ulteriormente le prestazioni, senza dover scalare lo stato in base al cambio dei coefficienti.

Questa comoda proprietà è dovuta alla struttura della matrice C , la quale, per variazioni **istantanee** dei coefficienti K_2, K_1, K_p , non propaga gli effetti sull'uscita $y(t)$, l'effetto della variazione arriverà in un secondo momento grazie all'integrazione dello stato, che ovviamente evolverà diversamente da prima a causa delle variazioni dei coefficienti. Per avere un'idea qualitativa degli effetti dei coefficienti

sul sistema complessivo, rifarsi alla sezione "[Conclusioni per il design Avanzato](#)".

I calcoli della trasformata Zeta, sono presenti solo per completezza e sono stati ottenuti usando la classica formula:

$$G(z) = C_d (zI - A_d)^{-1} B_d + D_d$$

Per i nostri scopi noi siamo interessati alle matrici A_d, B_d, C_d, D_d della conversione, così da implementare esattamente la discretizzazione di ordine Zero del sistema dinamico.

5.2 Codifica del controllore

La codifica del controllore [4.3.5](#) avviene attraverso le matrici A_d, B_d, C_d, D_d riportate in tabella [5.1](#), con queste matrici è stato costruito la classe di controllo `iiCTRL` riportata in appendice nel codice del controllore ([Header Classe controllore C\(s\)](#) e [Source Classe controllore C\(s\)](#)).

La classe viene chiamata all'interno del `Loop di Controllo` ogni istante di campionamento, e permette la modifica a richiesta del riferimento da inseguire.

All'interno della classe è già codificata una la logica di *Safe-Shutdown*, essa viene attivata quando il controllo ha raggiunto la saturazione per un tempo superiore ai 200ms, risulta infatti rischioso e dispendioso tenere connesso il trasformatore, poiché le correnti che scorrono sono dell'ordine dei 10A, e in ogni caso raggiungere gli obiettivi di controllo necessita di una rampa di corrente sul primario, che non si può ottenere se si è già saturato.

Questo contatore viene ripristinato a 0 all'arrivo di un nuovo riferimento da inseguire, causando di fatto il ripristino dell'esperimento.

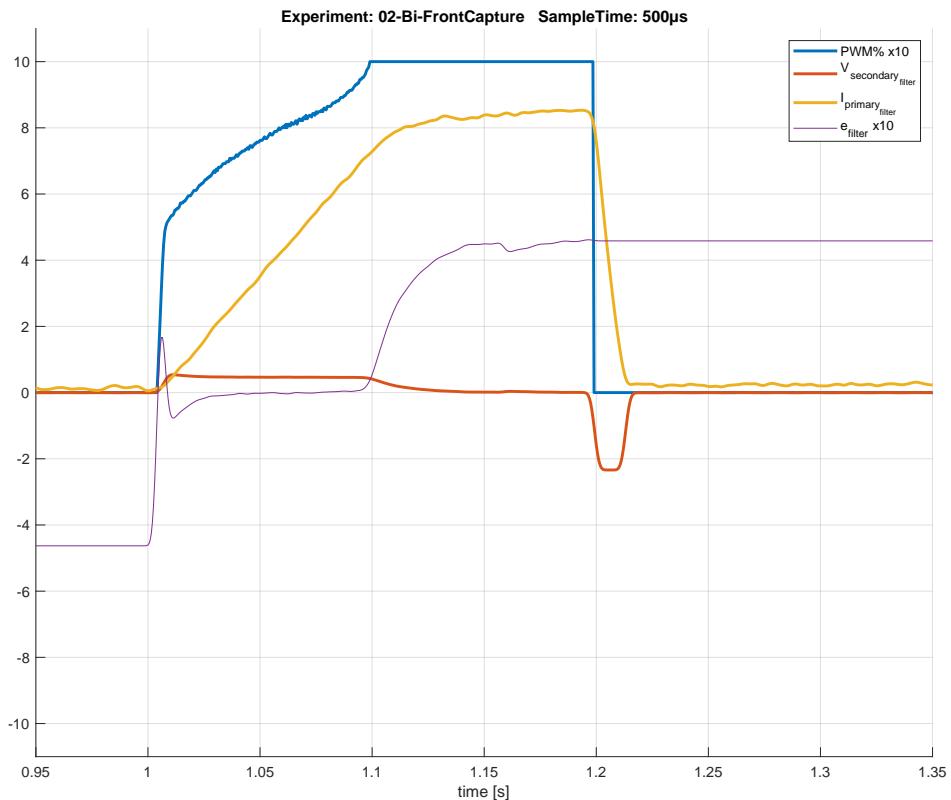
5.3 Esperimenti

Di seguito vengono riportati diversi spari catturati dal sistema in funzione, nella sua interezza, usando al posto di *MARTE2*, un programma scritto in C++ per Linux chiamato *MARTE2Moc*, quando il resto del sistema verrà controllato, gli stessi input di comando inviati da questa imitazione saranno ricevuti dal μ **Controllore** e attuati in maniera identica.

5.3.1 Inseguimento singolo

In questo "sparo" viene impostato il riferimento a $V_{2ref} = 0.5V$ e il controllo insegue prontamente il riferimento, mantenendo l'errore entro margini molto stretti e con una decrescita esponenziale.

