POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Elettronica

Orientamento Sistemi Elettronici

Sviluppo hardware e firmware di un attuatore elettromeccanico per comandi cambio automatici

Relatore

Prof. Luciano LAVAGNO

Tutor Aziendale

Edoardo MONGARLI

Candidato

Gianluca IACOVELLI

Anno Accademico 2020/2021

Abstract

In questa tesi è descritto il progetto svolto presso l'azienda Silatech S.r.l. riguardante lo sviluppo di un sistema di controllo per un attuatore elettromeccanico, da utilizzare relativamente ad un comando per un cambio automatico by wire. Partendo dal prototipo meccanico già progettato e realizzato nel reparto R&D dell'azienda è stato studiato un equipaggiamento hardware e un sistema di controllo, tradotto successivamente in firmware, adeguato per la movimentazione dell'attuatore. È stata utilizzata la tecnica del model based design; in una prima fase è stato caratterizzato sperimentalmente l'attuatore al fine di creare un modello completo nell'ambiente di sviluppo Simulink; grazie ad esso è stato possibile progettare accuratamente un sistema di controllo in anello chiuso. Dopo sono state utilizzate diverse simulazioni per testare il comportamento dell'intero sistema nelle varie condizioni operative. Gli esiti positivi hanno permesso di procedere con lo studio di un algoritmo firmware per microcontrollore adeguato alle richieste. In parallelo a questa attività è stato portato avanti lo sviluppo dell'architettura hardware. Sono stati scelti i sensori, gli attuatori e la piattaforma microcontrollore più conforme alle richieste meccaniche e del sistema di controllo. Tali componenti hardware sono stati installati nel prototipo meccanico e il firmware è stato caricato sulla piattaforma target. Infine l'attuatore è stato testato nelle varie condizioni operative e i risultati sperimentali sono stati comparati con quelli delle simulazioni.

Indice

Elenco delle figure

Elenco dei listati

Capitolo 1

Introduzione

1.1 L'azienda

L'azienda che ha gentilmente concesso lo sviluppo di questa tesi è la Silatech S.r.l. È un'azienda leader nel campo automotive; nello specifico è una multinazionale specializzata nella produzione e distribuzione di sistemi completi di comando per cambi meccanici per conto delle principali case automobilistiche di tutto il mondo. Oltre al sistema di cambio manuale, Sila Group ha applicato la sua esperienza nel campo dei cambi automatici e, più recentemente alle soluzioni di comando cambio full-by-Wire. Nell'azienda sono stato inserito nel reparto di ricerca e sviluppo, specializzato nella creazione e test di prototipi. Ho lavorato in collaborazione con una collega ingegnere meccatronico, anche lei tesista, che si è occupata principalmente della modellizzazione del prototipo e dello sviluppo di un sistema di controllo in ambiente Matlab Simulink. Un secondo collega ingegnere dell'autoveicolo si è occupato invece della progettazione meccanica del prototipo. Inoltre ho interagito frequentemente con i colleghi esperti nelle lavorazioni meccaniche e nell'utilizzo di stampanti 3D per realizzare facilmente piccoli pezzi meccanici e apportare modifiche al prototipo esistente.

1.2 Comandi cambio

Il lavoro di tesi è stato incentrato sulla costruzione di un prototipo funzionante di un prodotto molto richiesto nel mercato automotive. La maggior parte delle automobili moderne hanno come equipaggiamento di serie sistemi di cambio della modalità di guida automatici. Essi differiscono dai cambi manuali per diversi fattori. Nei sistemi classici il guidatore tramite un asta seleziona il rapporto di trasmissione che si traduce solitamente nel movimento di due cavi meccanici collegati alla scatola cambio. Nelle versioni automatiche invece l'utente sceglie soltanto la modalità di guida tra parking, retro, neutral e drive oppure soltanto tra parking e not parking; sarà poi compito di un sistema di controllo secondario la gestione dei vari rapporti di trasmissione quando viene selezionata la modalità drive. Questo tipo di attuazione può essere effettuato tramite la movimentazione di un'asta che traduce la forza fornita dal guidatore in movimento lineare del cavo meccanico; questa soluzione però vincola ad uno specifico posizionamento del cavo e dell'asta all'interno dell'abitacolo. Una soluzione innovativa invece è quella che prevede l'ausilio di un attuatore elettromeccanico per movimentare il cavo. Il vantaggio dell'utilizzo di questo tipo di prodotto è quello di svincolare totalmente l'azione del guidatore da quella del cavo, poiché l'input fornito diventa la pressione di un pulsante oppure la movimentazione di un selettore al posto dell'applicazione di una forza sul comando cambio tradizionale. I diversi tipi di comando sono mostrati in Figura 1.1. Sulla sinistra un comando cambio con leva tradizionale, sulla destra un comando con selettore rotativo o monostabile.

Figura 1.1: Tipologie di comando cambio presenti nell'abitacolo

Nel caso in cui si scelga un comando elettronico, la movimentazione del cavo collegato alla scatola cambio è poi effettuata da un motore elettrico accoppiato ad una particolare trasmissione vite-madrevite che permette la traduzione di un moto angolare in moto lineare. Una visione di sistema è mostrata in Figura 1.2. Si nota come la parte che include il selettore sia completamente svincolata dalla parte di attuazione che invece deve essere in una particolare posizione poiché collegata alla scatola cambio tramite uno specifico cavo meccanico.

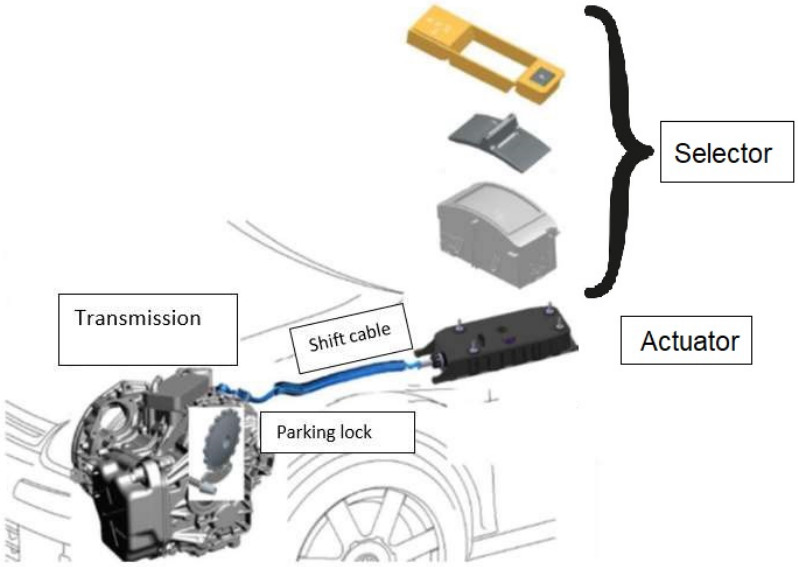


Figura 1.2: Vista di sistema

1.2.1 Scatola cambio

Come già descritto, l'attuatore ha come obiettivo la movimentazione di un cavo meccanico che è collegato al cambio automatico. Nello specifico questa componente meccanica dell'automobile presenta al suo esterno una leva, in grado di compiere dei movimenti angolari intorno ad un fulcro. Movimentando questa leva si induce il sistema interno di trasmissione a cambiare la modalità di funzionamento tra quelle disponibili (P, not P oppure P, R, N, D). Un esempio è mostrato in Figura 1.3.

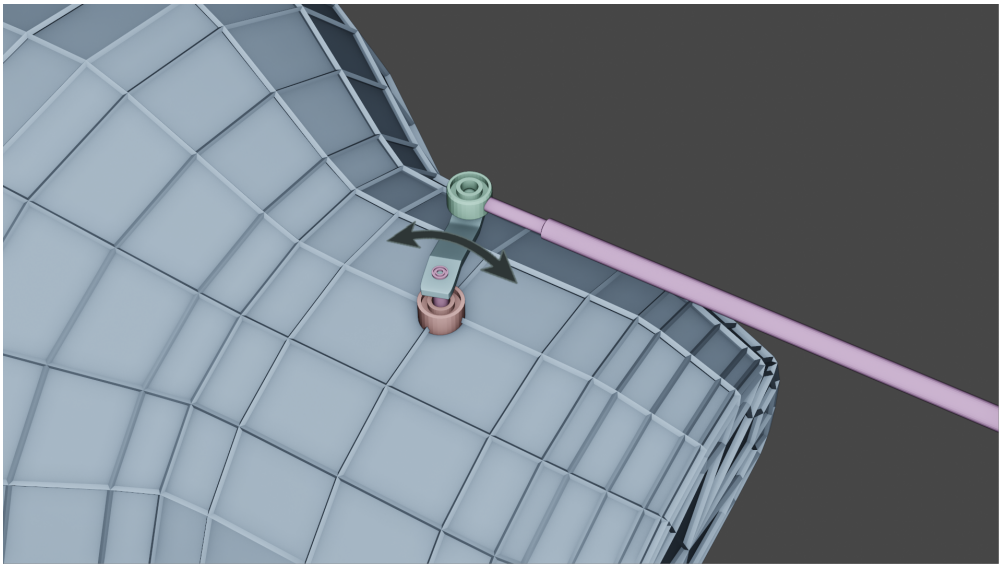


Figura 1.3: Posizionamento dell'attuatore nell'abitacolo

Ogni movimento è caratterizzato da uno specifico spostamento angolare e uno specifico profilo di forza da vincere. Il cavo collegato al comando cambio viene fissato sull'estremità della leva ed è studiato con delle proprietà meccaniche tali da poter effettuare delle specifiche corse e resistere alle forze previste dalla caratteristica del cambio. Questo aspetto è particolarmente importante ai fini dello sviluppo dell'attuatore perché essendo il movimento angolare direttamente proporzionale al movimento lineare, bisogna prevedere un sistema che sia in grado di effettuare delle movimentazioni con una corsa precisa e ci sia un motore in grado di vincere le forze resistenti in gioco.

1.3 Il funzionamento meccanico

Il lavoro di sviluppo dell'hardware e del firmware necessari per il corretto funzionamento dell'attuatore parte della comprensione del meccanismo messo a punto e dai componenti scelti preventivamente.

1.3.1 Funzionamento base

Una rappresentazione del prototipo di partenza è mostrata in Figura 1.4.

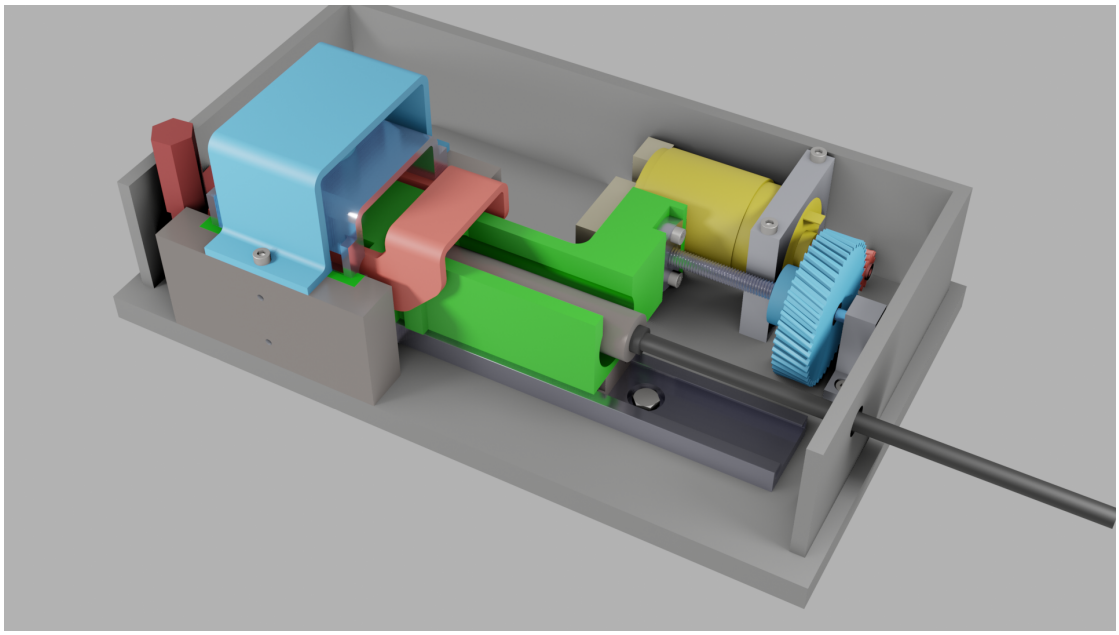


Figura 1.4: Protipo meccanico realizzato

La funzionalità principale dell'attuatore è quella di sostituire l'asta e il leveraggio che in un comando automatico classico permette il movimento del cavo collegato alla scatola cambio. La soluzione adottata è stata quella dell'utilizzo di un motore elettrico che tramite una trasmissione con ruote dentate ed un sistema vite-madrevite trasforma il moto angolare dell'albero del motore elettrico in moto lineare del cavo come mostrato in Figura 1.5.

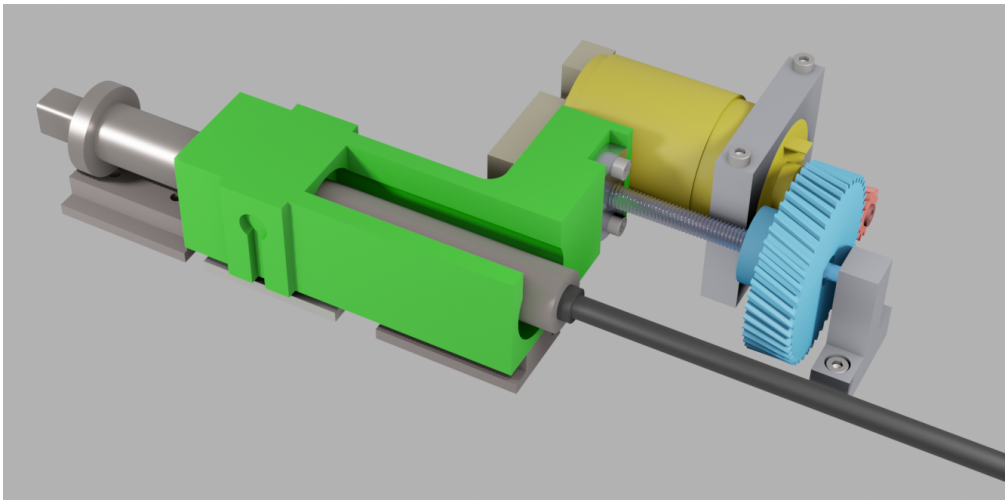


Figura 1.5: Accoppiamento vite-madrevite

Questa particolare scelta permette di poter movimentare agilmente il cavo tramite l'azionamento elettrico. Facendo riferimento alla Figura 1.4, si nota come l'alloggio del cavo sia costituito da una slitta, rappresentata in verde, che si muove in un binario tramite l'utilizzo di particolari pattini che riducono gli attriti; il movimento di tale slitta è garantito dall'accoppiamento di quest'ultimo con la madrevite.

1.3.2 Funzionalità di sicurezza

Il prototipo prevede inoltre una funzionalità di sicurezza. In mancanza di alimentazione, dovuta ad un fault elettrico o semplicemente quando il veicolo è spento, è previsto che il cavo debba essere movimentato e mantenuto in una specifica posizione. Questo è necessario per mantenere la leva sulla scatola cambio nella modalità parking, nella quale di default l'auto è ferma.
Per rendere possibile questo comportamento è stato pensato di aggiungere una seconda slitta a quella principale, collegata da un sistema di bielle, e una molla, come mostrato in Figura 1.6.

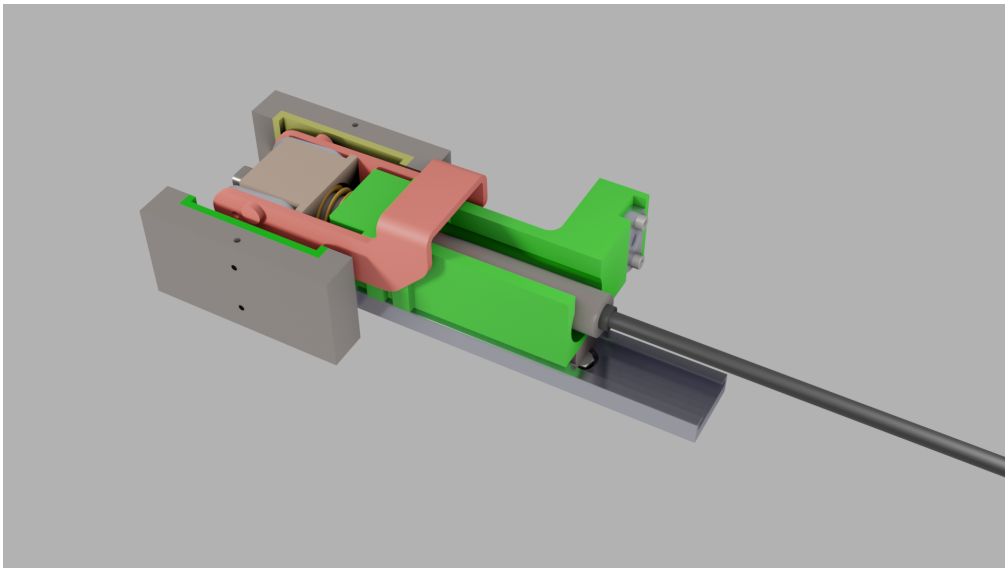


Figura 1.6: Seconda slitta e molla

Quando l'auto è spenta la molla posta tra le due slitte esercita una forza tale da mantenere il cavo in posizione parking, impedendo il libero movimento di quest'ultimo. All'accensione del veicolo, la molla viene compressa grazie al moto del motore e viene mantenuta in tale posizione tramite il sistema di bielle e l'elettromagnete di mantenimento, visibile in celeste in Figura 1.4. Dopo questa fase di setup iniziale del sistema, le due slitte e il cavo si muoveranno in maniera solidale. Se viene a mancare alimentazione all'elettromagnete, la molla si decomprimerà in automatico forzando il cavo nella posizione P. Per poter uscire nuovamente da questo stato sarà poi necessario tentare di ricomprimere la molla e permettere all'elettromagnete di mantenere la stessa compressa e le due slitte solidali. Bisogna precisare però che la modalità parking non coincide direttamente con l'inserimento del freno di stazionamento e il conseguente bloccaggio delle ruote, ma solo dalla modifica di modalità nella parte del cambio che va a disinserire gli ingranaggi della trasmissione. Sarà compito del sistema del freno di stazionamento andare a bloccare effettivamente le ruote.

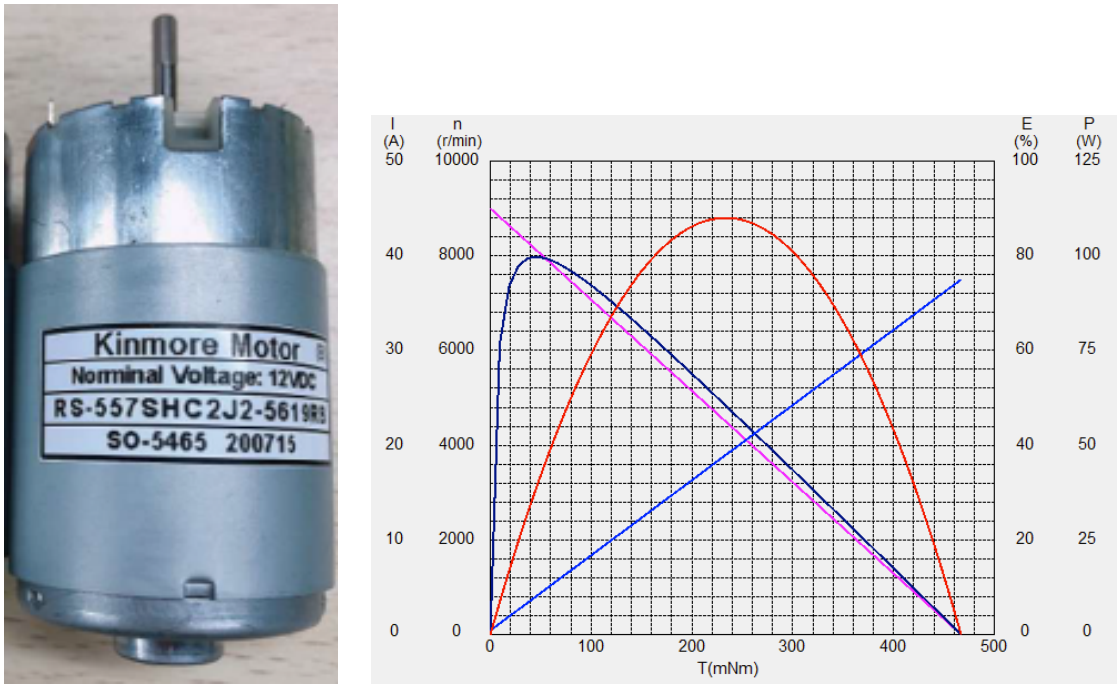
1.4 I componenti elettromeccanici

La parte di sviluppo meccanico dell'attuatore ha incluso la scelta di due componenti elettromeccanici che sono stati poi gestiti e pilotati all'interno del sistema realizzato in questo lavoro di tesi. È stato scelto il motore elettrico e l'elettromagnete di mantenimento.

1.4.1 Motore elettrico

Un componente cardine dell'attuatore è il motore elettrico utilizzato per le varie movimentazioni. Lo studio già presente della parte meccanica include anche la ricerca di un motore adatto a vincere i carichi in gioco. È stato evidenziato che per la maggior parte delle scatole cambio, per movimentare i leveraggi interni, è necessario vincere nelle condizioni peggiori delle forze di circa 500 N. Inoltre la tensione a cui si può accedere all'interno di un'automobile e quella della batteria che solitamente è di 12 V. La scelta del progettista meccanico è ricaduta su un motore con spazzole a magneti permanenti. L'azienda che ha fornito tale motore è la *Kinmore Motors* e il modello in questione è il *RS-557*.

La caratteristica di funzionamento alla tensione nominale di 12 V e l'immagine del motore sono mostrati in Figura 1.7. Nel grafico di sinistra viene mostrato la corrente assorbita, le ripetizioni al minuto, l'efficienza e la potenza assorbita in funzione della coppia erogata ad una tensione di funzionamento di 12 V. La coppia per la quale il motore è in stallo (velocità nulla) è di circa 450 mN m; dallo studio meccanico del prototipo è emerso che tale motore è ben dimensionato per l'applicazione in questione.



