Politecnico
di Torino

Politecnico di Torino

Ingegneria Aerospaziale

A.A. 2024/2025

Sessione di laurea Ottobre 2025

Studio e analisi elettromagnetica di valvole a solenoide

Progettazione, simulazione e validazione di un comando di pilotaggio di una valvola di tipo latching

Relatore: Paolo Maggiore

Candidato: Aurora Boccone

Ringraziamenti

Sommario

Il presente lavoro di tesi si inserisce all'interno di un'attività di sviluppo tecnologico condotta in collaborazione con **APR srl**, attualmente impegnata nella progettazione e realizzazione di una nuova valvola a ritenuta ad alta pressione, della cui tipologia, ad oggi, non ne esistono ancora fornitori in Europa.

L'obiettivo principale consiste nella progettazione, realizzazione e validazione di un circuito elettronico in grado di simulare e testare il funzionamento della valvola e dei suoi componenti, riproducendo in modo realistico il comando di pilotaggio della valvola.

Il circuito è stato sviluppato tenendo conto delle specifiche tecniche imposte dal progetto in corso, tra cui la necessità di generare impulsi di corrente brevi e controllati per l'attivazione dei solenoidi responsabili dell'apertura e della chiusura.

Attraverso simulazioni preliminari e prove su breadboard, il lavoro ha permesso di verificare la fattibilità del sistema, producendo un primo prototipo funzionante, utile per la successiva fase di test funzionale del componente. I risultati ottenuti rappresentano un contributo concreto alle attività di validazione del prodotto, e costituiscono la base per future evoluzioni del circuito in vista della qualifica spaziale.

Indice

Elenco delle figure

Elenco delle tabelle

Abbreviazioni

AC

Alternating current

ADC

Analog-to-Digital converter

CPU

Central processing unit

DAC

Digital-to-Analog converter

DC

Direct current

GND

Ground

GSE

Ground support equipment

HPLV

High pressure latching valve

I2C

Inter-Integrated Circuit

IC

Integrated circuit

IDE

Integrated development environment

LV

Latching valve

MCU

Microcontroller unit

PCB

Printed circuit board

PWM

Pulse width modulation

Capitolo 1

Introduzione

Il presente capitolo fornisce un'introduzione generale al lavoro, con particolare attenzione al contesto di riferimento, alle motivazioni e agli obiettivi della tesi.

1.1 Contesto generale del lavoro

Nel campo spaziale, in particolare nell'ambito dei sistemi propulsivi, sono di grande importanza le valvole. Questi particolari dispositivi sono parte integrante dei sottosistemi presenti a bordo di un generico satellite. Essi permettono di svolgere importanti funzioni al fine di permettere al satellite di portare a termine la sua missione.

Le valvole svolgono ruoli cruciali. Se ci si sofferma sulle valvole presenti nel sistema propulsivo è possibile individuare diverse funzioni alle quali le valvole sono associate: controllo del flusso di fluidi, regolazione della pressione, mantenimento della corretta miscelazione di propellenti, ...
A seconda quindi della funzione richiesta, vengono classificate in diverse tipologie alle quali corrispondono architetture e caratteristiche diverse.

Nel seguente lavoro di tesi si prende in considerazione quella tipologia di valvola che gestisce il mantenimento della posizione, detta *latching valve* (HPLV). Il passaggio del fluido all'interno della valvola viene controllato tramite un comando di tipo elettrico (presenza di avvolgimenti interni) che definisce lo stato della valvola: aperta o chiusa.
Il mantenimento dello stato, e quindi il passaggio o meno del fluido, viene garantito fintanto che non viene fornito un altro comando, sempre di tipo elettrico, che va ad invertire la condizione in cui il sistema si trova al momento dell'attivazione del comando stesso.
Nel Capitolo 2.1 viene approfondita meglio l'architettura e il funzionamento di questa tipologia di valvole.

Le LV fanno parte della grande famiglia delle valvole a solenoide, in quanto il loro funzionamento è basato proprio sul comando elettrico fornito dal passaggio della corrente all'interno degli avvolgimenti del solenoide.

1.2 Scopi e obiettivi

Lo scopo del seguente lavoro è quello di svolgere uno studio delle caratteristiche elettriche delle valvole a solenoide.
Il lavoro si articola in diversi obiettivi. In primo luogo viene svolta la caratterizzazione degli avvolgimenti interni ad una valvola a solenoide, mediante una misura effettiva delle grandezze elettriche del sistema. Successivamente viene sviluppato un dispositivo di simulazione del pilotaggio elettrico della valvola di tipo latching, con l'obiettivo di comprendere l'architettura dietro ad un comando di commutazione. E' inoltre prevista una verifica della tensione di Pull-In[1] sul prototipo sviluppato, al fine di valutarne la compatibilità con le

[1]Il Pull-In è la massima tensione richiesta per l'apertura della valvola negli intervalli di pressione operativa e temperatura specificati nei requisiti di progetto.

specifiche di progetto.
Per raggiungere tali obiettivi vengono realizzati diversi circuiti elettronici, che consentono di soddisfare le varie fasi dello studio.

Il progetto di un dispositivo aerospaziale prevede di svolgere le cosiddette operazioni di validazione, fondamentali per accertare il corretto funzionamento e l'affidabilità dei componenti prima dell'impiego operativo.
Queste operazioni vengono condotte utilizzando appositi sistemi e strumenti noti come **GSE** – *Ground Support Equipment*, di cui il prototipo, sviluppato nel corso del seguente lavoro, ne è un esempio. Si tratta di equipaggiamenti di supporto a terra progettati per simulare le condizioni operative reali, fornire alimentazione, controllare parametri di funzionamento, monitorare le risposte del sistema e acquisire dati durante le prove.
L'utilizzo dei GSE consente di effettuare test in un ambiente controllato, riproducendo scenari realistici e garantendo così una validazione accurata e sicura delle funzionalità del componente o del sistema in esame.

In sintesi, il lavoro riproduce il comportamento elettrico dei solenoidi e della valvola latching mediante l'impiego congiunto di piattaforme hardware, software e microcontrollori.

1.3 Motivazione dello studio

Nel caso di componenti meccanici come le valvole, che svolgono una funzione pratica e operano in condizioni reali, la sola progettazione teorica e la successiva produzione non sono sufficienti a garantirne l'affidabilità e le prestazioni.
Occorre dunque includere una fase di pre-test (una fase di testing sviluppata durante la progettazione della valvola, quindi prima dell'effettiva produzione del dispositivo) in modo da poter studiare diverse opzioni di realizzazione e

osservarne il comportamento.

Durante la fase di progettazione della valvola è molto importante definire le varie caratteristiche elettriche dei componenti interni, verificando quali combinazioni di valori rispecchiano meglio ciò che si vuole ottenere come comportamento del sistema nell'utilizzo finale. Così facendo sarà possibile scegliere l'opzione più opportuna in termini di risposta e tempi di funzionamento.

È quindi indispensabile svolgere questa fase per verificare che il componente funzioni correttamente nelle condizioni operative previste e che rispetti i requisiti prestazionali definiti in fase di progetto. Questo processo consente di validare le scelte progettuali e di individuare eventuali criticità prima della produzione, in modo da ottenere un componente che sia il più adatto all'impiego finale.

1.4 Struttura del lavoro

Il lavoro di concentra su due macro-obiettivi principali: analisi delle caratteristiche elettriche delle valvole a solenoide e sviluppo di un sistema per il pilotaggio della valvola latching. Per garantire un'analisi completa e coerente con tali obiettivi, lo studio è stato articolato in diverse fasi.
La scelta di strutturare il lavoro in fasi permetterà di analizzare in modo più chiaro ogni aspetto, facendo emergere gli obiettivi principali per ciascuna fase.

Le attività condotte nel corso del lavoro, volte al conseguimento degli obiettivi della tesi, possono essere articolate nei seguenti step operativi:

1. Comprensione e studio dell'argomento in esame, mediante documentazione ed approfondimento delle principali tematiche trattate con l'obiettivo di individuare gli strumenti più adatti alla realizzazione del lavoro;
2. Caratterizzazione degli avvolgimenti in una valvola a solenoide;

3. Realizzazione e simulazione del circuito elettrico rappresentativo del comando di commutazione della valvola mediante l'utilizzo del software MATLAB/Simulink;

4. Sviluppo del codice volto alla programmazione del microcontrollore sfruttando la piattaforma software-hardware di Arduino e realizzazione del circuito per il pilotaggio della valvola;

5. Osservazione dei risultati.

Prima di procedere con lo svolgimento dell'attività, è opportuno soffermarsi sui vari componenti coinvolti (già citati o che verranno menzionati), al fine di fornire una visione chiara e precisa dei concetti e dei particolari funzionamenti.

Capitolo 2

Materiali, strumenti e metodi

Oggetto della tesi è dunque la realizzazione di un GSE che permetta di validare la risposta elettrica della HPLV.
Nel presente capitolo viene posta particolare attenzione ai supporti fisici (hardware) e non (software) che verranno citati e/o utilizzati nel corso del progetto.

Come primo elemento si presentano le caratteristiche fondamentali della LV, oggetto centrale dello studio. Pur non trattandosi di un componente fisico direttamente impiegato nelle attività sperimentali, si ritiene opportuno dedicarle una sezione introduttiva.

2.1 Latching Valve

Le **latching valves**, ovvero le valvole a ritenuta, sono utilizzate nei sistemi di propulsione aerospaziale per isolare i rami fluidici, sia in caso di guasto di un componente a valle, sia durante lunghi periodi di inattività. Le valvole a ritenuta sono progettate per mantenere la loro posizione anche dopo la

rimozione dell'alimentazione elettrica. L'alimentazione viene applicata alla valvola solo per cambiarne lo stato, rendendole componenti ideali per questa funzione [1].

Il funzionamento della HPLV, in corso di progettazione, è basato su un solenoide a doppia bobina in cui viene installato un magnete permanente. Il magnete, del tipo SmCo[1], viene utilizzato per mantenere in posizione lo *spool* ferromagnetico situato all'interno alla valvola che, a sua volta, permette il passaggio o meno del fluido.

Il solenoide, essendo composto da due bobine, comanda sia l'apertura che la chiusura della valvola, a seconda di quale dei due avvolgimenti riceva corrente. Inoltre, la valvola prevede la presenza di una coppia di sensori ad effetto Hall[2] usati per identificare e fornire il feedback dello stato della valvola.

Per comandare la posizione della valvola (chiusa o aperta) viene utilizzato un impulso di corrente applicato per un intervallo di tempo prestabilito.

Nella figura 2.1 è possibile avere una prima visualizzazione della valvola in esame.

2.1.1 Applicazioni della valvola

Come detto in precedenza l'utilizzo di questa valvola è importante dal punto di vista della propulsione di un satellite.
Questa tipologia di valvole solitamente viene utilizzata posizionandola in serie ad un'altra, solitamente una valvola della tipologia regolatrice di portata.

[1]Elementi della tavola periodica: Samario (Sm) e Cobalto (Co)

[2]Per approfondimenti sull'effetto Hall vedere l'Appendice A

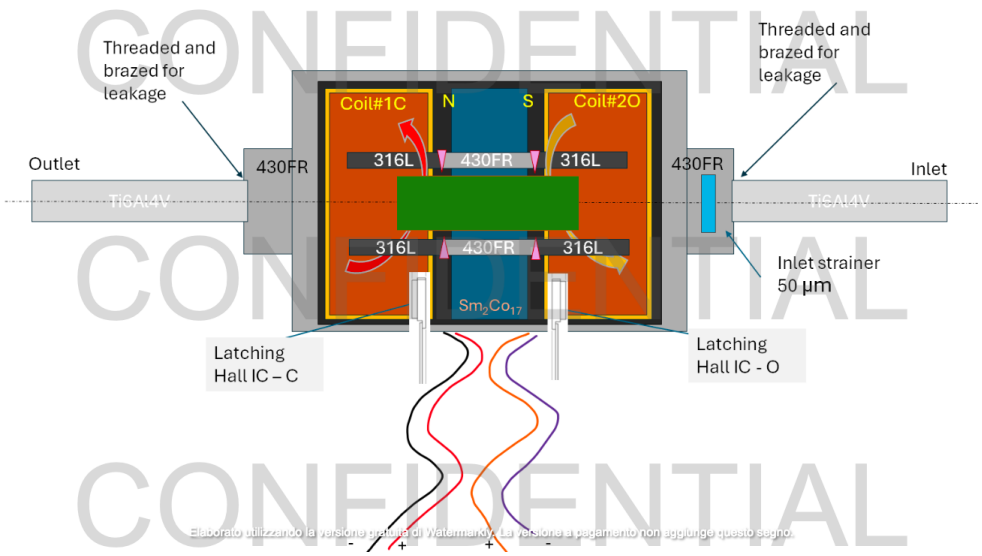


Figura 2.1: Rappresentazione schematica della HPLV di proprietà di APR srl.

Lo scopo della LV è quello della gestione ed erogazione del gas propulsivo che alimenta la valvola a valle.

Andando a gestire l'emissione di gas attraverso queste valvole è possibile svolgere diverse operazioni in orbita:

- Controllo dell'assetto → Variazione dell'orientamento del veicolo spaziale nello spazio [2]. Il sistema propulsivo viene quindi utilizzato per controllare l'assetto del satellite in quanto, nel corso della missione in orbita, il satellite è soggetto a variazioni dell'assetto e dunque potrebbe essere necessario controllarlo e riposizionarlo;

- Controllo (e mantenimento) dell'orbita → Gestione/Modifica di uno o più elementi orbitali. Una volta che il veicolo spaziale si è separato dal lanciatore o dallo stadio superiore, non è più possibile esercitare alcun controllo sull'orbita del satellite, il quale sarà soggetto alla resistenza aerodinamica e al decadimento orbitale [2]. Per questo motivo è necessario implementare a bordo del satellite un sistema di controllo

orbitale in modo che ci sia la possibilità di controllare e mantenere l'orbita desiderata.

Una delle possibili applicazioni di questo tipo di valvola riguarda la gestione delle fasi finali di un trasferimento in orbita geostazionaria.

Raggiunta l'orbita desiderata, il sistema propulsivo entra nuovamente in gioco e la LV, a seconda della portata richiesta dal sistema di spinta, si attiverà in modo da fornire la quantità necessaria di gas per consentire la gestione del controllo dell'assetto, il mantenimento dell'orbita e la correzione dell'orientamento del satellite.

2.2 Architettura hardware

In questa sezione viene presentata la descrizione dei componenti fisici utili alla realizzazione del comando di pilotaggio della valvola.

2.2.1 Driver pilotaggio solenoidi

I driver di pilotaggio sono dispositivi elettronici. Si tratta di circuiti integrati (IC), ovvero chip che racchiudono al loro interno tutta l'elettronica necessaria a gestire l'attività dei dispositivi che devono comandare; questi dispositivi possono essere solenoidi e/o motori elettrici.

Nel caso dei solenoidi, i driver gestiscono la corrente che pilota le bobine. Questo fa si che il driver, tramite la gestione della corrente che entra nel solenoide, controlli lo stato della valvola.

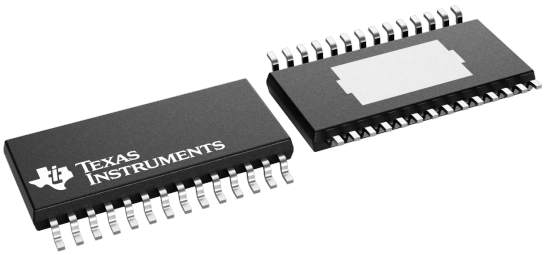


Figura 2.2: Driver DRV8844. Fonte: Texas Instruments, TI-DRV8844.

I driver consistono in involucri di plastica dotati di "piedini", detti pin, che servono a mettersi in contatto elettrico con il resto del circuito stampato e hanno funzionalità differenti come canali di input/output, di alimentazione, di terra, ecc.

Per poter funzionare e controllare i dispositivi a cui è collegato, il driver richiede dei segnali di input, che possono essere forniti da sistemi esterni, come ad esempio un microcontrollore.

Esistono differenti tipologie di driver, selezionate in base alle esigenze applicative, e si distinguono in base alla disposizione dei pin e alle caratteristiche interne del driver stesso: nel presente lavoro è stato individuato un driver in particolare in funzione delle esigenze richieste dal progetto.

La scelta effettuata (Driver **DRV8844** mostrato in Figura 2.2) e le relative motivazioni sono illustrate nella Sezione 3.1.

E' opportuno specificare che per la realizzazione pratica del comando di pilotaggio, il driver non è indispensabile. Questo strumento trova utilità nel momento in cui si andrà a costruire realmente la valvola in modo che i comandi, le istruzioni e il dialogo con il microcontrollore avvengano correttamente e in modo preciso.

Nel seguito viene brevemente descritto il funzionamento di quest'ultimo dispositivo.

2.2.2 Microcontrollore

Il microcontrollore, detto anche MCU, è anch'esso un circuito integrato che racchiude al suo interno diversi componenti. I principali, ovvero quelli che caratterizzano questo particolare dispositivo, sono i seguenti:

- CPU → unità centrale di elaborazione che ha il compito di eseguire le istruzioni di un programma, effettuare calcoli, gestire i dati e coordinare il funzionamento delle altre componenti del sistema, come memoria e periferiche;

- Memoria → sia volatile che non volatile, utilizzata per conservare temporaneamente e definitivamente delle istruzioni;

- Periferiche di input → ricevono informazioni dall'ambiente esterno o da altri dispositivi;

- Periferiche di output → inviano comandi o segnali dal microcontrollore verso dispositivi esterni.

Nel presente lavoro è stato scelto di utilizzare il microcontrollore integrato nella scheda Arduino UNO, ovvero l'**ATmega328P** prodotto da Atmel.

Tale scelta è stata motivata dalla facilità d'uso della piattaforma, supportata dalla disponibilità dell'ambiente di sviluppo integrato (IDE) che permette di programmare e configurare il microcontrollore in modo semplice ed efficace.