Figura 5.4: Esperimento singolo $V_{2ref} = 0.5V$



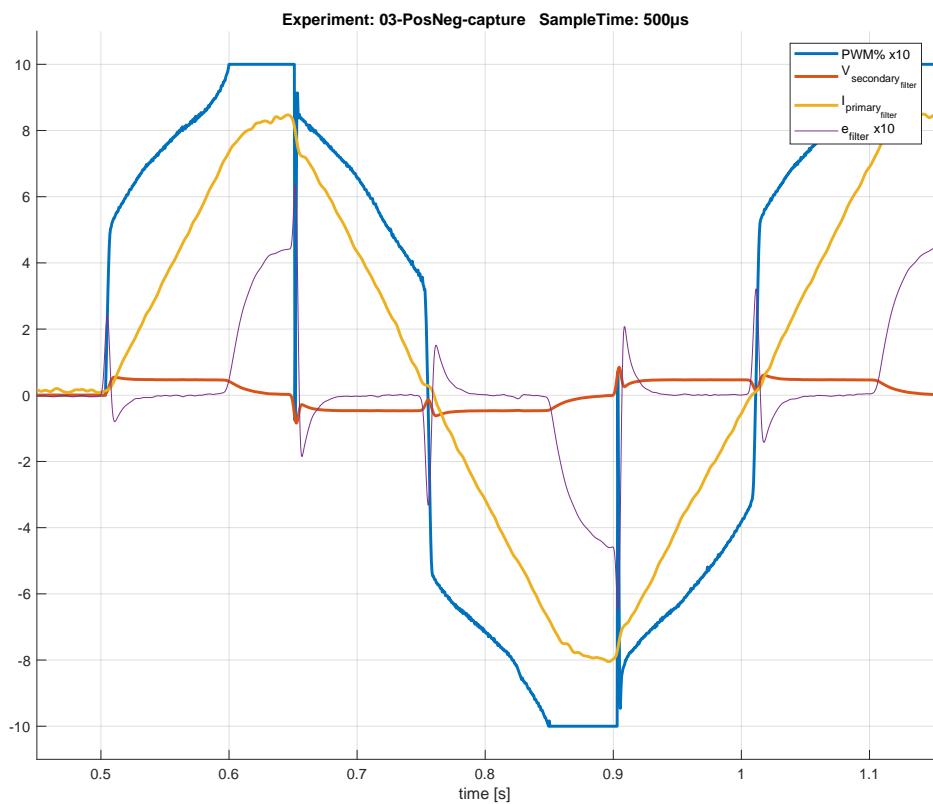
Come abbiamo avuto modo di vedere nel capitolo "[Stima del modello del sistema](#)", il modello lineare che abbiamo trovato, taglia tutta una serie di *Non-Linearità* presenti invece nell'impianto e particolarmente evidenti nei pressi della **Dead-Zone** e della **Saturazione**, ma il controllo in catena chiusa riesce a gestire questi problemi e il sistema sin dall'inizio tende a 0 esponenzialmente.

Passati i 200ms in saturazione il controllo automaticamente va a 0, realizzando la logica di *Safe-Shutdown* descritta prima. Negli istanti subito successivi è possibile vedere come la tensione sul secondario vada fuori scala e si saturi attorno a $-2.2V$, questo è concorde con i limiti di misura che abbiamo sul prototipo, e impone che le pendenze massime ineseguibili siano non superiori ai $\pm 2V$.

5.3.2 Inseguimento Triangolare

In questo secondo esperimento, *MARTe2Moc* è stato programmato per invertire il segno del riferimento ogni volta che il controllo arriva in saturazione, l'obiettivo del test è vedere la risposta del controllo nei pressi della **Dead-Zone** e in corrispondenza di inversione di polarità, il continuo ribaltamento del $V_{2_{ref}}$ impedisce di arrivare in saturazione troppo a lungo, rendendo l'esperimento periodico e illimitato nel tempo.

Figura 5.5: Inseguimento Triangolare $V_{2_{ref}} = \pm 0.5V$

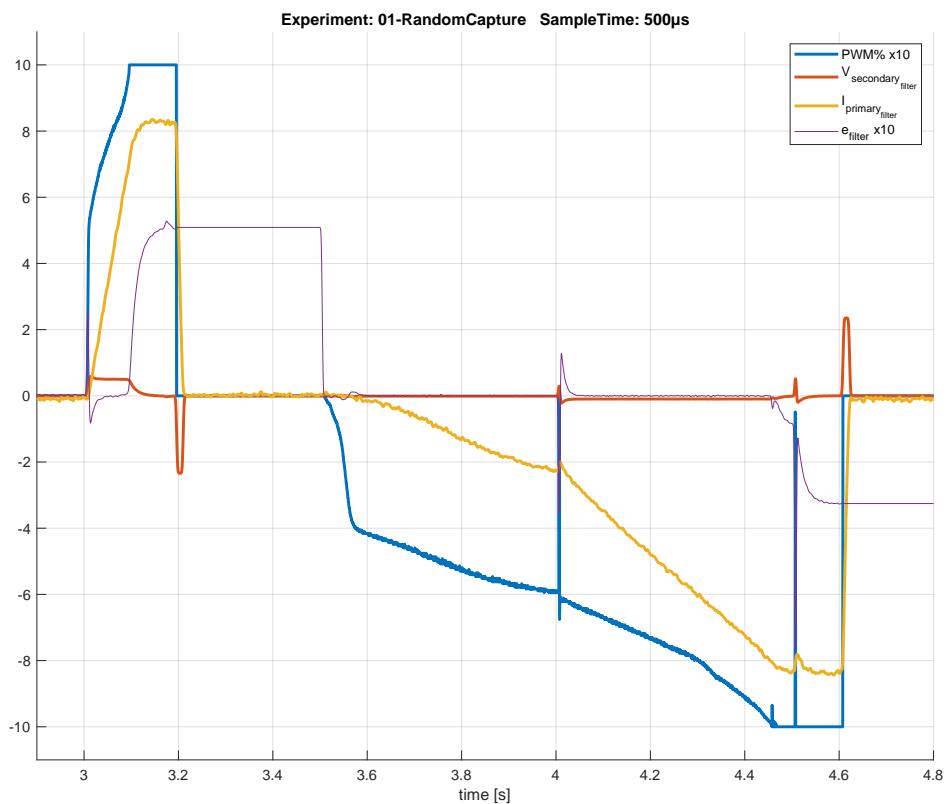


Come è possibile vedere, nei pressi della **Dead-Zone** la presenza della zona morta fa sì che l'errore cresca di $0.02V$, ma questo errore viene rapidamente ripreso e portato in 0.

5.3.3 Cambio riferimento in corsa

In questo terzo esperimento, *MARTE2Moc* è stato programmato per inviare un nuovo riferimento casuale ogni 500ms, nel grafico riportato vi è stato prima un inseguimento "ripido", seguito da uno molto morbido che è stato interrotto ben prima del raggiungimento della saturazione da un altro riferimento dello stesso segno negativo ma più ripido:

Figura 5.6: Cambio di V_{2ref} prima della saturazione



Risulta evidente che il sistema accetta bene i cambi di riferimenti, bisogna solo correggere il problema nella classe che implementa il controllore modificando che il cambio di riferimento non influisca sullo stato automaticamente.