(a) *Il motore* **(b)** *La caratteristica di funzionamento*

Figura 1.7: Motore Kinmore RS-557

1.4.2 Elettromagnete

Come descritto in sottosezione 1.3.2, il comportamento corretto dell'attuatore è subordinato al corretto funzionamento dell'elettromagnete di mantenimento. Anche in questo caso dalle valutazioni meccaniche sul sistema composto dalle bielle, le due slitte e la molla compressa è emerso che la forza necessaria per mantenere la molla compressa è di circa 30 N. L'elettromagnete scelto funziona ad una tensione nominale di 12 V, compatibile con quella presente nell'automobile, e assorbe una corrente di 0.25 A; la forza di attrazione nominale è di 50 N, valore maggiore a quello necessario anche considerando un margine di sicurezza.

Capitolo 2

Introduzione al lavoro svolto

2.1 Fasi di progetto

Il prototipo meccanico già realizzato in azienda è stato progettato e realizzato rispettando delle richieste fornite da un cliente. Nello specifico ci si è concentrati sulla costruzione di un tipo di attuatore che fosse in grado di fare delle movimentazioni P, notP, in grado quindi di inserire e disinserire la modalità parking. Il lavoro da me svolto è partito dallo studio della specifica che definisce le diverse funzionalità e gli standard prestazionali che deve avere il prodotto finale. Una volta aver estratto le informazioni necessarie si è potuti partire con la vera fase progettuale. Il lavoro su cui mi sono concentrato è stato lo studio e lo sviluppo della parte hardware e firmware necessaria alla movimentazione dell'attuatore. In parallelo, la collega con cui ho collaborato ha modellizzato il sistema ed ha studiato una strategia di controllo per l'attuatore.

In una prima fase è stata definita l'architettura hardware del sistema, in maniera conforme a quanto richiesto dalla legge di controllo, progettata in ambiente Simulink. Sono stati quindi scelti dei componenti che fossero

adeguati alle performance richieste. Tra questi, sono stati selezionati quelli che garantissero un facile utilizzo e debug e che fossero reperibili sul mercato a dei prezzi ragionevoli.

Definita l'architettura hardware è stato progettato un firmware che prevedesse varie modalità di funzionamento, da quella standard a quelle necessarie nella fase finale del progetto. In questa fase sono stati effettuati dei test sperimentali sul prototipo per validare il modello progettato in Simulink e per verificare che il comportamento reale fosse comparabile con quello previsto dalle simulazioni.

2.2 Architettura del sistema

Per la progettazione dell'architettura di sistema è fondamentale capire nello specifico che tipo di movimenti devono essere previsti per l'attuatore. Obiettivo fondamentale della movimentazione del cavo è il raggiungimento e mantenimento di posizioni ben precise tramite l'attuazione del motore elettrico. Per questo motivo è stato necessario progettare un sistema in anello chiuso. Il feedback principale su cui è basato il sistema è un sensore di posizione che permette di tenere traccia della posizione del cavo meccanico istante per istante.

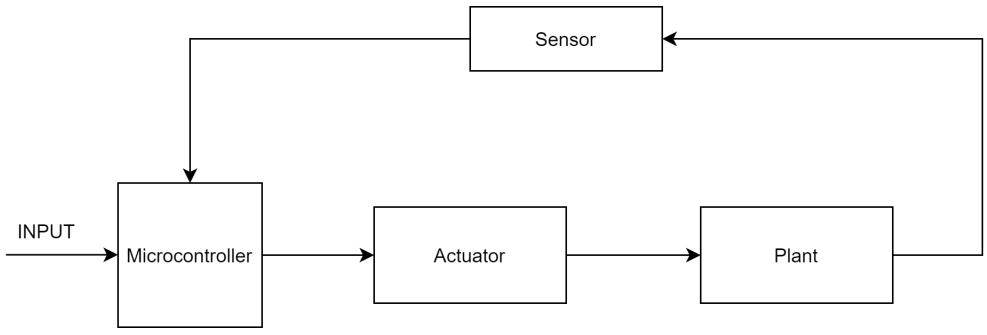


Figura 2.1: Vista di sistema dell'attuatore

Come è rappresentato in Figura 2.1 il sistema nella sua interezza è costituito dalla parte già progettata e realizzata, il plant, che è costituito dal motore con il cavo, le slitte e l'elettromagnete. A questo sono stati aggiunti tutti i componenti hardware e firmware per creare un sistema di controllo completo. È stata necessaria l'aggiunta di una parte di sensing, per rilevare cambiamenti e stato del plant istante per istante; un microcontrollore, la mente del sistema, capace di valutare la legge di controllo, interpretare i dati provenienti dai sensori, i comandi forniti dall'utente e pilotare il plant. Ultima parte anch'essa fondamentale per completare il sistema è l'attuazione, indispensabile per adattare i comandi che il microcontrollore fornisce al plant.

Capitolo 3

Creazione del modello e sviluppo del sistema di controllo

La progettazione del sistema che permette la movimentazione dell'attuatore è stata effettuata secondo un approccio model based, il quale prevede che in una prima fase venga studiato e modellizzato il sistema, successivamente venga effettuata la progettazione della legge di controllo e infine venga tradotta in codice C per microcontrollore.

3.1 Simulink

Il tool di simulazione Simulink fa parte della famiglia di software Matlab; è un comodo strumento che permette di realizzare per via grafica, inserendo degli appositi blocchi collegati da frecce, degli interi sistemi anche molto complicati al fine di poter simulare i più svariati comportamenti in differenti condizioni operative. Nel caso in questione è stato molto utile l'utilizzo di questo strumento perché ha permesso di progettare e tarare in maniera

rapida la legge di controllo, fulcro centrale dello sviluppo del progetto. Inoltre permette di effettuare svariate tipologie di simulazioni, tra cui quelle nel tempo continuo e nel tempo discreto. In una prima fase è stato utile utilizzare quelle nel tempo continuo per studiare il comportamento del sistema. In una fase avanzata invece le simulazioni nel tempo discreto hanno permesso di prevedere il comportamento che avrebbe avuto il microcontrollore nella valutazione della legge di controllo.

3.2 Modello Simulink dell'attuatore

L'analisi del prototipo già esistente ha portato alla creazione di un modello che rappresenta tramite equazioni i componenti già presenti nel prototipo. Una vista ad alto livello è mostrata in Figura 3.1.

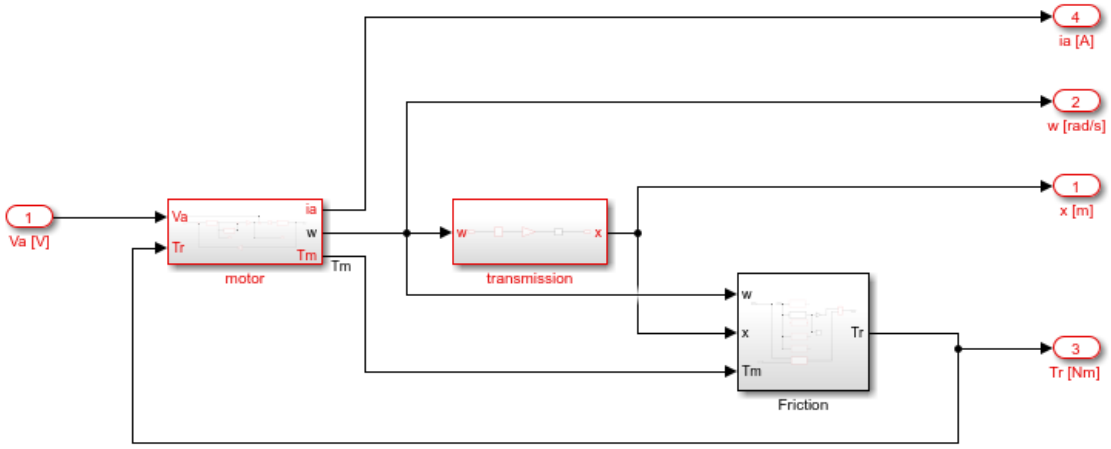


Figura 3.1: Modello Simulink dell'attuatore

Data una tensione in ingresso vengono generate, tramite la simulazione, le informazioni sulla corrente assorbita, la velocità del motore, lo spostamento del cavo e la conseguente forza resistente. I tre blocchi principali realizzati modellizzano il motore, la trasmissione, costituita dalle ruote dentate e dall'accoppiamento vite madrevite e permettono il calcolo delle forze resistenti. Questo blocco include i calcoli dovuti alle varie forze di attrito, alla forza

resistente generata dalla meccanica della scatola cambio e quella della molla presente tra le due slitte.

3.2.1 Modello Simulink del motore

Una parte centrale del progetto è il modello Simulink realizzato per il motore. Il tipo di motore presentato in sottosezione 1.4.1 è con spazzole a magneti permanenti, può essere descritto facilmente da un set di equazioni, schematizzate in Figura 3.2. Tale blocco richiede in ingresso il valore di tensione e la coppia resistente da vincere e fornisce i valori di corrente assorbita, coppia motrice generata e velocità angolare del motore. Queste valutazioni vengono effettuate istante per istante durante le simulazioni.

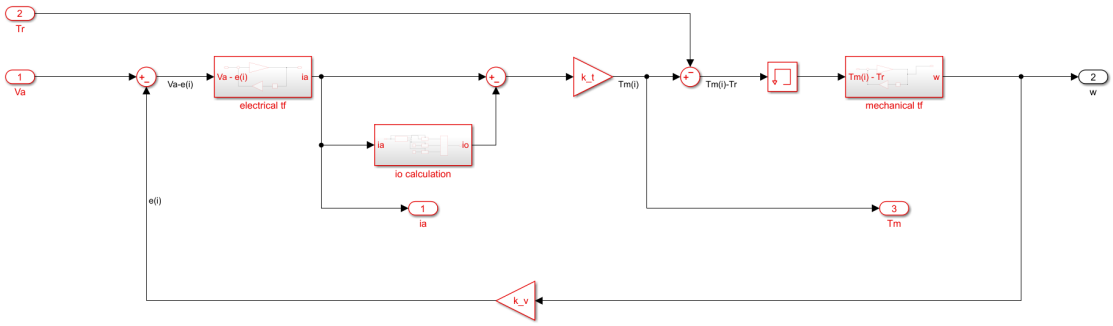


Figura 3.2: Modello Simulink del motore

3.3 Modello completo

Creato il modello dell'attuatore si è passati allo sviluppo di una legge di controllo per pilotare il motore e di conseguenza il cavo meccanico. È stato introdotto un controllo ad anello chiuso con un feedback sulla posizione in tempo reale del cavo. Lo schema realizzato in Simulink è mostrato in Figura 3.3.

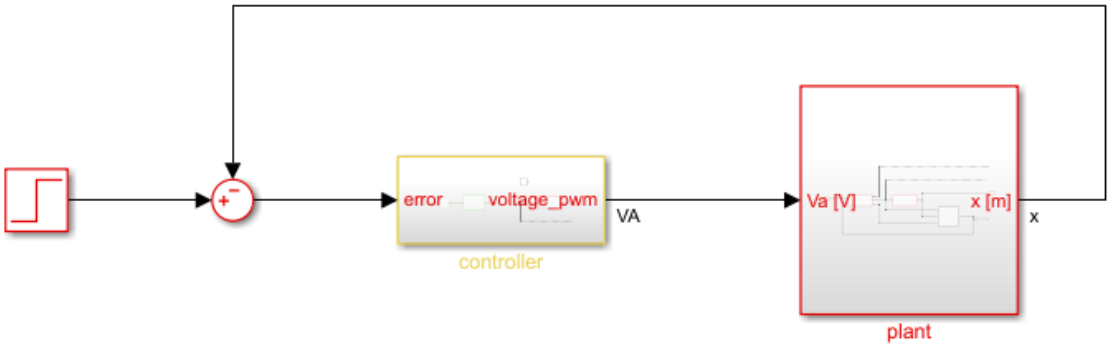


Figura 3.3: Modello Simulink dell'intero sistema

Si possono evidenziare in tale schema tutte le caratteristiche di un sistema ad anello chiuso. In ingresso è presente un segnale di riferimento, chiamato target, nel caso in questione rappresenta la posizione finale che deve essere raggiunta dal cavo meccanico. A questo segnale viene sottratto quello della posizione in tempo reale calcolata nel blocco plant, presente in Figura 3.1. Il segnale risultante viene chiamato segnale errore ed è l'input per il controllore. Questo fondamentale blocco si occupa di valutare una particolare legge di controllo a partire dal segnale in ingresso e fornisce in uscita un valore di tensione che andrà direttamente al motore elettrico.

3.4 Il controllore

L'obiettivo principale dello studio e della modellizzazione in ambiente Simulink è finalizzata alla progettazione efficace di un controllore. In applicazioni in cui è necessario avere un'elevata precisione nel raggiungimento di un target, come in questo caso di posizione, è strettamente necessario prevedere l'inserimento nel sistema di un controllore. Tra le varie tecniche di controllo è stata preferita quella PID (Proporzionale Integrativa Derivativa). L'utilizzo di questa tecnica richiede un adattamento per essere introdotta nel prototipo reale, ossia l'introduzione di un driver che possa tradurre l'informazione riguardante un valore analogico a cui alimentare il motore in un valore PWM

(Pulse Width Modulation), facilmente realizzabile via hardware. Lo schema di massima è presentato in Figura 3.4.

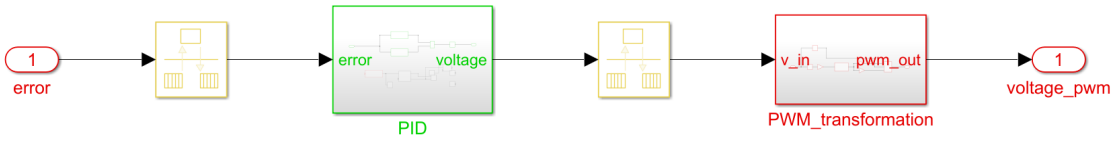


Figura 3.4: Modello del controllore

Il controllore è il punto cardine di questo lavoro di tesi, perché il blocco Simulink con la legge di controllo è la parte fondamentale del firmware che è stato realizzato. Il secondo blocco presente, chiamato *PWM_transformation* è la rappresentazione tramite equazioni di quello che è il vero è proprio driver hardware che è stato aggiunto all'attuatore per permettere il corretto pilotaggio del motore. Gli altri blocchi in giallo servono per poter effettuare delle simulazioni nel tempo discreto, fondamentali per prevedere il comportamento del microcontrollore.

3.5 La tecnica di controllo automatico PID

Il mondo dei controlli automatici negli ultimi tempi ha fatto passi da gigante, soprattutto nel campo automotive. Basti pensare agli sviluppi per quanto riguarda i sistemi di controllo di velocità, trazione, frenata; oppure i diversi sistemi che stanno portando verso la presenza sempre più marcata nel mercato automotive di veicoli con elevati gradi di guida autonoma. Una delle tecniche di controllo più utilizzate, per l'efficacia che ha e per la semplicità di sviluppo e realizzazione è quella del PID. Questa sigla sta per Proportional Integrative Derivative, che rappresentano i tre tipi di controlli inclusi in questa metodologia.
Consiste nella valutazione di una specifica equazione, a partire da un segnale di in ingresso, per fornire un comando per un attuatore in uscita. L'utente fornisce un proprio obiettivo da raggiungere, che può essere una velocità

target, una temperatura o una posizione target nel caso trattato. Lo scopo della legge di controllo è quello di raggiungere questo obiettivo rispettando alcune specifiche che possono essere tempo di salita, sovraelongazione e oscillazione massima a regime. Degli esempi di risposte al comando di ingresso, spesso chiamato impulso, sono rappresentate in Figura 3.5.

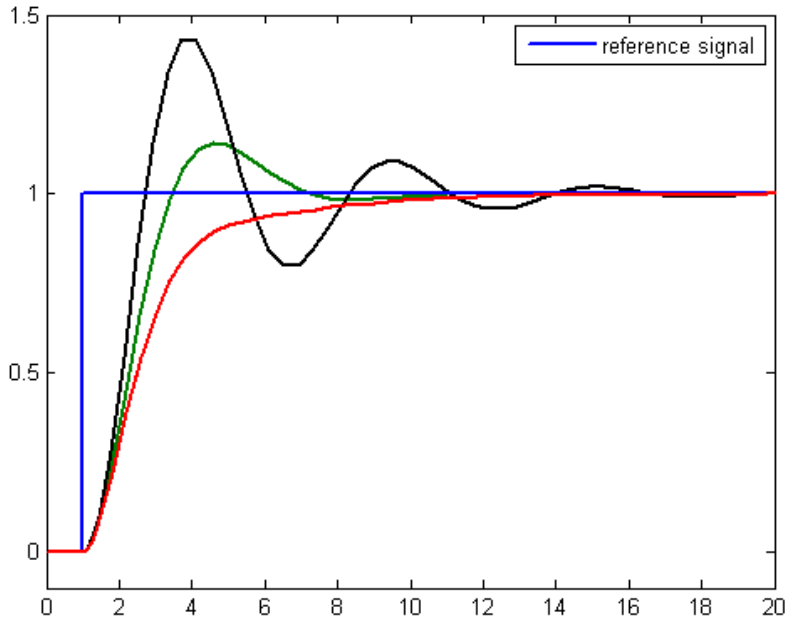


Figura 3.5: Diversi tipi di risposta all'impulso

Il segnale in blu, *reference signal*, è l'impulso fornito dall'utente. Si nota come la risposta in rosso abbia un lungo tempo di salita ma senza sovraelongazione, la risposta in verde presenti una piccola sovraelongazione ma è più veloce mentre la risposta in nero presenta una grossa sovraelongazione e anche un'oscillazione che si prolunga nel tempo.
Il sistema di controllo, per poter decidere in automatico che comando dare all'impianto da pilotare, non ha solo bisogno del comando dell'utente ma anche del feedback da parte dell'impianto sullo stato attuale della specifica variabile da controllare. Questo aspetto si nota molto bene nella rappresentazione del modello completo Simulink mostrato in Figura 3.3.

3.5.1 L'equazione nel dominio del tempo continuo

L'equazione che regola il controllore PID è data dalla somma di tre contributi: proporzionale, integrativo e derivativo. Bisogna definire il segnale di ingresso, errore $e(t)$, come la differenza tra il comando $c(t)$ e il feedback $s(t)$. Di conseguenza:

$$e(t) = c(t) - s(t)$$

La legge di controllo nel dominio del tempo è così definita:

$$u(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(t)dt + K_d \cdot \frac{de(t)}{dt} \tag{3.1}$$

Il primo contributo, quello proporzionale, è semplicemente la moltiplicazione di un coefficiente per l'errore all'istante t. Il secondo contributo, quello integrativo, dipende dall'integrale fino all'istante t del segnale errore. Infine il terzo contributo, quello derivativo, dipende dalla derivata dell'errore. Il design della legge di controllo consiste nella determinazione dei coefficienti K_p, K_i e K_d che permettano di rispettare i criteri di tempo di salita, sovraelongazione e oscillazione a regime per raggiungere le prestazioni volute. Per scegliere tali valori ci sono diverse tecniche che spesso richiedono delle prove reiterate, per questo motivo diventa fondamentale avere degli strumenti di simulazione che abbattono i tempi e i costi di progettazione.

3.5.2 L'equazione nel dominio di Laplace

La legge di controllo può essere anche studiata nel dominio della frequenza. Effettuando la trasformata di Laplace di Equazione 3.1, si ottiene la funzione di trasferimento del controllore:

$$\begin{aligned} H(s) &= K_p + K_i\frac{1}{s} + K_d s \\ &= \frac{K_d s^2 + K_p s + K_i}{s} \end{aligned}$$

Esaminando l'equazione nel dominio della frequenza si nota come siano presenti due zeri e un polo nell'origine; la posizione dei poli e degli zeri è data dalla scelta dei tre coefficienti. Lo studio in questo dominio è utile per effettuare valutazioni sulla stabilità del sistema e la risposta in frequenza.

3.5.3 L'equazione nel dominio del tempo discreto

Le precedenti analisi sono molto utili in fase di progetto del sistema di controllo; successivamente c'è necessità di tradurre l'equazione in codice C compilabile per poter utilizzare una piattaforma hardware. Inoltre il mondo digitale è legato ad una frequenza fissa di funzionamento, dipendente dalla frequenza di clock del microcontrollore su cui verrà caricato il codice; di conseguenza bisogna sostituire le equazioni studiate nel tempo continuo con delle altre legate al campionamento del sistema hardware. Una forma dell'equazione più comoda che aiuta la stesura del codice è quella al tempo discreto. Al posto della variabile indipendente t, che rappresenta il tempo, si introduce la variabile n che conta il numero di campioni che vengono valutati. L'equazione al tempo discreto risulta quindi essere:

$$u[n] = K_p \cdot e[n] + K_i \cdot \sum_{0}^{n} e[k]\Delta t + K_d \cdot \frac{e[n] - e[n-1]}{\Delta t} \tag{3.2}$$

Si nota come l'integrale definito si trasformi in una sommatoria; questo implica il dover prevedere una memoria per salvare tutti i valori istante per istante della variabile errore e un hardware in grado di effettuare in maniera rapida svariate somme. Un'altra osservazione che si può fare è sulla derivata che si trasforma in un rapporto incrementale. Particolare attenzione bisogna

fare all'intervallo di tempo Δt, legato alla frequenza di campionamento del sistema, in quanto la sua durata determina la precisione nella valutazione della legge di controllo.
Per l'applicazione in questione non è stato necessario prevedere un tipo di controllo con il contributo derivativo ma è stato sufficiente avere un controllore proporzionale e integrativo (PI).

Capitolo 4

Hardware

Una parte fondamentale del lavoro di tesi svolto, consiste nella progettazione di un'architettura hardware adeguata per assecondare le esigenze dettate dal tipo di sistema di controllo e poter adattarlo all'attuatore già presente. Una rappresentazione di massima dell'architettura hardware è presente in Figura 4.1.