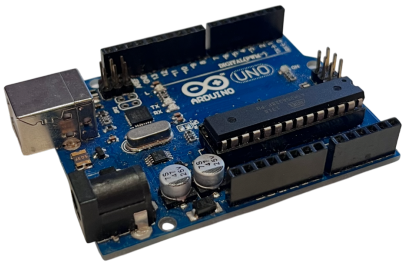


Figura 2.3: Scheda Arduino UNO.

Sulla scheda sono presenti dei fori, denominati anch'essi pin, che consentono di realizzare collegamenti con strumentazione esterna, permettendone l'integrazione e il funzionamento congiunto.

2.2.3 Componenti di completamento

Infine è importante andare a citare i vari componenti elettronici che si andranno ad utilizzare per l'assemblaggio del circuito rappresentativo della HPLV.

Breadboard

Si tratta di uno strumento utilizzato per la realizzazione di circuiti elettronici senza necessità di saldature. Funge da banco di partenza per la costruzione fisica del circuito, è composta da fori collegati elettricamente in maniera interna, che permette di inserite i terminali di altri componenti elettronici (cavi jumper, resistori, induttori, micro-controllori, ...).

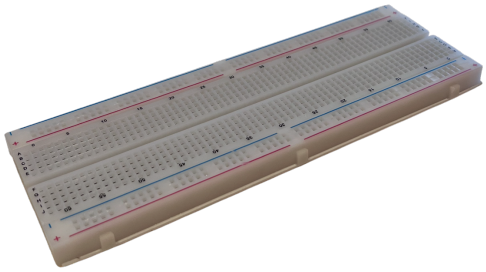


Figura 2.4: Illustrazione della breadboard (tavoletta sperimentale).

Cavi jumper

Sono dei fili elettrici, di ridotte dimensioni e in rame che presentano dei connettori in corrispondenza delle estremità. Questi cavi vengono utilizzati per fare da collegamento tra i vari componenti elettronici che vengono posizionati sulla breadboard e per collegare la breadboard a componenti esterni come il microprocessore.

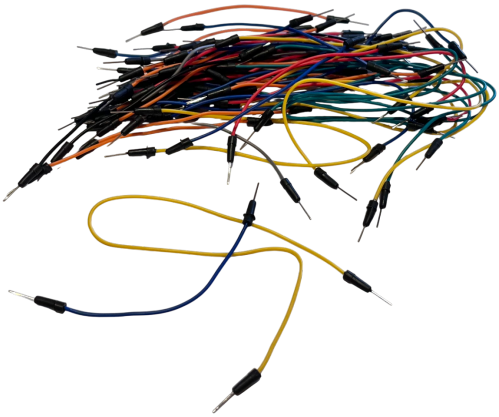


Figura 2.5: Illustrazione di una serie di cavi jumper di collegamento.

DAC

Il DAC (Digital-to-Analog Converter) è un piccolo chip che prende un numero digitale proveniente da un dispositivo esterno (che può trattarsi della scheda Arduino UNO) e genera in uscita una tensione continua reale proporzionale. Una tipologia molto utilizzata con Arduino è l'**MCP4725**, che si collega alla scheda tramite due uscite: le porte SDA e SCL del chip.

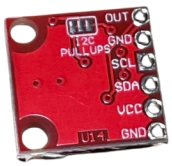


Figura 2.6: Illustrazione del DAC (chip per la conversione di un segnale digitale in uno analogico).

Regoltore di tensione

Un regolatore di tensione consente di modulare il valore della tensione fornita a un circuito, permettendone la variazione entro un intervallo prestabilito. In particolare, il regolatore **MIC29502WT** sarà integrato in un circuito alimentato a 5 V e consentirà di erogare una tensione compresa tra 0 V e 26 V.

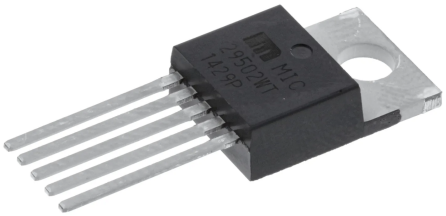


Figura 2.7: Regolatore di tensione MIC29502WT. Fonte: RS Components S.r.l., MIC29502WT.

Potenziometro digitale

Il potenziometro digitale è un dispositivo elettronico che consente di emulare una resistenza variabile, permettendone la regolazione in modo controllato. La regolazione della resistenza avviene in modo lineare rispetto al comando digitale. Integrato all'interno di un circuito, ha il vantaggio di poter modificare il valore di resistenza in maniera automatizzata, senza necessità di intervento manuale. Il modello scelto (**DS3502**) può essere utilizzato con i microcontrollori da 3,3 V e 5 V come Arduino.

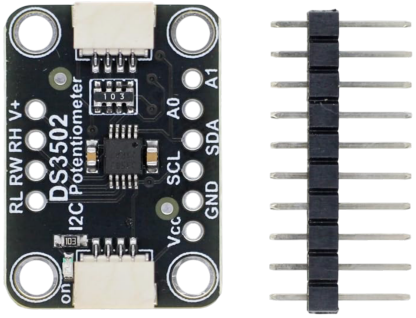


Figura 2.8: Illustrazione del potenziometro digitale (chip per la regolazione della resistenza).

Resistori

Un resistore ideale è un dispositivo che evidenzia proprietà di resistenza lineare in accordo con la legge di Ohm[3] [...] cioè che la tensione ai capi di un elemento è direttamente proporzionale al flusso di corrente lungo di esso, applicando la convenzione dell'utilizzatore [3].

[3] V = IR, dove V è la tensione, I è la corrente e R è la resistenza in un circuito elettrico.

Figura 2.9: Illustrazione di un gruppo di resistori.

Induttori

Nei circuiti elettrici esistono due diversi meccanismi per accumulare energia: la capacità e l'induttanza, che consentono entrambi di accumulare energia in un campo elettromagnetico. L'induttore rappresenta le proprietà ideali dell'accumulo induttivo di energia [3].

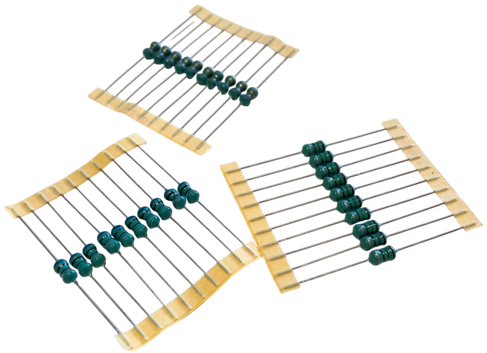


Figura 2.10: Illustrazione di un gruppo di induttori.

Condensatori

Il condensatore è un dispositivo che può immagazzinare energia separando delle cariche elettriche quando viene opportunamente polarizzato da un campo elettrico (ossia da una tensione) [3].

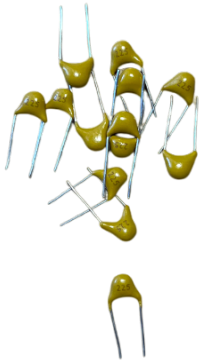


Figura 2.11: Illustrazione di un gruppo di condensatori.

2.3 Architettura software

In questa sezione viene presentata una breve introduzione ai software che hanno rivestito un ruolo fondamentale nello svolgimento delle attività necessarie al raggiungimento degli obiettivi del lavoro.

2.3.1 MATLAB/Simulink

MATLAB/Simulink è un ambiente di sviluppo integrato prodotto da MathWorks, ampiamente utilizzato in ambito scientifico e ingegneristico.

- **MATLAB** è un linguaggio di programmazione e una piattaforma numerica pensata per l'elaborazione di dati, la simulazione e l'analisi matematica. Il codice utile alla programmazione viene articolato nello *Script*, le variabili/costanti create vengono visualizzate nel *Workspace* mentre i risultati nella *Command Window* (si veda Figura 2.12).

- **Simulink** è un'estensione grafica di MATLAB che permette di modellare, simulare e analizzare sistemi dinamici attraverso schemi a blocchi. Questi vengono trascinati dal *Library Browser* verso l'ambiente di simulazione nel quale si può andare a modellare un circuito elettronico.

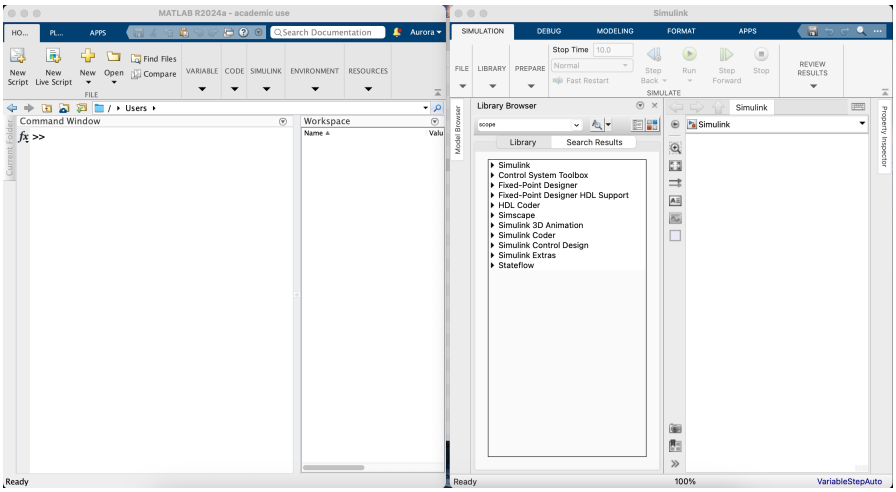


Figura 2.12: Schermata principale di avvio di MATLAB/Simulink.

L'utilizzo di questa piattaforma è stato indispensabile per comprendere meglio il funzionamento, dal punto di vista elettronico, della parte di circuito che comprende le bobine dei solenoidi.
Nel Capitolo 3 verrà illustrato il procedimento svolto nel corso del lavoro.

2.3.2 Arduino

Come anticipato nel Capitolo 1, nel corso del seguente lavoro è stato necessario l'utilizzo anche della piattaforma Arduino, vediamo ora nello specifico di che cosa si tratta.

Arduino è una piattaforma elettronica open-source basata su hardware e software di facile utilizzo. Le schede Arduino sono in grado di leggere input e

convertirli in output, ad esempio azionando un motore, accendendo un LED o pilotando una valvola.
E' possibile istruire la scheda inviando un insieme di comandi al microcontrollore integrato, utilizzando il linguaggio di programmazione Arduino e l'Arduino Software (IDE).

Per poter programmare la scheda è quindi necessaria l'installazione del software **Arduino IDE**. Una volta installato apparirà la finestra principale, come illustrato in Figura 2.13, nella quale è possibile procedere con la scrittura del codice.

Figura 2.13: Schermata principale di avvio dell'Arduino IDE.

Nel Capitolo 3 viene illustrato l'impiego della piattaforma nell'ambito del presente lavoro di tesi.

Capitolo 3

Svolgimento dell'attività sperimentale

In questo capitolo verranno analizzate le fasi da 2 a 4 elencate nella Sezione 1.4. Prima di entrare nel merito del vero e proprio progetto è stato opportuno svolgere una ricerca riguardante il driver di pilotaggio.

Dal momento che fisicamente non è richiesto di implementare il driver nel progetto, la ricerca è stata utile al fine della progettazione finale della valvola, fungendo da punto di partenza per la raccolta dei dispositivi integrativi alla valvola.

Il capitolo si articola nei seguenti argomenti principali:

1. Scelta del driver di pilotaggio per solenoidi più opportuno al progetto della valvola;

2. Misura della resistenza e dell'impedenza dei solenoidi con Arduino;

3. Simulazione del comando di pilotaggio della valvola mediante l'utilizzo di MATLAB/Simulink;

4. Implementazione fisica del comando di pilotaggio della valvola;
5. Verifica della tensione di Pull-In degli avvolgimenti.

3.1 Scelta del driver

Importante scelta da fare è quella del driver per poter pilotare i solenoidi. Come anticipato nella Sezione 2.2.1, esistono molteplici varianti di driver per poter pilotare un solenoide. Dal momento che la richiesta è quella di pilotare una valvola latching è necessario andare a svolgere una selezione attenta sulla base di alcune specifiche note. In particolare il driver di nostra scelta dovrà avere le seguenti caratteristiche:

- Tensione massima fino a 60 V ;
- Possibilità di gestire due canali/solenoidi contemporaneamente;
- Interfaccia con microcontrollori;
- Canali di misura della corrente per il monitoraggio e lettura diretta senza dover aggiungere ulteriori sensori;
- Diodo di protezione;
- Scheda PCB già preinstallata.

Date le suddette caratteristiche si è svolta una ricerca tra i principali fornitori e produttori di driver e si sono evidenziati i risultati ottenuti nella Tabella 3.1.

Una volta individuata la lista di driver tra i quali scegliere, è possibile procedere con l'analisi delle caratteristiche di ciascun driver in modo da poterne individuare il più adatto.

- **DRV101**: Dispone di scheda prestampata e di un canale per la misurazione diretta della corrente, rendendolo potenzialmente adatto. Tuttavia,

Nome	Tipo Canali	Vmax	Corrente sensing	Canali	Interfaccia MCU	PCB integrata	Diodo protezione
DRV101 (*)	Singolo	60 V	No (solo flag over/under)	1	Analogico/GPIO	Sì (modulo)	Sì
DRV8803 (*)	Quad (low side)	60 V	PWM only	4	Parallel IN	No	Sì
DRV8844 (*)	Quad (half bridge)	60 V	No	4	IN/IN	No	Sì
DRV8873-Q1 (*)	Dual (bridge/half)	38 V	Sì (IPROPI)	2	PWM + fault	No	Sì
L9305 (-)	Quad (high/low)	36 V	Sì (SPI)	4	SPI + fault pin	No	Sì
MCP236/MCP236 (**)	Dual (bridge)	60 V	Sì	2	USB/TTL/CAN/RC PWM/ISET analogico.	Sì (modulo)	Sì
MPQ6610 (*-)	Dual (half bridge)	55 V	Parziale (ISET)	2	Compatible with 2.5V, 3.3V, and 5V Logic	No	Body diode in MOSFET
NSD5604(N) (–)	Quad (low side)	50 V	Limit config.	4	Parallel IN	No	Sì

(*) → Texas Instruments
(**) → Basicmicro
(*-) → MPS
(-) → STMicroelectronics
(–) → NOVOSENSE

Figura 3.1: Confronto tra driver.

essendo dotato di un solo canale di pilotaggio, può gestire un solenoide alla volta; nel caso del presente lavoro, essendo presenti due solenoidi, sarebbe necessario utilizzare due driver separati.

- **DRV8803**: Non dispone di PCB né di funzionalità di misura della corrente.

- **DRV8844**: Integra quattro mezzi ponti ad H (1/2-H-bridge), ognuno controllabile singolarmente. Non possiede PCB né canale per la misura della corrente integrata. Il vantaggio di questo dispositivo risiede nel fatto che l'azienda produttrice prevede una scheda PCB opzionale, acquistabile separatamente, progettata per l'integrazione con questo driver. Inoltre, la misura della corrente può essere implementata semplicemente aggiungendo un resistore di sensing opzionale tra i pin SRC12 o SRC34 e VNEG.

- **DRV8873-Q1**: Non supporta la tensione richiesta dal progetto, gestendo valori troppo bassi oltre che la mancanza della scheda PCB integrata.

- **L9305**: Di origine automotive, progettato per lavorare a tensioni inferiori rispetto a quelle necessarie al progetto.

- **MCP236 / MCP266**: Rispetta tutte le caratteristiche necessarie, il problema risiede nel fatto che è stato pensato in particolare per il pilotaggio di motori brushed (DC), anche se tecnicamente potrebbe pilotarli non sarebbe la soluzione ideale o più economica per semplici solenoidi.

- **MPQ6610**: La tensione supportabile è di poco inferiore al limite imposto però presenta una modalità di misura della corrente già integrata al contrario della scheda PCB che non è presente.

- **NSD5604 / NSD5604N**: La tensione supportbile è di poco inferiore al limite imposto e il sensore di corrente ha un funzionamento limitato in quanto fornisce un'indicazione di soglia e non di valore. Sono inoltre privi di PCB.

Nella seguente tabella si riassume in modo visivo l'analisi appena svolta.

✗→ Non presente
✓→ Presente
●→ Parziale o implementabile

	DRV101	DRV8803	DRV8844	DRV8873	L9305	MCP236/MCP236	MPQ6610	NSD5604(N)
Vmax	✓	✓	✓	✗	✗	✓	●	●
Corrente sensing	●	✗	●	✓	✓	✓	●	●
Canali	✗	✓	✓	✓	✓	✓	✓	✓
Interfaccia MCU	✓	✓	✓	✓	✓	✓	✓	✓
PCB integrata	✓	✗	●	✗	✗	✓	✗	✗
Diodo di protezione	✓	✓	✓	✓	✓	✓	✓	✓
	✗	✗	✓	✗	✗	✗	✗	✗

Tabella 3.1: Confronto visivo tra driver.

Confrontando le caratteristiche richieste all'inizio del capitolo con quelle effettivamente disponibili nei driver candidati, è possibile individuare la soluzione che soddisfa in modo più completo le specifiche progettuali e che, al contempo, offre alternative per quei requisiti non pienamente soddisfatti. Questo driver è il **DRV8844** della Texas Instruments.

Dopo aver svolto l'analisi dei driver, è possibile quindi proseguire con il lavoro andando a svolgere in primo luogo la caratterizzazione degli avvolgimenti e successivamente sviluppare il prototipo del comando di commutazione.

3.2 Caratterizzazione degli avvolgimenti di una valvola a solenoide

Per svolgere uno studio esaustivo di una valvola a solenoide, è fondamentale andare a svolgere un'analisi mirata delle caratteristiche elettriche degli avvolgimenti interni alla valvola.

Viene prodotto un circuito elettrico che simuli la bobina del solenoide in modo da poter svolgere le opportune verifiche.