Capitolo 6

Conclusioni e sviluppi futuri

Conclusioni

In conclusione, con questo prototipo si è dimostrato e realizzato un controllore *PID-style* a doppio Polo nell'origine, è un design di controllore perfetto da usare per governare la corrente di una bobina in un impianto Tokamak.

La realizzazione di questo prototipo però, come per ogni progetto pratico, ha dovuto confrontarsi con vari problemi di *Non-Linearità*, problemi implementativi, limiti di attuazione e campionamento, etc...

La risoluzione di questi problemi ha portato a compromessi e semplificazioni volte a catturare gli aspetti principali dell'esperimento, trascurando quelli secondari.

Il controllo così creato, oltre a funzionare bene nella teoria, risulta robusto alle variazioni dal modello lineare presenti nella realtà ma trascurate durante la modellizzazione del sistema, e ciò rende un simile design di controllo general-purple per impianti tokamak, poiché i coefficienti trovati nel corso di questa tesi permettono l'ottimizzazione per questo impianto in particolare, ma porterebbero a convergenza un qualunque impianto Tokamak avente la stessa struttura nella dinamica.

Sviluppi Futuri

Anche se per i fini della tesi il progetto termina qui, il progetto in sé ancora avanza, e tra gli sviluppi futuri abbiamo:

- L'intenzione di portare un controllo **Switching** nella legge di Up-date, variando i coefficienti del controllore [4.3.5](#)

In tal senso il controllore implementato nel μ **Controllore**, ottenuto dal modello nello spazio di stato in Forma Compagna di Osservatore (Zanasi, «[Forme Canoniche Spazio di Stato](#)») già rende il sistema pronto per questo tipo di modifica a livello di codice, è necessario tuttavia studiare e simulare per quali soglie lo switch dei parametri può incrementare le performance, prima di poter implementare una simile tecnica.

- Attualmente è in cantiere la realizzazione all'interno di *MARTe2* della libreria [Embedded Messege Pack\(EMP\)](#) per permettere l'integrazione di questo firmware con l'ecosistema *MARTe2*.
- Risulta di interesse eseguire un upgrade del μ **Controllore** dal'**Arduino Uno** a una scheda più performante, e che permetta comunicazioni, campionamenti e PWM a velocità superiori, permetterebbe di superare l'attuale soglia di $2Khz$ di controllo, e realizzare dei controlli molto più performanti con tempi di campionamento molto inferiori, e quindi ritardi di fase ancora più azzerati.
- In fine, per raggiungere performance superiori è necessario avere un motore superiore, in questo caso il ponte-H, trovarne uno con una dinamica più lineare e che possa funzionare a frequenze maggiori di PWM permetterebbe di avere un controllo sul Primario molto più fine e potente, portando a migliorare le prestazioni in maniera sensibile e tangibile.

Appendice A

Arduino Code

7.1 Set-up Registri

7.1.1 Tic Timer

```
1 void periodicTask(int time) { // time in micro secondi
2     // PWM pin Disable, motalita CTC(pt1)
3     TCCR2A = (0x0 << COM2A0) | (0x0 << COM2B0) | (0x2 << WGM20);
4     // CTC(pt2), Prescalere 256
5     TCCR2B = (0 << WGM22) | (0x6 << CS20);
6     // T_ckclock * Twant / Prescaler = valore Registro
7     OCR2A = (int)(16UL * time / 256);
8     TIMSK2 = (1 << OCIE2A); // attivo solo l'interrupt di OC2A
9 }
```

Listing 7.1: Tic Timer

Questa funzione imposta il TIMER2 in modalità Fast PWM, ovvero che si resetta quando arriva al conteggio finale, e calcola il valore da mettere nel registro affinchè il conteggio sia il più vicino possibile a tempo desiderato

7.1.2 Frequenza PWM

```
1 enum pwmFreq: char {
2     hz30, hz120, hz490, hz4k, hz30k
3 };
4
5 void setMotFreq(pwmFreq freq) {
6     // TCCR0B is for Timer 0
7     #define myTimer TCCR0B
8     switch (freq) {
9         // set timer 3 divisor to 1024 for PWM frequency of 30.64 Hz
10        case hz30:
11            myTimer = (myTimer & B11111000) | B00000101;
12            break;
13        case hz120:
14            // set timer 3 divisor to 256 for PWM frequency of 122.55 Hz
15            myTimer = (myTimer & B11111000) | B00000100;
16            break;
17        case hz490:
18            // set timer 3 divisor to 64 for PWM frequency of 490.20 Hz
```

```

19     myTimer = (myTimer & B11111000) | B00000011;
20     break;
21     case hz4k:
22     // set timer 3 divisor to 8 for PWM frequency of 3921.16 Hz
23     myTimer = (myTimer & B11111000) | B00000010;
24     break;
25     case hz30k:
26     // set timer 3 divisor to 1 for PWM frequency of 31372.55 Hz
27     myTimer = (myTimer & B11111000) | B00000001;
28     break;
29     default:
30     setMotFreq(hz4k);
31     break;
32   }
33 #undef myTimer
34 }
```

Listing 7.2: Frequenza PWM

Mediante questa funzione si modifica il valore del Prescaler per il TIMER 0, modificando la velocità di conteggio si ottiene un PWM con una periodo, e quindi frequenza, che varia.

Offset Calculation

```

1 #define offsetCalc(pin,n)
2 ({ \
3   long read = 0; \
4   for (int i = 0; i < 1 << n; i++) { \
5     read += analogRead(pin); \
6     delay(1); \
7   } \
8   read = read >> 5; \
9   read; \
10 })
```

Listing 7.3: Macro per calcolo Offset

La macro misura con una distanza temporale di 1 ms 2^n -volte, la misura del Pin, e successivamente ne fa la media eseguendo un revers shift, questa tecnica di divisione semplicemente minimizza il tempo di calcolo, poiché ogni revers shift equivale a una divisione per 2, e si ha l'effetto di dividere per $\frac{read}{2^n}$ in $O(n)$ con un operazione per il processore molto semplice.