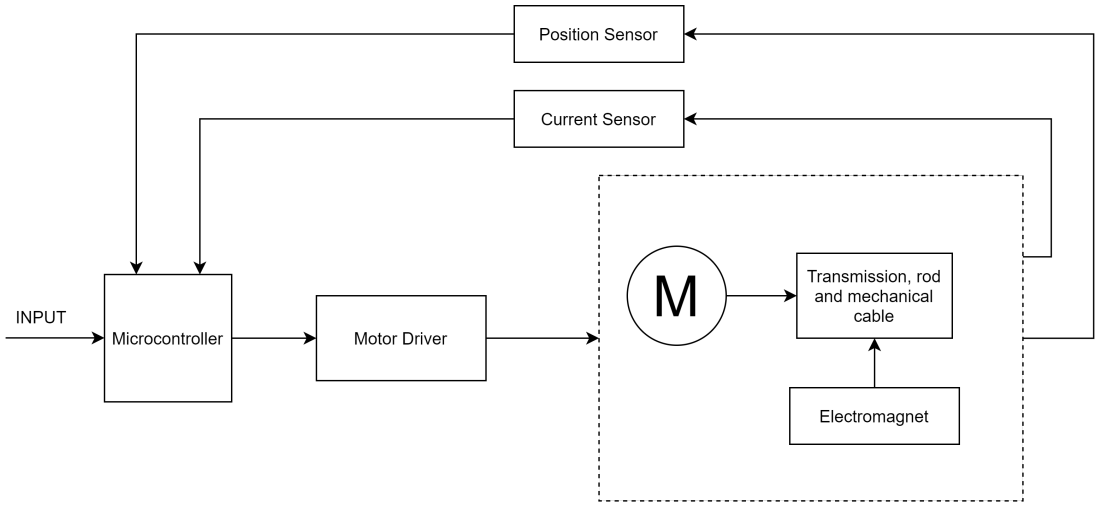


Figura 4.1: Vista di sistema dell'hardware

La progettazione ha incluso la scelta di adeguati componenti e la realizzazione di piccoli driver prototipali. Punto di partenza della scelta è stata la

conoscenza approfondita del prototipo già esistente e delle performance che bisognava raggiungere. In particolare per la scelta del sensore di posizione sono state valutati non solo i parametri di accuratezza ma anche eventuali ingombri meccanici. Per il driver necessario a pilotare il motore è stato valutato il tipo di attuazione necessaria per l'applicazione, i livelli massimi di tensione e corrente necessari al corretto funzionamento e il tipo di input che poteva essere fornito. Per il microcontrollore sono state fatte diverse valutazioni: in primis che il processore avesse una frequenza di funzionamento adeguata, che ci fossero tutte le periferiche necessarie e infine la possibilità di poter effettuare la programmazione e il debug in maniera comoda con degli strumenti di facile utilizzo. Infine per il sensore di corrente si è verificato che fosse compatibile con i livelli massimi di corrente da misurare.

4.1 Microcontrollore

Come già anticipato la scelta del microcontrollore è centrale nello sviluppo di questo progetto. Nel mercato odierno sono presenti tantissime soluzioni di questo tipo dalle più semplici alle più complesse, che spesso complicano la scelta della piattaforma adeguata. Inoltre nel mondo automotive il prezzo del prodotto finale ne fa da padrona, di conseguenza diventa fondamentale effettuare un trade-off tra caratteristiche necessarie e opzionali che deve avere il microcontrollore e costo finale.

4.1.1 Caratteristiche richieste

Dopo uno studio accurato delle specifiche di progetto e un confronto con la collega meccatronica sono state stilate una serie di caratteristiche che sono ritenute la base per la ricerca sul mercato di un microcontrollore.

Frequenza di clock

La frequenza di clock è dettata dalla velocità che deve avere il microprocessore nell'elaborare i segnali di ingresso e dal tipo di comando che deve essere fornito al motore. Per pilotare il motore si utilizza la tecnica del PWM (Pulse Width Modulation) che consiste nell'utilizzo di un'onda quadra a frequenza fissa e duty cycle variabile. La scelta sulla frequenza di funzionamento del PWM è ricaduta nel range $5\,\text{kHz} \backsim 40\,\text{kHz}$. Inoltre un altro parametro discriminante è la frequenza con cui deve essere valutata la legge di controllo, ossia l'inverso del valore Δt spiegato nella sottosezione 3.5.3. In questo caso la scelta è ricaduta nel range $5\,\text{kHz} \backsim 20\,\text{kHz}$. La necessità di dover generare queste frequenze prevede che il periodo di clock sia nettamente inferiore poiché oltre alla generazione di un'onda quadra e la valutazione di un'equazione, devono essere svolte anche altre operazioni dal microcontrollore. Il requisito sulla frequenza di clock quindi è stato fissato a tre ordini di grandezza superiore:

$$f_{ck} \geq 40\,\text{MHz}$$

Timer

La necessità di dover generare un'onda quadra e di avere un timing preciso per le attività di valutazione della legge di controllo danno luogo ad un'altra caratteristica che il microcontrollore deve possedere. La periferica che si occupa della generazione di segnali a frequenze specifiche e di precisi intervalli di tempo è chiamata timer; di solito in tutte le piattaforme embedded ne è presente almeno uno. Per l'applicazione in questione è necessaria la presenza di almeno due timer separati, uno per la generazione dell'onda quadra e l'altro per la generazione dell'intervallo di tempo Δt.

ADC

Poiché nel sistema in anello chiuso c'è un sensore di feedback, si richiede la presenza di una periferica in grado di acquisire i segnali provenienti da tale dispositivo. Generalmente i sensori più semplici presenti in commercio generano un valore di tensione analogica proporzionale alla grandezza misurata; questo valore per poter essere memorizzato in memoria centrale deve essere trasformato in un valore digitale. La periferica che si occupa di questo compito è il convertitore analogico digitale (ADC). Il microcontrollore da scegliere dovrà necessariamente disporre di questa periferica, inoltre poiché potrebbe essere interessante acquisire altri dati non necessari ma utili ai fini del testing, il convertitore dovrà presentare più di un canale di acquisizione dati.

GPIO

Il sistema da realizzare prevede il collegamento fisico di sensori, attuatore e device di comando al microcontrollore di conseguenza un altro fattore fondamentale nella scelta è la presenza di porte di input e output. Il sistema da progettare ha molteplici input, sia digitali, i comandi utente, sia analogici, i segnali dei sensori ma anche di output, basti pensare ai segnali PWM da mandare all'attuatore o eventuali feedback da mandare all'utente. Per questo motivo è fondamentale accertarsi della presenza di sufficienti periferiche GPIO.

Connectivity

I sistemi di comunicazione non sono un requisito fondamentale che deve possedere il microcontrollore, nonostante ciò la presenza di periferiche ad hoc come quelle per la comunicazione seriale, I2C o SPI permettono di avere più libertà nella scelta di protocolli di comunicazione con l'utente, con il driver di attuazione e con i sensori. Anche per il caricamento del

firmware sul microcontrollore diventa indispensabile avere un sistema di comunicazione facile e intuitivo con il PC che permetta agevolmente di effettuare caricamento e debug del firmware. Infine durante la fase di testing è fondamentale prevedere un forma di comunicazione tra microcontrollore e PC per poter raccogliere i dati in real time e poterli paragonare con i risultati attesi dalle simulazioni.

Interrupt

La progettazione di firmware per applicazioni real time, come quella in questione, richiede particolare attenzione alla presenza di particolari caratteristiche software. Una di questa è la possibilità di poter progettare dei codici nei quali sia possibile inserire delle routine che devono essere svolte con cadenza periodica o a seguito dell'arrivo di particolari eventi esterni. Ad esempio l'interazione con l'utente di solito è asincrona e imprevedibile, l'invio di un comando al microcontrollore non può essere previsto con precisione. Tutte queste azioni, a seguito di un'opportuna programmazione, possono interrompere il normale funzionamento del programma ed eseguire la routine di codice predisposta. Questa caratteristica è chiamata interrupt e necessita di particolari librerie software e un hardware dedicato che gestisca le varie richieste con un sistema di priorità, specificato dal programmatore, il NVIC ossia Nested Vectored Interrupt Controller.

DMA

Un'altra caratteristica che può essere comoda e snellire la scrittura del codice è la presenza di un sistema DMA, ossia Direct Memory Access. Quando viene acquisito un dato dal convertitore A/D può essere scatenato un interrupt, spesso però è troppo dispendioso dal punto di vista computazionale e può rallentare anche di molto il sistema; basti pensare che l'overhead per la gestione di un interrupt si aggira intorno alle decine di colpi di clock e che la frequenza di acquisizione dei dati è molto elevata. Una soluzione consiste

nel programmare l'hardware in modo tale da permettere in automatico la memorizzazione del valore acquisito nella memoria centrale. Quando poi sarà necessario il dato del sensore si potrà accedere comodamente alla porzione di memoria e leggere i valori acquisiti.

Tool di sviluppo

Un'ultima caratteristica da tenere in considerazione nella scelta del microcontrollore è quella riguardante gli strumenti di sviluppo che permettono la programmazione del firmware. Gli step necessari che si compiono dall'idea di firmware al caricamento sulla piattaforma target sono diversi. In una prima fase di solito si determinano quali siano le periferiche che devono essere utilizzate e si crea un file di configurazione e abilitazione per le stesse; per alcuni tipi di microcontrollori esistono dei software che permettono di programmare l'hardware per via grafica e generano il file di configurazione in linguaggio C in maniera automatica. Successivamente si passa alla fase di sviluppo vero e proprio del codice; anche qui per la stesura del codice embedded esistono delle librerie dedicate allo specifico microcontrollore che facilitano la scrittura del programma. Spesso questi software permettono la compilazione, la creazione del file eseguibile e il caricamento tramite porta USB sul microcontrollore target in maniera molto semplice. Nella fase finale della scrittura del firmware è spesso necessario trovare gli errori nel codice e correggerli; per questo motivo la presenza di un software che permetta il debug riga per riga del codice diventa fondamentale per lo sviluppo del progetto. Si evince quindi che l'esistenza o meno dei tool software e delle librerie dedicate sono fondamentali per procedere in maniera agevole nella parte di sviluppo firmware dell'applicazione.

4.1.2 Il microcontrollore scelto

A seguito delle valutazioni fatte nella sottosezione 4.1.1 sono stati interpellati i principali produttori di microcontrollori sul mercato. La scelta finale è

ricaduta sul microcontrollore dell'azienda*ST Microelectronics*. In particolare è stato scelto il modello *STM32F411RE* che presenta le seguenti caratteristiche principali:

- processore ARM Cortex serie M4
- Frequenza di clock $f_{ck} = 100\,\text{MHz}$.
- Tre timer a 16 bit e un timer a 32 bit, tutti con 4 canali separati; queste periferiche danno la possibilità di generare in un automatico onde quadre a duty cycle variabile, oppure impulsi a frequenze predefinite.
- Un convertitore analogico digitale con risoluzione 12 bit a 12 canali con la possibilità di essere configurato con la memorizzazione automatica dei valori.
- Svariati pin di comunicazione GPIO sia analogici che digitali per poter acquisire dati dai sensori e generare o leggere segnali digitali. Una rappresentazione grafica del microcontrollore è mostrata in Figura 4.2.
- Una periferica SPI, una I2C e due UART. Di queste ultime due, una é disponibile per l'utente, l'altra è collegata fisicamente ad una porta USB. Tramite tale porta è possibile scambiare informazioni con un personal computer.
- Possibilità di poter programmare diversi interrupt, da quelli esterni, che attendono un evento asincrono, a quelli software e quelli a seguito di una lettura del convertitore. La priorità dei vari interrupt è gestibile tramite l'opportuna programmazione del NVIC.
- È presente una periferica DMA per effettuare trasferimenti di dati in memoria senza interrompere il normale funzionamento del processore.
- I tool di sviluppo realizzati ad hoc dall'azienda produttrice sono molteplici. Per l'abilitazione delle periferiche è presente un software di

sviluppo grafico, *STM32CubeMX*, che da la possibilità di attivare le eventuali periferiche che si intende utilizzare nella stesura del codice. Le impostazioni vengono poi tradotte in codice C sfruttando delle librerie apposite, chiamate HAL (Hardware Abstraction Layer). Viene così creato un progetto con un template nel quale è possibile inserire il proprio codice. Il progetto generato da *STM32CubeMX*, può essere aperto tramite un altro tool che permette la scrittura del codice, *System Workbench for STM32*. È una piattaforma basata sull'editor *Eclipse* nella quale è possibile creare il proprio firmware a partire da un template o da un progetto vuoto; successivamente permette di sincronizzarsi con il microcontrollore target tramite porta USB in modo da poter caricare il codice ed effettuare comodamente il debug.
Queste ultime due funzionalità sono supportate grazie alla presenza sulla scheda di sviluppo, di un secondo processore dedicato, chiamato debugger, che facilita di molto le operazioni di comunicazione e debug con i software per PC tramite porta USB; il debugger è visibile nella parte superiore di Figura 4.2.

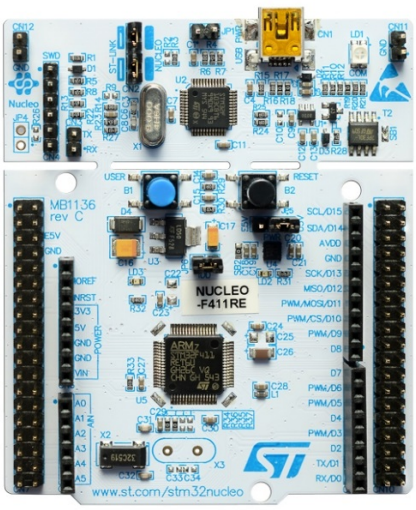


Figura 4.2: Microcontrollore *STM32F411RE*

4.2 Sensing

Una seconda parte fondamentale nello sviluppo dell'hardware riguarda la scelta dei sensori. Il sensore che ha un ruolo centrale nello sviluppo del progetto è quello che fornisce il feedback per il sistema di controllo. Come già anticipato e mostrato in Figura 4.1, il sensore presente in retroazione è quello di posizione. Inoltre ai fini della validazione del modello, del testing, e della sicurezza nell'effettuare le prove diventa fondamentale monitorare la corrente elettrica assorbita dal motore. Essendoci delle forze ingenti in gioco, è molto semplice finire in situazioni in cui il motore assorbe molta corrente, il che può essere pericoloso per la nostra incolumità e per la vita del motore stesso.

4.2.1 Sensore di posizione

I sensori di posizione sono dispositivi destinati a determinare la posizione lineare o angolare di un oggetto e a convertire tale informazione in un segnale che viene trasmesso al microcontrollore tramite un sistema di comunicazione. La misurazione della posizione è la più frequente in ambito industriale dopo quella della temperatura. I sensori di posizione, di conseguenza, trovano impiego in un'ampia gamma di applicazioni industriali e commerciali. Di conseguenza esistono in commercio diverse tipologie di sensori con svariate caratteristiche. Per la scelta di un sensore di posizione bisogno tenere conto di diversi fattori:

- Lineare o rotativo
- Con o senza contatto
- Caratteristiche meccaniche per l'installazione
- Range di misura e precisione
- Tipo di segnale di uscita

Le valutazioni effettuate

Le esigenze dettate dal tipo di applicazione da realizzare indirizzano su un sensore che sia lineare, poiché la grandezza da misurare è lo spostamento del cavo meccanico che si muove nel binario. La valutazione delle corse in gioco richiede la misura di un range di almeno 20 mm di corsa. Il tipo di segnale di uscita deve essere compatibile con uno dei tipi di comunicazione che il microcontrollore accetta: un segnale analogico oppure tramite i vari protocolli di comunicazione decritti nella sottosezione 4.1.2. La scelta invece di contatto o meno tra il sensore e l'oggetto di cui misurare la posizione è stata effettuata facendo un trade off tra la difficoltà di utilizzo sia dal punto di vista firmware che dell'installazione e il costo. Solitamente i sensori di posizione contactless sono basati sull'effetto Hall o altri fenomeni magnetici; questo li rende robusti e precisi, ma con costi elevati e specifiche condizioni di installazione da rispettare. I sensori non contactless invece sono solitamente dei potenziometri, quindi delle resistenza variabili; questo li rende a volte meno precisi e più sensibili all'usura ma ad un costo più contenuto e di facile utilizzo. Dopo un confronto con il collega ingegnere dell'autoveicolo è stata individuata un area nel prototipo nel quale fosse efficace l'installazione del sensore. Come si può vedere nella Figura 4.3, la zona adiacente alla slitta che sostiene il cavo è quella ottimale per effettuare una misura diretta della posizione del cavo stesso.

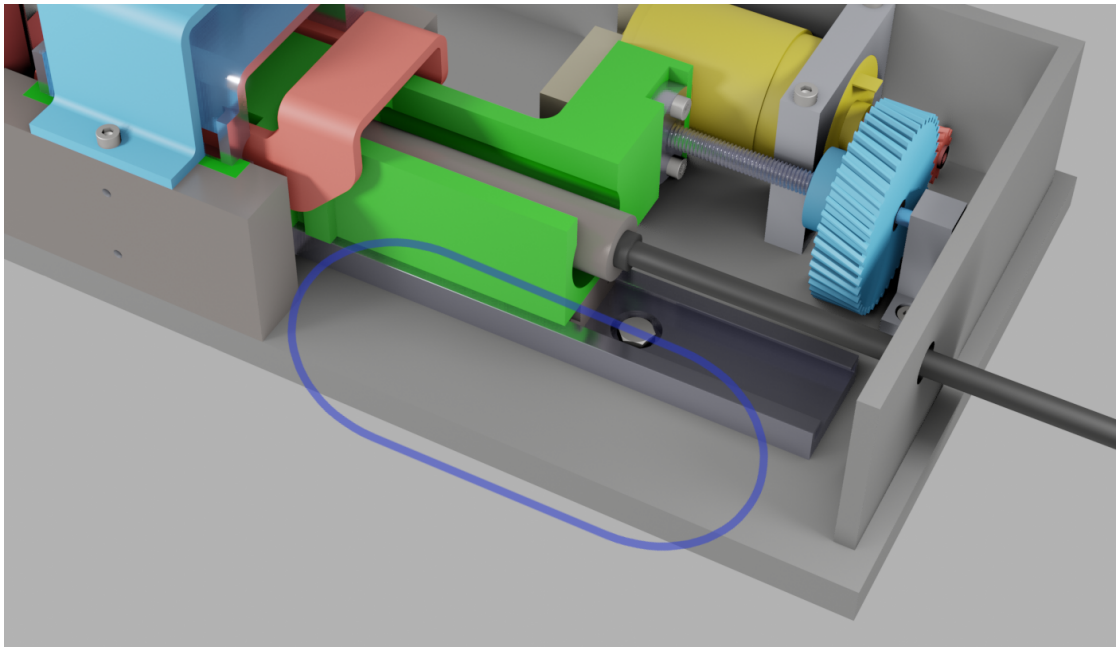


Figura 4.3: Area individuata per l'installazione del sensore di posizione

Il sensore di posizione scelto

Dopo aver effettuato le valutazioni opportune sulle varie tipologie di sensori di posizione, la scelta è ricaduta su un sensore resistivo, un potenziometro lineare con una corsa di 30 mm. Il segnale generato da questo sensore è puramente analogico e può essere letto comodamente tramite una periferica analogica di I/O e tradotta dal convertitore analogico digitale. Per un applicazione automotive, dove l'affidabilità e la sicurezza la fanno da padrone, un sensore contactless sarebbe stato più adeguato ma vista la natura prototipale del progetto si è optato comunque per un potenziometro.
Il sensore scelto è dell'azienda *Bourns*, una rappresentazione è mostrata in Figura 4.4.

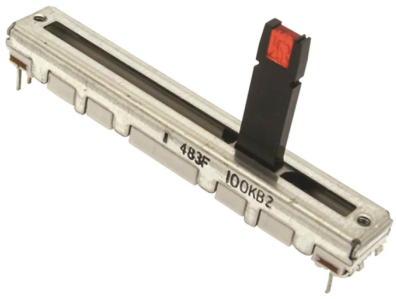


Figura 4.4: Potenziometro

L'incertezza di misura si aggira intorno al 20% e il valore massimo di resistenza è di 10 kΩ.

Il circuito elettronico per l'utilizzo

Il potenziometro viene alimentato ad una tensione nota tramite due pin e viene letto il valore di tensione proporzionale alla posizione tramite un terzo pin. Il circuito utilizzato è mostrato in Figura 4.5. È stato utilizzato il riferimento di tensione interna a 3.3 V e quello di *GND* per l'alimentazione; il valore letto è configurato come ingresso del ADC.

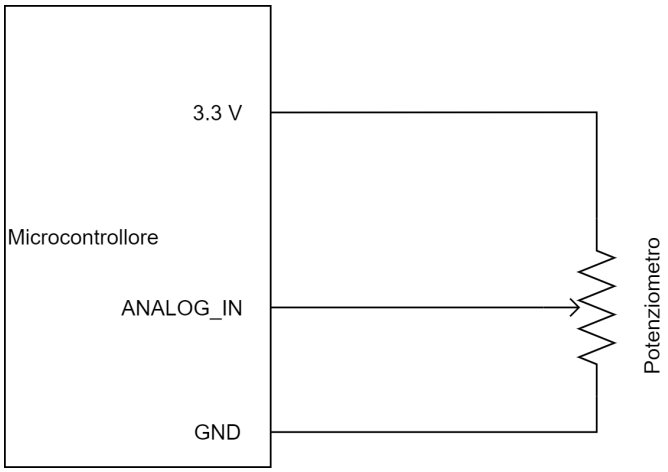


Figura 4.5: Il circuito utilizzato per il potenziometro

L'installazione nel prototipo

Per l'installazione fisica del potenziometro c'è stato un attento confronto con i colleghi del reparto di ricerca e sviluppo con i quali abbiamo cercato la migliore soluzione. Alla fine è stato progettato da loro un piccolo supporto al CAD che è stato successivamente stampato in 3D sfruttando le stampanti presenti in azienda. I componenti stampanti sono rappresentati in blu nella Figura 4.6.

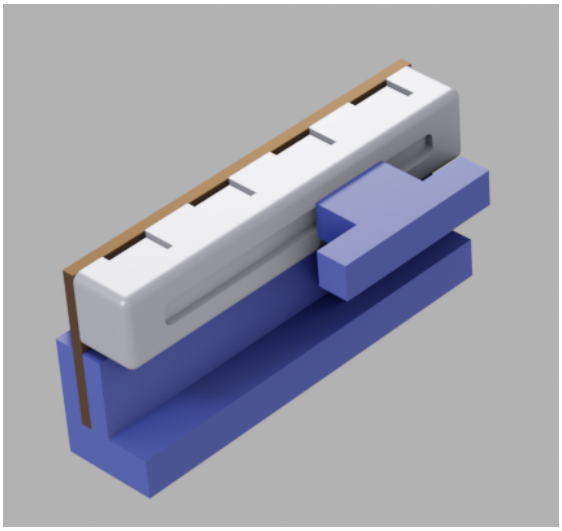


Figura 4.6: Il supporto realizzato

La base del sostegno è stata avvitata sulla base dell'attuatore nell'area indicata e la parte mobile è stata avvitata alla slitta a cui è collegato il cavo e la madrevite. Il risultato finale è visibile in Figura 4.7.

