



Scuola Politecnica e delle Scienze di Base  
Corso di Laurea Triennale in Ingegneria Informatica

Tesi di Laurea in Tecnologie Informatiche per  
l'Automazione Industriale

***PROGETTO E IMPLEMENTAZIONE DI  
UN CONTROLLORE DIGITALE PER UN  
MOTORE CC***

Anno Accademico 2019/2020

Relatore  
**Ch.mo prof. Adriano Mele**  
Candidato  
**Giuseppe Francesco Di Cecio**

*Un giorno le macchine riusciranno  
a risolvere tutti i problemi,  
ma mai nessuna potrà porne uno.*

Albert Einstein

# Indice

<b>Introduzione</b>	<b>1</b>
<b>1 Modellistica</b>	<b>3</b>
1.1 Motore elettrico . . . . .	3
1.2 Sistema di controllo . . . . .	8
1.2.1 Regolatori PID . . . . .	9
1.2.2 Tecnica di taratura con le formule di inversione	11
<b>2 Hardware di base</b>	<b>15</b>
2.1 Attuatore . . . . .	15
2.1.1 Scheda tecnica . . . . .	16
2.1.2 Pulse-Width Modulation . . . . .	18
2.2 Trasduttore . . . . .	20
2.2.1 Scheda tecnica . . . . .	21
2.2.2 Comparatore . . . . .	22
2.3 Microcontrollore . . . . .	23
2.3.1 Generazione PWM . . . . .	24
2.3.2 Interfacciamento con l'encoder . . . . .	29
2.4 Motore elettrico . . . . .	31
2.5 Configurazione completa . . . . .	32

<b>3 Identificazione del modello matematico</b>	<b>35</b>
3.1 Posizione angolare . . . . .	36
3.2 Velocità angolare . . . . .	41
3.2.1 Digitalizzazione . . . . .	43
<b>4 Controllo digitale</b>	<b>48</b>
4.1 Regolatore PI . . . . .	50
4.1.1 Implementazione su microcontrollore . . . . .	52
4.2 Approssimazione al secondo ordine . . . . .	53
4.3 Approssimazione del primo ordine . . . . .	55
<b>Conclusioni</b>	<b>62</b>
<b>Bibliografia</b>	<b>63</b>

# Introduzione

I motori elettrici appartengono alla categoria delle macchine elettriche più utilizzate sia quotidianamente che in ambito industriale e di ricerca. Negli ultimi tempi si sta puntando molto sulle tecnologie innovative che permettono di ridurre l'inquinamento per avere un impatto ridotto sulla natura. Uno dei passi fondamentali dell'ingegneria è senza dubbio il passaggio dai motori a combustione interna ai motori elettrici. Essi non sono di certo una tecnologia recente, tuttavia il loro controllo è sempre stato molto complesso. Il grande passo che ha reso possibile l'utilizzo dei motori elettrici in moltissime applicazioni come automobili, aeronautica, robotica è stato sicuramente l'avvento dell'elettronica digitale.

In tale contesto si è scelto di realizzare un prototipo di controllo PID digitale per un motore a corrente continua sfruttando i concetti teorici assunti durante il percorso di studi triennale. Per la loro semplicità, i regolatori PID possono essere realizzati con le tecnologie più varie: meccanica, pneumatica, idraulica, elettronica analogica e digitale. Questo implica una grande disponibilità commerciale, che permette la realizzazione in tempi brevi e con costi contenuti. Il controllo in forma digitale mette in risalto le potenzialità, sia in termini di efficienza che in termini economici, che offre questo tipo di implementazione.

Il motore è una macchina elettrica analogica, deve essere quindi accoppiata con un apposito trasduttore ed attuatore che ne applicano la conversione analogico/digitale durante il controllo.

L'attuatore utilizzato è il *TB66112FNG* della Toshiba, si occupa di trasformare un segnale di ingresso digitale in un segnale di pilotaggio analogico per il motore in questione. Il segnale digitale in ingresso deve essere un segnale PWM (Pulse-Width Modulation).