Prima di procedere è importante specificare che, in termini di rappresentazione elettrica, un solenoide è possibile rappresentarlo utilizzando una combinazione di componenti elettrici che riproducano il suo funzionamento.

3.2.1 Rappresentazione dei solenoidi

Dal punto di vista elettrico, un solenoide può essere rappresentato come un circuito equivalente costituito da un resistore e un induttore in serie.
Nel seguito, ogni riferimento a solenoidi o bobine farà sempre riferimento a questo modello equivalente.

In ottica della rappresentazione circuitale della bobina e, successivamente, del comando della valvola, vengono presi in considerazione dei valori di riferimento di resistenza e induttanza e fatti variare in un intervallo a passi di due. Questo per osservare combinazioni diverse e simulare scenari differenti. Il valore di riferimento del resistore che si prende in considerazione è di $150\,\Omega$, il range di variazione del valore è compreso tra $100\,\Omega$ e $220\,\Omega$.

Di seguito la tabella dei valori presi in questo range con quel valore di riferimento.

Nome	**Resistenza target**	**Combinazione proposta**
R1	$100\,\Omega$	singola: $100\,\Omega$
R2	$135\,\Omega$	serie: $100\,\Omega + 22\,\Omega + 10\,\Omega = 132\,\Omega$
R3	$150\,\Omega$	singola: $150\,\Omega$
R4	$170\,\Omega$	serie: $100\,\Omega + 68\,\Omega = 168\,\Omega$
R5	$220\,\Omega$	singola: $220\,\Omega$

Tabella 3.2: Combinazioni di resistenze per raggiungere il valore target.

Lo stesso procedimento viene fatto per gli induttori. Il range di riferimento è compreso tra i valori di $0{,}7\,\mu H$ e $50\,\mu H$ con un valore di riferimento pari al valore intermedio in quel range (circa $25\,\mu H$).

Nome	Induttanza target	Combinazione proposta
I1	$0{,}7\,\mu H$	parallelo: $1\,\mu H$ e $2{,}2\,\mu H \rightarrow 0{,}69\,\mu H$
I2	$13\,\mu H$	serie: $10\,\mu H + 3{,}3\,\mu H = 13{,}3\,\mu H$
I3	$25\,\mu H$	serie: $10\,\mu H + 15\,\mu H = 25\,\mu H$
I4	$38\,\mu H$	serie: $33\,\mu H + 4{,}7\,\mu H = 37{,}7\,\mu H$
I5	$50\,\mu H$	serie: $47\,\mu H + 3{,}3\,\mu H = 50{,}3\,\mu H$

Tabella 3.3: Combinazioni di induttori per ottenere i valori target.

La terza colonna delle tabelle riportate indica i valori reali di resistenza da utilizzare per approssimare al meglio i valori teorici desiderati, in funzione dei componenti effettivamente disponibili.

Per quanto riguarda la caratterizzazione degli avvolgimenti, la simulazione delle combinazioni dei valori presenti nelle Tabelle 3.2 e 3.3 risulta utile al fine di sviluppare diverse prove di misura. Nella Tabella 3.4 vengono calcolate le principali caratteristiche di un circuito RL per ognuna delle configurazioni: costante di tempo τ e corrente massima ammissibile I_{max}.

3.2.2 Misura di resistenza e impedenza con Arduino

In questa sezione si andrà a sviluppare il circuito elettrico e la routine Arduino necessari ad effettuare la misura della resistenza R, in DC, e dell'impedenza Z, in AC.

Per quanto riguarda in particolare la misura dell'impedenza, è opportuno specificare che essa viene effettuata modulando la corrente in frequenza con

R [Ω]	**L** [μ**H**]	$\tau = \frac{L}{R}$ [μs]	$I_{max} = \frac{V}{R}$ [mA]
100	0.7	0.007	**50.0**
100	13	0.13	**50.0**
100	25	0.25	**50.0**
100	38	0.38	**50.0**
100	50	0.50	**50.0**
135	0.7	0.005	37.0
135	13	0.096	37.0
135	25	0.185	37.0
135	38	0.281	37.0
135	50	0.370	37.0
150	0.7	0.005	33.3
150	13	0.087	33.3
150	25	0.167	33.3
150	38	0.253	33.3
150	50	0.333	33.3
170	0.7	0.004	29.4
170	13	0.076	29.4
170	25	0.147	29.4
170	38	0.224	29.4
170	50	0.294	29.4
220	0.7	0.003	22.7
220	13	0.059	22.7
220	25	0.114	22.7
220	38	0.173	22.7
220	50	0.227	22.7

Tabella 3.4: Combinazioni R–L con costante di tempo τ e corrente massima I_{max}. In grassetto i valori rischiosi per Arduino (> 40 mA).

il canale digitale PWM[1] della scheda Arduino.

[1]Il PWM è un segnale digitale (0 1) che viene accesso e spento molto velocemente. Serve a simulare un'uscita analogica o a generare segnali per pilotare circuiti di potenza.

Implementazione del circuito

Si è precedentemente visto che la bobina è rappresentabile da un resistore e un'induttore posti in serie tra loro. E' però necessario andare ad implementare un resistore aggiuntivo, detto resistenza *shunt*, per ottenere una misura di corrente reale. Questo perché si andranno a misurare le cadute di tensione ai capi della bobina e della resistenza e, per mezzo della legge di Ohm, si calcolerà il valore della corrente effettiva che scorre nel circuito.
Vengono inizialmente utilizzati i valori di riferimento citati in precedenza ($150\,\Omega$, $25\,\mu H$) per effettuare una prima misura di verifica, successivamente è possibile andare a variare a piacere i valori tra quelli elencati nella Tabella 3.4.

La routine di Arduino viene utilizzata per istruire la scheda dotata di microcontrollore affinché questa possa pilotare il circuito. Il codice viene riportato nella Sezione C.1 in appendice.

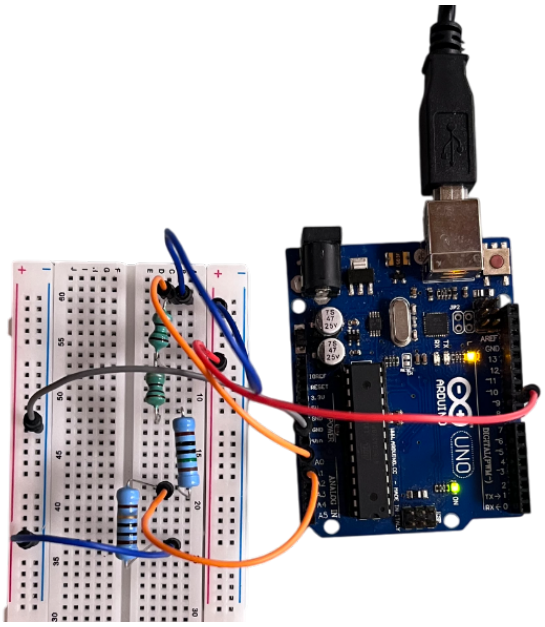


Figura 3.2: Realizzazione del circuito rappresentativo della singola bobina per il calcolo di R e Z.

La scheda Arduino vedrà quindi i collegamenti di alimentazione, di terra e degli ingressi analogici per la misura della tensione.
Nella Figura 3.2 si possono distinguere i diversi collegamenti tramite il seguente codice colore: il cavo rosso fornisce l'alimentazione alla board, il cavo grigio collega il pin GND della scheda alla board, ossia il collegamento al polo negativo, mentre i cavi arancioni permettono di prelevare la misura della tensione e della corrente ai capi della bobina e della resistenza shunt.

Con il valore di tensione letto è possibile svolgere i calcoli per valutare R, Z.

Calcolo della resistenza R

Per la misura della resistenza R, il codice genera una corrente continua (DC) tramite il PWM dell'Arduino. Vengono letti i valori della tensione ai capi della bobina e della resistenza shunt grazie agli ingressi analogici A0 e A1. In realtà, l'ingresso analogico A0 va a misurare la caduta di tensione della bobina e della resistenza shunt insieme, quindi è necessario andare a svolgere una sottrazione tra i due valori ottenuti dalla lettura di A0 e A1 per ottenere direttamente quello della sola bobina.
La tensione, dal momento che viene prelevata dal comando **analogRead()**, viene convertita in Volt tramite la relazione:

$$V = V_{ADC} \cdot \frac{V_{\text{ref}}}{1023}$$

Dove la V_{ref} non è altro che la tensione di alimentazione della scheda Arduino $(5V)$ e 1023 è il valore corrispondente[2].
Una volta nota la caduta di tensione ai capi della resistenza shunt, è possibile calcolare la corrente che vi scorre attraverso tramite la legge di Ohm.

[2]Il convertitore Analogico - Digitale ADC di Arduino è a 10 Bit corrispondenti a 1024 valori, da 0 a 1023, dove il primo valore corrisponde a 0 V mentre l'ultimo valore ai 5 V [4].

$$I = \frac{V}{R} \tag{3.1}$$

Dove le grandezze V, R sono rispettivamente V_{shunt}, R_{shunt}.
Nota quindi il valore reale della corrente che scorre nel circuito, si calcola la resistenza della bobina invertendo la Formula 3.1 rispetto a R.
Per ridurre il rumore e migliorare la stabilità della misura, la tensione viene campionata più volte (20 campioni) e viene calcolata la media aritmetica.

Calcolo dell'impedenza Z

Per la misura dell'impedenza Z in corrente alternata, il codice genera una corrente sinusoidale approssimata tramite il PWM della scheda Arduino. La tensione in questo caso viene campionata più volte durante la simulazione a diverse frequenze. La tensione efficace (RMS), sia della bobina che della shunt, viene calcolata come segue:

$$V_{\mathrm{RMS}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} V_i^2}$$

Dove V_i corrisponde ai campioni di tensione prelevata a diverse frequenze e N è il numero di campionamenti svolti.
La corrente RMS viene calcolata nuovamente con la legge di Ohm riferita ai valori della shunt.

$$I_{\mathrm{RMS}} = \frac{V_{\mathrm{RMS}}}{R_{shunt}}$$

Dunque, l'impedenza AC viene calcolata, riferendosi alla bobine, come:

$$Z = \frac{V_{\mathrm{RMS}}}{I_{\mathrm{RMS}}}$$

Una volta svolta la prima analisi è possibile testare le varie configurazioni di resistenza e induttanza e verificare la coerenza con i risultati ottenuti.

I risultati vengono riportati nella Sezione 4.1.

Si prosegue la trattazione entrando nel merito del pilotaggio della valvola latching. In primo luogo si svolge una simulazione con MATLAB/Simulink e successivamente si implementa il circuito vero e proprio per il comando di commutazione.

3.3 Simulazione del comando di pilotaggio della valvola mediante l'uso di MATLAB/Simulink

Dopo aver individuato le caratteristiche dei componenti elettronici rilevanti per la rappresentazione dei solenoidi, è stato utile realizzare una rappresentazione schematica della valvola mediante l'ambiente Simulink di MATLAB, con l'obiettivo di simulare il suo funzionamento e analizzarne la risposta.

Grazie all'utilizzo di questo software è stato possibile riprodurre in modo schematico la valvola andando a rappresentare i suoi componenti grazie ai blocchi logici di Simulink.

L'intero funzionamento della valvola si può semplicemente schematizzare soffermandosi sui solenoidi in quando questi sono fondamentali per la risposta elettrica.

3.3.1 Simulazione della risposta elettrica

Per prima cosa si è svolta la schematizzazione elettrica del solenoide utilizzando l'ambiente *Simscape* di Simulink, che consente la rappresentazione dei vari componenti elettrici tramite blocchi selezionabili dal *Library Browser* e trascinabili nell'area di lavoro.

Come illustrato nella Sezione 3.2.1, un solenoide può essere rappresentato da un resistore e un'induttanza in serie, che costituiscono il primo ramo dello schema. Ricordiamo che i valori di riferimento per il resistore e l'induttore sono rispettivamente $150\,\Omega$ e $25\,\mu H$.

È importante considerare la presenza di un diodo di ricircolo in parallelo al ramo resistore-induttore, questo perché, in ottica della realizzazione di un circuito fisico, è necessario proteggerlo in caso di scariche improvvise.

Poiché il passaggio di corrente nel solenoide dovrebbe avvenire in risposta a un comando proveniente dal driver di pilotaggio, è necessario introdurre un generatore di impulsi e un interruttore che ne riproducano il comportamento. In particolare, il segnale in ingresso al solenoide, equivalente a quello fornito dal driver, viene rappresentato tramite il blocco *Pulse Generator*, che genera un impulso ad onda quadra in grado di simulare il comando di apertura/chiusura della valvola. Il segnale passa poi attraverso un interruttore, implementato con il blocco *Switch*, che chiude il circuito quando il comando è attivo e lo apre quando il comando viene rimosso. Questi due componenti sono collegati al ramo costituito dal resistore e dall'induttore in serie.

Dal momento che il principale interesse è quello di monitorare l'andamento della corrente nel solenoide durante la simulazione, è possibile inserire in serie al resistore e all'induttore un sensore di corrente (blocco *Current Sensor*). In questo modo si ottiene una simulazione completa del funzionamento di

un solenoide.

Poiché il sistema prevede due solenoidi, è sufficiente duplicare lo schema, prestando attenzione al fatto che il segnale di chiusura debba essere ritardato rispetto a quello di apertura, in modo da garantire che la valvola abbia il tempo necessario per aprirsi completamente prima di chiudersi. Per fare ciò, è dunque importante definire la *Phase Delay* per il segnale di ingresso fornito dal blocco *Pulse Generator* corrispondente al solenoide che pilota la chiusura della valvola.

I due schemi dei solenoidi vengono quindi collegati in parallelo, con l'aggiunta di un riferimento a terra tramite il blocco *Electrical Reference*. Per garantire il corretto funzionamento del modello, sono stati aggiunti un generatore di tensione (blocco *DC Voltage Source*) con tensione di alimentazione pari a 5 V, e un blocco *Solver Configuration*. Quest'ultimo definisce l'algoritmo numerico e definisce i parametri di simulazione per far funzionare il modello.

La Figura 3.3.1 mostra lo schema completo.

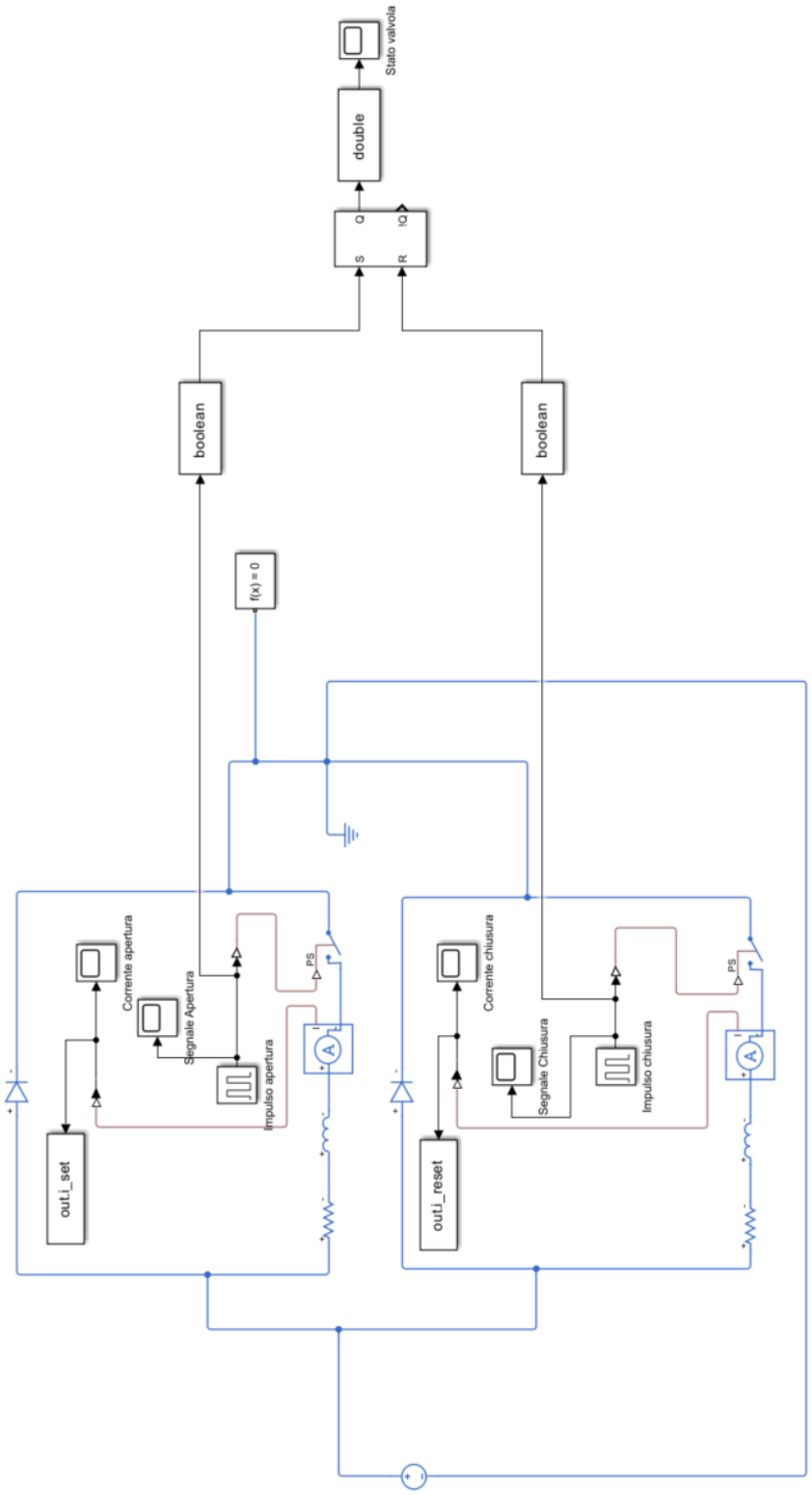


Figura 3.3: Schema rappresentativo della valvola prodotto con Simulink.