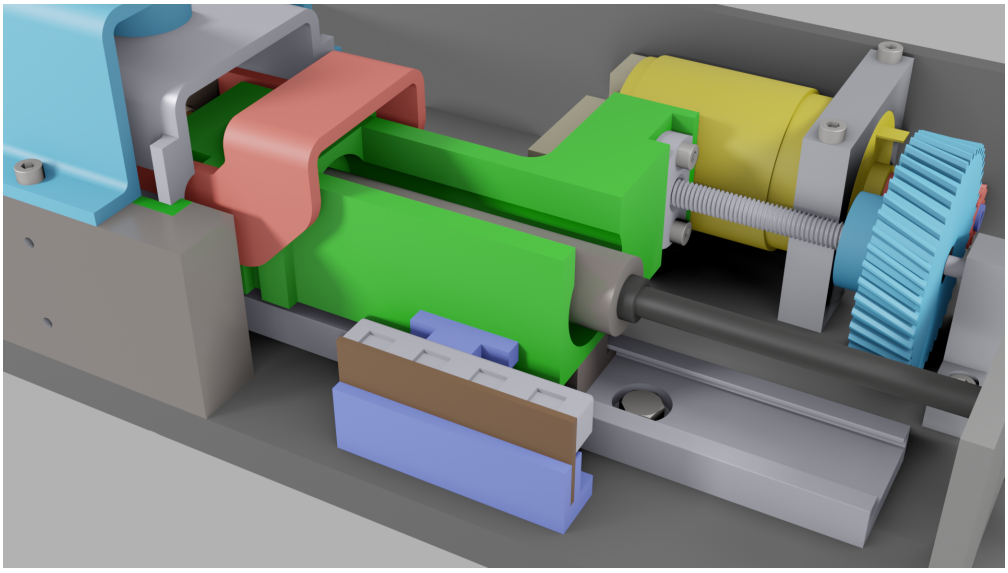


Figura 4.7: L'installazione del potenziometro

4.2.2 Sensore di corrente

I sensori di corrente sono dispositivi destinati a determinare l'intensità di corrente che scorre in un cavo elettrico e a convertire tale informazione in un segnale che viene trasmesso al microcontrollore. La misura della corrente può essere effettuata in due modi differenti: per contatto diretto con la corrente, come ad esempio un amperometro, nel quale la corrente passa da una resistenza di shunt della quale ne viene letta la tensione; oppure misurando il campo elettromagnetico. La misura della corrente in questo caso viene fatta sfruttando la legge di Biot-Savart che regola il campo magnetico generato da un filo percorso da corrente:

$$B = \frac{\mu_0 I}{2\pi r}$$

Si nota come ci sia una diretta proporzionalità tra campo magnetico e corrente. Il campo elettromagnetico viene misurato da particolari strumenti che sfruttano l'effetto Hall.

Le valutazioni effettuate

Nel caso in questione, ai fini del corretto funzionamento del sistema di controllo, non esiste una reale necessità nell'utilizzo di un sensore di corrente. Nonostante ciò, come è stato evidenziato in sottosezione 3.2.1, una variabile di sistema del modello del motore è proprio la corrente che scorre nella armature. La presenza quindi di una misura sperimentale della stessa fornisce delle indicazioni sulla correttezza del modello Simulink creato. Dalle simulazioni è emerso che nelle condizioni peggiori la corrente assorbita dal motore può arrivare fino a 15 A e che la frequenza di variazione della corrente non è molto elevata.

Il sensore di corrente scelto

L'unica specifica restrittiva che ha guidato la scelta del sensore di corrente è quella sul valore massimo di corrente da misurare. È stato scelto un sensore di corrente che sfrutta l'effetto Hall e che ha fondoscala 20 A; l'errore di misura dichiarato è ±1.5%. Il modello in questione è il *ACS712ELC-20A* dell'azienda *Allegro MicroSystems*. È stata acquistata una versione installata su una piccola board che permette il collegamento in maniera agevole al microcontrollore. Una rappresentazione di tale sensore è mostrata in Figura 5.12.

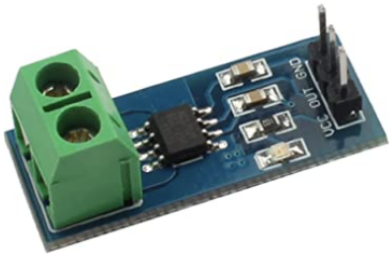


Figura 4.8: Il sensore di corrente

Il circuito elettronico per l'utilizzo

Il circuito elettronico per l'utilizzo del sensore di corrente è simile a quello utilizzato per il potenziometro. Viene sfruttato il riferimento di tensione interno al microcontrollore di 5 V per l'alimentazione. La lettura della corrente avviene interrompendo il collegamento diretto presente tra il driver e il motore; viene quindi inserito il sensore in serie al circuito di pilotaggio. Una rappresentazione schematica è presente in Figura 4.9.

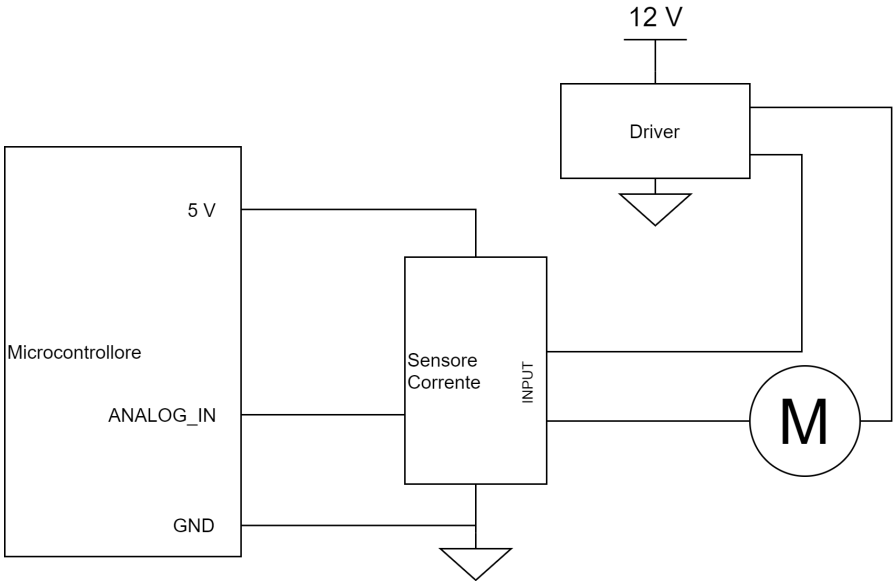


Figura 4.9: Il circuito utilizzato per il sensore di corrente

4.3 Attuazione

Anello fondamentale del sistema di controllo oltre alla parte di sensing è quella di attuazione. Per diversi motivi non è possibile inviare direttamente il comando generato dal microcontrollore al motore. Come spiegato nella sezione 3.5, il segnale di uscita del controllore PID è un comando per il motore. Nello specifico, regolare la posizione del cavo corrisponde indirettamente a regolare la posizione angolare e il verso di rotazione del motore. Con un motore in corrente continua l'unico modo per poter controllare queste variabili è cambiando la tensione di alimentazione; incrementando la tensione aumenta la velocità di rotazione mentre cambiando il segno si cambia il verso. L'unico riferimento di tensione interno all'autoveicolo è quello a 12 V, da qui bisogna ricavare tutti gli altri valori di tensione necessari. Per variare in maniera dinamica il valore di tensione è necessario l'utilizzo della tecnica PWM.

4.3.1 La tecnica del PWM

Il PWM, Pulse Width Modulation, è una strategia che consiste nell'avere una tensione che sia un'onda quadra la cui frequenza è fissa e il duty cycle variabile. Grazie a quest'onda è possibile codificare il valore voluto di tensione all'interno del duty cycle. Il motore a valle percepirà il valor medio di tale onda, che corrisponderà a quello imposto dal sistema di controllo. Per calcolare il valor medio di un segnale periodico qualsiasi è necessario applicare la seguente formula:

$$V_m = \frac{1}{T} \int_0^T v(t)dt$$

Nel caso di un'onda quadra l'equazione si trasforma in:

$$V_m = \frac{t}{T} \cdot t\, V_{max} = DC \cdot V_{max}$$

La variabile t rappresenta la fase alta dell'onda quadra, mentre T simboleggia il periodo; il rapporto tra questi due intervalli di tempo da il duty cycle. Da questa equazione si nota chiaramente come il valor medio sia direttamente proporzionale la duty cycle dell'onda quadra. Una rappresentazione grafica del PWM è presente nella Figura 4.10. Si nota come all'aumentare del duty cycle aumenti anche il valor medio.

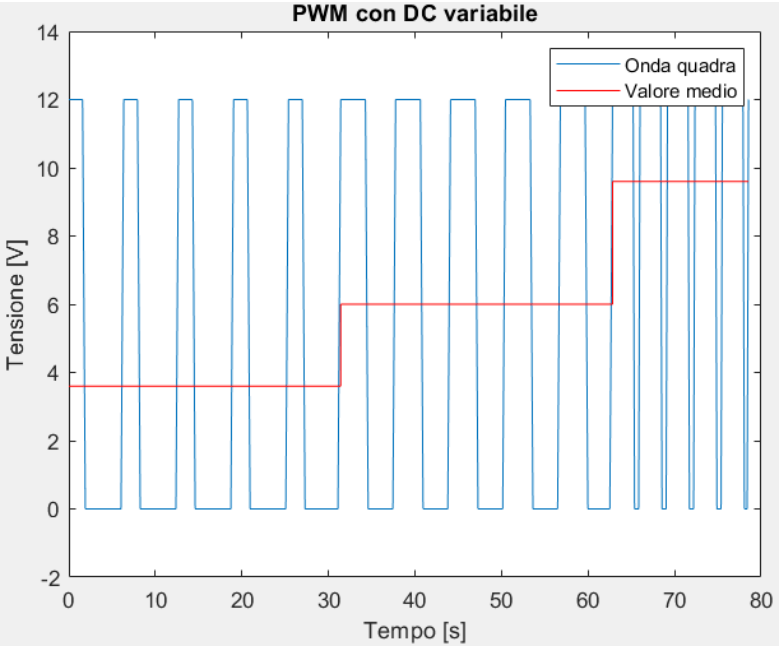


Figura 4.10: Rappresentazione grafica PWM

La necessità quindi di avere un microcontrollore che dia la possibilità di generare un'onda quadra nasce proprio dall'esigenza di pilotare il motore con la tecnica del PWM.
Un parametro che bisogna scegliere però è la frequenza alla quale generare l'onda quadra. Dal punto di vista del microcontrollore non ci sono grosse limitazioni in quanto i timer sono molto performanti. Delle riflessioni sono necessarie invece per quanto riguarda il motore. In primis bisogna sottolineare un comportamento singolare che ha il motore DC, esso funge da amplificatore; questo fa si che se la frequenza utilizzata è troppo bassa si hanno delle

vibrazioni anche molto forti e se è nel campo delle frequenze audio si traduce in un suono, che a tratti può essere fastidioso. Se la frequenza utilizzata è al di fuori del range audio, il motore genera un'onda che può essere percepita da un analizzatore di spettro ma non dall'orecchio umano. Un'altra osservazione può essere fatta sulla corrente: anch'essa seguirà un profilo ad onda quadra, oscillazioni troppo elevate però possono danneggiare il motore; di solito si cerca quindi di aumentare la frequenza del PWM per fare in modo di diminuire sempre di più queste oscillazioni rendendole dei piccoli ripple intorno ad un valore medio. D'altra parte non è consigliato utilizzare una frequenza troppo elevata perché entrano in gioco fattori di secondo ordine che degradano l'efficienza del motore.
A seguito di queste osservazioni è stata scelta una frequenza per il PWM:

$$f_{PWM} = 10\,\text{kHz}$$

4.3.2 H bridge driver

Come appena detto è necessario generare dei valori di tensione tramite PWM nel range $[-12\,\text{V}; 12\,\text{V}]$. Il problema della generazione di un'onda quadra è facilmente risolvibile tramite l'opportuna programmazione della periferica timer e GPIO del microcontrollore. Il segnale di uscita sarà però un'onda che varia tra 0 V e 3.3 V che è il valore di tensione massimo che può essere generato dallo stesso. È evidente come sia necessario non solo amplificare il segnale, ma anche avere la possibilità di invertirlo di segno.
Il driver più usato per pilotare i motori in corrente continua è il ponte ad H. Una prima versione circuitale è presente in Figura 4.11.

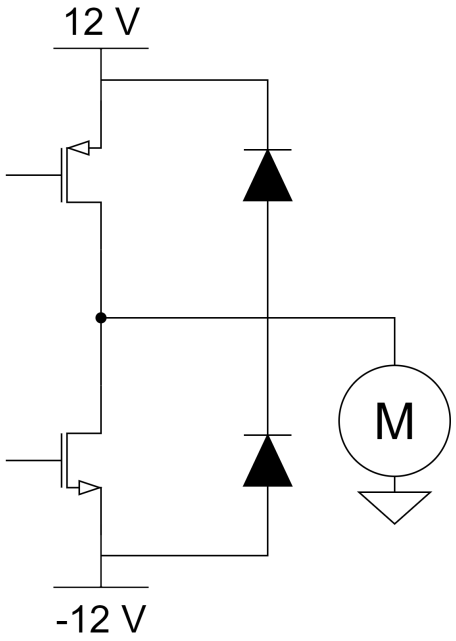


Figura 4.11: Circuito dell'half bridge driver

È costituito da una coppia di transistor, uno a canale N ed uno canale P, che devono essere pilotati da due diversi segnali PWM a seconda che si voglia una tensione positiva o negativa. Lo svantaggio principale che rende inutilizzabile questa tecnica è la necessità di avere due riferimenti di tensione, che nella stragrande maggioranza delle applicazioni è molto difficile da ottenere. I diodi presenti, detti diodi di flyback, sono necessari per avere un circuito che colleghi ai riferimenti di tensione il motore una volta che non vengono più pilotati i transistor. Essendo il motore dal punto di vista circuitale comparabile ad un induttore, quando la tensione ai capi varia genera una corrente indotta, essa nel caso dello spegnimento del motore può danneggiare i transistor in assenza dei diodi.
Un'evoluzione di questo circuito è mostrato in Figura 4.12.

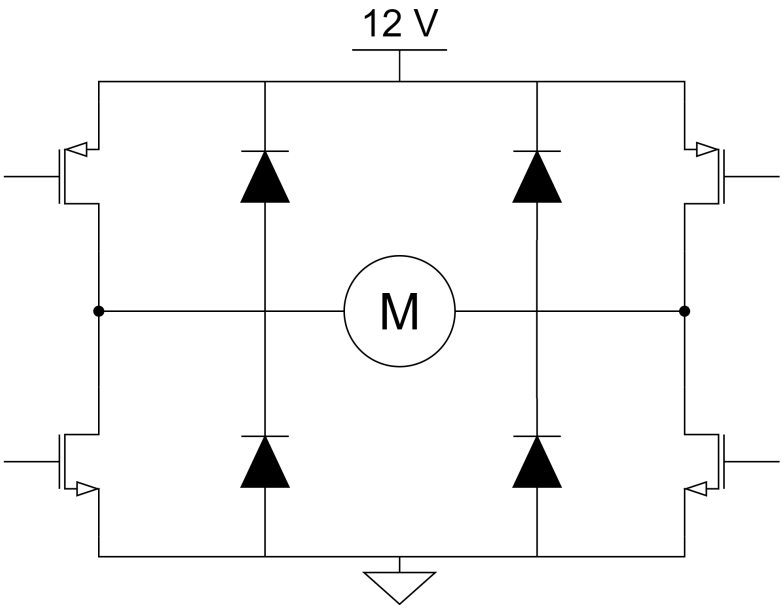


Figura 4.12: Circuito del H bridge driver

In questa configurazione sono presenti due half bridge e il motore e collegato tra i due terminali centrali. Si evita in questo modo di avere due riferimenti di tensione, aspetto fondamentale per la realizzazione. Per avere tutti i valori possibili basta pilotare i transistor in maniera alternata: NMOS di un ramo e PMOS dell'altro ramo. Cambiando le coppie di transistor si ottengono tensioni positive e negative. Anche in questo caso sono sufficienti due onde PWM, al patto che per pilotare il PMOS si utilizzi un inverter.
La scelta del driver per il motore quindi è stata indirizzata su un H bridge. Un altro fattore da considerare per l'acquisto è la corrente massima che deve essere in grado di fornire. Dalle specifiche del motore emerge che la corrente di picco assorbita può arrivare a 20 A; in aggiunta dalle simulazione è emerso che nelle condizioni peggiori di carico il motore dovrebbe assorbire 14 A, valore inferiore a quello massimo. Infine bisogna anche tenere conto del tipo di segnale che il driver deve ricevere dal microcontrollore.

Il driver scelto

Il modello scelto è un modulo contenente due half bridge *BTS 7960* dell'azienda *Infineon Technologies*. La tensione massima di alimentazione che riesce a gestire è nettamente superiore ai 12 V necessari per l'applicazione. Inoltre riesce ad erogare intensità di corrente che possono arrivare sino a 40 A di picco. Per poter attivare tale driver è necessario un comando di enable per i due lati del ponte e i due segnali PWM per pilotare tensioni positive o negative. Un'immagine del modulo utilizzato è presente in Figura 4.13.

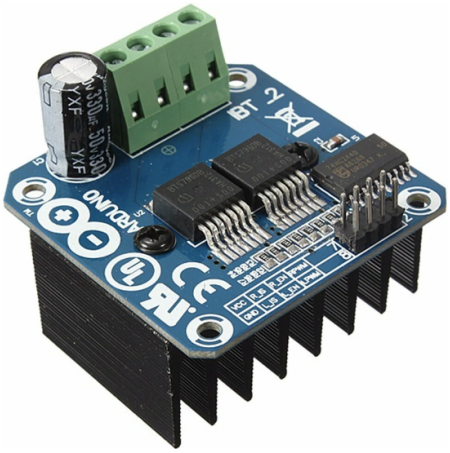


Figura 4.13: Rappresentazione grafica del driver

I pin in basso sono da collegare al microcontrollore, mentre i connettori verdi in alto sono per l'alimentazione e il motore. Sotto la board è presente un dissipatore termico che evita il surriscaldamento dei due chip in presenza di elevate potenze. Infine questo driver è creato per essere automotive compliant, quindi rispetta gli standard di sicurezza necessari per poter commercializzare un prodotto e installarlo nell'autovettura.

Capitolo 5

Firmware

Secondo pilastro della progettazione è stato il design di un firmware per microcontrollore. È la parte centrale del loop di controllo, si occupa come già anticipato di svariate funzioni. Queste caratteristiche devono essere però programmate. Il lavoro di stesura del codice è partito dallo studio di un algoritmo efficiente per la traduzione della legge di controllo. Un altro occhio è stato dato alle diverse condizioni di funzionamento dell'attuatore; l'obiettivo è stato quello di rendere il codice il più versatile possibile. In questo modo cambiando solo pochi parametri o utilizzando delle particolari routine piuttosto che altre è stato possibile testare il prototipo nelle varie situazioni. Come spiegato nella sottosezione 4.1.2, la progettazione del firmware parte dalla configurazione delle periferiche e passa in un secondo momento alla scrittura del software. Per la scrittura del codice sono venute in aiuto le librerie proprietarie che permettono di sfruttare le periferiche. Il lavoro sviluppato è stato organizzato in una libreria costruita appositamente, dove sono contenuti tutti i file sorgente e header che gestiscono separatamente tutte le varie parti.

5.1 L'algoritmo base

Il punto di partenza per la scrittura del codice per il PID è l'Equazione 3.2 dove viene data un'interpretazione nel tempo discreto della legge di controllo. Dal punto di vista software, dopo che il guidatore ha inviato il comando, è necessario in primis identificare la posizione attuale e quella di arrivo. Fatto ciò deve essere valutata la legge di controllo in maniera periodica fino al completamento del movimento, che si traduce nell'avere il segnale errore nullo. La Figura 5.1 mostra il diagramma a blocchi dello schema software realizzato.

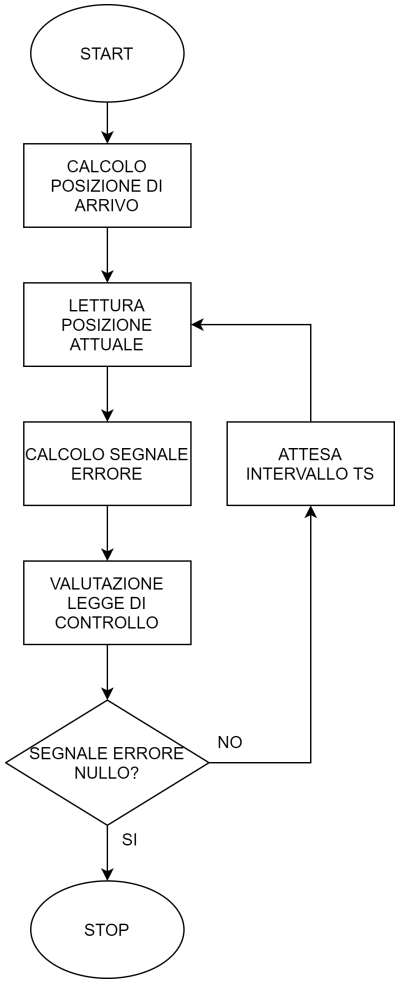


Figura 5.1: Lo schema a blocchi dell'algoritmo

Lo start per la routine viene dato dall'utente nel funzionamento normale; in alcune versioni di firmware progettate per il testing, lo start viene dato in maniera automatica periodicamente. Quando finisce la routine, il sistema è di nuovo pronto per ricevere altri segnali.

5.1.1 Valutazione della legge di controllo

In questa sezione viene descritto e commentato la parte di codice C che è stata scritta per tradurre nello specifico le legge di controllo.