7.2 Generatore di Segnale

Per generare i segnali di controllo in Feed-Forward usati nel sistema, sono stati usati 2 diversi livelli di programmazione.

Un primo livello segnali di base, definiti su tutto \mathbb{R} , e usabili a piacere, e dei segnali composti e periodici da mandare durante l'esperimento. Tutti i segnali sono pensati per andare da -100% <-> 100%, è compito dell'attuazione eliminare le deadzone e traslare il controllo al valore più opportuno

7.2.1 Segnali Base

Rampa

```

1 int ramp(uint64_t t, int vStart, uint64_t tStart, int vEnd, uint64_t tEnd) {
2     // Saturazione
3     if (t < tStart)
4         return vStart;
5     else if (t > tEnd)
6         return vEnd;
7     // Retta
8     unsigned int dt = t - tStart;
9     return vStart + int((vEnd - vStart) / float(tEnd - tStart) * dt);
10 }
```

Listing 7.4: Rampa Saturata

La rampa è descritta come una retta nell'intervallo di interesse, saturata prima e dopo il tempo desiderato

$$RampaSat(t) = \begin{cases} v_{start} + \frac{v_{end}-v_{start}}{t_{end}-t_{start}} * (t - t_{start}) & \forall t \in [t_{start}, t_{end}] \\ v_{start} & t < t_{start} \\ v_{end} & t > t_{start} \end{cases}$$

7.2.2 Segnali Composti

Onda Triangolare

```

1 int triangleSignal(uint64_t t, int msQuartPeriod) {
2     static uint64_t startTic = 0;
3     int dTic = t - startTic;
4     int pwm = 0;
5     if (dTic < ticConvert(msQuartPeriod))
6         pwm = ramp(dTic, 0, 0, 100, ticConvert(msQuartPeriod));
7     else if (dTic < (ticConvert(msQuartPeriod) * 3))
8         pwm = ramp(dTic, 100, ticConvert(msQuartPeriod), -100, ticConvert(msQuartPeriod)
9             * 3);
10    else if (dTic < (ticConvert(msQuartPeriod) * 4))
11        pwm = ramp(dTic, -100, ticConvert(msQuartPeriod) * 3, 0, ticConvert(msQuartPeriod
12            ) * 4);
13    else {
14        pwm = 0;
15        startTic = t;
16    }
17    return pwm;
18}

```

Listing 7.5: Onda Triangolare Periodica

Onda Trapezoidale (RapidShot)

```

1 int rapidShot(uint64_t t) {
2     static uint64_t startTic = 0;
3     int pwmRapidShot;
4     long dTic = t - startTic;
5     if (dTic > t4) {
6         startTic = t;
7         pwmRapidShot = 0;
8         dTic = t - startTic;
9     }
10    if (dTic <= t1) {
11        pwmRapidShot = ramp(dTic, 0, 0, 100, t1);
12    } else if (dTic <= t2) {
13        pwmRapidShot = 100;
14    } else if (dTic <= t3) {
15        // falling ramp
16        pwmRapidShot = ramp(dTic, 100, t2, 0, t3);
17    } else if (dTic <= t4) {
18        pwmRapidShot = 0;
19    }
20    return pwmRapidShot;
21}
22}

```

Listing 7.6: Onda Trapezoidale Periodica (RapidShot)

7.3 Codici Controllore

Setup dell'esperimento

```

1 void setup() {
2     mpSerial.begin(2000000);
3
4     memset(&pRead, 0, sizeof(pRead));
5     memset(&pWrite, 0, sizeof(pWrite));
6
7     // Motori
8     setMotFreq(hz30k);
9     mot = new DCdriver(enPwm, inA, inB);
10
11    // Mean find
12    pMean.V2_mean = offsetCalc(V2, 5);
13    pMean.Isense_mean = offsetCalc(Isense, 5);
14    pMean.dt = dtExperiment;
15
16    // Ctrl Reference
17    Ctrl.setNewRef(tic, 0);
18
19    // ##### Start Experiment #####
20    mpSerial.bufClear();
21    // Start Delay
22    memset(&pWrite, 0, sizeof(pWrite));
23    pWrite.type = sampleType;
24    delay(1000);
25    periodicTask(pMean.dt);
26    sei();
27 }
```

Listing 7.7: Setup del Controllore

Loop di Controllo

```

1 volatile u32 oldTic = tic;
2 int readData;
3 int pwm;
4 void loop() {
5     mpSerial.updateState();
6     do {
7         readData = mpSerial.getData_try(&pRead);
8         if (readData >= 0) {
9             serialExe(&pRead);
10        }
11    } while (tic == oldTic);
12
13    pWrite.read.V2_read = analogRead(V2);
14    pWrite.read.Isense_read = analogRead(Isense);
15    pwm = mot->drive_motor(Ctrl.ctrlStep(oldTic, pWrite.read.V2_read - pMean.V2_mean));
16    pWrite.read.pwm = pwm;
17    pWrite.read.err = Ctrl.lastErr;
18
19    oldTic++; // Suppose no over time
20    mpSerial.packSend(&pWrite, sizeof(pWrite.type) + sizeof(pWrite.read));
21 }
```

Listing 7.8: Loop di Controllo

7.3.1 Classe Controllore

```

1 // Common define
2 #define volt2adc(x) ((int)((((x) / (5.0 / (1023)) + 0.5)))
3 #define dtExperiment 500UL // us
4 #define ticConvert(ms) ((ms * 1000UL) / dtExperiment)
5 #define Ts (dtExperiment / 1000000.0) // second
6
7 class iiCTRL {
8     // Saturation showdown experiment
9     unsigned int ticSatCount = 0;
10    // Adc Current Voltage reference
11    int V2currRef = 0;
12
13    float kp = 0.0;
14    float k1 = 0.8;
15    float k2 = 0.05;
16
17    // State Space Cii(s)
18    float xCii1 = 0, xCii2 = 0;
19    // State Space Ci(s)
20    float xCi1 = 0;
21 public:
22     int lastCtrl;
23     int lastErr;
24     iiCTRL();
25     iiCTRL(float k2, float k1, float kp);
26
27     // State update
28     void setNewRef(uint64_t ticSet, int v2AdcNewRef); // v2AdcNewRef senza offset
29     int ctrlStep(uint64_t t, int v2Adc); // v2Add gi senza offset
30
31     // For switching logic
32     void changeK(float k2, float k1, float kp);
33     void changeK2(float k2);
34     void changeK1(float k1);
35     void changeKp(float kp);
36
37 private:
38     void stateReset(int setOutput);
39 }
```