Per completare il modello e consentire un'analisi visiva dei risultati della simulazione, sono stati introdotti ulteriori blocchi:

- *Flip-flop*, consente di osservare lo stato finale della valvola;
- *Scope*, impiegati per visualizzare l'andamento delle grandezze elettriche nel tempo. In particolare, essi sono stati posizionati sui segnali generati dai *Pulse Generator*, sugli output dei *Current Sensor* e sull'uscita del *Flip-flop*;
- *To Workspace*, permettono di estrarre il valore in output fornito dal *Current Sensor*.

Una volta definito lo schema, è possibile eseguire la simulazione del modello. I risultati ottenuti permettono di analizzare l'andamento dei segnali di ingresso, della corrente e dello stato della valvola. I grafici generati dai blocchi *Scope* sono riportati nella Sezione 4.2.

Dopo aver eseguito la simulazione con i valori di riferimento dei componenti elettronici, si possono effettuare ulteriori simulazioni variando i valori di resistenza e induttanza per osservare le corrispondenti variazioni nell'andamento della corrente.

A questo scopo si utilizza uno script MATLAB, che consente di eseguire automaticamente la simulazione in Simulink per tutte le combinazioni di resistenze e induttanze riportate nella Tabella 3.4.

I valori dei componenti vengono raccolti in due vettori: R_{set}, L_{set} per il solenoide che gestisce l'apertura della valvola, e R_{reset}, L_{reset} per il solenoide che gestisce la chiusura. Tali variabili sostituiscono i valori numerici nel modello, permettendo allo script di iterare automaticamente su ciascun elemento dei vettori.

Lo script include una fase di preparazione della simulazione, in cui viene richiamato il file Simulink contenente lo schema elettrico della valvola.

Al termine della simulazione, i dati di corrente vengono estratti tramite i blocchi *To Workspace* e salvati in una struttura MATLAB insieme ai valori temporali corrispondenti. Infine, i risultati vengono rappresentati graficamente, permettendo un confronto visivo tra i diversi andamenti della corrente (si veda Sezione 4.2).

Il codice MATLAB utilizzato per questa simulazione è riportato nella Sezione B.1 in appendice.

In questo modo è stato possibile schematizzare il funzionamento elettrico della valvola, fornendo un supporto utile per la comprensione e la realizzazione del circuito fisico.

3.4 Realizzazione del comando di pilotaggio della valvola

Questa sezione descrive le modalità e le fasi di sviluppo del sistema di pilotaggio della valvola.

In una prima fase è stato realizzato il codice per la scheda Arduino, così da consentire al microcontrollore di gestire correttamente il circuito fisico previsto. Successivamente viene presentata l'idea di rappresentazione del comando mediante la costruzione di un circuito elettrico dedicato.

3.4.1 Istruzione del microcontrollore

Dopo aver schematizzato il circuito elettrico grazie a Simulink, è dunque possibile entrare più nel merito del lavoro passando all'utilizzo della logica Arduino (per approfondimenti vedere Sezione 2.3).

Per istruire il microcontrollore è necessario sviluppare un codice nell'Arduino IDE il quale andrà a simulare il comando di pilotaggio della valvola.
Il codice viene riportato per intero nella Sezione C.3 in appendice.

Nelle prime righe vengono determinate le costanti e le variabili utilizzate nel corso del codice. Le costanti rappresentano i pin digitali di Arduino utilizzati, mentre le variabili corrispondono a due parametri imposti a priori: la durata dell'impulso e l'intervallo tra il funzionamento di un solenoide e l'altro.

Successivamente sono state definite delle funzioni che andranno ad essere richiamate nella parte principale del codice.

- *Setup* → corrisponde alla funzione che viene eseguita una sola volta appena viene azionato il microcontrollore;

- *PulseCoil* → è la funzione che richiama il funzionamento della bobina andando a raggruppare le istruzioni di accensione, mantenimento e spegnimento del pin relativo alla bobina che si sta richiamando;

- *OpenValve* e *closeValve* → sono le funzioni che danno il comando di accensione delle bobine, una sarà relativa all'apertura e l'altra alla chiusura;

- *CheckSensors* → è la funzione che gestisce la ricevuta del segnale dal sensore ad effetto Hall.

Nella parte finale del codice è presente la sequenza di istruzioni che costituisce il vero e proprio comando di pilotaggio, andando a richiamare le funzioni precedentemente definite, organizzandole in modo da ripetersi ciclicamente. Viene inoltre implementata la possibilità di inserire un comando tramite tastiera: l'utente, attraverso l'immissione di un carattere, può impartire al microcontrollore l'istruzione di commutazione dello stato della valvola. In particolare, digitando la lettera "O" (open), viene attivato il solenoide che gestisce l'apertura, mentre con la lettera "C" (close) si attiva quello di chiusura.

Una volta completata la stesura del codice nell'ambiente di sviluppo (IDE), si procede con la fase di verifica per individuare eventuali errori di compilazione. Successivamente, il programma viene caricato sul microcontrollore tramite collegamento USB tra il PC e la scheda Arduino UNO. In questo modo il microcontrollore viene programmato per pilotare alternativamente i due solenoidi, consentendo così di realizzare le azioni di apertura e chiusura della valvola.

Una volta configurato il microcontrollore, è quindi possibile passare alla realizzazione fisica del circuito elettrico. Il circuito realizzato ricalca lo schema precedentemente sviluppato con Simulink (Sezione 3.3).

3.4.2 Implementazione del circuito

Il flusso di elettroni (carica negativa) si muove partendo dal polo negativo a quello positivo quindi, nel nostro circuito, dal pin di terra (GND) al pin dell'alimentazione 5 V. Per convenzione il senso della corrente va dal polo positivo a quello negativo [3]; si farà riferimento a questa convenzione per orientarsi nella spiegazione del circuito.

Si procede quindi con l'allestimento della breadboard, predisponendo un LED con due funzioni: la prima destinata a segnalare l'avvenuto comando da parte del microcontrollore, la seconda ad indicare il mantenimento dello stato della valvola. In questa fase del progetto, il LED sostituisce il sensore ad effetto Hall inizialmente previsto nel progetto aziendale. Per evitare danneggiamenti, il LED viene protetto mediante una resistenza posta in serie.

Successivamente è necessario selezionare i valori di resistenze e induttanze tra quelli riportati nelle Tabelle 3.2 e 3.3. Una volta definiti tali valori, è possibile calcolare la corrente massima sviluppata, data dalla Formula 3.1. Dove $V = 5\,\mathrm{V}$ è la tensione erogata dalla scheda Arduino UNO, mentre R è il valore della resistenza in Ohm [Ω]. Nella Tabella 3.4 è possibile leggere i valori di corrente ottenuti dalle varie combinazioni.
I pin della scheda Arduino UNO presentano un valore di soglia della corrente pari a $40\,mA$ [5]. Perciò è opportuno fare attenzione alle combinazioni di valori che vengono scelte in modo da rimanere sempre in sicurezza. Si scartano dunque le combinazioni che eccedono i limiti di corrente imposti da Arduino (nella Tabella 3.4, i valori i valori che eccedono il limite vengono evidenziati in grassetto).

Per iniziare, si scelgono come valori di resistenza e di induttanza sempre quelli di riferimento ($150\,\Omega$ e $25\,\mu H$). Per semplicità di realizzazione è stato prima implementato un singolo solenoide (si veda Figura 3.4) in modo da comprendere bene i vari collegamenti e come realizzarli.
Si vuole realizzare un comando nel quale, in seguito ad un'istruzione proveniente dall'esterno, venga commutato lo stato del sistema la quale avvenuta viene identificata dall'accensione del LED.
Per questo il circuito vedrà la presenza di due rami in parallelo:

- Ramo 1 → solenoide composto da resistenza e induttanza
- Ramo 2 → LED con resistenza

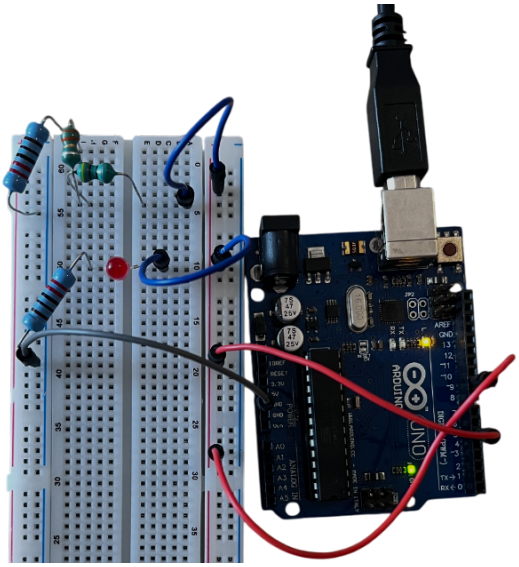


Figura 3.4: Circuito bobina singola con LED spento.

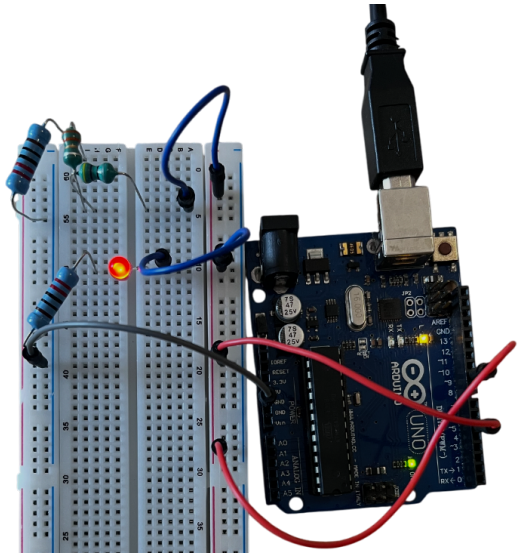


Figura 3.5: Circuito bobina singola con LED acceso.

Per l'alimentazione della board è necessario utilizzare due pin distinti della scheda Arduino: il primo è dedicato al comando di commutazione (attivazione del solenoide), mentre il secondo mantiene acceso il LED che indica lo stato della valvola.

Come già visto nella Sezione 3.2.2, i vari collegamenti sono individuati dal seguente codice colore: il cavo rosso fornisce l'alimentazione alla board, il cavo grigio rappresenta la messa a terra (GND), mentre i cavi blu hanno la funzione di ponti di collegamento. Questi ultimi consentendo il passaggio della corrente tra le sezioni della board non direttamente collegate tra loro.

Nella sequenza rappresentata dalle Figure 3.4 e 3.5 è possibile visualizzare il circuito in funzione: dopo aver fornito il comando a tastiera, il LED si è acceso. Il codice Arduino relativo all'eccitazione della singola bobina è possibile consultarlo nella Sezione C.2 dell'appendice.

Completata la prima implementazione, pensata unicamente per verificare la validità concettuale del circuito e il suo corretto funzionamento, è possibile introdurre un secondo solenoide in parallelo al primo, così da simulare in modo più realistico il comportamento complessivo del sistema.

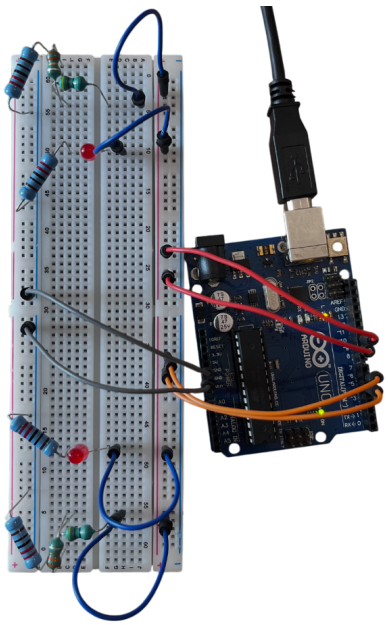


Figura 3.6: Circuito rappresentativo del pilotaggio della valvola.

Come si vede nella Figura 3.6 i due rami principali sono esattamente specchiati. Si distinguono i cavi dell'alimentazione della board, individuati dai colori rosso per la prima bobina e arancione per la seconda, i cavi della messa a terra grigi e i cavi di collegamento di colore blu.

Per rendere più immediata la visualizzazione dello stato della valvola è possibile utilizzare un codice colore anche per i LED: il rosso viene associato alla condizione di shut-off (valvola disattivata), mentre il verde a quella di turn-on (valvola attivata).

La sequenza delle Figure 3.7 e 3.8 mostra il circuito in funzione: viene fornito il comando "O" e si illumina il LED verde (la valvola si è aperta), viene fornito il comando "C" e si illumina il LED rosso (la valvola si è chiusa) .

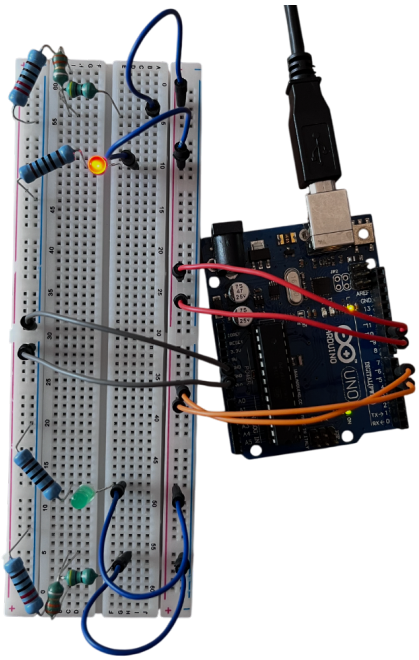


Figura 3.7: Circuito rappresentativo della valvola chiusa: LED rosso acceso.

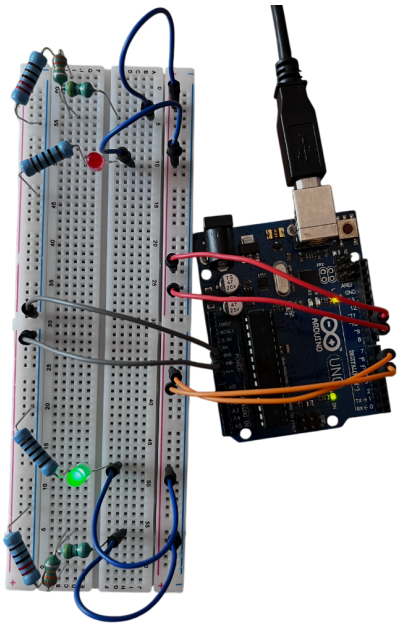


Figura 3.8: Circuito rappresentativo della valvola aperta: LED verde acceso.

Dopo aver realizzato il circuito rappresentativo del comando di pilotaggio della valvola, è possibile svolgere la verifica della tensione di Pull-In, come anticipato nella Sezione 1.2.

3.4.3 Verifica della tensione di Pull-In degli avvolgimenti

Il test ha lo scopo di verificare che, ad ogni incremento della tensione applicata, gli avvolgimenti raggiungano il cosiddetto Pull-In (si ricorda che il Pull-In è la massima tensione richiesta per l'apertura della valvola). Poiché la scheda Arduino non consente di variare direttamente la tensione ad incrementi fissi oltre i 5 V, è necessario introdurre un dispositivo esterno in grado di filtrare il segnale e fornire al circuito la tensione desiderata.
Per lo svolgimento di questo lavoro la scelta è ricaduta sul regolatore di tensione citato nella Sezione 2.2.3. Questo dispositivo è in grado di far variare la tensione entro un range desiderato, superiore alla tensione massima erogabile dalla scheda.

L'obiettivo dell'attività è ottenere un incremento graduale, a step discreti, della tensione di alimentazione della bobina, in modo da esplorare l'intero intervallo di tensioni gestibile dal regolatore e verificare l'effettiva attivazione del comando di accensione. Da progetto, la tensione di Pull-In della valvola risulta compresa tra un valore minimo di 4 V e un valore massimo di 20 V; al di fuori di tale intervallo la valvola non è in grado di aprirsi.
Per realizzare questa funzionalità è stato sviluppato un codice nell'ambiente Arduino IDE, che consente la comunicazione tra la scheda Arduino e il regolatore di tensione. In questo modo, la scheda gestisce l'alimentazione del circuito, mentre il regolatore varia la tensione fornita alla bobina.
Il regolatore di tensione comunica con Arduino tramite l'ingresso V_{IN} e restituisce l'alimentazione al circuito tramite l'uscita V_{OUT} (si veda Figura 3.9).

Il codice sviluppato per istruire il microcontrollore ha l'obiettivo di verificare la tensione di Pull-In della bobina di apertura della HPLV. Il circuito utilizzato per svolgere questa verifica è quello rappresentativo della singola bobina. L'implementazione risulterà diversa da quella fatta in precedenza (Figura 3.4) per via dei dispositivi che si andranno ad aggiungere. Il circuito prevede sempre la presenza di due rami principali: il primo costituito dalla bobina, rappresentata da un induttore in serie a una resistenza, e il secondo formato da un LED con la relativa resistenza di protezione.

Nel *datasheet* del regolatore di tensione [6] è riportato uno schema di riferimento per la regolazione della tensione fino a 26 V, mostrato in Figura 3.9. In tale configurazione sono presenti due resistenze, R_1 e R_2, di cui la prima è variabile mentre la seconda è fissa.

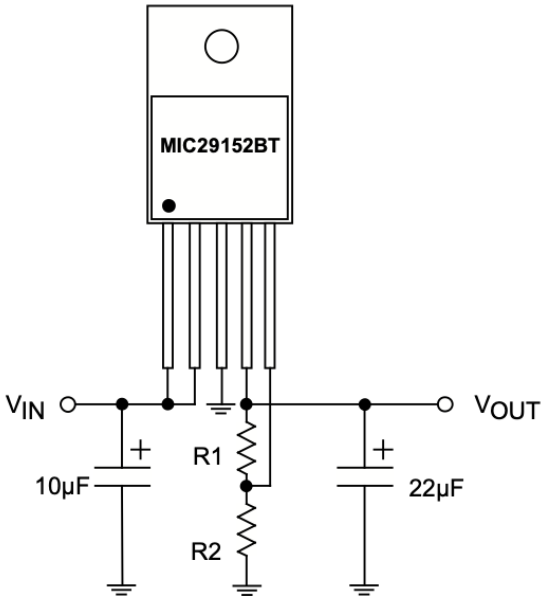


Figura 3.9: Setup per la regolazione del regolatore di tensione MIC29502WT. Fonte: Micrel, Inc., datasheet.