Il trasduttore utilizzato è un trasduttore ottico. E' un elemento di elevata importanza in quanto permette di attuare una retroazione al sistema. Permette la conversione della posizione angolare della mac-

china in un segnale digitale.

I segnali digitali dunque devono essere opportunamente elaborati e generati da una scheda digitale.

La scheda di controllo utilizzata è la *STM32F303Discovery* della ST-Microelectronics con un microcontrollore incorporato che si occupa di eseguire l'algoritmo necessario al controllo della macchina elettrica. Tale algoritmo è sintetizzato a partire da uno studio teorico dei componenti aventi a disposizione, per poi essere tradotto in un opportuno linguaggio di programmazione per permettere l'esecuzione sulla scheda digitale.

In questo lavoro di tesi sono presentati i vari strumenti necessari (teorici e pratici) per portare a termine l'obbiettivo.

Il primo capitolo riguarda principalmente i concetti teorici necessari per la realizzazione e l'implementazione del controllore. Nel secondo capitolo vengono analizzati dettagliatamente gli strumenti hardware che si hanno a disposizione. Nel terzo capitolo si applicano le conoscenze di sistemi dinamici per identificare un modello matematico del motore che si ha a disposizione. Il quarto capitolo, invece, descrive il controllo vero e proprio.

Lo scopo di questo elaborato è fornire una procedura di progettazione principalmente per una macchina a corrente continua, ma i concetti possono essere estesi per un qualunque impianto si voglia controllare, che sia esso meccanico, elettrico, elettro-meccanico, termico o idraulico.

# Capitolo 1

## Modellistica

### 1.1 Motore elettrico

Il motore in corrente continua rientra nell'ambito dei motori elettrici utilizzati per gli azionamenti di tipo domestico ed industriale. Il motivo del suo grande successo è che, a differenza di altri tipi di motori, il suo controllo è estremamente semplice ed efficace.

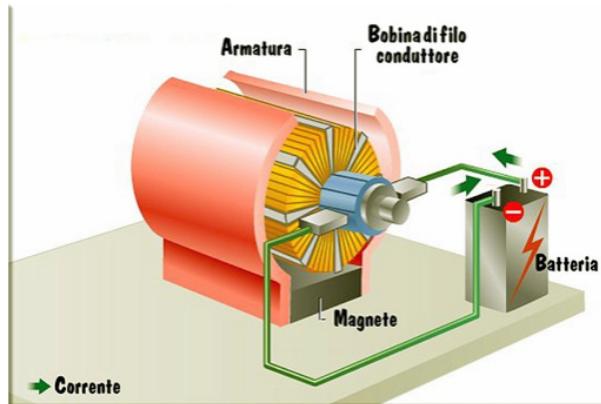


Figura 1.1: *Principio di funzionamento di una macchina a corrente continua*

La parte rotante del motore è detta appunto *rotore*. La parte esterna statica è detta *statore*. Il rotore è composto da avvolgimenti di rame i quali generano un campo elettromagnetico al passaggio della corrente. Esso è immerso in un campo magnetico generato dallo stator e l'interazione tra i due campi introduce la rotazione. Il campo magnetico interno può essere generato da magneti permanenti situati sullo stator o da altri avvolgimenti di conduttore che lo generano con il

passaggio di corrente elettrica.

Per comprenderne il funzionamento si parte con la forza di Lorentz.