Listato 5.1: Codice per la legge di controllo

```
1  //evaluate PID
2  pos_error = (PID_true_target - pos_measured)/1000; //m
3  ts_PID = (float)ts * TIM4_CLK;
4  //ts PID value control
5  actual_PID_tolerance = ts_PID - 1/(float)PWM_FREQ;
6  if( actual_PID_tolerance <= TS_TOLERANCE && actual_PID_tolerance
       >= -TS_TOLERANCE){
7      integrative += pos_error*ts_PID;
8      v_out = PIDkp*pos_error + PIDki*integrative;
9  }
10 //saturation mechanism
11 if (v_out > V_MAX)
12     v_out = V_MAX;
13 if (v_out < -V_MAX)
14     v_out = -V_MAX;
15 //dc calculation
16 dc = v_out / V_MAX;
```

Le righe di codice nel Listato 5.1, mostrano il core dell'applicazione. Nella riga 3 viene calcolato l'errore di posizione come differenza tra la posizione attuale e quella target. Nelle righe successive sono effettuati dei controlli sull'intervallo di tempo `ts`; per avere dei conti più precisi esso viene calcolato di volta in volta. Alla riga 8 viene calcolato l'integrale dell'errore tramite sommatoria; la variabile `integrative` viene azzerata all'inizio di ogni

movimentazione e ogni volta che viene eseguita la routine viene aggiornata. Alla riga 9 viene valutata la legge di controllo PID tramite la somma della parte integrativa e della parte proporzionale. Infine viene effettuato un controllo di saturazione sul valore della tensione di uscita e viene calcolato il duty cycle che avrà l'onda come rapporto con la tensione massima;

5.1.2 Calcolo dell'intervallo di tempo ts

La routine appena descritta, come si nota nella Figura 5.1, deve essere eseguita ad intervalli di tempo regolari. Una questione che è stata subito affrontata è quella della scelta di questa pausa tra due routine. In linea di massima verrebbe da dire che aumentare la risoluzione nel tempo sia una cosa positiva anche perché ad essa è legata l'acquisizione della posizione dal sensore. Bisogna però ricordare che avere un intervallo `ts` troppo piccolo, al di sotto del periodo PWM diventa difficile da gestire se non deleterio. Inoltre la routine ha un suo tempo di esecuzione e a seguito della valutazione del comando viene cambiato il valore del duty cycle dell'onda quadra. La pausa non può essere quindi più piccola del tempo di esecuzione stesso della routine; questo vincolo tuttavia non è molto stringente. Un fattore invece da tenere bene a mente è il periodo dell'onda quadra: il valore `ts` deve rispettare il seguente criterio:

$$ts \geq T_{PWM}$$

per evitare di cambiare il duty cycle più volte in un periodo. Imposto questo limite quindi si è scelto di avere comunque la risoluzione più alta possibile:

$$ts = T_{PWM}$$

5.1.3 Realizzazione software della pausa ts

Per realizzare con un microcontrollore una pausa, o meglio, per eseguire periodicamente una routine ci sono diverse strade. Una di queste può essere configurare una periferica timer in modalità output compare con abilitazione del relativo interrupt. In questo modo ogni qual volta il timer raggiunge una soglia impostata scatena un interrupt; la service routine consiste appunto nell'algoritmo base della legge di controllo. L'utilizzo però di questo genere di soluzione si è dimostrata non ottimale. Bisogna sottolineare che l'utilizzo della tecnica interrupt ha come svantaggio il fatto di aggiungere un overhead di istruzioni, utili al cambio di contesto, che può arrivare fino a 50 colpi di clock. In un primo momento è stato utilizzata questa soluzione e, essendo `ts` abbastanza piccolo come intervallo, spesso si notavano dei malfunzionamenti per i ritardi introdotti dalla presenza di interrupt. La soluzione finale quindi è stata quella di non adottare una interrupt service routine, ma di eseguire una vera funzione. La pausa tra due esecuzioni successive è stata implementata via software eseguendo una funzione che avesse il solo scopo di aspettare un tempo predefinito. Questo attesa è stata comunque calcolata prendendo come riferimento un timer. Inoltre per essere ancora più precisi, ogni qualvolta viene eseguita la funzione viene calcolato il tempo intercorso tra due esecuzioni successive, in modo da tenere in considerazione anche il tempo di esecuzione della funzione. Il ciclo di funzionamento base del programma è mostrato nel Listato 5.2.

Listato 5.2: Loop principale

```
1 while (1){
2     PID();
3     pause(&htim3, PID_PAUSE);
4 }
```

Come si può notare viene eseguito ciclicamente la funzione *PID()* e quella *pause()*, alla quale bisogna specificare il timer da utilizzare e la pausa da

aspettare. Nel Listato 5.3 invece è mostrata la funzione per creare delle pause predefinite.

Listato 5.3: Funzione per la creazione di pause

```
void pause(TIM_HandleTypeDef *htim, uint32_t microsec){
    for(int i = 0; i < microsec / 50000; i++){
        __HAL_TIM_SET_COUNTER(&htim3, 0);
        while(__HAL_TIM_GET_COUNTER(&htim3) < 50000);
    }
    __HAL_TIM_SET_COUNTER(&htim3, 0);
    while(__HAL_TIM_GET_COUNTER(&htim3) < microsec % 50000);
}
```

Si può notare come, ricevuto il valore in microsecondi della pausa, il codice consista semplicemente in un ciclo *while* senza nessuna azione da svolgere. È presente un ciclo *for* per permettere di creare delle pause più grandi del valore massimo che può contare il timer scelto. La funzione progettata è bloccante, ossia nell'attesa il processore non può svolgere altre attività a meno che non arrivi un interrupt esterno; questo però non è un grosso svantaggio perché una volta che arriva il comando il processore deve solo eseguire le istruzioni necessarie alla movimentazione del motore.

5.2 La periferiche utilizzate

Prima di proseguire con la descrizione delle diverse funzionalità implementate nel codice è giusto fare un passo indietro al punto di partenza della scrittura del codice. Come annunciato nella sottosezione 4.1.2 prima di scrivere le varie funzioni software è necessario abilitare e configurare le periferiche che devono essere utilizzate, tramite il tool *STM32CubeMX*. Nell'applicazione in questione sono: Timer, ADC e GPIO, Interrupt e DMA; la lo loro configurazione è descritta qui di seguito.

5.2.1 Configurazione timer

La necessità di utilizzare la periferica timer è emersa nelle sezione precedenti: servono per la generazione dell'onda quadra PWM e delle pause. Inoltre poiché è stata scritta una versione di software nella quale il comando per l'attuatore è dato da un pulsante, un altro timer serve come temporizzazione per il debouncing. Per attivare il timer bisogna scegliere i canali da abilitare e le relative modalità, il prescaler, un fattore di scala per il clock di sistema e il counter period ossia il valore di fine conta del timer.

Timer PWM

La configurazione del timer utilizzata per il PWM è mostrata in Figura 5.2.

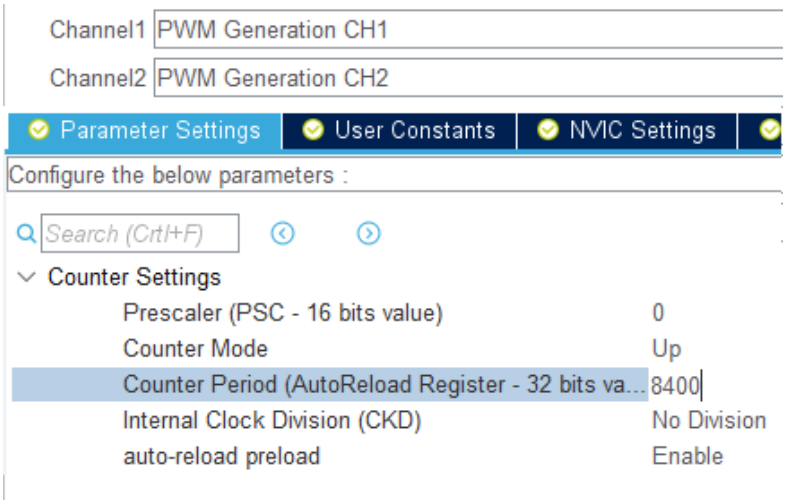


Figura 5.2: Configurazione del timer per il PWM

Sono stati abilitati due canali nella modalità *PWM Generation* e il prescaler è stato impostato ad un valore nullo. Conoscendo la frequenza di clock di sistema e quella dell'onda che si vuole generare, il counter period è dato da:

$$period = \frac{f_{ck}}{f_{PWM}} = \frac{84\,\text{MHz}}{10\,\text{kHz}} = 8400$$

Le onde generate infine vengono mandate su due differenti canali GPIO che andranno a pilotare il driver H bridge. Un'onda servirà per le tensioni positive e una per le tensioni negative.

Timer pause

La configurazione utilizzata per questo timer è mostrata in Figura 5.3.

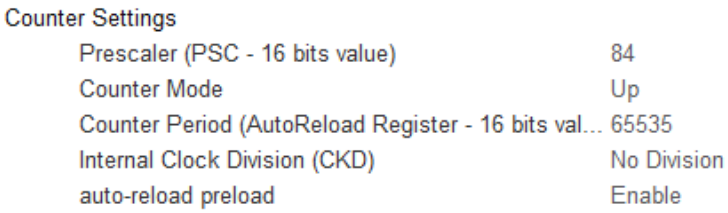


Figura 5.3: Configurazione del timer per le pause

Non è stato abilitato nessun canale hardware perché il timer viene usato solo come riferimento temporale per le pause software. Il valore del prescaler è:

$$prescaler = 84$$

In questo modo ogni tick del timer vale un microsecondo ed è più semplice impostare delle pause. Il valore del counter period è:

$$period = 2^{16} - 1 = 65535$$

È stato impostato il valore massimo perché non è richiesto uno specifico valore.

Timer debouncing pulsante

Le ragioni dell'utilizzo di un timer per il debouncing di un pulsante sono mostrate nella sottosezione 5.4.2. La configurazione utilizzata per questo timer è mostrata in Figura 5.4.

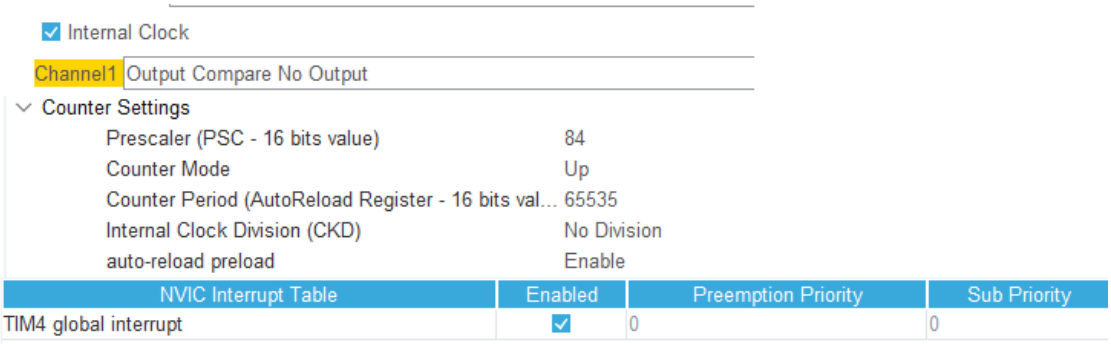


Figura 5.4: Configurazione del timer per il debounce

È stato impostato in modalità *output compare no output* con l'abilitazione dell'interrupt. In questa modalità ogni volta che il timer raggiunge una certa soglia scatena un interrupt. Anche in questo caso poichè non ci sono delle specifiche stringenti la configurazione prevede:

$$prescaler = 84$$
$$period = 65535$$

5.2.2 Configurazione ADC

Per il convertitore analogico digitale bisogna configurare in primis i canali di acquisizione e successivamente il tipo di trasferimento dati. La configurazione dei canali è mostrata in Figura 5.5.

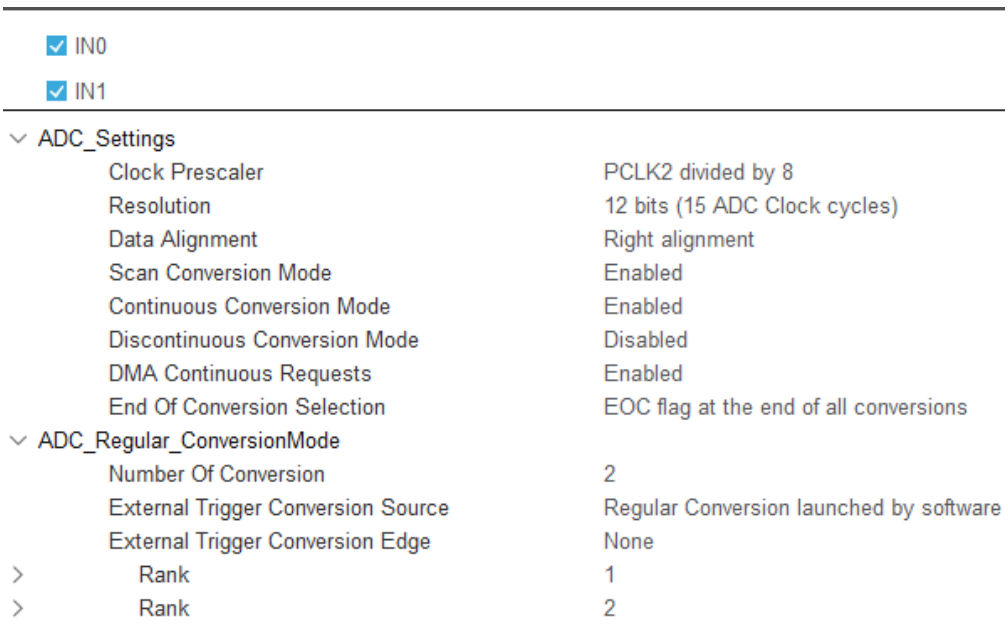


Figura 5.5: Configurazione dei canali del ADC

Sono stati abilitati i canali `IN0` e `IN1` per la lettura del sensore di posizione e corrente. Il parametro *clock prescaler* è stato impostato a *PCLK2 divided by 8* che comporta avere:

$$f_{ADC} \simeq 656\,\text{kHz}$$

Questo riferimento di clock serve per il sample & hold presente nel convertitore. È stato scelto di abbassare il più possibile tale frequenza per avere un'acquisizione più accurata. Sempre a questo fine la risoluzione è stata impostata al valore massimo di 12 bit. Inoltre è stata abilitata la modalità *Scan Conversion Mode*, *Continous Conversion Mode* e *DMA Continous Conversion Mode*; in questo modo i due canali verrano convertiti su continue richieste DMA e il comando di conversione comprenderà la lettura in successione dei due canali attivi.

La configurazione del trasferimento dei dati in DMA è mostrata in Figura 5.6.

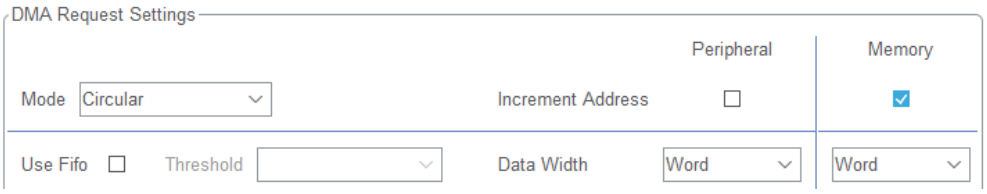


Figura 5.6: Configurazione del DMA per il trasferimento dati

È stata scelta la modalità di trasferimento in DMA Circular, con trasferimento a gruppi di word ossia 32 bit. In questo modo ogni qualvolta siano presenti dati da salvare in memoria verranno trasferiti senza interrompere il processore dal normale funzionamento. Infine è stato abilitato anche l'interrupt relativo al DMA: ogni qualvolta il trasferimento dei dati sarà completo verrà scatenato un interrupt, utilizzato per la conversione dei valori adimensionati acquisiti.

5.2.3 Configurazione interrupt

Come già detto gli interrupt che sono stati abilitati sono quelli relativi al timer per il debouncing e per il DMA. E' stato aggiunto poi un altro interrupt non relativo ad alcuna periferica che può essere scatenato solo tramite una funzione software.
La tabella delle priorità del controller interrupt NVIC è mostrata in Figura 5.7.

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
Non maskable interrupt	☑	0	0
Hard fault interrupt	☑	0	0
Memory management fault	☑	0	0
Pre-fetch fault, memory access fault	☑	0	0
Undefined instruction or illegal state	☑	0	0
System service call via SWI instruction	☑	0	0
Debug monitor	☑	0	0
Pendable request for system service	☑	0	0
Time base: System tick timer	☑	0	0
PVD interrupt through EXTI line 16	☐	0	0
Flash global interrupt	☐	0	0
RCC global interrupt	☐	0	0
ADC1 global interrupt	☑	0	0
TIM2 global interrupt	☐	0	0
TIM3 global interrupt	☐	0	0
TIM4 global interrupt	☑	0	0
DMA2 stream0 global interrupt	☑	0	0
FPU global interrupt	☐	0	0
EXTI line[15:10] interrupts	☑	1	0

Figura 5.7: Tabella di priorità NVIC

Si nota la presenza di molti interrupt a priorità zero non disattivabili; essi sono creati e abilitati dal processore per la gestione di aspetti fuori dal controllo del programmatore. Gli interrupt aggiunti, mostrati in basso sono stati impostati a priorità zero tranne quelli software, `EXTI_LINE[15:10]`, che hanno priorità uno.

5.2.4 Configurazione GPIO

La configurazione delle periferiche GPIO dipende in parte dalle altre periferiche utilizzate, basti vedere i timer e l'ADC, in parte da altre esigenze scollegate.
Una rappresentazione grafica della configurazione GPIO è mostrata in Figura 5.8.

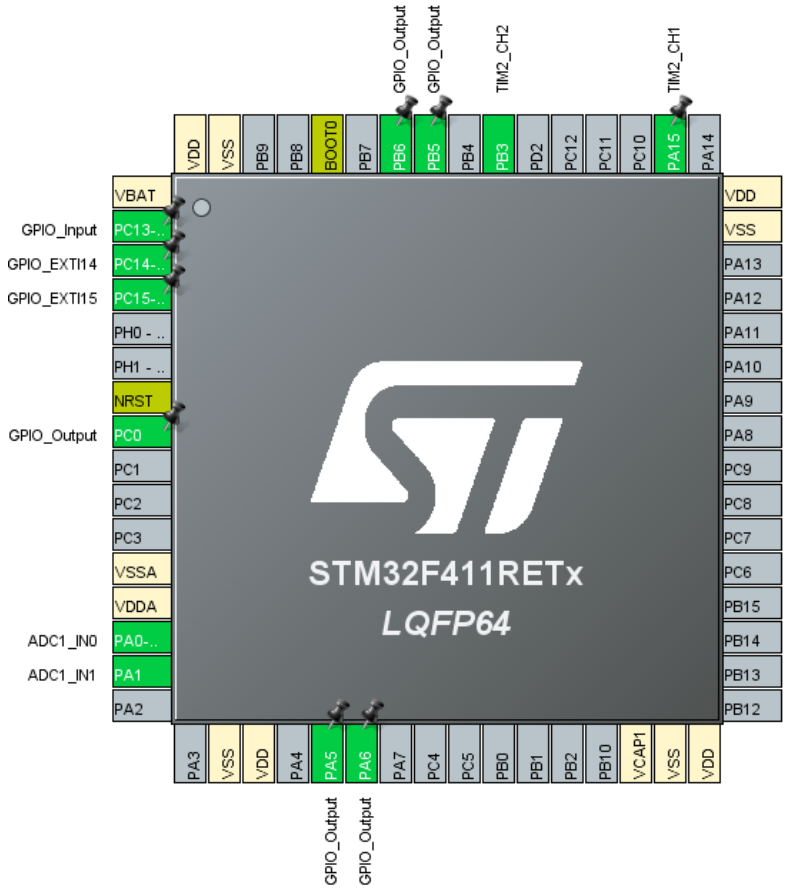


Figura 5.8: Configurazione della periferica GPIO

I pin configurati come `GPIO_Outuput` sono relativi ai segnali di abilitazione del driver H bridge e di due led di segnalazione. Si possono inoltre notare i pin di uscita delle onde quadre e i pin di ingresso dell'ADC. Il pin configurato come `GPIO_Input` infine è relativo al segnale di ingresso del pulsante.

5.3 La struttura del codice C

Il codice C scritto è stato raccolto in una libreria; sono stati creati diversi file header e sorgente nei quali sono raccolte separatamente le funzioni in base alla periferica che gestiscono o la funzionalità software che svolgono. I diversi file sono così organizzati:

- *version.h*: contiene diversi parametri di configurazione per selezionare la versione di firmware da caricare e per l'impostazione delle costanti per la legge di controllo
- *FSM.h* e *FSM.c*: contengono le funzioni per le varie macchine a stati implementate
- *control.h* e *control.c*: contengono le funzione relative all'algoritmo di controllo
- *ADC.h* e *ADC.c*: contengono le routine di conversione dei valori letti relativi ai sensori
- *timer.h* e *timer.c*: contengono le varie funzioni di supporto alla gestione dei timer
- *pushbutton.h* e *pushbutton.c*: contengono le funzioni di lettura del pulsante e debouncing
- *bridge.h* e *bridge.c*: contengono le varie funzioni per la gestione dei comandi da mandare al driver ponte ad H
- *electromagnet.h* e *electromagnet.c*: contengono i segnali di abilitazione dell'elettromagnete per una specifica versione di firmware.

Inoltre è stato aggiunto il codice e le librerie nei file *main.h* e *main.c* generati in automatico dal tool di configurazione.
Per lo sviluppo dell'applicazione è stato necessario pensare versioni differenti

di firmware, mantenendo ovviamente l'algoritmo base come punto fisso. Le versioni pensate oltre a quella base sono servite successivamente a fare i diversi test.

5.4 Versione firmware per il funzionamento normale

Il firmware principale per il normale funzionamento è stato implementato per la versione di attuatore nel quale ci sono solo due posizione possibili, P e notP. Essa prevede che a seguito della ricezione del comando da parte dell'utente il sistema agisca sul motore. Il mezzo tramite il quale l'utente può fornire il comando al microcontrollore è un pulsante presente sulla piattaforma. L'accensione del sistema è simulata tramite la pressione dello stesso dopo che il microcontrollore è stato resettato. A seguito di questa prima pressione, deve essere effettuata una prima movimentazione atta alla compressione della molla presente tra le due slitte, come descritto in sottosezione 1.3.2. Da quel momento la pressione del pulsante comporta l'azionamento in maniera alternata al fine di spostare il cavo nelle due posizioni disponibili.

5.4.1 Macchina a stati implementata

Il metodo più comodo per l'implementazione di quanto appena descritto è l'utilizzo di una macchina a stati software. Il diagramma di transizione degli stati è mostrato in Figura 5.9.

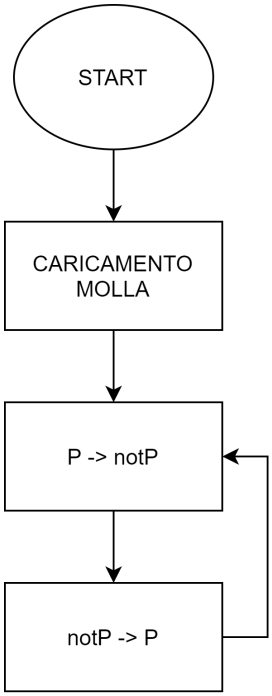


Figura 5.9: Transizioni della macchina a stati

La transizione tra i diversi stati avviene ogni qualvolta viene premuto il pulsante. Gli stati sono tre, uno per il caricamento della molla e l'altro per le due movimentazioni concesse. L'implementazione software della FSM è stata fatta tramite due processi: uno per la transizione degli stati e l'altro per le azioni da compiere, come mostrato nel Listato 5.4 e Listato 5.5.