Per la resistenza variabile si è scelto di impiegare un potenziometro digitale, il modello DS3502, che consente di variare il valore resistivo in un intervallo

compreso tra $0\,\Omega$ e $10\,\text{k}\Omega$. Il potenziometro è controllato dalla scheda Arduino tramite interfaccia I^2C[3], permettendo di impostare via software la tensione di alimentazione della bobina.

La resistenza fissa, invece, è stata determinata in funzione dei valori di R_1 e delle caratteristiche del regolatore stesso. La relazione utilizzata dal MIC29502WT per il calcolo della tensione di uscita V_{OUT} è la seguente:

$$V_{\text{OUT}} = 1.240 \left(1 + \frac{R_1}{R_2}\right) \tag{3.2}$$

Si desidera variare la tensione di uscita all'interno dell'intervallo $0\,\text{V} - 26\,\text{V}$. Considerando i casi estremi di tensione e resistenza, si risolve per R_2 l'Equazione 3.2, ottenendo un valore approssimato pari a $R_2 \approx 500\,\Omega$. Nel circuito reale è stato impiegato un resistore da $470\,\Omega$, in quanto rappresenta il valore commerciale più vicino a quello teorico calcolato.

In sintesi, il codice sviluppato consente la variazione progressiva della resistenza del potenziometro digitale all'interno del range operativo del dispositivo. Il valore di resistenza impostato viene letto dal regolatore di tensione, il quale, applicando l'Equazione 3.2, determina la tensione di alimentazione della bobina.

La tensione così ottenuta viene confrontata con l'intervallo di riferimento del Pull-In ($4\,\text{V} - 20\,\text{V}$) al fine di valutare l'attivazione della valvola. L'accensione o lo spegnimento del LED rappresentano rispettivamente gli stati di apertura e chiusura della valvola (ON/OFF).

Infine, il codice prevede la stampa sul Serial Monitor dei valori di resistenza, tensione e stato del LED a ciascun step di variazione, permettendo di monitorare in tempo reale il comportamento del sistema.

[3]L'interfaccia I^2C è un protocollo di comunicazione seriale che consente lo scambio di dati tra più dispositivi elettronici, come sensori e microcontrollori, utilizzando solo due linee di collegamento.

Nella sequenza mostrata nelle Figure 3.10 e 3.11 è possibile osservare il funzionamento del circuito. In particolare, quando il LED risulta spento significa che la valvola non ha raggiunto la tensione minima necessaria per l'alimentazione corretta del solenoide di apertura. Al contrario, l'accensione del LED indica che la tensione di alimentazione rientra nel range previsto dalla specifica e che la valvola è in grado di aprirsi correttamente.

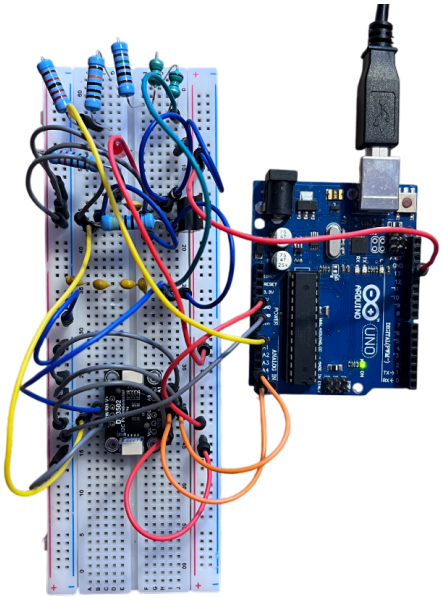


Figura 3.10: Circuito rappresentativo del solenoide di apertura della valvola con aggiunta dei dispositivi per effettuare la verifica della tensione di Pull-In.

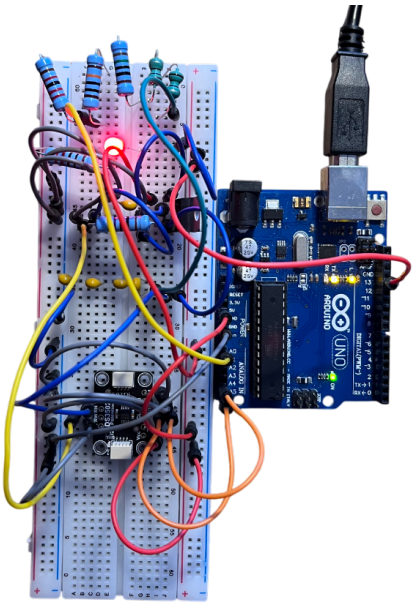


Figura 3.11: Circuito in funzione rappresentativo del solenoide di apertura della valvola con aggiunta dei dispositivi per effettuare la verifica della tensione di Pull-In.

I risultati vengono riportati nella Sezione 4.3. L'analisi viene svolta ripetendo il test più volte, in questo modo è possibile ottenere una valutazione della ripetibilità del fenomeno.

I codici sono riportati per intero nelle Sezioni B.2 e C.4.

Capitolo 4

Analisi dei risultati

In questo capitolo viene svolta l'analisi dei risultati ottenuti nel corso delle simulazioni svolte nel Capitolo 3.
Per chiarezza di lettura, le sezioni vengono suddivise alla maniera del capitolo precedente.

4.1 Misura di resistenza e impedenza con Arduino

In questa sezione vengono osservati i risultati della Sezione 3.2.2.
Nel Capitolo 3 è stato implementato un circuito rappresentativo della bobina di solenoide. L'obiettivo è stato quello di svolgere una caratterizzazione degli avvolgimenti e quindi di misurarne i valori di resistenza ed impedenza.
La misura è stata effettuata attraverso diversi tentativi, al fine di ottenere un valore il più preciso possibile.

Nella Figura 4.1 vengono riportati i risultati ottenuti durante il primo tentativo di misura. Questo è stato svolto andando ad implementare il circuito rappresentativo della bobina (Figura 3.2) e prelevando i segnali tramite gli ingressi A0 e A1 della scheda Arduino.

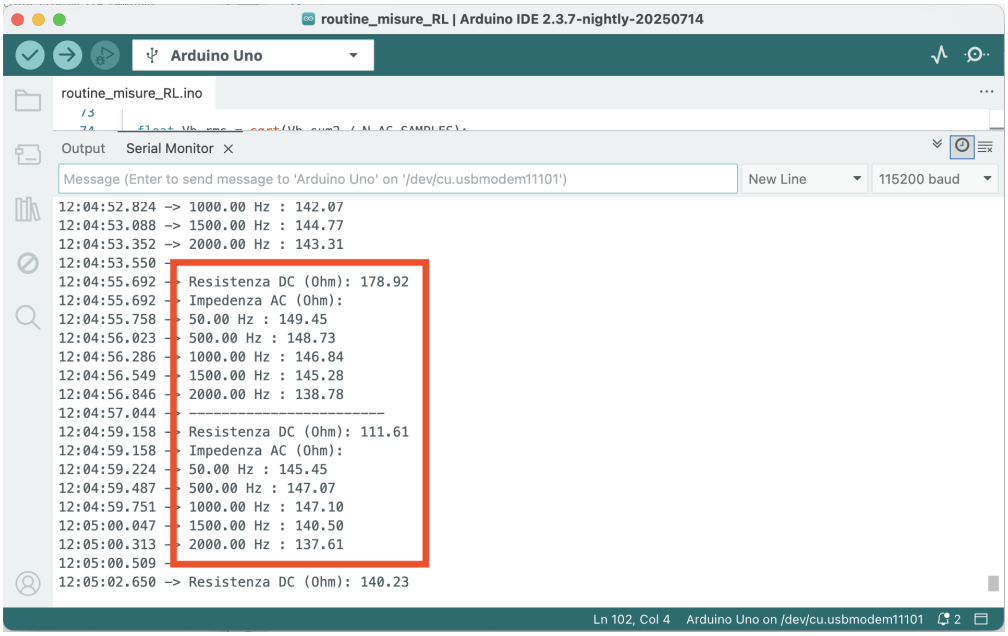


Figura 4.1: Primo tentativo di misura dei valori di resistenza ed impedenza.

Si può osservare che i valori riportati (riquadrati) non corrispondono esattamente al valore della resistenza reale, soprattutto i valori misurati in DC: invece di misurare circa 150 Ω, i valori rilevati oscillano molto. Mentre per i valori in AC la misura sembrerebbe essere più stabile.

Questa forte oscillazione della misura è dovuta al tipo di rilevamento effettuato da Arduino. La rilevazione non è diretta e risulta influenzata dalla modalità con cui la scheda genera il segnale. Arduino, infatti, non dispone di un convertitore digitale-analogico (DAC), ma utilizza un segnale PWM per simulare una tensione continua o variabile. Tale segnale è un'onda quadra che alterna rapidamente i livelli logici tra 0 V e il valore massimo.

Se il PWM viene collegato direttamente al circuito RL, l'ADC della scheda legge valori che cambiano rapidamente e risultano instabili. Di conseguenza, le misure di resistenza in DC e di impedenza in AC non sono precise, quindi la tensione rilevata ai capi del circuito non riflette il valore reale.

Per ovviare a questo problema si inserisce un filtro RC (composto da resistenza e condensatore) tra l'uscita PWM e il circuito RL (si veda Figura 4.2). Il condensatore, collegato a massa, e la resistenza in serie smussano le rapide variazioni del segnale, fornendo al circuito RL una tensione più stabile. In questo modo l'ADC legge valori più corretti, rendendo le misure di resistenza e impedenza più affidabili.

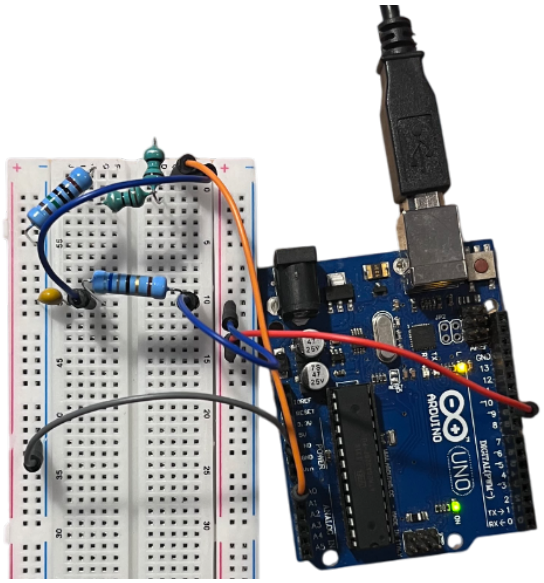


Figura 4.2: Circuito rappresentativo della bobina con rilevazione della misura di R e Z.

In Figura 4.3 si può notare come i valori in DC si siano stabilizzati e si avvicinino maggiormente a quelli effettivamente presenti.

Per avere una misura ancora più precisa, è possibile andare ad utilizzare un multimetro[1] per misurare direttamente la resistenza.

[1]Il multimetro è uno strumento in grado di rilevare una misura diretta delle grandezze elettriche reali (resistenza, corrente, tensione, ...)

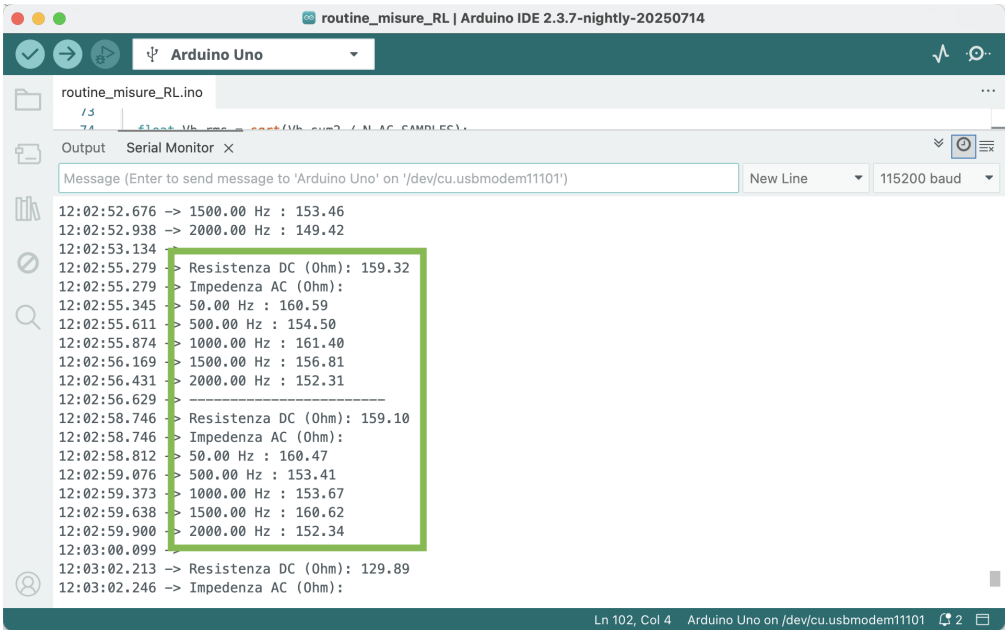


Figura 4.3: Secondo tentativo di misura dei valori di resistenza ed impedenza.

Come si vede nella Figura 4.4 la resistenza misurata ai capi della bobina è pari a 150.5 Ω.

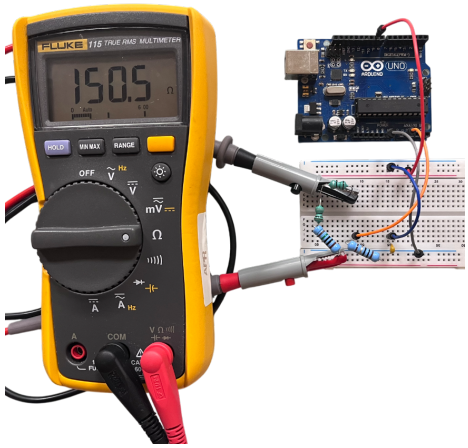


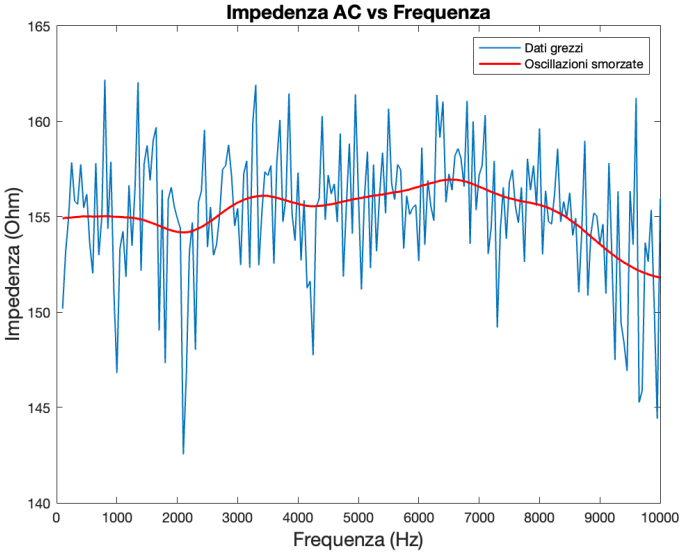
Figura 4.4: Misura della resistenza con il multimetro.

Per quanto riguarda la misura diretta dell'impedenza occorre utilizzare un multimetro particolare che abbia la funzione di misura diretta dell'impedenza.

Essendo un dispositivo non di uso comune, purtroppo non è stato possibile svolgere la verifica diretta della misura dell'impedenza della bobina.

Per questioni relative al progetto della HPLV, le frequenze di interesse sono nel range 100 Hz − 10 kHz, è dunque interessante svolgere la misura adattando il codice a questi valori estrapolandone l'andamento dell'impedenza al variare della frequenza.

In Figura 4.5 è possibile notare come le oscillazioni della curva siano importanti. La curva rossa permette di osservare l'andamento andando a smussare le oscillazioni ed è facilmente notabile come il valore dell'impedenza sia quasi costante e addirittura all'aumentare della frequenza tenda a diminuire.



P

Figura 4.5: Andamento dell'impedenza con la frequenza.

L'andamento teorico atteso mostrerebbe una lieve crescita dell'impedenza al crescere della frequenza. La discrepanza con i risultati ottenuti è dovuto alla generazione del segnale della scheda Arduino. Le oscillazioni sono dovute al rumore elettronico causato dalla resistenza capacitativa del circuito.

4.2 Simulazione del comando di pilotaggio della valvola mediante l'uso di MATLAB/Simulink

In questa sezione vengono osservati i risultati della simulazione svolta nella Sezione 3.3.1. Vengono riportati i grafici prodotti dei blocchi *Scope*.

In primo luogo si è svolta la simulazione del comando di pilotaggio andando a simulare, con i blocchi *Pulse Generator*, l'impulso di apertura e chiusura. Dal momento che l'impulso fornito dal *Pulse Generator* come input al sistema è del tipo onda quadra, questo andamento sarà caratterizzante per tutta l'analisi.
Si è scelto di fornire un impulso con le seguenti caratteristiche:

Parametro	**Valore**
Ampiezza	1
Periodo	2 s
Durata impulso	20 % del periodo
Ritardo accensione	1 s
Ritardo chiusura	2.25 s*s*

Tabella 4.1: Parametri del segnale di comando applicato alla valvola.

Si considera un ciclo di apertura e chiusura della valvola, corrispondente a una singola sequenza di comando. L'andamento atteso prevede che, trascorso un intervallo di 1 s, venga inviato il segnale di apertura che porta la valvola nello stato aperto. Quest'ultima permane in tale condizione fino alla ricezione del segnale di chiusura, applicato con un ritardo di 2.25 s rispetto al comando di apertura. Successivamente, la valvola rimane nello stato chiuso fino all'invio di un nuovo segnale di apertura, che si presenta nuovamente con un ritardo di 1, s rispetto al comando di chiusura.