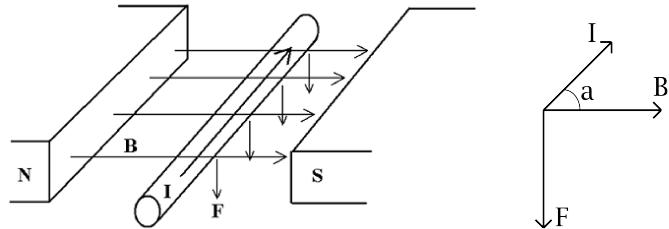


Figura 1.2: *Filo immerso perpendicolarmente in un campo magnetico*

Un conduttore di lunghezza  $l$  immerso in un campo magnetico  $B$  con un'angolazione  $\alpha$  e percorso da una corrente  $I$  è soggetto ad una forza [8]

$$\vec{F} = I \vec{l} \times \vec{B} = I |l| |B| \sin(\alpha) \quad (1.1)$$

In cui  $\alpha$  è l'angolo formato dal vettore lunghezza (con verso riferito al senso di percorrenza della corrente e direzione riferita all'orientamento del filo nello spazio) e dal vettore campo magnetico.

Una spira rettangolare immersa in un campo magnetico (Fig 1.3) ha due lati perpendicolari ( $\alpha = 90^\circ$ ) e due paralleli ( $\alpha = 0^\circ$ ) al campo. Applicando la 1.1 ad ognuno di essi si ottengono due forze uguali ed opposte a due a due. Definendo  $r$  la distanza tra un lato e il centro della spira si può definire un momento torcente:

$$\vec{\tau} = \vec{r} \times \vec{F} = |r| |F| \sin(\beta)$$

In cui  $\beta$  è l'angolo formato dal vettore braccio e dal vettore forza.

$\vec{\tau}$  è nullo quando il braccio  $\vec{r}$  e la forza  $\vec{F}$  sono paralleli ( $\beta = 0^\circ$ ).

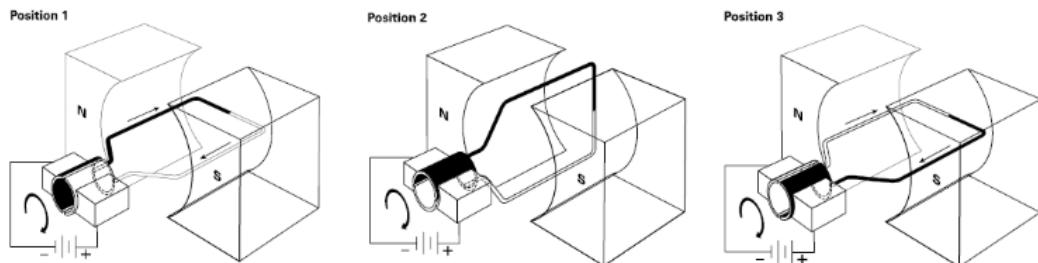


Figura 1.3: *Rotazione completa di una spira*

Quando  $\beta = 0^\circ$  il momento torcente è nullo, ma grazie all'inerzia del rotore tale angolo sarà per un'istante diverso da zero. Se in questo istante si inverte il verso della corrente, la spira risente di un nuovo momento torcente di verso opposto al precedente. Tale soluzione si effettua anteponendo un anello diviso a metà in modo che l'alimentazione si inverte dopo la rotazione di  $90^\circ$ . L'anello prende il nome di *anello di Pancinotti*.

Un motore elettrico tradizionale è composto da  $N$  spire, quindi il ragionamento precedente viene replicato su ognuna di esse. L'anello, di conseguenza, è diviso in  $N$  settori e prende il nome di *collettore*.

Nella Fig.1.3 il momento torcente che agisce nel mettere in rotazione il motore prende il nome di *coppia*.

In generale la coppia generata da un insieme di spire vale [3]:

$$C = K_a \cdot \phi \cdot I$$

In cui  $K_a$  rappresenta una costante che dipende dai parametri costruttivi del motore e  $\phi$  è il flusso del campo magnetico, concatenato con gli avvolgimenti, generato dallo statore con magneti permanenti.

Quando il rotore ruota, l'angolazione degli avvolgimenti con il flusso di campo magnetico varia nel tempo, generando per la legge di Faraday-Neumann-Lenz una forza elettromotrice indotta a capi del motore [3]:

$$E = K_e \cdot \phi \cdot \omega$$

In cui  $K_e$  è anch'essa una costante che dipende dalla costituzione della macchina, e  $\omega$  è la velocità angolare del rotore.