Listing 7.9: Header Classe controllore C(s)

La classe `iiCTRL` implementa il controllore 4.3.5 e usa le matrici di discretizzazione riportate nella tabella "Funzioni di trasferimento nello spazio di Stato, da Tempo Continuo a Tempo Discreto" per implementarle a tempo continuo.

```

1 iiCTRL::iiCTRL() : iiCTRL(0.05, 0.8, 0) {}
2
3 iiCTRL::iiCTRL( float k2, float k1, float kp) {
4     changeK(k2, k1, kp);
5     setNewRef(0, 0);
6 }
7
8 void iiCTRL::setNewRef(uint64_t ticSet, int v2AdcNewRef) {
9     V2currRef = v2AdcNewRef;
10    ticSatCount = 0;
11 }
12
13 int iiCTRL::ctrlStep(uint64_t t, int v2Adc) {
14     //Saturation Check
15     if (ticSatCount >= ticConvert(200)){
16         stateReset(0);
17         return 0;
18     }
19     //Error Calc (ADC value), u_k per i sistema dinamico
20     lastErr = (V2currRef - v2Adc);
21     // Caso speciale: Riferimento a 0 => spengo tutto
22     if (V2currRef == 0){
23         stateReset(0);
24         return 0;
25     }
26     // ##### Space State Update #####
27     // State update Cii(s)
28     xCii1 = xCii1 + Ts * xCii2 + sq(Ts) / 2 * k2 * lastErr;
29     xCii2 = xCii2 + Ts * k2 * lastErr;
30     // State update Ci(s)
31     xCi1 = 0;
32     float yCp = kp * lastErr;
33
34     // ##### Output calc #####
35     lastCtrl = xCii1 + xCi1 + yCp; // y_k
36     lastCtrl = constrain(lastCtrl, -1000, 1000); // Limitazione +-1000
37     if (abs(lastCtrl) == 1000) {
38         ticSatCount++;
39         stateReset(lastCtrl);
40     } else
41     ticSatCount = 0;
42     return lastCtrl;
43 }
44
45 void iiCTRL::stateReset(int setOutput) {
46     // Idea Stati proporzionali
47     // float rap = (float)dIntegral2 / (float)dIntegral1;
48     // dIntegral1 = (int)((float)setOutput / (rap * k2 + k1));
49     // dIntegral2 = (int)(rap * (float)dIntegral1);
50     // Riassegno tutto l'output su Cii(s)
51     float rap = xCii1 / xCi1;
52     xCii1 = setOutput;
53     xCii2 = 0;
54     xCi1 = 0;
55 }
56
57 void iiCTRL::changeK( float k2, float k1, float kp) {
58     changeK2(k2);
59     changeK1(k1);
60     changeKp(kp);
61 }
62
63 void iiCTRL::changeK2( float k2) { this->k2 = k2; }
64 void iiCTRL::changeK1( float k1) { this->k1 = k1; }

```

```
65 void iiCTRL::changeKp( float kp ) { this->kp = kp; }
```

Listing 7.10: Source Classe controllore C(s)

Appendice B

EMP Code

8.4 Pacchetti in EMP

8.4.1 Definizione di 2 Pacchetti

Ecco di seguito un esempio di definizione asimmetrica per pacchetti da un Device (Linux), verso un altro (Arduino), e viceversa:

```
1 #include <stdint.h> // To be sure for the footPrint in any platform
2 struct _packLinux2Ard {
3     int16_t num;
4     char buf[20];
5 } __attribute__((packed));
6 typedef struct _packLinux2Ard packLinux2Ard;
7
8 struct _packArd2Linux {
9     int16_t num;
10    char buf[10];
11 } __attribute__((packed));
12 typedef struct _packArd2Linux packArd2Linux;
```

Listing 8.11: Esempio di definizione Pacchetto in EMP

Come si può osservare, il sistema non necessita di alcun formato particolare per la definizione dei pacchetti, essi sono delle semplici strutture in C, con l'attributo `__attribute__((packed))` per eliminare Padding e ottimizzare la memoria. Come è possibile notare, le regole non vietano la definizione di 2 pacchetti diversi e asimmetrici.

8.4.2 Pacchetti Multipli

Per definire un pacchetto di pacchetti, il metodo migliore è creare un `enum` per numerare univocamente il tipo (nell'esempio `uint8_t` ma può essere reso maggiore all'occorrenza), e una `union` con tutti i tipi in fila.

Il pacchetto di interesse per Embedded Message Pack(EMP) è `dataSend`, ed è responsabilità dell'utente della libreria riempire correttamente i campi.

```

1 //(DataType1 ,DataType2 , DataType3  typedef  above;
2 enum packTypeSend : uint8_t{
3     DataType1code ,
4     DataType2code ,
5     DataType3code ,
6 };
7 typedef union{
8     DataType1 t1;
9     DataType2 t2;
10    DataType3 t3;
11 } __attribute__((packed)) dataSendUnion;
12
13 typedef struct{
14     packTypeSend type;
15     dataSendUnion pack;
16 } __attribute__((packed)) dataSend;
```

Listing 8.12: Definizione Pacchetti Multipli

Per ottimizzare l'invio al minimo strettamente necessario si può quindi usare la variante nel metodo di `packSend(...)`, in cui viene specificata la dimensione:

```

1 //...
2 dataSend.type = DataType2code;
3 dataSend.pack.t2 = /*{data}*/ // fill correctly dataSend
4 //...
5 MP->packSend(&data , sizeof(packTypeSend) + sizeof(DataType2));
6 //...
```