Come già anticipato, trattandosi di un segnale ad onda quadra, esso assume esclusivamente due valori:

- **0** → assenza del comando;
- **1** → invio del comando.

Nelle Figure 4.6 e 4.7 è possibile osservare le caratteristiche del segnale appena evidenziate. È importante sottolineare che, nel caso del segnale di apertura, il valore 1 viene assunto nel momento in cui il comando è inviato e la valvola risulta aperta. Lo stesso accade per la chiusura: il segnale passa a 1 quando viene trasmesso il comando di chiusura.

Ciò avviene perché, in questa fase, lo stato 1 non distingue tra apertura o chiusura, ma indica semplicemente l'invio del comando.

Tale interpretazione cambierà invece quando si considererà lo stato finale della valvola.

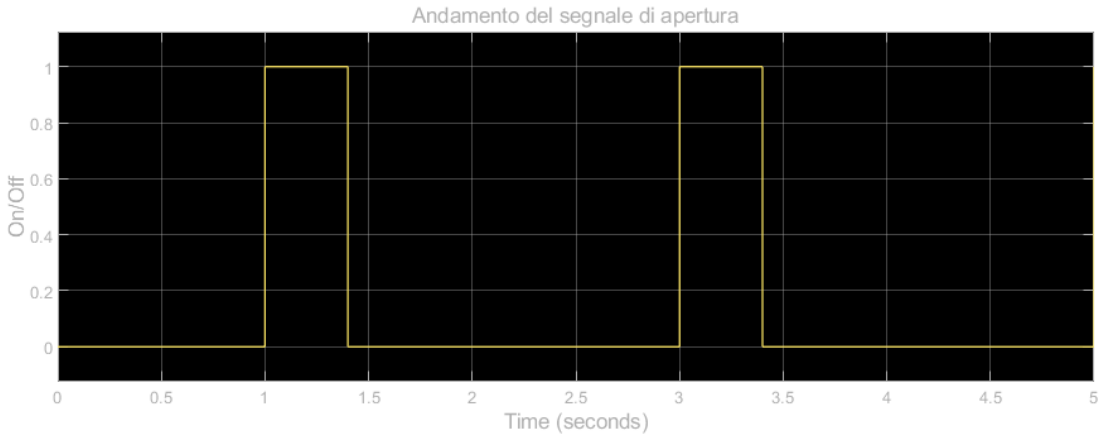


Figura 4.6: Andamento del segnale di input per eseguire l'apertura della valvola.

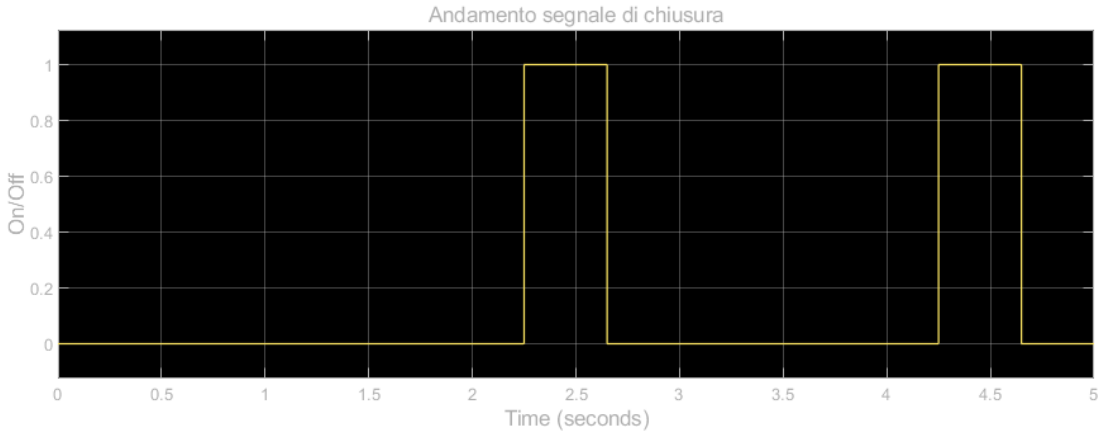


Figura 4.7: Andamento del segnale di input per eseguire la chiusura della valvola.

Insieme al comando di pilotaggio, è stata fatta una misura della corrente in itinere alla simulazione, in modo da tenere sotto controllo l'andamento è che questo potesse essere compatibile con il segnale inviato e con i valori delle caratteristiche del sistema.

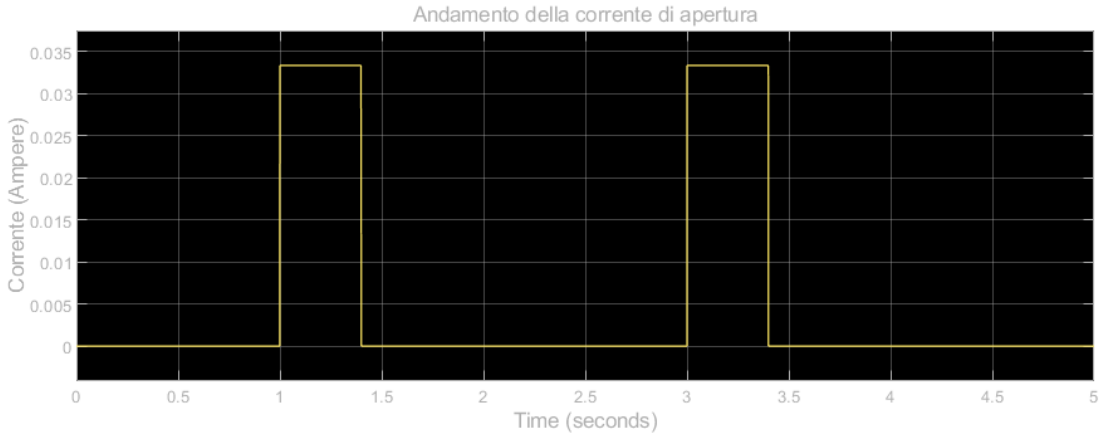


Figura 4.8: Andamento della corrente durante l'eccitazione della parte di circuito corrispondente all'apertura della valvola.

Nelle Figure 4.8 e 4.9 viene riportato l'andamento dell'output prodotto dal *Current Sensor*. Il picco di corrente massima lo si ottiene quando lo stato del

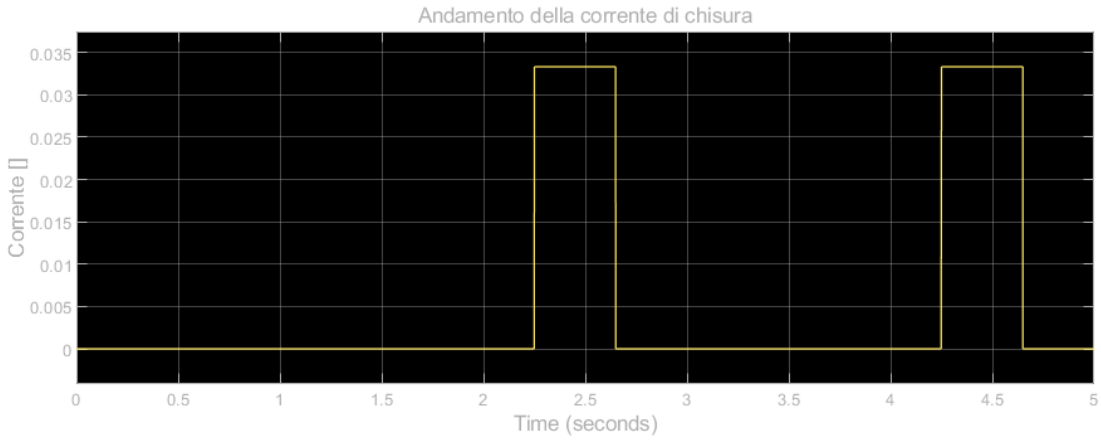


Figura 4.9: Andamento della corrente durante l'eccitazione della parte di circuito corrispondente alla chiusura della valvola.

segnale è 1, non appena questo cessa anche la corrente torna ad assumere un valore nullo. Il valore sembra essere in accordo con il corrispettivo nella Tabella 3.4.

Grazie all'inserimento del blocco *Flip-flop* nello schema circuitale è stato possibile combinare i segnali di apertura e chiusura in modo da poter ottenere la simulazione complessiva del comando di pilotaggio e osservare dunque lo stato della valvola durante l'avanzamento della simulazione.

La Figura 4.10 permette la visualizzazione chiara dello stato della valvola. Come anticipato in precedenza, nel caso del segnale il valore 1 indicava semplicemente l'invio del comando, indipendentemente che fosse di apertura o di chiusura. Se invece si considera lo stato finale della valvola, i valori assumono il seguente significato:

- **0** → valvola chiusa;
- **1** → valvola aperta.

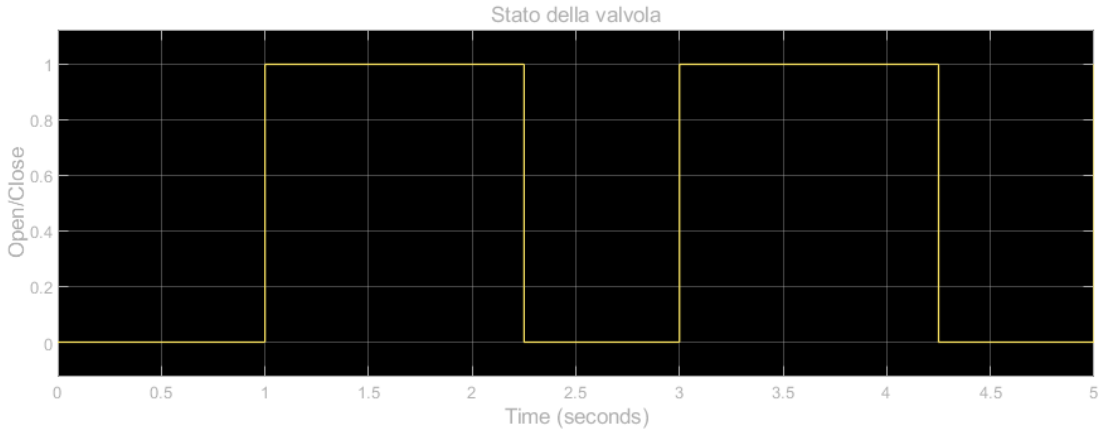


Figura 4.10: Visualizzazione grafica dello stato della valvola.

Si osserva che lo stato della valvola commuta da 0 a 1 in corrispondenza del comando di apertura, trasmesso all'istante $T = 1\,\text{s}$. Successivamente, quando viene inviato il comando di chiusura, $T = 2{,}25\,\text{s}$, lo stato ritorna dal valore 1 al valore 0, evidenziando la corretta risposta della valvola anche nella fase di spegnimento. Perciò la valvola riceve l'impulso relativo al comando di apertura, esegue la commutazione, mantiene lo stato finché non viene inviato il segnale relativo al comando di chiusura, ritardato di 2,25 s rispetto al segnale di apertura. Lo stato della valvola rispecchia esattamente l'andamento previsto.

Dopo aver eseguito la simulazione con i valori di riferimento dei componenti elettronici, si possono effettuare ulteriori simulazioni variando i valori di resistenza e induttanza per osservare le corrispondenti variazioni nell'andamento della corrente. Si può quindi osservare il grafico di confronto dei diversi andamenti della corrente, corrispondenti alle diverse combinazioni, ottenuto utilizzando MATLAB. In particolare si distinguono due set di curve ognuna rappresentativa di un solenoide:

- Curve tratteggiate → rappresentano i picchi di corrente relativi al comando di apertura della valvola;

- Curve continue → rappresentano i picchi di corrente relativi al comando di chiusura della valvola.

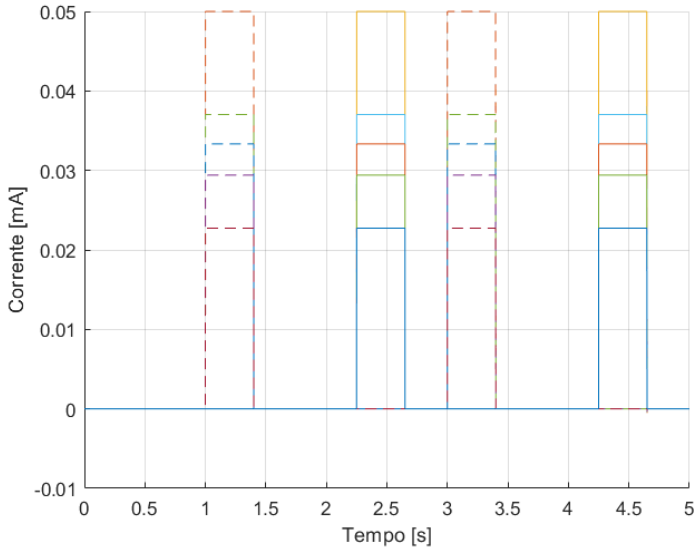


Figura 4.11: Confronto tra i diversi andamenti di corrente corrispondenti alle differenti combinazioni di resistori e induttori.

Come già stabilito in precedenza, i valori di corrente massima delle varie combinazioni sono coerenti con i valori presenti nella Tabella 3.4.

È stato quindi possibile simulare e visualizzare il comando di pilotaggio della valvola. Tale simulazione si è rivelata particolarmente utile, in quanto ha permesso di analizzare in maniera più approfondita il funzionamento del sistema di comando e di evidenziare le sue principali caratteristiche operative. Inoltre, lo studio preliminare del comportamento simulato ha contribuito a semplificare e a rendere più immediata la successiva realizzazione fisica del circuito di pilotaggio.

4.3 Verifica della tensione di Pull-In degli avvolgimenti

In questa sezione vengono osservati i risultati della Sezione 3.4.3.
Nel Capitolo 3 è stato sviluppato il circuito rappresentativo del comando di pilotaggio della valvola. A questo circuito è stata modificata la modalità di alimentazione della bobina dedicata all'apertura. Questo viene reso necessario dal momento che, l'intervallo di valori di tensione desiderato per svolgere la verifica, eccede il range gestibile dalla sola scheda Arduino.

In primo luogo è stata realizzata una simulazione volta a rappresentare il comportamento atteso del sistema implementato nel codice. In Figura 4.12 è illustrato l'andamento previsto per la valvola in funzione della variazione di tensione.

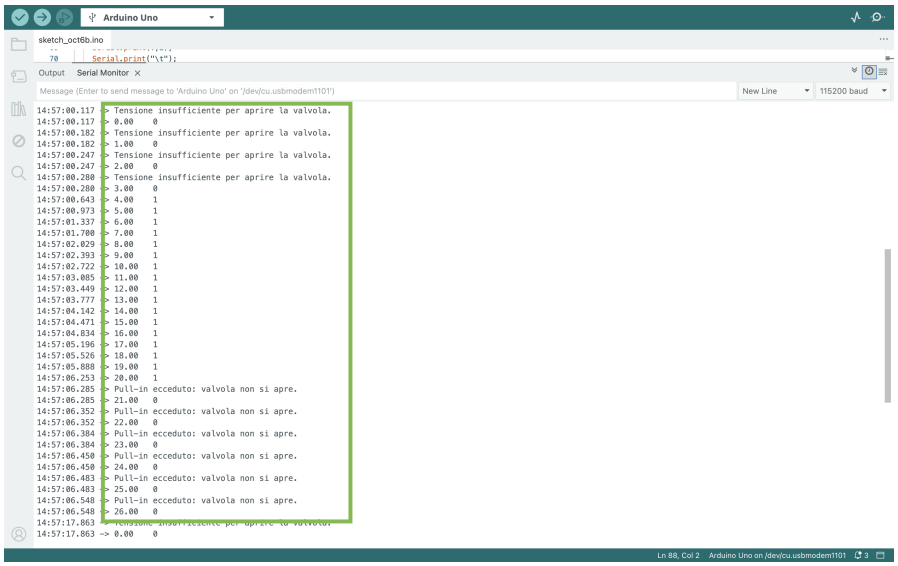


Figura 4.12: Simulazione dei risultati che si vogliono ottenere dalla verifica della tensione di Pull-In.

L'obiettivo di questa fase è ottenere una rappresentazione grafica dell'andamento della tensione applicata agli avvolgimenti durante l'invio del comando

di apertura della valvola. Il segnale di comando viene inviato e la tensione, regolata dal dispositivo di controllo, viene incrementata progressivamente con step fissi da 0 V fino a V_{PULL}.

Al termine del primo ciclo di variazione, una volta raggiunti i valori massimi dell'intervallo, il comando viene ripetuto per valutare la ripetibilità del comportamento del sistema. Secondo le specifiche di progetto della valvola HPLV, il limite minimo della tensione di Pull-In è pari a 4 V in corrente continua, mentre il limite massimo è di 20 VDC.

Come mostrato in Figura 4.12, l'apertura della valvola avviene quando il regolatore imposta una tensione compresa all'interno di tale intervallo. Poiché il range di valori di tensione utilizzato per la verifica sperimentale ($0\,\mathrm{V} - 26\,\mathrm{V}$) è più ampio rispetto a quello definito dalla specifica di progetto ($4\,\mathrm{V} - 20\,\mathrm{V}$), si osservano condizioni per le quali la valvola non effettua la commutazione di apertura.

In particolare, se la tensione fornita al comando risulta inferiore al valore minimo di attivazione $V_{\mathrm{PULL}}^{\mathrm{min}}$, la valvola non dispone dell'energia sufficiente per cambiare stato. Analogamente, per tensioni superiori al limite massimo $V_{\mathrm{PULL}}^{\mathrm{max}}$, l'alimentazione risulta eccessiva e non conforme alle condizioni operative nominali.

Successivamente è stato sviluppato il codice effettivo che consente alla scheda Arduino di pilotare sia il potenziometro digitale sia il regolatore di tensione (si veda la Sezione C.4 in appendice). I risultati dei cicli di variazione della tensione sono riportati in Figura 4.13.