Intuitivamente si può rappresentare un modello elettromeccanico del motore per impostare altre equazioni che ne governano la dinamica.

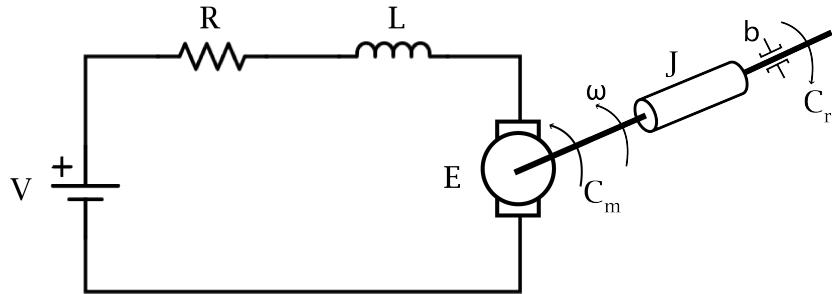


Figura 1.4: *Modello elettromeccanico del motore elettrico in corrente continua con magneti permanenti*

Supponendo che il flusso magnetico generato dallo statore sia costante e dovuto a magneti permanenti si ha:

$$\begin{cases} V = RI + L \frac{dI}{dt} + E \\ C_m = K_a \phi I \\ C_m = b\omega + J \frac{d\omega}{dt} + C_r \\ E = K_e \phi \omega \end{cases} \quad (1.2)$$

$V$ Tensione di alimentazione	$R$ Resistenza degli avvolgimenti
$L$ Induttanza degli avvolgimenti	$I$ Corrente nel circuito
$E$ Forza contro-elettromotrice	$J$ Momento di inerzia
$C_m$ Coppia motrice	$C_r$ Coppia resistente
$\omega$ Velocità angolare	$b$ Coefficiente di attrito viscoso

Per rappresentare il sistema dinamico in forma ISU si scelgono le grandezze fisiche che rappresentano lo stato, l'ingresso e l'uscita.  
 $\omega$  e  $I$  sono le uniche grandezze differenziate, dunque rappresentano lo stato del sistema.  $V$  e  $C_r$  rappresentano l'ingresso. L'uscita potrebbe essere scelta  $\omega$ .

$$\begin{aligned} u_1 &= I, \quad u_2 = \omega \quad \Rightarrow \quad \mathbf{x} = [I \quad \omega]^T \\ x_1 &= I, \quad x_2 = \omega \quad \Rightarrow \quad \mathbf{u} = [V \quad C_r]^T \\ &\quad \quad \quad y = \omega = x_1 \end{aligned}$$

Il sistema ISU diventa:

$$\begin{cases} \dot{x}_1 = -\frac{R}{L}x_1 - \frac{K_e\phi}{L}x_2 + \frac{1}{L}u_1 \\ \dot{x}_2 = \frac{K_a\phi}{J}x_1 - \frac{b}{J}x_2 - \frac{1}{J}u_2 \\ y = x_2 \end{cases}$$

In forma matriciale:

$$\begin{cases} \dot{\mathbf{x}} = \begin{bmatrix} -\frac{R}{L} & -\frac{K_e\phi}{L} \\ \frac{K_a\phi}{J} & -\frac{b}{J} \end{bmatrix} \mathbf{x} + \begin{bmatrix} \frac{1}{L} & 0 \\ 0 & -\frac{1}{J} \end{bmatrix} \mathbf{u} \\ y = \begin{bmatrix} 0 & 1 \end{bmatrix} \mathbf{x} \end{cases} \quad (1.3)$$

$$\mathbf{A} = \begin{bmatrix} -\frac{R}{L} & -\frac{K_e\phi}{L} \\ \frac{K_a\phi}{J} & -\frac{b}{J} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} \frac{1}{L} & 0 \\ 0 & -\frac{1}{J} \end{bmatrix} \quad \mathbf{C} = [0 \ 1] \quad \mathbf{D} = 0$$

Avendo il modello ISU si può anche ricavare la funzione di trasferimento  $W(s)$ , nel dominio di Laplace con variabile complessa  $s$ , ricordando:

$$\mathbf{W}(s) = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}$$

Essa sarà un vettore di due funzioni di trasferimento poiché il sistema è stato modellato con due ingressi e una sola uscita.