Listing 8.13: Definizione Pacchetti Multipli

Appendice C

Matlab Post Elaborazione

9.5 Importazione dei Dati

9.5.1 Parsing Tabelle

```
1 function [Table, Info] = importExperiment(filename)
2 % #####
3 % ### Mean Import ####
4 % #####
5 opts = delimitedTextImportOptions("NumVariables", 4);
6 % Specify range and delimiter
7 opts.DataLines = [2, 2];
8 opts.Delimiter = "\t";
9 % Specify column names and types
10 opts.VariableNames = ["V2_mean", "Isense_mean", "dt", "V2Ref_set"];
11 opts.SelectedVariableNames = ["V2_mean", "Isense_mean", "dt", "V2Ref_set"];
12 opts.VariableTypes = ["double", "double", "double", "double"];
13 % Specify file level properties
14 opts.ExtraColumnsRule = "ignore";
15 opts.EmptyLineRule = "read";
16 % Import the data
17 Info = readtable(filename, opts);
18 Info.dt = Info.dt * 1E-6; % dt shuld be in second
19 [~, name, ~] = fileparts(filename);
20 Info.Properties.Description = name;
21 % #####
22 % ### Data Import ####
23 % #####
24 % Set up the Import Options and import the data
25 opts = delimitedTextImportOptions("NumVariables", 3);
26 dataLines = [4, Inf];
27 % Specify range and delimiter
28 opts.DataLines = dataLines;
29 opts.Delimiter = "\t";
30 % Specify column names and types
31 opts.VariableNames = ["PWM", "V2_read", "Isense_read"];
32 opts.VariableTypes = ["double", "double", "double"];
33 % Specify file level properties
34 opts.ExtraColumnsRule = "ignore";
35 opts.EmptyLineRule = "read";
36 % Import the data
37 Table = readtable(filename, opts);
38 Table = tableConvert(Table, Info);
39 end
```

Listing 9.14: Parsing delle tabelle

9.5.2 Conversione dei Dati

Il parsing delle tabelle si appoggia alla conversione e filtraggio dei dati qui riportato:

```

1 function tCon = tableConvert(table , Info)
2 % Utility Factor
3 vScale = 5/(2^10-1);           % ADC sensitivity
4 IScale = vScale / 0.020;      % Isensitivity = 20mv/A
5 V2Mean = Info.V2_mean;
6 IsenseMean = Info.Isense_mean;
7 dt = Info.dt;
8 vRef = Info.V2Ref_set;
9 % New Table
10 tCon = table;
11 tCon.PWMDead = pwm2duty(tCon.PWM,60,210);
12 tCon.PWM=tCon.PWM/255;
13 tCon.V2_read = (table.V2_read - V2Mean) * vScale;
14 tCon.Isense_read = (table.Isense_read - IsenseMean) * IScale;
15 tCon.error = (tCon.V2_read - vRef * vScale);
16
17 d = designfilt('lowpassfir', ...
18 'PassbandFrequency',0.01,'StopbandFrequency',0.4, ...
19 'PassbandRipple',1,'StopbandAttenuation',60, ...
20 'DesignMethod','equiripple');
21 tCon.V2_readFilter = filtfilt(d,tCon.V2_read);
22 end

```

Listing 9.15: Parsing delle tabelle

Usando come funzioni accessorie:

```

1 function u = pwm2duty(pwmList , downDead , upDead)
2 u = zeros(length(pwmList),1);
3 for i = 1:length(pwmList)
4     pwm = pwmList(i);
5     if (abs(pwm)>=upDead)
6         u(i)=sign(pwm);
7         continue
8     end
9     if (abs(pwm)<=downDead)
10        u(i) = 0;
11        continue
12    end
13    if (pwm>0)
14        u(i) = map(pwm,downDead,upDead,0,1);
15        continue
16    else
17        u(i) = -map(-pwm,downDead,upDead,0,1);
18        continue
19    end
20 end
21 end
22
23 function u = map ( var , varMin , varMax , outMin , outMax)
24     u = ((var-varMin)/(varMax-varMin));
25     u = (u * outMax) + ((1-u)*outMin);
26 end

```

Listing 9.16: Parsing delle tabelle

9.6 Stima Parametri

9.6.1 Stima parametri automatica

```

1 % ##### PWM-I2 Auto Estimation #####
2 % DAT = iddata(Y,U,Ts)
3 PrimaryFrameData = iddata(experiment.Isense_read, experiment.PWMDead, Info.dt);
4
5 % ##### PWM-I1 Estimation #####
6 % DAT = iddata(Y,U,Ts)
7 InPriRiseFrameData = iddata(experiment.Isense_read(1:1200), experiment.PWMDead
8 (1:1200), Info.dt);
9
10 % SYS = tfest(DATA, NP, NZ, IODELAY)
11 tfExtPWM_I1 = tfest(InPriRiseFrameData, 1, 0, NaN)
12 % lsim(SYS,U,T)
13 yExtPWM_I1Fit = lsim(tfExtPWM_I1, experiment.PWMDead(1:1200), t(1:1200));
14
15 % ##### Primary - Secondary Rise Estimation #####
16 % DAT = iddata(Y,U,Ts)
17 PriSecRiseFrameData = iddata(experiment.V2_read(1:1200), yExtPWM_I1Fit, Info.dt);
18 tfExtI1_V2 = tfest(PriSecRiseFrameData, 1, 1, NaN)
19
20 % ##### Simulation #####
21 yExtPWM_I1 = lsim(tfExtPWM_I1, experiment.PWMDead, t);
22 yExtI1_V2 = lsim(tfExtI1_V2, yExtPWM_I1, t);
23
24 figure(1)
25 plotTable(experiment, Info, [0.2, 1.2], [-inf, inf], [yExtPWM_I1, yExtI1_V2], {'yExtPWM-I1',
26 'yExtI1-V2'}, 'autoExt');

```

Listing 9.17: Stima corrente Primario automatica

9.6.2 Stima parametri manuale

```

1 % ##### PWM-I2 Manual Estimation #####
2 % fPWM_I1 = tf([0.022 1]);
3 % yfPWM_I1 = lsim(fPWM_I1, experiment.PWMDead, t);
4
5 fI1_V2 = tf([0.0085 0],[1/2800 1]);
6 yfI1_V2 = lsim(fI1_V2,yfPWM_I1,t);
7
8 figure(2)
9 plotTable(experiment, Info, [0.2, 1.2], [-inf, inf], [yfPWM_I1, yfI1_V2], {'yfPWM-I1',
10 'yfI1-V2'}, 'manualExt');