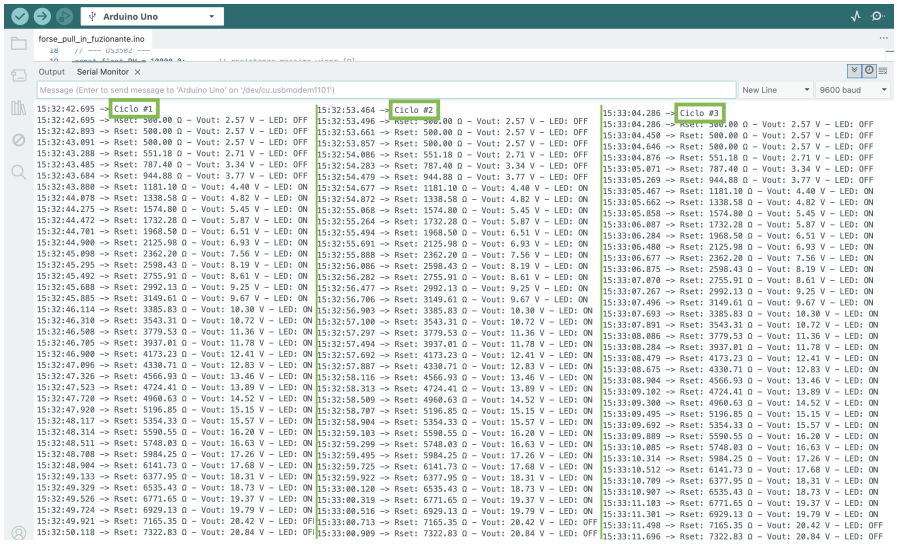


Figura 4.13: Risultati ottenuti dall'Arduino IDE per la verifica della tensione di Pull-In.

Ad ogni step è stato stampato il valore di resistenza impostata dal potenziometro e di tensione calcolata dal regolatore. I dati sono stati successivamente estrapolati ed inseriti in uno script MATLAB (si veda Sezione B.2 in appendice) in modo da osservarne meglio gli andamenti.
Si ricorda che la tensione di uscita del regolatore è espressa dalla relazione 3.2 riportata di seguito.

$$V_{OUT} = 1.240 \left(1 + \frac{R_1}{R_2}\right)$$

Nel caso in esame, la resistenza R_1 è variabile mentre R_2 è fissa. Nelle Figure 4.14 e 4.15 si può facilmente notare come la tensione di uscita risulti direttamente proporzionale al valore assunto da R_1, ovvero le due grandezze presentano lo stesso andamento. Ciò implica che all'aumentare della resistenza R_1 si osserva un incremento lineare di V_{OUT}, partendo da un valore minimo pari a 1.240 V fino al valore massimo corrispondente alla resistenza $R_1^{max} = 10\,\mathrm{k}\Omega$.
Dai grafici ottenuti è possibile osservare visivamente la proporzionalità che

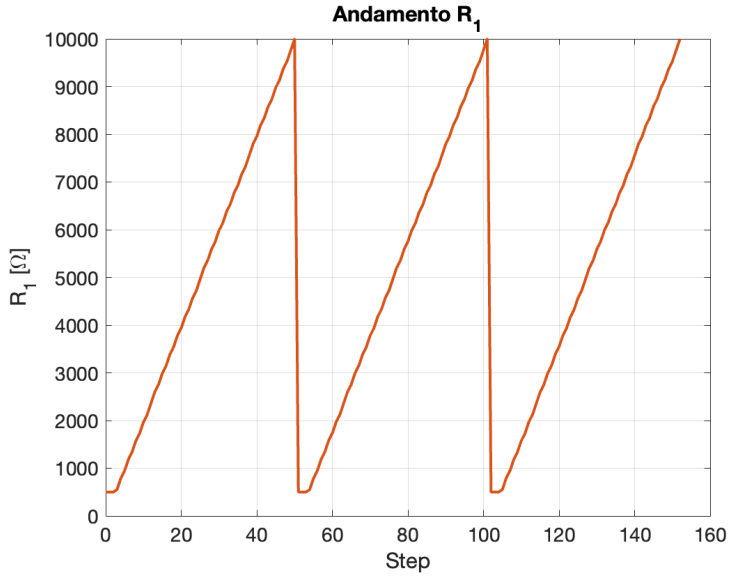


Figura 4.14: Andamento della resistenza R_1 impostata dal potenziometro digitale DS3502.

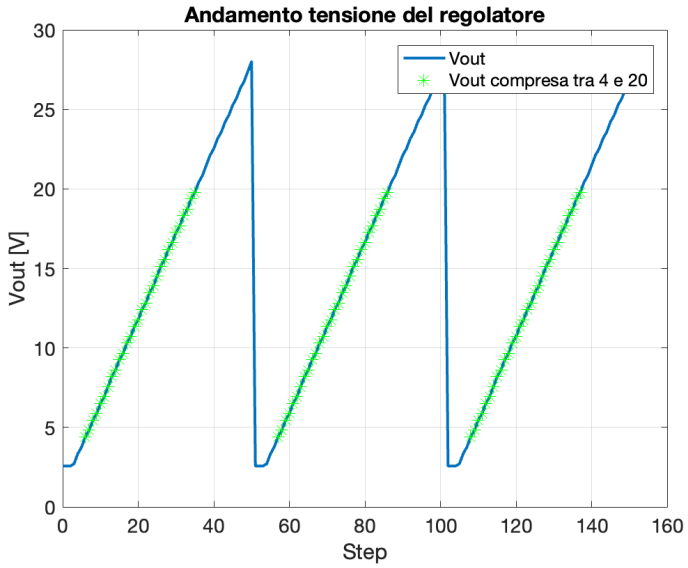


Figura 4.15: Andamento della tensione impostata dal regolatore di tensione MIC29502WT.

nasce tra la tensione e la resistenza.

$$V_{\mathrm{OUT}} \propto R_1$$

Oltretutto, come accennato nella Sezione 2.2.3, il potenziometro permette una regolazione lineare della resistenza, di conseguenza gli andamenti sia di R_1 che di V_{OUT} che si otterranno saranno lineari.
Nel grafico della tensione (Figura 4.15) sono stati inseriti dei marker colorati distinti dalla curva principale, al fine di evidenziare i valori di tensione che determinano la commutazione della valvola e la sua apertura.

Per chiarezza di rappresentazione, è stato costruito un grafico che rappresentasse l'andamento dello stato della valvola in modo chiaro ed immediato. Si osserva che per tensioni inferiori a 4 V la valvola rimane chiusa, così come per tensioni superiori a 20 V.

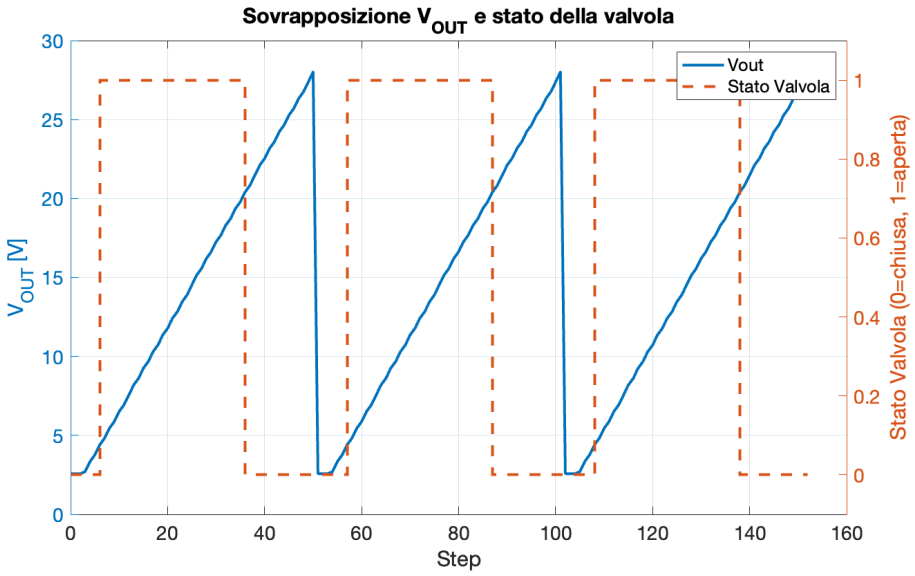


Figura 4.16: Andamento dello stato della valvola (1=aperta, 0=chiusa).

Capitolo 5

Conclusioni e sviluppi futuri

Grazie allo sviluppo di questa tesi prevalentemente a carattere progettuale, è stato possibile entrare in prima persona nel processo aziendale di progettazione e produzione di un componente per l'aerospazio.

La realizzazione del comando di pilotaggio è stato utile per verificare l'effettiva funzionalità del modello e della sua realizzazione.

Nonostante la resa più semplificata del progetto, è stato comunque possibile riprodurre il funzionamento dei principali componenti elettrici interni alla valvola e all'occorrenza è stato sostituito il componente fisico, presente nel progetto originale dell'azienda, con un'alternativa riproducibile in maniera semplificata ma funzionale per un progetto di tesi.
In particolare si parla del driver e dei sensori ad effetto Hall che nel progetto originale sono richiesti ma nel progetto svolto per la seguente tesi non sono stati utilizzati per questione di reperibilità del materiale. Nonostante ciò si sono utilizzate delle alternative per riprodurre e simulare il funzionamento del dispositivo mancante:

- I solenoidi sono stati rappresentati come una serie tra resistori ed induttori;

- I comandi forniti dal driver per pilotare i solenoidi sono stati direttamente implementati nel codice usato per istruire la scheda elettronica contenente il microprocessore.

- Il segnale di feedback di avvenuta commutazione fornito dai sensori ad effetto Hall, è stato fornito grazie all'inserimento di LED appositi che segnalassero lo stato della valvola.

Le simulazioni hanno riportato dei risultati conformi con la teoria trattata nel corso del lavoro andando a riprodurre visivamente il funzionamento del comando di pilotaggio.

Dal punto di vista delle analisi delle caratteristiche elettromagnetiche, è stato possibile andare a caratterizzare gli avvolgimenti dei solenoidi per mezzo di una misura delle grandezze delle caratteristiche elettriche del circuito e attraverso lo svolgimento di test relativo all'avvenuto passaggio di corrente all'interno dell'avvolgimento.
Questo è stato utile al fine di fornire una base di partenza per la scelta delle caratteristiche degli avvolgimenti una volta che il progetto aziendale possa procedere con la realizzazione del prodotto vero e proprio.

Un possibile sviluppo e aggiornamento di questo comando di pilotaggio potrebbe essere sicuramente la sostituzione dei dispositivi elettronici mancanti (driver e sensori Hall) in modo da poter svolgere delle analisi ancora più accurate e particolarizzate per gli effettivi valori e caratteristiche dei componenti.

Successivamente a questa fase iniziale di test, sarà finalmente possibile andare a produrre la valvola vera e propria anche sulla base dei risultati e delle conclusioni che sono emersi durante questo lavoro di tesi.
Seguiranno le prove di test e validazione sul prodotto finito ed infine la messa in commercio e l'equipaggiamento di un satellite.

Si conclude pertanto un percorso di ricerca che ha permesso di consolidare le conoscenze teoriche ed applicative sul tema trattato, aprendo la strada a nuove indagini e potenziali miglioramenti.

Appendice A

Cenni teorici

A.1 Effetto Hall

L'effetto Hall può essere interpretato come una conseguenza diretta della forza di Lorentz sugli elettroni in moto all'interno di un conduttore immerso in un campo magnetico perpendicolare alla corrente [7].

Si tratta di un fenomeno che entra in gioco quando avviene un'interazione tra campo elettrico e campo magnetico; consiste in un campo elettrico aggiuntivo che nasce all'interno di un conduttore percorso da corrente elettrica ed immerso in un campo magnetico esterno.
Vediamo di seguito come questo effetto si genera.

All'interno di un conduttore, ai cui capi viene applicata una differenza di potenziale, quindi un campo elettrico, avviene uno spostamento interno di elettroni dotati di una velocità detta velocità di deriva $\overline{v_d}$. Questo spostamento genera una corrente che sarà caratterizzata da un verso di percorrenza all'interno del conduttore. Per la teoria dell'elettrodinamica è noto che lo spostamento degli elettroni all'interno del conduttore è in verso opposto a quello della corrente.

Se si inserisce il conduttore all'interno di un campo magnetico in modo che quest'ultimo sia perpendicolare alla direzione della corrente, sugli elettroni in movimento immersi nel campo magnetico si genera una forza, detta **forza di Lorentz**, che tenderà a spostare ulteriormente gli elettroni. Questa forza la si esprime come segue.

$$\overline{F}_L = q\overline{v} \times \overline{B} \tag{A.1}$$

Dove i simboli si riferiscono rispettivamente a:

- $\overline{F_L}$: forza di Lorentz;
- q: carica;
- $\overline{v}$: velocità;
- $\overline{B}$: campo magnetico.

La direzione della forza di Lorentz la si ottiene sfruttando la regola della mano destra che risulterà in direzione perpendicolare a quella della corrente. Dunque gli elettroni vengono trascinati sia dalla velocità di deriva sia dalla forza di Lorentz e questo comporta l'accumulo di carica negativa su una particolare superficie del conduttore e quindi una carica positiva sull'altra superficie caricando di fatto con cariche opposte le pareti del conduttore. Tra le superfici del conduttore si instaura un campo elettrico aggiuntivo $\overline{E}_H$, il cosiddetto campo di Hall. Insieme al campo elettrico si genera anche una differenza di potenziale detta tensione di Hall

$$\Delta V_H = E_H d$$

L'equilibrio non viene raggiunto finché il campo elettrico $\overline{E}_H$ non compensa lo spostamento degli elettroni dovuto alla forza di Lorentz. Questo perché il campo elettrico risulta uscente dalla carica positiva ed entrante verso quella negativa, ovvero in verso opposto alla forza di Lorentz.

Il fenomeno persiste finché il conduttore risulta essere immerso nel campo magnetico il quale funge proprio da motore dell'**effetto Hall**.

In ottica della definizione e dello sviluppo degli equipaggiamenti di supporto a terra è importante introdurre i driver di pilotaggio dei solenoidi.

Appendice B

Codici Matlab e Simulink

B.1 Simulazione della risposta elettrica

```
%% Parametri generali
R_vals = [100, 135, 150, 170, 220];                % Ohm
L_vals = [0.7e-6, 13e-6, 25e-6, 38e-6, 50e-6];   % Henry
t_final = 0.1; % secondi
results = struct;
%% Ciclo sui valori di R e L
for i = 1:length(R_vals)
    for j = 1:length(L_vals)
        % Parametri per le bobine
        R_set = R_vals(i);  L_set = L_vals(j);
        R_reset = R_set;    L_reset = L_set;

        % Preparazione simulazione
        simIn = Simulink.SimulationInput('
   schema_valvola_quasi_completo');
        simIn = simIn.setVariable('R_set', R_set);
        simIn = simIn.setVariable('L_set', L_set);
        simIn = simIn.setVariable('R_reset', R_reset);
        simIn = simIn.setVariable('L_reset', L_reset);
```

```
19          simIn = simIn.setVariable('t_final', t_final);
20
21          % Esecuzione simulazione
22          simOut = sim(simIn);
23
24          % Recupero variabili esportate
25          i1 = simOut.i_set.Data;    % corrente bobina SET
26          i2 = simOut.i_reset.Data;  % corrente bobina
    RESET
27          t  = simOut.i_set.Time;    % tempo
28
29          % Salvataggio dati
30          results(i,j).R = R_set;
31          results(i,j).L = L_set;
32          results(i,j).t = t;
33
34          % --- Bobina 1 ---
35          results(i,j).bobina(1).i = i1;
36          y_final1 = i1(end);
37          idx1 = find(abs(i1 - y_final1) <= 0.05*abs(
    y_final1));
38          if isempty(idx1)
39              results(i,j).bobina(1).t_settling = NaN;
40          else
41              results(i,j).bobina(1).t_settling = t(idx1(
    end));
42          end
43
44          % --- Bobina 2 ---
45          results(i,j).bobina(2).i = i2;
46          y_final2 = i2(end);
47          idx2 = find(abs(i2 - y_final2) <= 0.05*abs(
    y_final2));
```

```
48          if isempty(idx2)
49              results(i,j).bobina(2).t_settling = NaN;
50          else
51              results(i,j).bobina(2).t_settling = t(idx2(
    end));
52          end
53      end
54  end
55
56  %% Plot comparativo
57  figure; hold on;
58  for i = 1:length(R_vals)
59      for j = 1:length(L_vals)
60          plot(results(i,j).t, results(i,j).bobina(1).i, '
    --', 'DisplayName', ...
61              sprintf('i1: R=%g, L=%g', results(i,j).R,
    results(i,j).L));
62          plot(results(i,j).t, results(i,j).bobina(2).i, '-
    ', 'DisplayName', ...
63              sprintf('i2: R=%g, L=%g', results(i,j).R,
    results(i,j).L));
64      end
65  end
66  xlabel('Tempo [s]');
67  ylabel('Corrente [mA]');
68  %legend('show','Location','bestoutside');
69  grid on;
```