Eseguendo semplici moltiplicazioni tra matrici si ottiene:

$$\mathbf{W}(s) = \begin{bmatrix} \frac{K_a\phi}{LJ} & -\frac{1}{J}(s + \frac{R}{L}) \\ (s + \frac{R}{L})(s + \frac{b}{J}) + \frac{K_e K_a \phi^2}{LJ} & (s + \frac{R}{L})(s + \frac{b}{J}) + \frac{K_e K_a \phi^2}{LJ} \end{bmatrix}$$

Nell'elaborato è assunto  $C_r$  trascurabile. La funzione di trasferimento è quella associata solo all'ingresso  $u_1 = V$ :

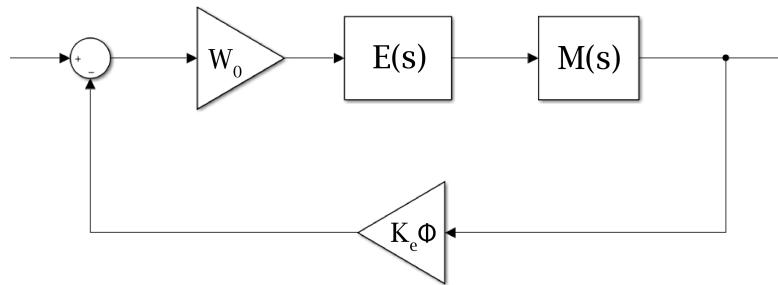
$$W(s) = \frac{\frac{K_a\phi}{LJ}}{(s + \frac{R}{L})(s + \frac{b}{J}) + \frac{K_e K_a \phi^2}{LJ}} \quad (1.4)$$

In realtà  $W(s)$  ricorda molto un'espressione di una funzione di trasferimento a ciclo chiuso. Se infatti si mette in evidenza il fattore

$(s + \frac{R}{L})(s + \frac{b}{J})$  si ottiene (ponendo  $W_0 = \frac{K_a \phi}{LJ}$ ):

$$W(s) = \frac{\frac{W_0}{(s + \frac{R}{L})(s + \frac{b}{J})}}{1 + K_e \phi \frac{W_0}{(s + \frac{R}{L})(s + \frac{b}{J})}}$$

Si notano immediatamente le funzioni di trasferimento sulla linea di azione e retroazione di un sistema con feedback.



$$W_0 = \frac{K_a \phi}{LJ} \quad E(s) = \frac{1}{(s + \frac{R}{L})} \quad M(s) = \frac{1}{(s + \frac{b}{J})}$$

con  $E(s)$  associata alla parte elettrica e  $M(s)$  associata alla parte meccanica. Come si può notare la retroazione dovuta dalla forza elettromotrice indotta dalla rotazione, introduce una retroazione negativa che ha un effetto stabilizzante sul sistema complessivo.

Per le finalità di controllo dell'elaborato è possibile utilizzare un'approssimazione a singolo polo dominante.

## 1.2 Sistema di controllo

Il progetto di un sistema di controllo in retroazione si riconduce alla progettazione di un regolatore in modo che il sistema retroazionato possieda determinate caratteristiche: tempo di assestamento, sovraelongazione, reiezione ai disturbi, stabilità.

Il tempo di assestamento è il tempo necessario perché la risposta entri in una certa fascia vicina al valore di regime (in genere si fa riferimento a scostamenti del 1% o 5%) senza più uscirne. Se la risposta durante il transitorio supera il valore di regime si ha una sovraelongazione. Essa fa riferimento al valore massimo percentuale rispetto al regime.