```

Listing 9.18: Stima parametri manuale

9.6.3 Plot dell'esperimento

```

1 function plotTable(table, Info, xLim, yLim, ySim, ySimLegend, nameSave)
2 V2Mean = Info.V2_mean;
3 IsenseMean = Info.Isense_mean;
4 dt = Info.dt;
5 vRef = Info.V2Ref_set;
6
7 name = Info.Properties.Description;
8 [len, ~] = size(table);
9
10 t = [0:len-1]' * dt;
11
12 clf
13 grid on
14 hold on
15 plot(t,table.PWMDead,'LineWidth',2)
16 plot(t,table.V2_readFilter,'LineWidth',2)
17 plot(t,table.Isense_readFilter,'LineWidth',2)
18 for y = ySim
19     plot(t,y,'LineWidth',1);
20 end
21 legNorm = { 'PWM%-DeadFilter', 'V_{secondary}', 'I_{primary}' };
22
23 legend(cat(2,legNorm,ySimLegend));
24
25 title("Experiment: "+name+ " SampleTime: " + mat2str(dt * 1000000) + "us")
26 xlabel("time [s]")
27 xlim(xLim);
28 ylim(yLim);
29
30 % Figure Export and Save
31 savePath = "./img/";
32 [~,~] = mkdir(savePath);
33 set(gcf, 'PaperUnits', 'normalized')
34 set(gcf, 'PaperPosition', [0 0 1 1])
35 set(gcf, 'PaperOrientation', 'landscape');
36 saveas(gcf,savePath+name+'_'+nameSave+'.png')
37 saveas(gcf,savePath+name+'_'+nameSave+'.pdf')
38 end

```

Listing 9.19: Plot dell'esperimento

Elenco delle figure

1	ACS770LCB-100B-PFF-T Schema di collegamento dal Datasheet	4
1.1	Schema interno di un Tokamak	7
1.2	Trasformatore ideale con Campo magnetico	10
1.3	Circuito Equivalente Bobina/Plasma all'interno di un Tokamak	12
1.4	Circuito Equivalente Bobina/Plasma all'interno di un Tokamak trascurando l'induzione del plasma verso la bobina	13
1.5	Schema a Blocchi della funzione di trasferimento della corrente di Plasma	16
1.6	Circuito reale di misura dell'esperimento	17
1.7	Sensore di Corrente ACS770LCB-100B-PFF-T	18
1.8	ACS770LCB-100B-PFF-T Schema a Blocchi	20
1.9	ACS770LCB-100B-PFF-T Sensibilità rispetto Temperatura	21
1.10	ACS770LCB-100B-PFF-T <i>Non-Linearità</i>	21
1.11	ACS770LCB-100B-PFF-T Schema di collegamento dal Datasheet	22
1.12	Driver Motori IBT-2 TopView & PinOut	24
1.13	IBT-2 Schema Elettrico	25
1.14	Esperimento con Onda Triangolare	27
1.15	Esperimento con Onda Trapezoidale	28
2.1	Schema finale dell'archittettura di controllo	29

2.2	Scheda Arduino Uno	30
2.3	Esempio di COBS	32
2.4	Esempio di COBS con distanza	33
2.5	Class Diagram UML di EMP	36
2.6	Sequence Diagram UML di EMP	38
2.7	EMP Benchmark Testing Flow	39
2.8	Circuito equivalente del Plasma con l'offset delle Misure	40
3.1	Simulazione con stima automatica	46
3.2	Simulazione con parametri tarati a mano	47
3.3	Esperimento di verifica della stima 1	48
3.4	Esperimento di verifica della stima 2	49
3.5	Esperimento di verifica della stima 3	49
4.1	Sistema da controllare a blocchi con $P(s)$	51
4.2	Sistema da controllare ingresso-uscita reale	52
4.3	Modelli simulati su Simulink	57
4.4	Modelli simulati su Simulink	58
4.5	Controllo Avanzato $K_p=0 K_1=0 K_2=100$	59
4.6	Controllo Avanzato $K_p=0 K_1=10 K_2=100$	60
4.7	Controllo Avanzato $K_p=0 K_1=100 K_2=500$	61
4.8	Controllo Avanzato $K_p=0.2 K_1=100 K_2=500$	62
4.9	Controllo Avanzato $K_p=0.8 K_1=100 K_2=500$	63
5.1	Impianto Reale	65
5.2	Discretizzazione Zero-Order Hold $H_d(z)$ del sistema Tempo continuo $H(s)$	67
5.3	Effetto sui segnali discretizzati con il metodo Zero-Order Hold	67

5.4 Esperimento singolo $V_{2ref} = 0.5V$	70
5.5 Inseguimento Triangolare $V_{2ref} = \pm 0.5V$	71
5.6 Cambio di V_{2ref} prima della saturazione	72

Elenco delle tabelle

1.1	Allegro, <i>High-Precision Linear Current Sensor</i> Riassunto caratteristiche	18
1.2	ACS770LCB-100B-PFF-T Particolarità	19
1.3	ACS770LCB-100B-PFF-T Parametri di Interesse	22
1.4	IBT-2 Specifiche	24
1.5	Fasce della Dead-Zone	28
2.1	Salvataggio di "struct setUpPack"	42
2.2	Salvataggio di "struct sample" $\forall dt$	42
3.1	Stime parametri con <i>System Identification Toolbox</i>	45
3.2	Stime parametri tarati a mano	47
5.1	Funzioni di trasferimento nello spazio di Stato, da Tempo Continuo a Tempo Discreto	68