Listato 5.4: Processo di transizione degli stati per il funzionamento base

```
1  void FSM(){
2      static uint16_t prev_state = 0;
3      static uint16_t prev_button_state = 1;
4
5      if(prev_button_state == 1 && button_state == 0){
6          //transitions
7          if(prev_state == 0)
8              state = 1;
9          else if(prev_state == 1 || prev_state == 3)
10             state = 2;
```

```
11          else if(prev_state == 2)
12              state = 3;
13
14          //operations
15          operation_3();
16          prev_state = state;
17      }
18
19      if (prev_button_state != button_state)
20          prev_button_state = button_state;
21  }
```

Listato 5.5: Processo delle azioni da compiere per il funzionamento base

```
1  void operation_3(){
2      switch (state){
3      case 1:
4          chargeSpring();
5      break;
6      case 2:
7          state1LEDs();
8          motorMove_start(TARGET_TYPE, 1, PID_VERSION);
9      break;
10     case 3:
11         state2LEDs();
12         motorMove_start(TARGET_TYPE, -1, PID_VERSION);
13     break;
14     default:
15         motorMove_start(0, 0, 0);
16         state0LEDs();
17     break;
18 }
```

5.4.2 Debouncing pulsante

La lettura del valore di input proveniente da un pulsante spesso richiede particolare accortezza. Come si può vedere in Figura 5.10 quando viene premuto un pulsante la commutazione porta a delle altre commutazioni spurie. Essendo un componente composto essenzialmente da una barretta metallica che va a chiudere un contatto, la pressione e il rilascio comportano oscillazioni che vengono spesso rilevate come altre commutazioni. Se questo problema non viene risolto si ha un malfunzionamento certo del sistema.

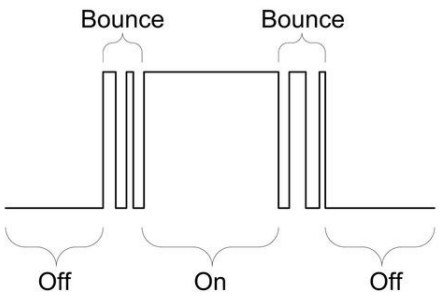


Figura 5.10: Il problema del debouncing

Per effettuare il debouncing di un pulsante esistono varie tecniche, sia hardware che software. Poiché il pulsante utilizzato è integrato sulla board di sviluppo, è stato preferito l'utilizzo di una tecnica software. Il debouncing software si può applicare tramite l'utilizzo di una periferica timer e dell'abilitazione di un interrupt. Consiste nel leggere in maniera periodica il valore presente sul pin digitale di ingresso del pulsante; dopo un numero definito di letture congruenti si può affermare lo stato della periferica. Nello specifico è stato configurato un timer come precedentemente descritto in sezione 5.2.1 per generare un interrupt ogni 250 µs. La routine che viene eseguita periodicamente è mostrata nel Listato 5.6.

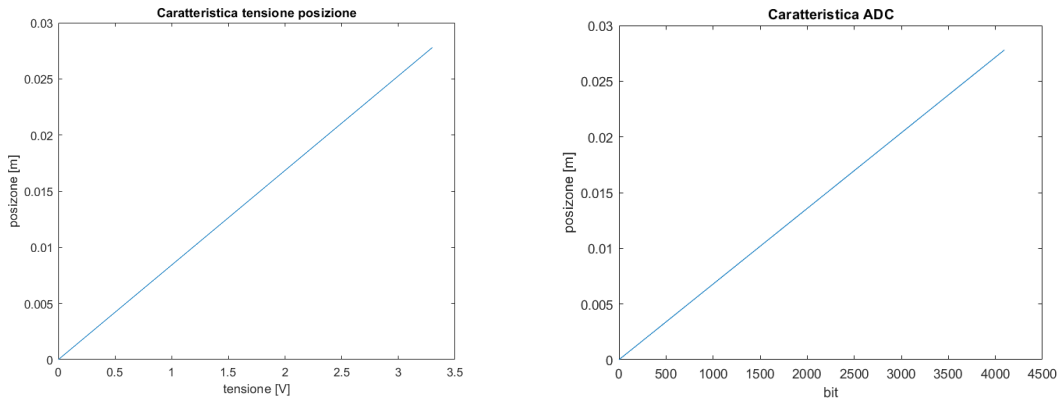
Listato 5.6: Funzione per il debouncing del pulsante

```
void buttonDebounce(){
    static uint16_t count_0 = 0;
    static uint16_t count_1 = 0;
    if(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) == GPIO_PIN_RESET){
        count_0 ++;
        count_1 = 0;
        if(count_0 >= DEBOUNCE){
            button_state = 0;
            __HAL_GPIO_EXTI_GENERATE_SWIT(GPIO_PIN_14);
        }
    }
    else{
        count_1 ++;
        count_0 = 0;
        if(count_1 >= DEBOUNCE){
            button_state = 1;
            __HAL_GPIO_EXTI_GENERATE_SWIT(GPIO_PIN_14);
        }
    }
}
```

Viene letto il valore sul pin di ingresso, in base ad esso viene incrementata la relativa variabile di conteggio `count_0` o `count_1`. Solo quando il valore di una delle due supera la soglia `DEBOUNCE` viene cambiata la variabile di stato `button_state`. In una soluzione classica si sarebbe potuto attivare l'interrupt per la periferica GPIO ma ci sarebbero stati i problemi qui citati; l'alternativa in questo caso è stata quella di scatenare un interrupt software quando viene accertato lo stato del pulsante.

5.4.3 Lettura del sensore di posizione

La posizione letta dal potenziometro si traduce in un valore di tensione analogica proporzionale alla posizione stessa. Tale grandezza deve poi essere convertita tramite software nella posizione effettiva.



(a) **(b)**

Figura 5.11: Grafici per la conversione della posizione

Nella Figura 5.11 è mostrata sulla sinistra la caratteristica del sensore di posizione, ossia quanto vale la tensione in uscita a seguito di una particolare posizione del potenziometro. Poiché il riferimento di tensione è quello interno al microcontrollore, 3.3 V, il valore massimo in uscita sarà quello e corrisponderà alla posizione massima di 27.8 mm, dato emerso da una taratura fatta in via preliminare. La tensione acquisita dall'ADC è convertita in valore digitale su 12 bit; sulla destra in Figura 5.11 è mostrata la retta di conversione per cercare la posizione. Per trovare tale valore è sufficiente determinare preliminarmente il coefficiente angolare della retta:

$$m = \frac{pos\_max}{ADC\_MAX} = \frac{27.8\,\text{mm}}{4095} = 6.78 \times 10^{-6}$$

e applicare l'equazione:

$$pos = m \cdot ADC\_val$$

Infatti ogni volta che le conversioni e il trasferimento DMA viene completato a seguito dell'interrupt scatenato, la routine eseguita è quella mostrata in Listato 5.7.

Listato 5.7: Funzione per la conversione della posizione

```
float posConversion(uint32_t ADC_val){
    float pos;
    float m = (float)LEN_MAX / ADC_MAX;

    pos = ADC_val * m;
    return pos;
}
```

Questo modo di acquisire il segnale permette di discriminare valori fino a 7 µm, misura comunque affetta da errore dovuto all'incertezza del sensore e del sistema di misura.

5.4.4 Lettura del sensore di corrente

Il valore di corrente acquisito dal sensore viene tradotto in una tensione analogica che successivamente viene convertita in valore digitale dal convertitore analogico digitale. Per ricavare da tale valore l'intensità di corrente è necessario applicare una formula per la conversione. La caratteristica tensione corrente del sensore è mostrata in Figura 5.12.

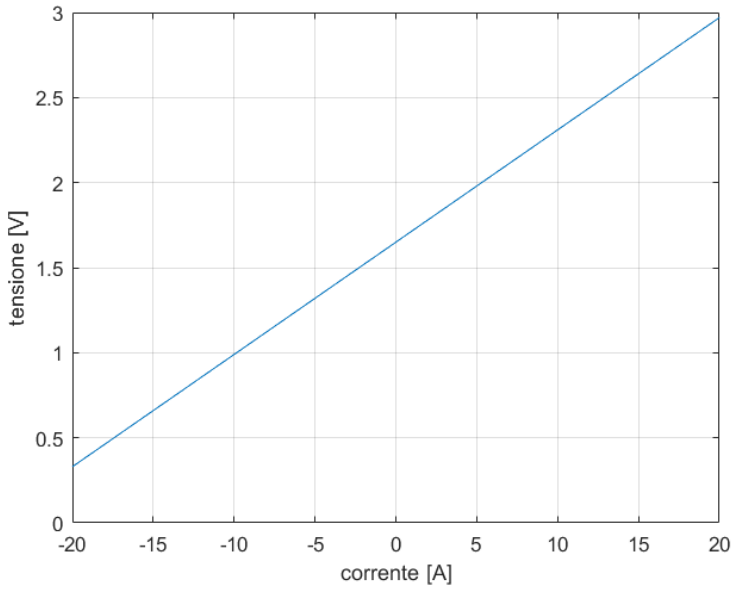


Figura 5.12: Caratteristica tensione corrente del sensore di corrente

La tensione in uscita dal sensore quando la corrente in ingresso è nulla è pari alla metà della dinamica totale, in questo caso di 3.3 V. L'espressione della caratteristica del sensore è:

$$V = m \cdot I + V_0$$

Il valore del coefficiente angolare m è stato tratto dal datasheet del sensore di corrente e risulta essere:

$$m = 0.066\,\mathrm{mV/A}$$

Invertendo tale espressione è possibile calcolare via software il valore di corrente:

$$I = \frac{V - V_0}{m} \tag{5.1}$$

La routine di conversione che ne deriva è mostrata nel Listato 5.8.

Listato 5.8: Funzione per la conversione dell'intensità di corrente

```
float currentConversion(uint32_t ADC_val){
    float mVoltage = (float)V_ADC_MAX / ADC_MAX;
    float voltage, current;

    voltage = ADC_val * mVoltage;
    current = (voltage - q_current) / M_CURRENT;
    return current;
}
```

Viene in primis calcolato il valore della tensione a partire dalla lettura digitale, successivamente si applica l'Equazione 5.1 per trovare il valore di corrente. Poiché il valore di tensione a corrente nulla può leggermente variare, dopo il reset del microcontrollore viene calcolato e salvato nella variabile `q_current` il reale valore misurato.
Questo modo di acquisire il segnale permette di discriminare valori fino a 12 mA, misura comunque affetta da errore dovuto all'incertezza del sensore e del sistema di misura.

5.5 Versioni firmware per il testing

Oltre la versione di firmware per il normale funzionamento sono state progettate delle versioni leggermente differenti per facilitare le operazioni di testing. Le modalità ideate sono:

- Versione senza algoritmo di controllo: viene fornito soltanto un valore fisso di PWM, utile ai fini della validazione del modello Simulink del motore.

- Versione per il test dei parametri PID: utile per la comparazione dei vari parametri forniti dallo studio della strategia di controllo

- Versione per i test ciclici: utile a testare l'affidabilità del software e dell'hardware e la resistenza dei componenti meccanici ad usura. Le versioni implementate a questo fine sono:
  - Ripetizione continua della routine di accensione
  - Ripetizione continua del ciclo completo

5.5.1 Firmware senza algoritmo di controllo

La necessità di avere un firmware che non preveda la presenza dell'algoritmo di controllo è dettata dalla modellizzazione del prototipo. Ad esempio per vedere gli effetti di un comando PWM su un motore, acquisendone la corrente o per estrarre alcuni parametri del motore. È stato anche utile per effettuare delle movimentazioni senza scatola cambio collegata in modo da stimare i parametri meccanici dell'attuatore. La struttura del firmware utilizzata è simile a quella del funzionamento normale, la sola differenza sta nel fatto che quando viene premuto il pulsante non viene eseguito alcun algoritmo di controllo ma viene solo generato il PWM con un duty cycle specificato prima del caricamento del firmware sul microcontrollore. La funzione di generazione del PWM è mostrata nel Listato 5.9.

Listato 5.9: Funzione per la generazione dell'onda PWM

```
1  void onlyPWM(int voltage){
2      if(state != 0 && movement_finished == 0){
3          v_out = voltage;
4          //dc calculation
5          dc = v_out / V_MAX;
6          //bridge dc setting
7          if (dc < 0){
8              setDCBridge(1, 0);
9              //pause of 2 us
10             pause(&htim3, 2);
11             setDCBridge(-1, -dc);
12         } else if (dc > 0){
```

```
13                setDCBridge(-1, 0);
14                //pause of 2 us
15                pause(&htim3, 2);
16                setDCBridge(1, dc);
17            } else{
18                setDCBridge(1, 0);
19                setDCBridge(-1, 0);
20            }
21        } else {
22            setDCBridge(1, 0);
23            setDCBridge(-1, 0);
24        }
25    }
```

Bisogna notare che per una questione di sicurezza, prima di impostare il valore del PWM e attivare un lato del ponte, viene spento l'altro lato e viene forzata una piccola pausa di 2 µs. L'abilitazione contemporanea dei due lati del ponte creerebbe un cortocircuito che danneggerebbe sia il driver che il motore di conseguenza diventa fondamentale avere questo tipo di accortezza.

5.5.2 Firmware per il test dei coefficienti PID

Il motivo principale per cui è stato pensato un firmware semplice per il solo test dei set di coefficienti PID è per verificare la congruenza dei dati che emergono dalle simulazioni con quelli sperimentali. Anche in questo caso la versione di firmware è molto simile a quella per il funzionamento normale; la sola differenza sta nel fatto che l'unica opzione disponibile alla pressione del tasto sia quella di effettuare movimentazioni P notP con un unico set di coefficienti. Il diagramma di transizione degli stati è uguale a quello mostrato in Figura 5.9, ma non è presente uno stato iniziale nel quale è previsto il caricamento della molla. Di conseguenza la routine di transizione degli stati è leggermente diversa come si può notare in Listato 5.10.

Listato 5.10: Processo di transizione degli stati per il test dei coefficienti PID

```
void FSM(){
    static uint16_t prev_state = 0;
    static uint16_t prev_button_state = 1;

    if(prev_button_state == 1 && button_state == 0){
        //transitions
        if(prev_state == 2 || prev_state == 0)
            state = 1;
        else //prev_state == 1
            state = 2;
        //operations
        operation_3();
        prev_state = state;
    }
    if (prev_button_state != button_state){
        prev_button_state = button_state;
    }
}
```

Gli stati sono due, di conseguenza anche la routine delle operazioni da compiere avrà solo due stati. Tramite un file header di configurazione è possibile scegliere prima del caricamento del programma la versione di coefficienti PID da utilizzare.

5.5.3 Firmware per il test ciclico

Al fine di effettuare dei test sulla resistenza del prototipo meccanico e sulla robustezza dell'hardware e dell'algoritmo di controllo sono state scritte due versioni di firmware apposite. A differenza delle versioni precedentemente descritte in questo caso l'interazione con l'utente è limitata solo al semplice comando di start. I comandi per le diverse attuazioni vengono forniti tramite istruzioni C; questo ha comportato l'implementazione di un meccanismo

per riconoscere e di conseguenza agire a seguito della terminazioni di uno specifico movimento.

Versione per la sequenza di accensione

Come spiegato nella sottosezione 1.3.2, la funzionalità di sicurezza prevede una particolare sequenza di accensione, nella quale la movimentazione dell'attuatore ha come scopo la compressione della molla posta tra le due slitte è l'entrata in funzione dell'elettromagnete, in modo da rendere solidale il cavo con la trasmissione. Per questa specifica movimentazione sono stati trovati dei parametri del controllore specifici per il carico da vincere. Poiché l'accensione è un punto cruciale per il funzionamento dell'applicazione, è stata pensata una versione di firmware al solo scopo di testare questa funzionalità. Per effettuare questo test automatico è stato necessario modificare leggermente il setup hardware dell'attuatore. Nel prototipo originale l'elettromagnete deve essere collegato direttamente alla tensione di alimentazione del sistema. Questa scelta si rileva scomoda ai fini del testing poiché si ha la necessità di accendere e spegnere in maniera dinamica l'elettromagnete. Di conseguenza è stato modificato leggermente il circuito collegando l'elettromagnete ad un semplice circuito di pilotaggio high side.
Per realizzare il firmware è stato studiato preliminarmente una macchina a stati adeguata, il cui diagramma di transizione è mostrato in Figura 5.13.

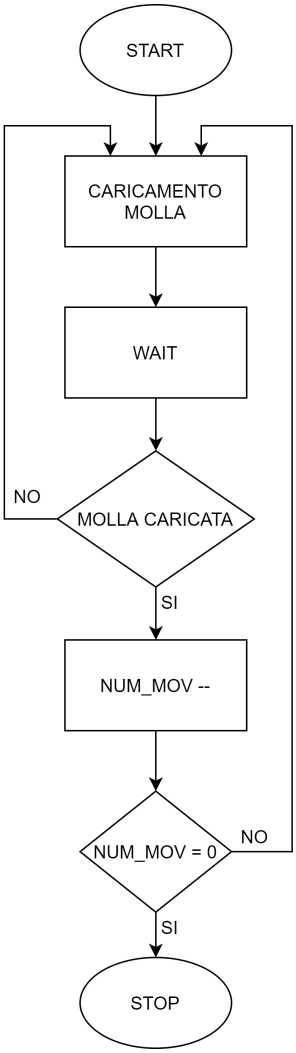


Figura 5.13: Transizioni della macchina a stati per il testing della sequenza accensione

Dopo aver acquisito il segnale di start, si procede con una routine che viene ripetuta un numero di volte specificato in fase di programmazione. Viene effettuato un tentativo di caricamento della molla, seguito da una piccola pausa. Dopo tale attesta viene verificato che il sistema abbia realmente compresso la molla e che l'elettromagnete stia funzionando; se così non fosse si riprova il caricamento. Una volta che questa fase va a buon fine, si decrementa la variabile che conta il numero di movimentazioni fatte e si ricomincia da

capo. Quando sono state effettuate correttamente il numero prefissato di movimentazioni specificate il programma finisce. Questo diagramma è stato quindi tradotto in codice C. La sequenza di transizione degli stati è mostrata nel Listato 5.11.

Listato 5.11: Processo di transizione degli stati per il test della sequenza di accensione

```
void FSM_5(){
    static uint16_t prev_button_state = 1;
    static uint32_t count_mov = 0;
    if(prev_button_state == 1 && button_state == 0){
        state1LEDs();
        springCharge();
        state = 1;
        count_mov = 0;
        //don't listen button
        stopTimer_OC_IT(&htim4, 1);
    }
    else if (state >= 1){ //movement finished
        state ++;
    }
    if (count_mov <= MOV_NUM && state > 1){
        state2LEDs();
        springCharge();
        count_mov ++;
    }
    else if (count_mov > MOV_NUM){
        //listen button
        stopTimer_Base(&htim4);
        startTimer_OC_IT(&htim4, 1);
    }
    if (prev_button_state != button_state){
            prev_button_state = button_state;
    }
}
```

Questa funzione viene richiamata dopo che viene ricevuto il comando di start o quando la precedente movimentazione è completata. Infatti si nota come le prime righe di codice siano relative alla prima movimentazione mentre le successive sono relative a quelle automatiche comandate via software. Ogni qualvolta si vuole fare una movimentazione viene invocata la funzione `springCharge()`, il cui corpo è mostrato nel Listato 5.12.

Listato 5.12: Funzione per la gestione del caricamento molla

```
void springCharge(){
    static int16_t first_round = 1;
    static int16_t attempt = 0;

    spring_charging = 1;
    if (first_round == 0){
        if (validSpringCharge()){
            spring_charging = 0;
            //prepare for a possibile next spring charge
            first_round = 1;
            attempt = 0;
            pause(&htim3, 1000000);
            motorMove_5_stop();
        }
        else
            pause(&htim3, 3000000);
    }
    if (attempt < SPRING_ATTEMPT_MAX){
        if (spring_charging == 1){
            motorMove_5_start(2, 1, 2);
            attempt ++;
            first_round = 0;
        }
    }
}
```

Viene impostato tramite la costante `SPRING_ATTEMPT_MAX` il numero massimo di tentativi che deve effettuare il sistema per una singola movimentazione;

lo start per la valutazione dell'algoritmo di controllo viene dato tramite la funzione `motorMove_5_start()` che si occupa del setting dei vari parametri e l'inizializzazione di alcune variabili. Una volta che la movimentazione è finita il comando del programma torna alla funzione `springCharge()` che va a verificare tramite la funzione `validSpringCharge()` se il caricamento della molla è andato a buon fine. Se la verifica è andata a buon fine viene invocata la funzione `motorMove_5_stop()` che si occupa di togliere alimentazione all'elettromagnete e aspettare che la molla venga rilasciata; infine genera un interrupt software la cui routine è la funzione `FSM_5()` presente nel Listato 5.11. Se invece il controllo sulla validità del caricamento della molla non va a buon fine, viene effettuato un altro tentativo richiamando nuovamente la funzione `motorMove_5_start()`.

Versione per il funzionamento completo

Una volta ultimati i test sui vari coefficienti PID, e sul comportamento del sistema a fronte dell'azione di caricamento della molla, è stato pensato un ultimo test di tutte le funzionalità del sistema in maniera ciclica. L'azienda ospitante ha messo a disposizione un banco prova grazie al quale è stato possibile collegare fisicamente l'attuatore alla scatola cambio tramite il cavo meccanico. A fronte di questa disponibilità è stata studiata una versione di firmware che fosse in grado di effettuare molte movimentazioni P notP in automatico, senza l'intervento di un comando utente. Questo tipo di test è stato utile a capire la risposta del sistema a fronte di svariate movimentazioni, al fine di verificare la robustezza dell'impianto meccanico, dei componenti hardware e del firmware. Per realizzare il firmware anche in questo caso è stato fatto uno studio preliminare su una macchina a stati adeguata, il cui diagramma di transizione degli stati è mostrato in Figura 5.14.

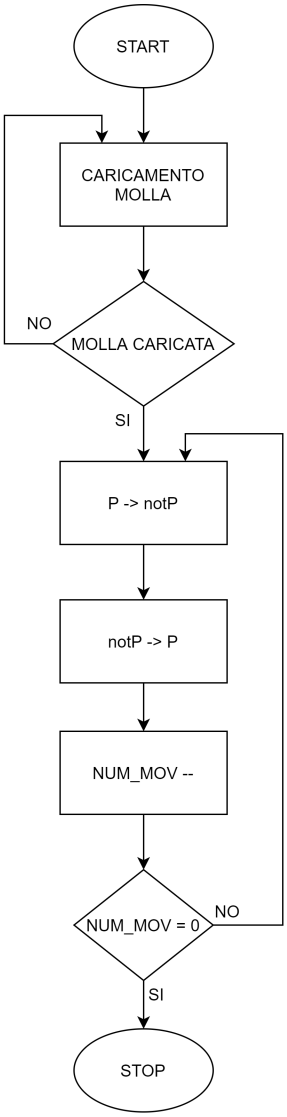


Figura 5.14: Transizioni della macchina a stati per il testing del funzionamento completo

Si può notare come il diagramma sia molto simile a quello per il test di caricamento della molla. Dopo che viene ricevuto il comando di partenza dal pulsante viene effettuata la routine di start che consiste come nel caso precedente nei tentativi ripetuti di caricamento della molla. Una volta finita questa fase vengono effettuate in sequenza le due movimentazioni disponibili; infine viene decrementata la variabile contatore `NUM_MOV`. Finché il valore

è diverso da zero vengono ripetute le due movimentazioni. La macchina a stati progettata si traduce nel codice mostrato nel Listato 5.13.