B.2 Verifica della tensione di Pull-In degli avvolgimenti

```
% --- Dati ---
Rset =
   [500,500,500,551.18,787.40,944.88,1181.10,1338.58,...

   1574.80,1732.28,1968.50,2125.98,2362.20,2598.43,...

   2755.91,2992.13,3149.61,3385.83,3543.31,3779.53,...

   3937.01,4173.23,4330.71,4566.93,4724.41,4960.63,...

   5196.85,5354.33,5590.55,5748.03,5984.25,6141.73,...

   6377.95,6535.43,6771.65,6929.13,7165.35,7322.83,...

   7559.06,7795.28,7952.76,8188.98,8346.46,8582.68,...

   8740.16,8976.38,9133.86,9370.08,9527.56,9763.78,...
         10000];
Vout =
   [2.57,2.57,2.57,2.71,3.34,3.77,4.40,4.82,5.45,5.87,...

   6.51,6.93,7.56,8.19,8.61,9.25,9.67,10.30,10.72,...

   11.36,11.78,12.41,12.83,13.46,13.89,14.52,15.15,...

   15.57,16.20,16.63,17.26,17.68,18.31,18.73,19.37,...

   19.79,20.42,20.84,21.48,22.11,22.53,23.16,23.58,...

   24.22,24.64,25.27,25.69,26.32,26.75,27.38,28.01];
```

```
17
18 % LED acceso = 1, spento = 0
19 LED = Vout >= 4 & Vout <= 20;
20
21 % --- Ripetizione dei dati 3 volte ---
22 rep = 3;
23 tempo_rep = [];
24 Rset_rep = [];
25 Vout_rep = [];
26 LED_rep = [];
27
28 for k = 0:(rep-1)
29     tempo_rep = [tempo_rep, (0:length(Vout)-1) + k*length
   (Vout)];
30     Rset_rep = [Rset_rep, Rset];
31     Vout_rep = [Vout_rep, Vout];
32     LED_rep = [LED_rep, LED];
33 end
34
35 %% --- Figura 1: Vout vs tempo con marker per 4<=Vout<=20
    ---
36 figure;
37 plot(tempo_rep, Vout_rep, '-', 'LineWidth', 2, 'Color', '
   b'); hold on;
38
39 % Marker verdi dove Vout è tra 4 e 20
40 idx_marker = find(Vout_rep >= 4 & Vout_rep <= 20);
41 plot(tempo_rep(idx_marker), Vout_rep(idx_marker), 'g*', '
   MarkerSize', 8);
42
43 xlabel('Step simulato');
44 ylabel('Vout [V]');
45 title('Andamento Vout con marker (4 <= Vout <= 20)');
```

```
46 grid on;
47 legend('Vout','Vout compresa tra 4 e 20');
48
49 %% --- Figura 2: Rset vs tempo ---
50 figure;
51 plot(tempo_rep, Rset_rep, '-', 'LineWidth', 2, 'Color', '
      m');
52 xlabel('Step simulato');
53 ylabel('Rset [\Omega]');
54 title('Andamento Rset');
55 grid on;
56
57 %% --- Figura 3: Stato valvola ---
58 figure;
59 stairs(tempo_rep, LED_rep, 'r-', 'LineWidth', 2);
60 xlabel('Step simulato');
61 ylabel('Stato Valvola (0=chiusa, 1=aperta)');
62 ylim([-0.1 1.1]);
63 title('Andamento Stato Valvola');
64 grid on;
65
66 %% --- Figura 4: Vout e Stato Valvola sovrapposti ---
67 figure;
68
69 yyaxis left
70 plot(tempo_rep, Vout_rep, '-', 'LineWidth', 2); hold on;
71 ylabel('Vout [V]');
72 grid on;
73
74 yyaxis right
75 stairs(tempo_rep, LED_rep, '--', 'LineWidth', 2);
76 ylabel('Stato Valvola (0=chiusa, 1=aperta)');
77 ylim([-0.1 1.1]);
```

```
78
79 xlabel('Step');
80 title('Sovrapposizione Vout e Stato Valvola');
81 legend('Vout','Stato Valvola');
```

Appendice C

Codici Arduino

C.1 Misura di resistenza e impedenza con Arduino

```
// --- CONFIGURAZIONE PIN ---
const int PWM_PIN = 9;          // uscita PWM
const int V_N1_PIN = A0;        // nodo tra PWM e bobina
const int V_N2_PIN = A1;        // nodo tra bobina e shunt

// --- PARAMETRI ---
const float Vref = 5.0;          // tensione di
   riferimento ADC
const float R_shunt = 10.0;      // resistenza shunt (Ohm)
const int N_DC_SAMPLES = 20;     // campioni DC
const int N_AC_SAMPLES = 100;    // campioni per AC

// --- PWM sinusoide grezza ---
int pwmFromSine(float angle) {
  float sinVal = (sin(angle) + 1.0) / 2.0; // scala tra 0
    e 1
  return (int)(sinVal * 255);
```

```
}

// --- Funzione DC ---
float measureResistanceDC(int duty) {
  analogWrite(PWM_PIN, duty);
  delay(10); // stabilizzazione

  const int N_avg = 50; // numero di campioni da mediare
  float Vb_sum = 0;
  float Vs_sum = 0;

  for(int i = 0; i < N_avg; i++){
    int rawN1 = analogRead(V_N1_PIN);
    int rawN2 = analogRead(V_N2_PIN);
    float Vn1 = rawN1 * Vref / 1023.0;
    float Vn2 = rawN2 * Vref / 1023.0;

    Vb_sum += (Vn1 - Vn2);  // tensione bobina
    Vs_sum += Vn2;          // tensione shunt

    delay(2); // piccolo ritardo tra campioni
  }

  float Vb_avg = Vb_sum / N_avg;
  float Vs_avg = Vs_sum / N_avg;

  float I = Vs_avg / R_shunt;
  float R = Vb_avg / I;

  return R;

}

```

```
49 // --- Funzione AC ---
50 float measureImpedanceAC(float freq) {
51   float Vb_sum2 = 0;
52   float Vs_sum2 = 0;
53
54   for(int i = 0; i < N_AC_SAMPLES; i++){
55     float t = (float)i / (freq * N_AC_SAMPLES);
56     float angle = 2.0 * PI * freq * t;
57     int duty = pwmFromSine(angle);
58     analogWrite(PWM_PIN, duty);
59
60     int rawN1 = analogRead(V_N1_PIN);
61     int rawN2 = analogRead(V_N2_PIN);
62     float Vn1 = rawN1 * Vref / 1023.0;
63     float Vn2 = rawN2 * Vref / 1023.0;
64
65     float V_bobina = Vn1 - Vn2;
66     float V_shunt = Vn2;
67
68     Vb_sum2 += V_bobina * V_bobina;
69     Vs_sum2 += V_shunt * V_shunt;
70
71     delayMicroseconds(200); // intervallo campioni
72   }
73
74   float Vb_rms = sqrt(Vb_sum2 / N_AC_SAMPLES);
75   float Vs_rms = sqrt(Vs_sum2 / N_AC_SAMPLES);
76   float I_rms = Vs_rms / R_shunt;
77   float Z = Vb_rms / I_rms;
78
79   return Z;
80 }
81
```

```
82 void setup() {
83   Serial.begin(115200);
84   pinMode(PWM_PIN, OUTPUT);
85 }
86
87 void loop() {
88   // --- Misura DC ---
89   float R_DC = measureResistanceDC(128); // duty 50%
90   Serial.print("Resistenza DC (Ohm): ");
91   Serial.println(R_DC, 2);
92
93  // --- Misura AC ---
94   Serial.println("Impedenza AC (Ohm):");
95   float freqs[5] = {50, 500, 1000, 1500, 2000}; // 5
      frequenze
96   for(int i = 0; i < 5; i++){
97     float Z_AC = measureImpedanceAC(freqs[i]);
98     Serial.print(freqs[i]);
99     Serial.print(" Hz : ");
100    Serial.println(Z_AC, 2);
101    delay(200);
102  }
103
104  Serial.println("-------------------------");
105  delay(2000);
106 }
```

C.2 Comando eccitazione singola bobina

```
1 // === Configurazione pin ===
2 const byte COIL_PIN = 5;   // bobina
```

```
3  const byte LED_PIN  = 6;   // LED corrispondente
4
5  // === Parametri regolabili ===
6  const unsigned int PULSE_MS = 200; // durata impulso
       bobina
7
8  void setup() {
9    pinMode(COIL_PIN, OUTPUT);
10   pinMode(LED_PIN,  OUTPUT);
11
12   // Tutto spento inizialmente
13   digitalWrite(COIL_PIN, LOW);
14   digitalWrite(LED_PIN,  LOW);
15
16   Serial.begin(115200);
17   Serial.println("Invia 'O' per attivare la bobina e il
       LED, 'C' per spegnere il LED");
18 }
19
20 void pulseCoil() {
21   digitalWrite(COIL_PIN, HIGH); // eccita bobina
22   digitalWrite(LED_PIN, HIGH);   // accende LED
23   delay(PULSE_MS);               // durata impulso
24   digitalWrite(COIL_PIN, LOW);   // spegne bobina
25   Serial.println("Bobina attivata per 200ms, LED acceso")
       ;
26 }
27
28 void loop() {
29   if (Serial.available()) {
30     char c = toupper(Serial.read());
31     if (c == 'O') pulseCoil();
32     if (c == 'C') {
```

```
33        digitalWrite(LED_PIN, LOW); // spegne LED
34        Serial.println("LED spento");
35      }
36    }
37  }
```

C.3 Comando di commutazione della valvola

```
1  // === Configurazione pin ===
2  const byte COIL_OPEN_PIN  = 5;  // D5 -> bobina APRI (
       driver IN1)
3  const byte COIL_CLOSE_PIN = 6;  // D6 -> bobina CHIUDI (
       driver IN2)
4  const byte HALL_OPEN_PIN  = 7;  // sensore effetto Hall
       posizione APERTA
5  const byte HALL_CLOSE_PIN = 8;  // sensore effetto Hall
       posizione CHIUSA
6
7
8  // === Parametri regolabili ===
9  const unsigned int PULSE_MS     = 200; // durata impulso
10 const unsigned int INTERLOCK_MS = 50;  // pausa di
       sicurezza tra una bobina e l'altra
11
12
13 void setup() {
14   // put your setup code here, to run once:
15   pinMode(COIL_OPEN_PIN,  OUTPUT); // dice ad Arduino che
       quei pin non leggono, ma inviano un segnale (comando
       al driver)
```

```
16   pinMode(COIL_CLOSE_PIN, OUTPUT); // dice ad Arduino che
       quei pin non leggono, ma inviano un segnale (comando
     al driver)
17   digitalWrite(COIL_OPEN_PIN,  LOW); // garantisce che
     all'inizio i pin siano spenti (nessuna bobina attiva)
18   digitalWrite(COIL_CLOSE_PIN, LOW); // garantisce che
     all'inizio i pin siano spenti (nessuna bobina attiva)
19   pinMode(HALL_OPEN_PIN,  INPUT); // questi pin servono a
       leggere uno stato digitale, non a inviare corrente
20   pinMode(HALL_CLOSE_PIN, INPUT); // questi pin servono a
       leggere uno stato digitale, non a inviare corrente
21
22   Serial.begin(115200); // apre la comunicazione seriale
     col PC a 115200 baud (velocita'). Serve per dare
     comandi da monitor seriale
23   Serial.println("Comandi: O=Open, C=Close"); //
     messaggio di istruzioni che compare appena avvio
24 }
25
26
27 // Funzione pulseCoil
28 void pulseCoil(byte pin, unsigned int ms) {
29   digitalWrite(pin, HIGH); //accende un pin (HIGH) ->
     bobina eccitata
30   delay(ms); // aspetta ms millisecondi (delay(ms))
31   digitalWrite(pin, LOW); // spegne il pin (LOW) ->
     bobina rilasciata
32 }
33
34
35 // Funzione openValve
36 void openValve() {
```

```
37   digitalWrite(COIL_CLOSE_PIN, LOW);   // spegne la
     bobina opposta (sicurezza)
38   delay(INTERLOCK_MS);                 // piccola pausa
39   pulseCoil(COIL_OPEN_PIN, PULSE_MS);  // invia impulso
     di apertura
40   Serial.println("Valvola: APERTA (impulso inviato)"); //
       Scrive sulla seriale un messaggio a video
41 }
42
43
44 // Funzione closeValve
45 void closeValve() {
46   digitalWrite(COIL_OPEN_PIN, LOW);
47   delay(INTERLOCK_MS);
48   pulseCoil(COIL_CLOSE_PIN, PULSE_MS);
49   Serial.println("Valvola: CHIUSA (impulso inviato)");
50 }
51
52  void checkSensors() {
53   int hallOpen  = digitalRead(HALL_OPEN_PIN);   // legge
     stato sensore APERTA
54   int hallClose = digitalRead(HALL_CLOSE_PIN);  // legge
     stato sensore CHIUSA
55
56   // Ricorda: 0 = campo magnetico presente, 1 = campo
     assente
57   Serial.print("Sensore APERTA: ");
58   Serial.print(hallOpen == LOW ? "CAMPO PRESENTE (0)" : "
     CAMPO ASSENTE (1)");
59   Serial.print(" | Sensore CHIUSA: ");
60   Serial.println(hallClose == LOW ? "CAMPO PRESENTE (0)"
     : "CAMPO ASSENTE (1)");
61  }
```

```
62
63 void loop() {
64   // Gestione comandi da seriale
65   if (Serial.available()) { // controlla se c'e' un
     carattere arrivato dal PC (dal monitor seriale)
66     char c = toupper(Serial.read()); // quello che si e'
     scritto a tastiera viene letto (.read) e reso
     maiuscolo (toupper)
67     if (c == 'O') openValve(); // a seconda di quello che
     si e' scritto viene chiamata una funzione piuttosto
     che l'altra
68     if (c == 'C') closeValve();
69   }
70
71   // Controllo sensori ad effetto Hall
72   checkSensors();
73   delay(200); // lettura ogni 200 ms, regolabile
74 }
```

C.4 Verifica della tensione di Pull-In degli avvolgimenti

```
1 #include <Wire.h>
2
3 // --- DS3502 ---
4 #define DS3502_ADDR 0x28
5 #define DS3502_WIPER 0x00
6 #define WIPER_MAX 127
7
8 // --- Pin Arduino ---
```

```
#define VOLTAGE_PIN A0
#define LED_PIN 13

// --- Parametri test ---
const int steps = 50;           // numero di incrementi
const float VpullMin = 4.0;
const float VpullMax = 20.0;
const float VoutMax = 26.0;
const int repetitions = 3;
const int delayStep = 200;

// --- Partitore resistivo per lettura Vout ---
const float Rtop = 100000.0;
const float Rbottom = 22000.0;

// --- DS3502 RH/RL ---
const float RH = 10000.0;       // resistenza massima
   wiper [\Omega]
const float RL = 0.0;           // resistenza minima
   wiper [\Omega]

// --- ADC Arduino ---
const float ADC_REF = 5.0;
const int ADC_MAX = 1023;

void setup() {
  Wire.begin();
  Serial.begin(9600);
  pinMode(LED_PIN, OUTPUT);
  Serial.println("=== Test Pull-In Bobina con LED ===");
}

// Imposta il wiper del DS3502
```

```
40 void setWiper(byte value) {
41   Wire.beginTransmission(DS3502_ADDR);
42   Wire.write(DS3502_WIPER);
43   Wire.write(value);
44   Wire.endTransmission();
45 }
46
47 // Legge la tensione reale sulla bobina
48 float readVoltage() {
49   int adcValue = analogRead(VOLTAGE_PIN);
50   float V_A0 = adcValue * (ADC_REF / ADC_MAX);
51   float Vout = V_A0 * ((Rtop + Rbottom) / Rbottom);
52   return Vout;
53 }
54
55 // Calcola resistenza equivalente del DS3502
56 float wiperResistance(byte wiperValue) {
57   return RL + ((RH - RL) * wiperValue / WIPER_MAX);
58 }
59
60 void loop() {
61   for (int rep = 1; rep <= repetitions; rep++) {
62     Serial.print("Ciclo #"); Serial.println(rep);
63
64     for (int i = 0; i <= steps; i++) {
65       byte wiperValue = map(i, 0, steps, 0, WIPER_MAX);
66       setWiper(wiperValue);
67       delay(50);
68
69       float voltage = readVoltage();
70       float Rset = wiperResistance(wiperValue);
71
72       // Controllo LED
```

```
73        digitalWrite(LED_PIN, (voltage >= VpullMin &&
    voltage <= VpullMax) ? HIGH : LOW);
74
75        // Stampa solo Rset e Vout
76        Serial.print("Rset: "); Serial.print(Rset); Serial.
    print(" \Omega");
77        Serial.print(" - Vout: "); Serial.print(voltage);
    Serial.print(" V");
78        Serial.print(" - LED: ");
79        Serial.println((voltage >= VpullMin && voltage <=
    VpullMax) ? "ON" : "OFF");
80
81        delay(delayStep);
82      }
83
84      // Reset a 0V
85      setWiper(0);
86      digitalWrite(LED_PIN, LOW);
87      Serial.println("Vout riportata a 0V");
88      delay(500);
89
90      Serial.println("--------------------------------");
91    }
92
93    Serial.println("=== Test completato ===");
94    digitalWrite(LED_PIN, LOW);
95    while(1);
96  }
```

Bibliografia

[1] EASA. *High-pressure latching valve for elettrical propulsion*. 2024. URL: https://esastar-publication-ext.sso.esa.int/ESATenderActions/details/91415 (cit. a p. 8).

[2] S. Corpino. *Sistemi aerospaziali – appunti del corso*. Dispensa del corso, Politecnico di Torino, Corso di Laurea in Ingegneria Aerospaziale. 2024 (cit. a p. 9).

[3] G. Rizzoni. *Elettrotecnica - Principi e applicazioni*. Milano, Mi: McGraw-Hill Education, 2008 (cit. alle pp. 16–18, 39).

[4] Arduino. *What is Arduino?* 2018. URL: https://www.arduino.cc/en/Guide/Introduction/?_gl=1*1osjbrn*_up*MQ..*_ga*MTgzMDA5NzAzOS4xNzU2ODgzNjMw*_ga_NEXN8H46L5*czE3NTY4ODM2MzAkbzEkZzEkdDE3NTY4ODM2NTAkajQwJGwwJGgxNTIwNTA1MDI3 (cit. a p. 30).

[5] Arduino. *Datasheet Arduino UNO R3*. User manual. 2025 (cit. a p. 40).

[6] Micrel Inc. *Datasheet regolatore di tensione MIC29502WT*. Datasheet. 2005 (cit. a p. 45).

[7] Paolo Mazzoldi, Massimo Nigro e Cesare Voci. *Fisica, vol. 2: Elettromagnetismo e ottica*. Bologna: Zanichelli, 2007 (cit. a p. 69).

[8] Arduino. *Pinout Arduino UNO R3*. User manual. 2022.