Elenco dei codici

2.1	Pacchetti Companion $\Rightarrow \mu\mathbf{Controllore}$	41
2.2	Pacchetti $\mu\mathbf{Controllore} \Rightarrow$ Companion	41
7.1	Tic Timer	75
7.2	Frequenza PWM	75
7.3	Macro per calcolo Offset	76
7.4	Rampa Saturata	77
7.5	Onda Triangolare Periodica	78
7.6	Onda Trapezoidale Periodica (RapidShot)	78
7.7	Setup del Controllore	79
7.8	Loop di Controllo	79
7.9	Header Classe controllore C(s)	80
7.10	Source Classe controllore C(s)	81
8.11	Esempio di definizione Pacchetto in EMP	83
8.12	Definizione Pacchetti Multipli	84
8.13	Definizione Pacchetti Multipli	84
9.14	Parsing delle tabelle	85
9.15	Parsing delle tabelle	86
9.16	Parsing delle tabelle	86
9.17	Stima corrente Primario automatica	87

9.18 Stima parametri manuale	87
9.19 Plot dell'esperimento	88

Bibliografia

Alfano: Embedded Message Pack(EMP)	EMP
Emanuele Alfano. <i>Embedded Message Pack(EMP)</i> . Repository. 24 Sep 2021. URL: https://github.com/Alfystar/EMP .	
Allegro: High-Precision Linear Current Sensor	ACS770
Allegro. <i>High-Precision Linear Current Sensor</i> . ACS770LCB-100B-PFF-T. Datasheet. 31 May 2019. URL: https://www.allegromicro.com/~/media/Files/Datasheets/ACS770-Datasheet.pdf .	
Carnevale: Performances and robustness	PerfAndRobust
Daniele Carnevale. «Performances and robustness». In: Università di Tor Vergata. 19 Oct 2017. URL: https://drive.google.com/file/d/15sorq07mvLST4o60CxF3DakUI8L3cFU_/view .	
Checksum Calculation 8 Bit (CRC8)	CRC8
<i>Checksum Calculation 8 Bit (CRC8)</i> . Rapp. tecn. Mouser Electronics, 15 Apr 2019. URL: https://www.mouser.com/pdfDocs/SFM4100_CRC_Checksum_Calculation.pdf .	
Cianfarani: MAGNETIC DIAGNOSTICS FOR FUSION MACHINES	MagneticDiagnostics
Cesidio Cianfarani. «MAGNETIC DIAGNOSTICS FOR FUSION MACHINES». In: <i>MAGNETIC DIAGNOSTICS FOR FUSION MACHINES</i> . Associazione Euratom-ENEA sulla Fusione. 11 Apr 2013. URL: https://www.fsn-fusphy.enea.it/Intranet/index.php/seminari-interni/download-file?path=Seminari-Workshop-WIP-Corsi+Interni%2FCorsi+Interni%2FCorso+Rd0+2015%2FSLIDES%26+AUDIO%2F13+Diagnostiche+Magnetiche%28Cianfarani%29%2FMagnetic+Diagnostics+2.2+-+Cianfarani.pdf .	
Handsontec: BTS7960 High Current 43A H-Bridge Motor Driver	IBT-2
Handsontec. <i>BTS7960 High Current 43A H-Bridge Motor Driver</i> . Datasheet. 18 Sep 2019. URL: https://www.handsontec.com/dataspecs/module/BTS7960%20Motor%20Driver.pdf .	
IEEE: Consistent Overhead Byte Stuffing (COBS)	COBS
IEEE. <i>Consistent Overhead Byte Stuffing (COBS)</i> . Rapp. tecn. IEEE, 29 Jun 2002. URL: https://www.ieee802.org/3/efm/public/jul02/copper/oksman_copper_1_0702.pdf .	

Infineon: BTS7960B High Current PN Half Bridge NovalithIC T **BTS7960b**

Infineon. *BTS7960B High Current PN Half Bridge NovalithIC T*. BTS7960b. Datasheet. 12 Jul 2016. URL: shorturl.at/cw0Z8.

Johnston: Creating a Circular Buffer in C and C++ **CircularBuffer**

Phillip Johnston. *Creating a Circular Buffer in C and C++*. Online Page. 10 june 2021. URL: <https://embeddedartistry.com/blog/2017/05/17/creating-a-circular-buffer-in-c-and-c/>.

Matlab: System Identification Toolbox **IdentificationToolbox**

Matlab. *System Identification Toolbox*. Online Page. URL: <https://it.mathworks.com/products/sysid.html>.

Matlab: Zero-phase digital filtering **zeroPhaseShiftFilter**

Matlab. *Zero-phase digital filtering*. Online Manual. 2006. URL: <https://it.mathworks.com/help/signal/ref/filtfilt.html>.

rdavide: Trasformata di Laplace **Laplace**

rdavide. «Trasformata di Laplace». In: Università di Pavia. 2 Apr 2015. URL: <http://sisdin.unipv.it/labsisdin/teaching/courses/falt/files/Laplace.pdf>.

Redazione: Nozioni di PWM **modulazionePWM**

Redazione. *Nozioni di PWM*. Online Page. 20 october 2019. URL: <https://www.ingegnerando.it/nozioni-di-pwm/>.

Romero et al.: Real time current profile control at JET **TokamakCircuit**

Jesús Romero et al. «Real time current profile control at JET». In: *Fusion Engineering and Design* 43 (feb. 1998), pp. 37–58. DOI: [10.1016/S0920-3796\(98\)00261-0](https://doi.org/10.1016/S0920-3796(98)00261-0).

Trasformatore Monofase **Transformatore**

Trasformatore Monofase. URL: <https://www.docenti.unina.it/webdocenti-be/allegati/materiale-didattico/34030176>.

Zanasi: Discretizzazione di un sistema tempo continuo **Discretizzazione**

Roberto Zanasi. «Discretizzazione di un sistema tempo continuo». In: Università di Modena e Reggio Emilia. 27 Jun 2012. URL: http://www.dii.unimo.it/~zanasi/didattica/Teoria_dei_Sistemi/Luc_TDS_2006_Discretizzazione_di_un_sistema_continuo.pdf.

Zanasi: Forme Canoniche Spazio di Stato **FormeCanoniche**

Roberto Zanasi. «Forme Canoniche Spazio di Stato». In: Università di Modena e Reggio Emilia. 4 Gen 2010. URL: http://www.dii.unimo.it/~zanasi/didattica/Teoria_dei_Sistemi/Luc_TDS_2006_Forme_Canoniche.pdf.