Listato 5.13: Processo di transizione degli stati per il test del funzionamento completo

```
void FSM_4(){
    /* state:
    * 0 -> nothing to be done
    * 1 -> spring charge
    * odd -> notP movement
    * even -> P movement
    */
    static uint16_t prev_button_state = 1;
    static uint32_t count_mov = 0;
    static uint16_t spring_charged = 0;
    static uint16_t small_mov_done = 0;

    if(prev_button_state == 1 && button_state == 0){
        if (spring_charged == 0){
            //charge spring
            state1LEDs();
            springCharge();
            spring_charged = 1;
            state = 1;
        }
        else
            state = 3;
        count_mov = 0;
        stopTimer_OC_IT(&htim4, 1); //don't listen button
        startTimer_Base(&htim4);
    }
    else if (state >= 1){ //movement finished
        state ++;
    }
    if (count_mov <= MOV_NUM && state > 1){
        if (state % 2 == 0){
```

```
32              state2LEDs();
33              //notP to P
34              if (!small_mov_done){
35                  motorMove_start(0, -1, PID_VERSION); //small
    movement
36                  small_mov_done = 1;
37                  state --;
38              } else {
39                  motorMove_start(TARGET_TYPE, -1, PID_VERSION);
40                  count_mov ++;
41              }
42          }
43          else if(state % 2 == 1){
44              state1LEDs();
45              motorMove_start(TARGET_TYPE, 1, PID_VERSION); //P to
     notP
46              count_mov ++;
47          }
48      }
49      else if (count_mov > MOV_NUM){
50          offElectromagnet();
51          stopTimer_Base(&htim4);
52          startTimer_OC_IT(&htim4, 1); //listen button
53          spring_charged = 0;
54      }
55      if (prev_button_state != button_state){
56              prev_button_state = button_state;
57      }
58  }
```

Il principio di funzionamento di questa funzione è molto simile a quello mostrato nel Listato 5.11. Dopo che viene acquisito lo start dal pulsante, si procede con la routine di accensione richiamando la funzione `springCharge()`. Con la terminazione della movimentazione viene indotto un interrupt software che richiama la funzione `FSM_4()`. A questo punto verranno effettuate

le movimentazioni in entrambi i versi in maniera sequenziale richiamando la funzione `motorMove_start()` e specificando la corsa da effettuare, la direzione e il set di coefficienti PID da utilizzare. Nella prima movimentazione dopo l'accensione viene fatto un piccolo movimento in direzione notP, è utile per evitare che durante i test ripetuti la slitta che sostiene il cavo vada urtare contro i finecorsa meccanici. Poiché il numero di movimentazioni specificate comprendono entrambe le direzioni, per poter discriminare dopo ogni interrupt quale movimento fare si è scelto di utilizzare la parità del numero. Quando la variabile `count_mov` vale un numero pari viene fatta la movimentazione da notP a P mentre quando vale un numero dispari viene fatta la movimentazione opposta.

Capitolo 6

Testing

Nella parte finale del progetto ci si è occupati di testare a fondo le varie funzionalità previste e progettate nelle precedenti fasi di lavoro. Gli scopi del testing sono stati molteplici. Dalla validazione del modello realizzato in Simulink alla validazione del sistema di controllo. Comparando i risultati delle simulazioni con quelli sperimentali si sono potuti trovare e correggere diversi bug presenti sia nel firmware che nella progettazione Simulink. Inoltre si è potuto verificare che il setup hardware fosse adeguato in termini di valori massimi di tensione e corrente piuttosto che di precisione dei vari sensori. Un ultimo obiettivo della fase di testing è stato quello di evidenziare, ove ce ne fossero, dei difetti di progettazione meccanica soprattutto per quanto riguarda gli aspetti legati all'usura e alla rigidezza del sistema; temi difficilmente prevedibili in fase di progettazione al CAD.

6.1 Tecnica utilizzata per l'acquisizione dati

Come appena detto, la fase di testing richiede necessariamente la presenza di un sistema che permetta l'acquisizione dei dati in real time. Questo tema è quindi centrale per garantire la corretta validazione e debug delle varie parti sia software che hardware. Un primo approccio utilizzato è stato quello di

aggiungere delle routine firmware dedicate che tramite interrupt periodico, o DMA, trasferissero i dati essenziali al PC. Era stato pensato un collegamento fisico tramite porta USB, e come protocollo di comunicazione quello UART (Universal Asynchronous Receiver Transmitter) che è un sistema semplice e veloce per comunicare. Da lato PC era stato progettato, in linguaggio di programmazione Python, un monitor che interpretasse correttamente i dati ricevuti e li stampasse a video. Questa soluzione però non è per niente ottimale in quanto l'inserimento di ulteriori routine concorrenti con l'algoritmo base vanno ad intaccare pesantemente il funzionamento stesso dell'applicazione rendendo inefficace la fase di testing.

6.1.1 Monitor di acquisizione dati

Dopo alcune ricerche sul sito ufficiale dell'azienda *ST Microelectronics* è stata notata l'esistenza di un tool creato appositamente per le piattaforme di sviluppo come quella utilizzata in questa applicazione. Il software sfrutta il debugger, un microprocessore secondario presente sulla board che permette di caricare il firmware sul microcontrollore principale, effettuare il debug e comunicare con il PC tramite porta USB. Il tool è chiamato *STM32Cube Monitor* è permette di creare dei monitor personalizzati per effettuare debug e testing. Per poter utilizzare il software è necessario caricare lo stesso file eseguibile che viene eseguito sul microcontrollore; tramite tale file si può entrare a conoscenza delle variabili globali dichiarate nel codice, che vanno solitamente salvate in memoria centrale. Configurando appositamente il monitor si permette al debugger di leggere e trasmettere periodicamente i dati presenti in memoria senza intralciare il normale funzionamento del microcontrollore. Il tipo di programmazione che viene utilizzata in questo tool è di tipo grafico, come si può notare nella configurazione utilizzata mostrata in Figura 6.1.

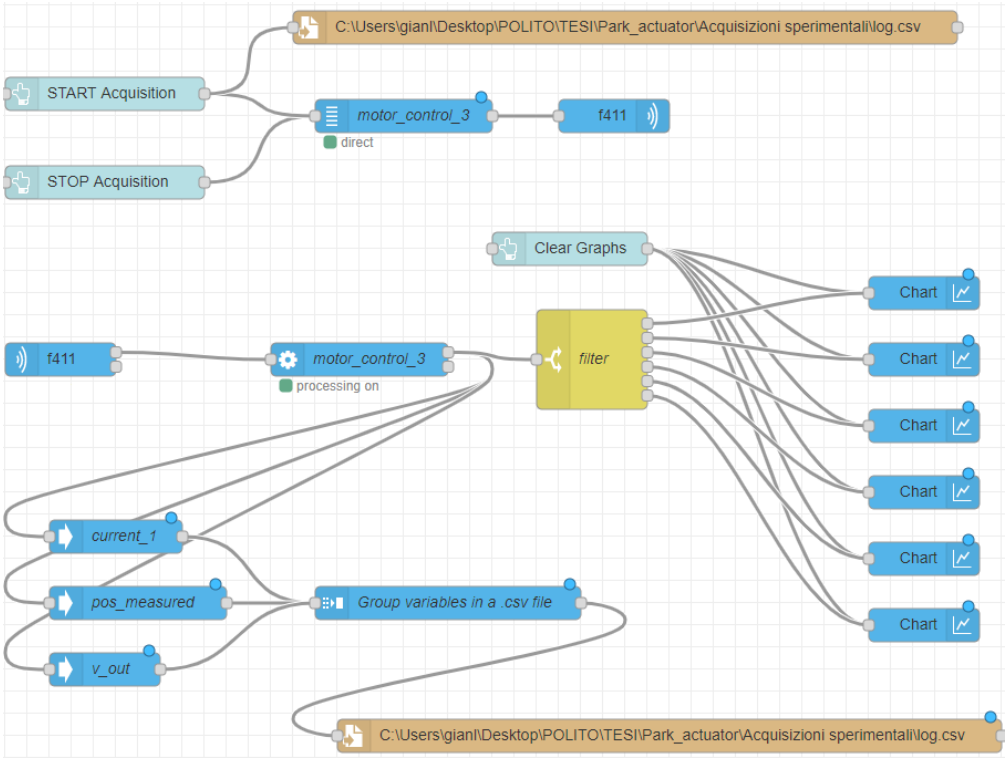


Figura 6.1: Configurazione monitor acquisizione dati

I blocchi presenti in alto permettono di inserire due pulsanti di inizio e fine acquisizione, e di impostare il file sorgente. Successivamente vengono aggiunti i due blocchi denominati `f411` che permettono di trasmettere e ricevere informazioni dal debugger. Dal flusso di dati in ingresso tramite un blocco filtro vengono estratte le variabili di cui si vuole creare il log nel monitor. Infine ogni blocco `Chart` rappresenterà un flusso dati relativo ad una singola variabile. La visione grafica del monitor di acquisizione dati dopo la compilazione del diagramma a blocchi realizzato è mostrato in Figura 6.2.

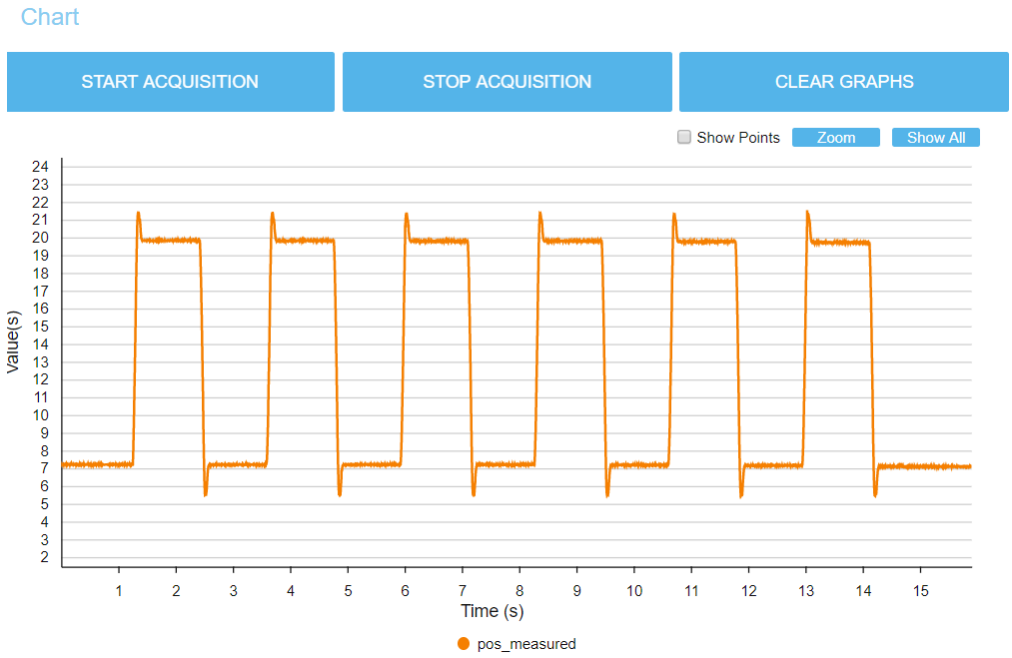


Figura 6.2: Esempio monitor acquisizione dati

Nello specifico è mostrato un log del sensore di posizione durante un test ciclico. Si nota in alto la presenza dei tre pulsanti inseriti in fase di programmazione per fornire start, stop e pulizia dei grafici. In arancione invece il log di tutte le posizioni acquisite in millimetri in funzione del tempo.

6.1.2 Salvataggio dati e importazione in Matlab

Per poter mettere a confronto i dati sperimentali con quelli della simulazione è necessario importare in ambiente Matlab i log; in questo modo è molto semplice sovrapporre i due grafici e fare valutazioni. Un'altra potenzialità del tool *MXCube Monitor* è quella di permettere di creare dei file di log con i valori che vengono mostrati a video durante l'acquisizione. Nella parte inferiore in Figura 6.1 è presente una serie di blocchi che permette di isolare la variabile di cui si vuole salvare il log e scegliere il nome e il tipo di file. Nel caso in questione è stato scelto un file di tipo csv (comma separated value) nel quale i valori sono organizzati in righe e colonne. I valori nelle righe sono

separati grazie a delle virgole. I file csv vengono poi importati in Matlab tramite il comando `readtable(filename)` che interpreta in automatico il formato file csv e crea una oggetto Table i cui attributi sono le colonne di dati. Accedendo a tale oggetto è stato possibile creare una moltitudine di grafici in maniera molto semplice.

6.2 Validazione del modello del motore

Una prima fase di testing ha come obiettivo la validazione del modello Simulink soprattutto per quanto riguarda la modellizzazione del motore e dei suoi parametri. L'individuazione dei vari coefficienti è spesso complicata e costosa; nel lavoro portato avanti in azienda, ci siamo avvalsi ove possibile del datasheet e del supporto dell'azienda produttrice. Spesso però alcuni dati non erano presenti o erano inesistenti di conseguenza sono stati studiati dei banchi prova per poter effettuare delle valutazioni sul motore. Per effettuare le stime dei parametri motore è molto utile raccogliere dati sul profilo di corrente nel tempo a fronte di uno stimolo di tensione costante. Per il banco prova realizzato è stato scollegato il motore dalla trasmissione a ruote dentate; è stato alimentato il motore con una tensione continua e si è letta la corrente assorbita tramite il sensore collegato al microcontrollore. I valori letti vengono mostrati tramite il monitor e salvati in un file di log. Grazie a questo tipo di test sono stati stimati in maniera più precisa i valori di resistenza e induttanza intrinsechi del motore. La comparazione dei profili di corrente sperimentali e quelli simulati è mostrata in Figura 6.3.

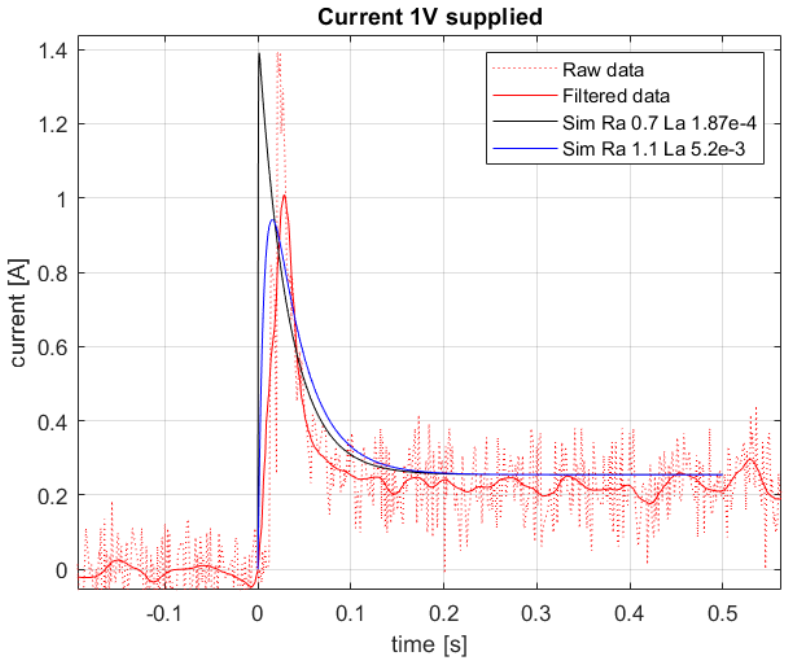


Figura 6.3: Comparazione valori di corrente per test motore

Si possono notare in rosso i dati sperimentali, ai quali è stato applicato un filtro per avere un andamento più leggibile; i risultati della simulazione con i coefficienti estratti dal datasheet sono rappresentati in nero. Si nota come, soprattutto negli istanti iniziali, il picco di corrente sia molto più alto rispetto a quello reale. Di conseguenza variando leggermente i valori di R_a e L_a e effettuando nuovamente la simulazione si è ottenuto il profilo in blu; si può subito notare come sia quasi sovrapponibile al profilo dei dati sperimentali. Osservando quindi i profili di corrente si è potuta fare una misurazione indiretta di questi due parametri fondamentali del motore.

6.3 Studio della frequenza di pilotaggio PWM

Un'altro parametro importante da scegliere è stata la frequenza dell'onda quadra PWM, come evidenziato in sottosezione 4.3.1. Per confrontare la risposta del sistema a fronte del cambiamento della frequenza PWM sono

stati acquisiti diversi set di dati e poi sovrapposti in un grafico. Si è scelto di mantenere la frequenza di aggiornamento del controllore t_s uguale a quella dell'onda quadra; inoltre si è preferito effettuare le movimentazioni senza il cavo meccanico collegato alla scatola cambio e di utilizzare un set di coefficienti PID adeguati per una movimentazione a vuoto. Sono stati provati alcuni valori di frequenza nel range [500 Hz; 20 kHz]; i valori letti dal sensore di posizione sono stati poi importati in Matlab ed è stato creato un plot che li contenesse tutti, come mostrato in Figura 6.4.

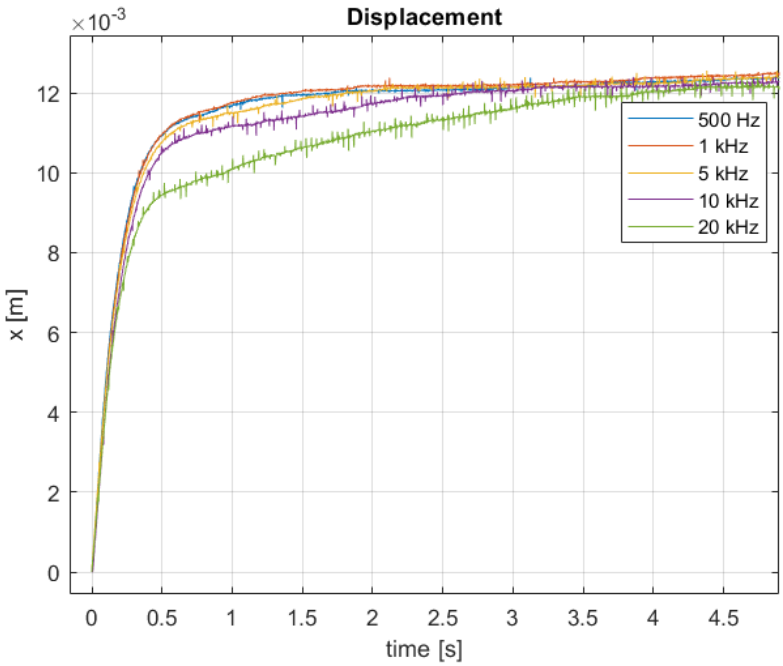


Figura 6.4: Comparazione valori di frequenza diversi per onda PWM

Si può notare come per le frequenze più basse fino a 5 kHz gli andamenti siano praticamente sovrapponibili, mentre per le frequenze maggiori gli andamenti si distacchino. L'aspetto che salta all'occhio è il rallentamento del sistema all'aumentare della frequenza; infatti il tempo impiegato a raggiungere il target per le frequenze più elevate è quasi triplicato rispetto alle frequenze più basse. Il motivo di questo comportamento sta nel fatto che frequenze di PWM elevate fanno entrare in gioco effetti di secondo ordine

che diminuiscono drasticamente l'efficienza del motore. A fronte di questi risultati verrebbe da dire che la scelta più corretta sarebbe una frequenza bassa, bisogna però ricordare il disturbo audio emesso dal motore a fronte di uno stimolo periodico. Si può affermare quindi che un giusto compromesso è costituito da una frequenza di 10 kHz, poiché l'efficienza non è intaccata molto e il suono emesso non risulta essere molto fastidioso.

6.4 Confronto tra risultati sperimentali e simulativi

Nella fase finale della parte di testing ci si è concentrati sul test di diverse funzionalità operative; l'obiettivo è quello di verificare la congruenza degli esiti delle simulazioni con quelli delle prove sperimentali.

6.4.1 Movimentazioni senza cavo meccanico

Dal punto di vista del setup meccanico da realizzare e dei rischi che si possono correre effettuando la movimentazione, quella senza cavo meccanico è certamente la più semplice. In questa modalità le forze da vincere sono molto basse di conseguenza anche le correnti assorbite dal motore, questo comporta una particolare facilità nel tarare i coefficienti del controllore. Sembrerebbe quindi superfluo effettuare questa tipologia di test ma è servito per capire se la tecnica utilizzata in questo progetto fosse corretta o meno. Per verificare la correttezza dei risultati sono stati raccolti e comparati i dati relativi alla posizione acquisita dal potenziometro, la corrente assorbita e il comando di tensione pilotato al motore. I grafici risultanti sono mostrati in Figura 6.5 e Figura 6.6.