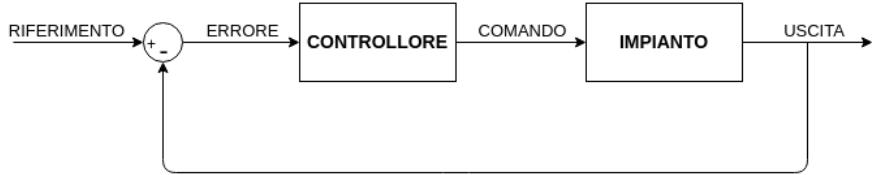
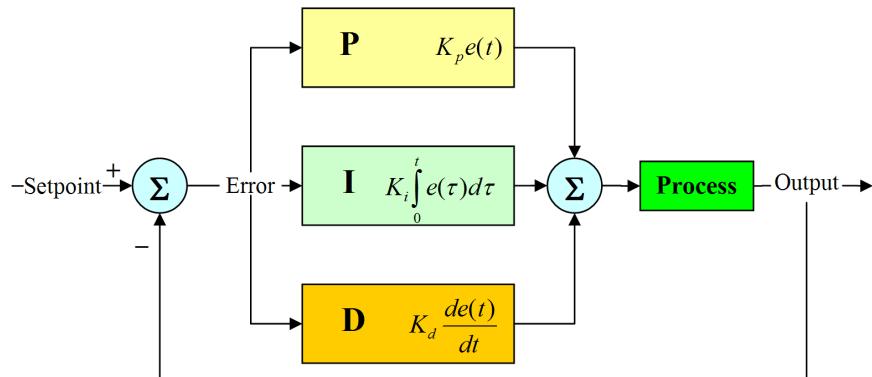


Figura 1.5: *Sistema di riferimento per la progettazione di un controllore*

### 1.2.1 Regolatori PID

Sono i regolatori lineari più utilizzati in ambito industriale. Essi compiono tre azioni di controllo: Proporzionale, Integrale e Derivativa (da cui l'acronimo PID) [7].



#### Azione Proporzionale

$$u(t) = K_p e(t)$$

L'azione di controllo dipende proporzionalmente dall'errore  $e(t)$ . Maggiore è l'errore maggiore è l'azione svolta dal controllore. Un regolatore dotato solo di azione proporzionale può aumentare la velocità risposta del sistema, e con un guadagno elevato, può diminuire la stabilità [7].

#### Azione Integrale

$$u(t) = K_i \int_0^t e(t) dt$$

E' particolarmente importante nelle applicazioni con riferimento a gradino, poiché garantisce errore nullo a regime. In generale l'azione integrale è associata all'azione proporzionale costituendo un regolatore

PI. Essi vengono utilizzati quando è richiesto un errore nullo a regime e una buona velocità di risposta. [7].

### Azione Derivativa

$$u(t) = K_d \frac{de(t)}{dt}$$

L'uscita del regolatore dipende dalla velocità con cui varia l'errore. Il controllore risulta molto sensibile alle variazioni del riferimento. Anch'esso può essere associato ad un'azione proporzionale costituendo un regolatore PD. E' particolarmente indicato per sistemi che hanno un notevole ritardo di fase; l'azione derivativa infatti fornisce un anticipo di  $90^\circ$  in fase (è detta anche azione anticipatrice) [7]. In generale la sola azione derivativa non è fisicamente realizzabile. E' utile dunque aggiungere un polo in alta frequenza che ne consente la realizzabilità. L'aggiunta del polo può anche portare alcuni benefici sullo smorzamento del rumore in uscita dal filtro.

In generale dunque:

$$u(t) = K_p e(t) + K_i \int_0^t e(t) dt + K_d \frac{de(t)}{dt} \quad (1.5)$$

Passando al dominio di Laplace si ha:

$$\begin{aligned} U(s) &= K_p E(s) + \frac{K_i}{s} E(s) + K_d s E(s) \\ K(s) &= \frac{U(s)}{E(s)} = K_p + \frac{K_i}{s} + K_d s \end{aligned} \quad (1.6)$$

A questo punto l'unico problema è determinare i valori di  $K_p$ ,  $K_i$  e  $K_d$  in modo che il sistema a ciclo chiuso rispetti le specifiche.

Esistono varie tecniche per la taratura dei parametri. In questo elaborato si è scelto di utilizzare la tecniche di taratura con le formule di inversione.

## 1.2.2 Tecnica di taratura con le formule di inversione

Questa tecnica si basa sulla conoscenza del modello matematico dell'impianto da controllare. In questo modo i parametri saranno in funzione delle proprietà dell'impianto.

Nell'elaborato si è scelto di implementare solo un controllore PI senza azione derivativa, per cui verranno descritte le tecniche di taratura per questo tipo di controllore. Supponendo che  $G(s)$  rappresenti la funzione di trasferimento dell'impianto e  $K(s)$  la funzione di trasferimento del controllore. Valutando  $s = j\omega$ :

$$K(s) = K_p + \frac{K_i}{s} \Rightarrow K(j\omega) = K_p + \frac{K_i}{j\omega} = K_p - \frac{jK_i}{\omega} \quad (1.7)$$

$G(s)$  e  $K(s)$  si possono rappresentare in termini di modulo e fase.

$$G(j\omega) = |G(j\omega)|e^{j\angle G(j\omega)} \quad K(j\omega) = |K(j\omega)|e^{j\angle K(j\omega)} \quad (1.8)$$

La funzione:

$$F(s) = G(s)K(s) = |G(j\omega)||K(j\omega)|e^{j\angle(G(j\omega)+K(j\omega))} \quad (1.9)$$

è la funzione di anello. Si possono definire alcune proprietà su  $F(s)$  che si ripercuotono sulla funzione a ciclo chiuso. L'idea è quindi quella di impostare le specifiche su  $F(s)$  in modo da ricavare i coefficienti  $K_p$  e  $K_i$  tali da rispettarle. Fissando una pulsazione  $\omega_c$  si ha il corrispettivo valore di modulo e fase di  $F(s)$ . Dalla 1.9:

$$|F(j\omega_c)| = F_0 \Rightarrow |G(j\omega_c)||K(j\omega_c)| = F_0$$

$$|K(j\omega_c)| = \frac{F_0}{|G(j\omega_c)|} \quad (1.10)$$

Analogamente per la fase:

$$\angle F(j\omega_c) = P \Rightarrow \angle G(j\omega_c) + \angle K(j\omega_c) = P$$

$$\angle K(j\omega_c) = P - \angle G(j\omega_c) \quad (1.11)$$

La 1.8 si può scrivere, applicando la formula di Eulero come:

$$K(j\omega_c) = |K(j\omega_c)| \cos(\angle K(j\omega_c)) + j|K(j\omega_c)| \sin(\angle K(j\omega_c)) \quad (1.12)$$

Eguagliando la 1.7 con la 1.12 e sostituendo le 1.10 e 1.11 si ottiene:

$$\begin{cases} K_p = \frac{F_0}{|G(j\omega_c)|} \cos(P - \angle G(j\omega_c)) \\ K_i = -\frac{F_0 \omega_c}{|G(j\omega_c)|} \sin(P - \angle G(j\omega_c)) \end{cases} \quad (1.13)$$

Bisogna dunque fissare un valore di  $\omega_c$  tale che la funzione di trasferimento a ciclo chiuso rispetti dei particolari requisiti.