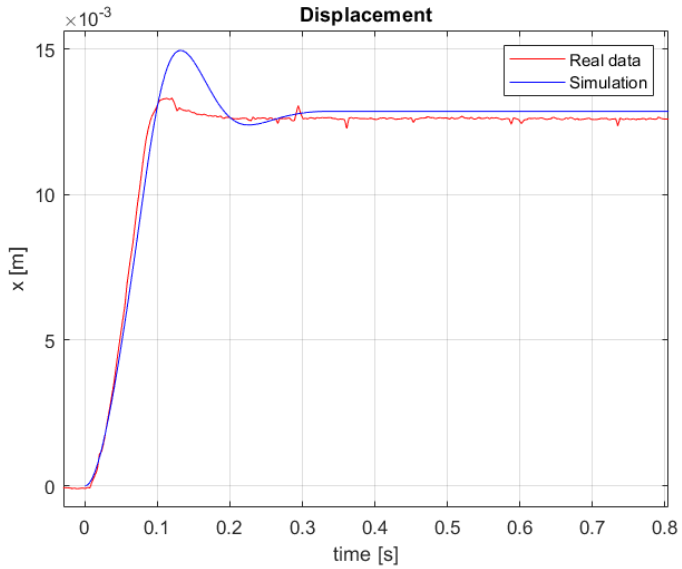
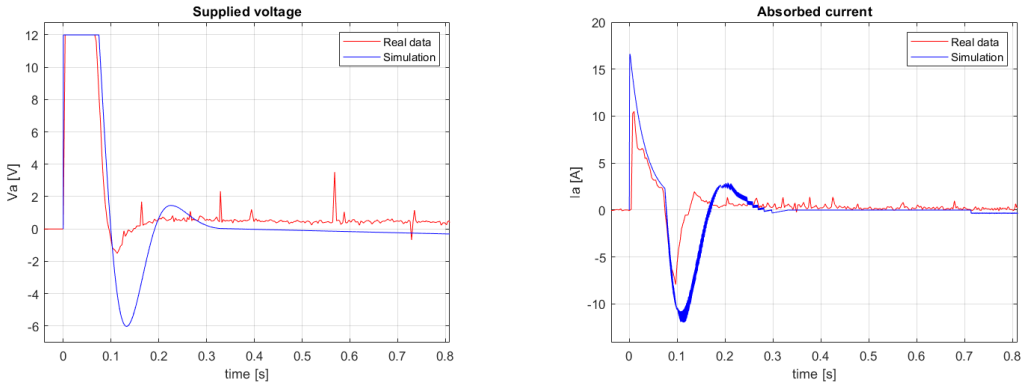


Figura 6.5: Comparazione profili di spostamento della movimentazione senza cavo meccanico



(a) **(b)**

Figura 6.6: Comparazione dati elettrici delle movimentazioni senza cavo meccanico

Nel grafico in Figura 6.5 sono messi a confronto i profili di spostamento reali e simulati. Si può notare come i due andamenti siano pressoché sovrapponibili; si evidenzia soltanto una discrepanza nell'overshoot che risulta essere più marcato nella simulazione. La causa di questa differenza è da ricercarsi

sicuramente in un'imprecisione nella stima di alcuni parametri legati agli attriti o al motore, che per loro natura sono difficili da stimare. Nei grafici in Figura 6.6 invece sono paragonati i valori di tensione e corrente; anche in questo caso gli andamenti sono paragonabili tranne in alcuni punti. I risultati di questi test si possono ritenere molto buoni soprattutto perché gli andamenti dei risultati sperimentali con quelli simulativi sono gli stessi.

6.4.2 Caricamento della molla all'accensione

Una volta appurato che la tecnica di progettazione fosse robusta ed efficiente si è passati a testare il sistema in una condizione di sforzo più importante. È stato tarato il controllore per poter vincere le forze maggiori dovute alla compressione della molla e si è testata quindi la routine di accensione con conseguente compressione della molla. Dalle simulazioni era emerso che le correnti assorbite sarebbero state più elevate quindi si è fatta particolare attenzione durante la fase di test. Anche in questo caso sono stati acquisiti i dati relativi a posizione, tensione e corrente come si nota in Figura 6.7 e Figura 6.8.

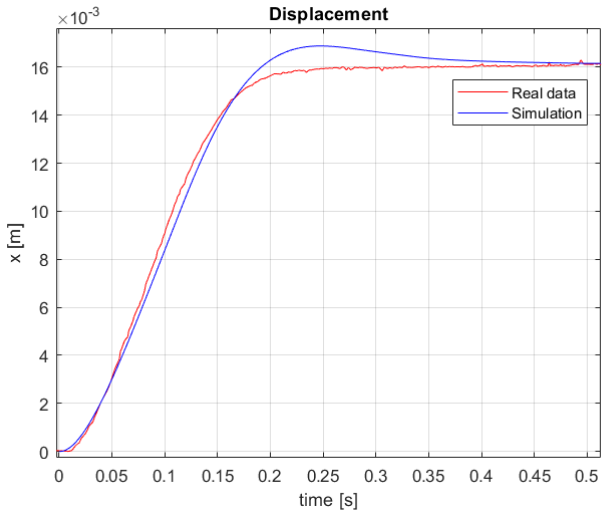
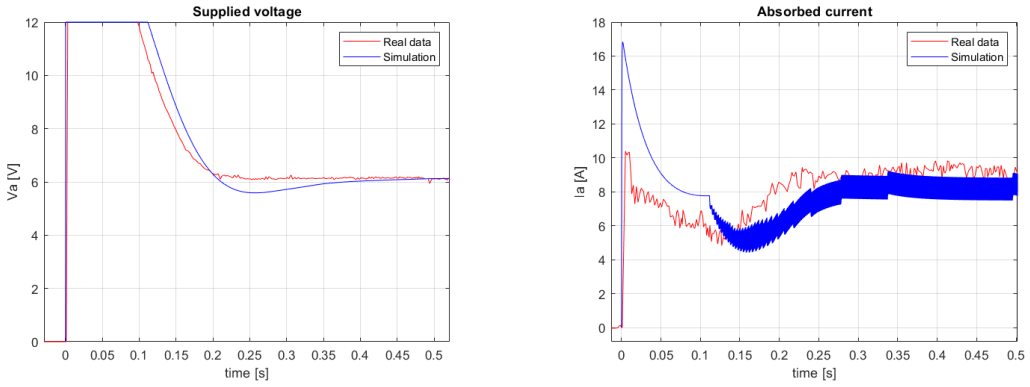


Figura 6.7: Comparazione profili di spostamento della movimentazione durante il caricamento della molla



(a) **(b)**

Figura 6.8: Comparazione dati elettrici delle movimentazioni durante il caricamento della molla

Nella Figura 6.7 si può vedere come i profili dello spostamento sono praticamente sovrapposti tranne sempre una piccola discrepanza nella zona di picco. Stesse riflessioni possono essere fatte per i profili di tensione e corrente in Figura 6.8. È interessante notare come nella fase iniziale la tensione pilotata sia di 12 V per circa 0.1 s; sembrerebbe come se il grafico fosse tagliato nella parte superiore, in realtà il comando di uscita del controllore sarebbe stato più alto se non fosse per la presenza di un saturatore. Essa può valere qualsiasi numero nel continuo però il limiti fisici impongono che la tensione massima pilotabile sia di 12 V. Per questo motivo nell'algoritmo software per il PID è stato inserito un saturatore che impedisce alla tensione di pilotaggio di assumere valori al di fuori del range concesso. Delle informazioni interessanti possono essere estratte confrontando i dati sperimentali in figura 6.6b e 6.8b. I valori di corrente media assorbita nel caso della compressione della molla sono molto maggiori rispetto al caso senza cavo meccanico. La forza da vincere per comprimere la molla è molto alta, infatti la corrente assorbita è di circa 8 A mentre per una movimentazione a vuoto a parte un transitorio iniziale è praticamente nulla. Questo dato è da tenere a mente perché dalle specifiche del motore si evince che la corrente che il motore può

assorbire per un tempo indefinito senza essere danneggiato è proprio di 8 A; quindi è un dato da tenere particolarmente sotto controllo tramite la taratura dei parametri PID. La scelta finale si può ritenere comunque soddisfacente e i test di movimentazioni ripetute non hanno comportato alcun degrado delle prestazioni del motore né tanto meno nessun aumento significativo della temperatura dello stesso.

6.4.3 Movimentazione con scatola cambio

L'ultimo obiettivo della fase di testing consiste nel verificare il corretto funzionamento dell'attuatore in una condizione operativa reale. Purtroppo non è stato possibile installare l'attuatore dentro l'abitacolo di un'automobile, nonostante ciò è stata utilizzata una scatola cambio gentilmente offerta dal reparto di testing dell'azienda ospitante. I colleghi esperti hanno creato un banco prova grazie al quale è stato possibile connettere fisicamente l'attuatore realizzato alla scatola cambio tramite cavo meccanico. Il progetto in partenza era stato studiato per funzionare con un altro modello di scatola cambio anche se uno degli obiettivi era quello di creare un attuatore che fosse il più versatile possibile. Infatti dal punto di vista meccanico non ci sono state particolari complicazioni per la creazione del banco prova. La scatola cambio era una tipologia diversa perché le posizioni disponibili erano P, R, N, D mentre la versione per cui è stata stato studiato l'attuatore era per scatole cambio con sole due posizioni disponibili P, notP. Questo non ha comportato grosse problematiche, la soluzione adottata è stata quella di sfruttare solo due posizioni: R, N. È bastato estrarre il profilo di forza da vincere da alcuni grafici sperimentali sul quale sono stati tarati i coefficienti PID adeguati quindi modificare le costanti riguardanti corse e parametri nel firmware. Si può affermare che il progetto portato avanti ha centrato in pieno l'obiettivo di versatilità perché in poche ore si è passati da una configurazione ad un'altra senza alcuna difficoltà. La versione di firmware utilizzata è stata quella descritta nella sezione 5.5.3. Sono stati acquisiti i dati relativi a due cicli di

test da più di un centinaio di movimentazioni; successivamente sono stati creati dei grafici comparativi come nelle precedenti fasi di test; i risultati sono mostrati in Figura 6.9 e Figura 6.10.

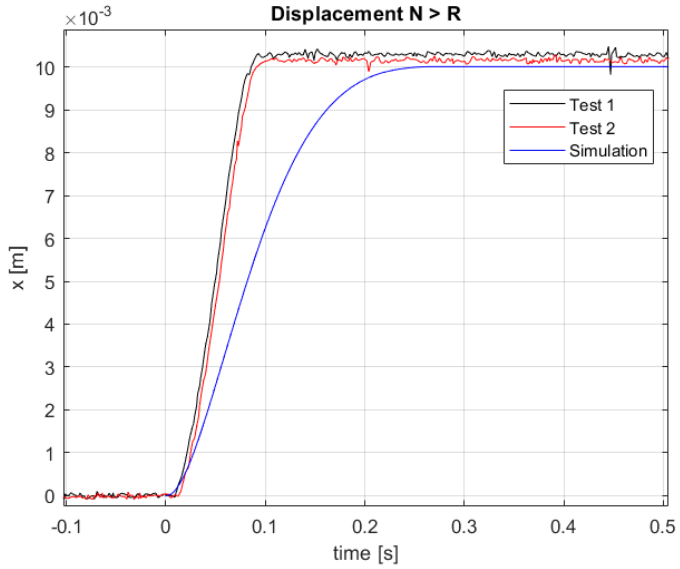
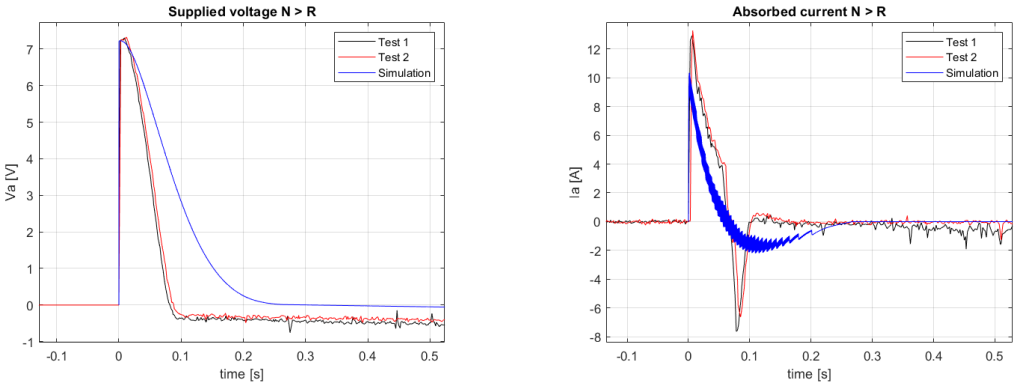


Figura 6.9: Comparazione profili di spostamento della movimentazione con scatola cambio



(a) **(b)**

Figura 6.10: Comparazione dati elettrici delle movimentazioni con scatola cambio

Dai grafici si può notare come i risultati dei due test, in rosso e nero, siano sovrapposti tra di loro, sintomo che l'attuatore non ha cambiato il su comportamento nel tempo. Confrontando invece i risultati dei test con quelli della simulazione si può notare un comportamento del sistema più veloce del previsto, come mostrato in Figura 6.9; anche il profili di corrente e tensione, in Figura 6.10, mostrano in alcuni istanti delle differenze. Il comportamento differente tra test e simulazione è da ricercare nella difficoltà che si è avuta nel ricavare da alcuni dati sperimentali che sono stati forniti dall'azienda i profili di forza da vincere e le corse di attuazione. Inoltre il collegamento meccanico dell'attuatore alla scatola cambio non può essere perfetto per via di tolleranze e giochi meccanici. Si possono considerare i risultati trovati soddisfacenti perché vedendo i grafici si nota comunque una netta similitudine negli andamenti sperimentali e simulativi.

Capitolo 7

Conclusioni

Dopo lo studio iniziale delle specifiche e del prototipo meccanico c'è stato un momento di confronto con i referenti aziendali; il filo conduttore fin dall'inizio del progetto è stato quello di realizzare un prototipo funzionante non solo dal punto di vista meccanico ma che fosse in grado di effettuare le movimentazioni nel modo corretto. Le richieste da parte dell'azienda non sono state incentrate sul costruire qualcosa che potesse essere subito industrializzabile, ma sul costruire un sistema che potesse essere un punto di partenza solido per una successiva fase di industrializzazione. Questo ha molto aiutato dal punto di vista dello sviluppo hardware e firmware perché ha permesso di non considerare delle variabili piuttosto importanti nel mondo automotive: le specifiche restrittive dovute ai requisiti di sicurezza, le funzionalità di diagnostica e i costi che devono essere piuttosto contenuti. I risultati ottenuti hanno dimostrato diversi successi per quanto riguarda l'esito del lavoro progettuale.

Uno dei primi obiettivi è stato quello di riuscire ad utilizzare una progettazione model based. La prima parte del progetto, che ha portato via molto tempo, si è incentrata fortemente sulla creazione di un modello Simulink che fosse il più accurato possibile, questo è stato ritenuto fondamentale per avere delle basi robuste per la progettazione. Sono state utilizzate efficacemente le

varie tecniche per creare un modello: ove possibile ci si è rifatti alle equazioni teoriche studiate nei vari corsi di studio o ricercate tra vari articoli scientifici e siti web specializzati. Altre volte è stato necessario improvvisare dei banchi prova non previsti per cercare di recuperare dei valori che altrimenti sarebbero stati impossibili da stimare. Per la costruzione del modello del motore molti parametri scritti nel datasheet erano imprecisi altri non presenti; c'è stato un lungo confronto telematico con l'azienda produttrice di motori elettrici che ha portato alla conoscenza di parametri che prima erano ignoti. Per quelli già presenti invece sono stati messi in atto ove possibile dei test; in alcuni casi hanno confermato i dubbi che erano emersi sulla veridicità dei dati presenti nel datasheet e hanno migliorato nettamente il modello. Un'altra parte su cui è stato impiegato molto lavoro è stata la modellizzazione e la stima degli attriti e delle varie forze in gioco. Anche in questo caso sono stati effettuati dei test sperimentali ove possibile oppure sono stati creati dei grafici di interpolazione da dati sperimentali forniti dall'azienda.

La validazione del modello e dell'algoritmo di controllo non sarebbe stata possibile se non si fosse realizzato un sistema hardware e firmware che riproducesse l'algoritmo di controllo messo a punto in Simulink. C'è stato un attento confronto con la collega che ha realizzato il sistema di controllo e un'analisi accurata delle varie tecniche utilizzabili e dei vari componenti necessari per mettere in atto ciò che era stato concordato. Prima di acquistare i vari componenti sono stati presi in considerazione pro e contro sia nelle caratteristiche sia nelle varie interfacce con gli altri componenti hardware. Grande attenzione è stata posta al rispetto delle specifiche di progetto, alle modalità, ai tempi necessari per poter utilizzare i vari componenti e ai costi. Dal punto di vista delle esigenze aziendali non c'erano delle grosse restrizioni di budget in quanto il progetto è ancora nelle fasi iniziali e i componenti utilizzati non dovevano avere il costo minore possibile. Nonostante ciò è stato interessante mostrare come, con l'acquisto di componenti a basso costo sia possibile realizzare un prototipo di qualità e funzionante secondo i requisiti

e in tempi relativamente brevi.

Nel complesso ci si può ritenere particolarmente soddisfatti perché i risultati sperimentali raccolti hanno mostrato una forte attinenza con quelli delle simulazioni fatte in Simulink. Questo dimostra non solo che il modello realizzato è ben fatto ma anche che il firmware scritto per il microcontrollore traduce in maniera fedele l'algoritmo di controllo PID. Infine il setup hardware messo a punto si è dimostrato abbastanza robusto e giustamente dimensionato rispetto alle richieste emerse durante la fase di simulazione. Le periferiche sono state configurate in maniera corretta e i dispositivi di attuazione e sensing giustamente connessi e abilitati dal microcontrollore.

Ultimo ma non per importanza è stato il feedback dei colleghi del reparto di ricerca e sviluppo che hanno dato opinioni positive non solo durante lo svolgimento del lavoro ma soprattutto a lavoro completato. A conclusione del lavoro progettuale in azienda sono stati presentati i risultati e le tecniche messe a punto al Presidente, CEO, Project Manager e R&D Manager; essi dopo aver appreso lo svolgimento e i risultati ottenuti si sono ritenuti molto soddisfatti del lavoro svolto e hanno reputato il progetto valido tanto da poter essere industrializzato e commercializzato in successivo momento.

7.1 Lavori futuri

Nonostante il lavoro realizzato sia completo dal punto di vista dell'obiettivo iniziale, ci sono diversi aspetti che potrebbero essere ancora approfonditi. Per quanto riguarda la parte di modellizzazione un punto su cui si potrebbe migliorare è quello della caratterizzazione del motore. Solitamente prima di utilizzare un motore elettrico si procede con una fase preliminare di caratterizzazione; essa richiede l'utilizzo di un banco prova specializzato, che spesso ha un costo molto elevato, o la delegazione di tale lavoro ad aziende specializzate.

Un altro punto su cui si può migliorare riguarda la generazione automatica del codice a partire dal modello Simulink. In un'ottica di aumento della produttività e della flessibilità del progetto impiegare uno sforzo maggiore nella creazione di un modello Simulink secondo specifici criteri può portare verso l'utilizzo del tool di generazione automatica del codice Embedded Coder della famiglia Matlab.

Dal punto di vista dei miglioramenti firmware, un aspetto su cui si potrebbe lavorare è sicuramente una maggiore integrazione dell'attuatore nell'ambiente automobile; questo comporta l'introduzione di una periferica CAN e l'aggiunta di particolari routine che permettano la ricezione dei comandi e l'invio di feedback di diagnosi alla centralina principale.

Un altro punto su cui è possibile apportare dei miglioramenti è quello dello sviluppo di un attuatore che sia in grado di interpretare i comandi P, R, N, D. Ove la meccanica e i driver lo permettano sarebbe solo necessario programmare il microcontrollore con una macchina a stati in grado di interpretare tutti i segnali disponibili e di attuare il motore nel giusto modo a seconda dei coefficienti PID scelti.

Infine in un'ottica di industrializzazione del prodotto è necessario apportare le modifiche ai componenti hardware acquistati, puntando sulla ridondanza e sulle specifiche che devono essere automotive compliant. Inoltre si rende necessaria l'integrazione di tutti i componenti su un unica PCB e la scrittura di codice che rispetti gli standard automotive.

Bibliografia

[1] *Sila Group*. https://www.grupposila.com/.

[2] *RS-557 data sheet*. Shenzen, China: Shenzhen Kinmore Motor Co.

[3] *STM32F411RE webpage*. https://www.st.com/en/microcontrollers-microprocessors/stm32f411re.html.

[4] *Come scegliere un sensore di posizione*. https://guide.directindustry.com/it/come-scegliere-un-sensore-di-posizione/.

[5] *PTL Series Slide Potentiometer data sheet*. California, USA: Bourns, Inc.

[6] *Come Effettuare Misure di Corrente Utilizzando un Trasduttore o un Sensore di Corrente*. https://dewesoft.com/it/daq/come-effettuare-misure-di-corrente-utilizzando-un-trasduttore-o-un-sensore-di-corrente.

[7] *ACS712ELC-20A data sheet*. New Hampshire, USA: Allegro MicroSystem, Inc.

[8] *Controlling Brushed DC Motors Using PWM*. https://www.machinedesign.com/materials/article/21125511/controlling-brushed-dc-motors-using-pwm.

[9] *BTS 7960 data sheet*. Neubiberg, Germany: Infinenon Technologies, AG.

Ringraziamenti

È mia volontà dedicare queste poche righe per mostrare la mia riconoscenza nei confronti delle persone che sono rimaste al mio fianco per la durata di questo percorso e che hanno contribuito a loro modo nel conseguimento di questo traguardo.

Un primo ringraziamento va all'azienda Silatech S.r.l. che ha permesso lo svolgimento di questo lavoro di tesi e ai responsabili del progetto, Massimiliano Gallo e Dennis Agnello.

Un ringraziamento speciale va al mio tutor aziendale Edoardo Mongarli; il suo sostegno giorno dopo giorno e i preziosi consigli dispensati hanno permesso l'ottenimento di brillanti risultati. Meritano inoltre menzione tutti i colleghi del reparto di ricerca e sviluppo: Mauro, Gaetano, Lino e Shob che mi hanno accolto con piacere nei loro uffici e sono stati sempre disponibili ad aiutarmi. A Federica, amica e collega di cui ho profonda stima, che ha collaborato con me in azienda; senza la sue conoscenze e il suo aiuto sarebbe stato impossibile portare a termine il progetto.

Vorrei inoltre ringraziare il mio relatore, il professor Luciano Lavagno, che ha accettato fin dal primo momento di appoggiarmi in questo lavoro ed è stato un ottimo riferimento accademico.

Non ci sono parole per descrivere la mia riconoscenza nei confronti della mia famiglia, mamma, papà e Fabrizio che da sempre sono dalla mia parte. Senza il loro appoggio non sarei mai potuto arrivare fin qui, la loro guida e i loro insegnamenti mi hanno dato la forza giorno dopo giorno per affrontare

tutte le difficoltà. Se oggi sono qui lo devo soprattutto a loro.

È mio desiderio dedicare questo traguardo ai miei nonni, Vito e Bice, che purtroppo non hanno potuto condividere con me questo lungo viaggio e Tonino e Lucia, che anche se dalla Puglia, sono sempre al mio fianco; i loro incoraggiamenti e il loro supporto sono stati linfa vitale durante tutto questo percorso.

A tutti i miei amici, infine, va la mia più sincera gratitudine; in particolare alla piccola famiglia torinese; ringrazio voi per avermi supportato, ma anche sopportato, in tutti questi anni. Gli infiniti momenti di gioia passati con voi li porterò sempre nel cuore.

Un ringraziamento speciale va, infine, al mio amico, compagno di classe fin dalla scuola media, collega di corso e coinquilino, Andrea; grazie per essere stato al mio fianco per tutto questo tempo; la tua presenza ogni giorno mi ha aiutato a scalare questa montagna e a raggiungerne la vetta. Spero di condividere con te in futuro altri mille successi.

A tutti voi, Grazie.