$$W(s) = \frac{F(s)}{1 + F(s)}$$

Può essere approssimata ad una funzione del primo o secondo ordine (con poli complessi e coniugati)  $W_a(s)$  in relazione al margine di fase  $\varphi_m$  di  $F(s)$  [7]. Fissando  $\omega_c$  tale che  $F(j\omega_c) = 0dB$  ( $\omega_c$  è detta *crossing frequency*) si ha:

$$\begin{cases} W_a(s) = \frac{1}{1 + \frac{2\zeta}{\omega_c} s + \frac{s^2}{\omega_c^2}} & \varphi_m < 60^\circ \\ W_a(s) = \frac{1}{1 + \frac{s}{\omega_c}} & \varphi_m > 60^\circ \end{cases} \quad (1.14)$$

In questo si può scegliere una fase  $P$  (1.11) tale che  $\varphi_m = 180^\circ + P$  approssimi  $W_a(s)$  all'ordine desiderato. Si devono distinguere le due casistiche di approssimazione.

### Approssimazione del secondo ordine

$$\varphi_m = 180^\circ + P < 60^\circ \Rightarrow P < -120^\circ$$

Il tempo di assestamento [7] di una funzione del secondo ordine con poli complessi e coniugati vale:

$$\begin{cases} t_{5\%} = \frac{3}{\zeta \omega_{-3dB}} & \zeta \ll 1 \\ t_{5\%} = \frac{4.75}{\omega_{-3dB}} & \zeta \simeq 1 \end{cases}$$

La pulsazione  $\omega_{-3dB}$  rappresenta la banda del sistema  $W_a(s)$ . Se  $F(s)$  è a stabilità regolare vale la relazione  $\omega_{-3dB} \simeq \omega_c$  per cui si può legare la crossing frequency al tempo di assestamento a ciclo chiuso [7]:

$$\begin{cases} \omega_c = \frac{3}{\zeta t_{5\%}} & \zeta \ll 1 \\ \omega_c = \frac{4.75}{t_{5\%}} & \zeta \simeq 1 \end{cases} \quad (1.15)$$

Ricordando che  $\zeta \simeq \frac{\varphi_m}{100}$  [7].

In questo tipo di approssimazione sono presenti anche oscillazioni e sovraelongazioni. In particolare la sovraelongazione percentuale vale [7].

$$s = e^{\frac{-\pi\zeta}{\sqrt{1-\zeta^2}}} \quad (1.16)$$

### Approssimazione del primo ordine

$$\varphi_m = 180^\circ + P > 60^\circ \Rightarrow P > -120^\circ$$

Il tempo di assestamento a ciclo chiuso vale:

$$t_{5\%} = \frac{3}{\omega_{-3dB}}$$

Le considerazioni fatte nell'approssimazione del secondo ordine continuano a valere, per cui:

$$\omega_c = \frac{3}{t_{5\%}} \quad (1.17)$$

In entrambi i casi  $F_0 = 0dB = 1$ , la 1.13 si può riscrivere:

$$\begin{cases} K_p = \frac{1}{|G(j\omega_c)|} \cos(P - \angle G(j\omega_c)) \\ K_i = -\frac{\omega_c}{|G(j\omega_c)|} \sin(P - \angle G(j\omega_c)) \end{cases} \quad (1.18)$$

# Capitolo 2

## Hardware di base

Il sistema di controllo per agire sull’impianto ha bisogno di un attuatore, per pilotare il sistema tramite segnali digitali, e un trasduttore, per ricavare l’uscita dell’impianto sottoforma di segnale digitale. Un primo schema logico da prendere in considerazione è il seguente:



Figura 2.1: *Rappresentazione schematica dei componenti necessari*

### 2.1 Attuatore

L’attuatore utilizzato è un driver (Fig.2.2) che trasforma una segnale digitale in ingresso in un segnale analogico in uscita per pilotare il motore. Come visto durante la fase di modellistica l’uscita deve essere una tensione, che rappresenterà l’ingresso del motore.

Esso si basa su un circuito elettronico a transistor che permette di selezionare sia la tensione in uscita che la direzione di rotazione del motore. In particolare permette di gestire al più due motori indipendentemente.

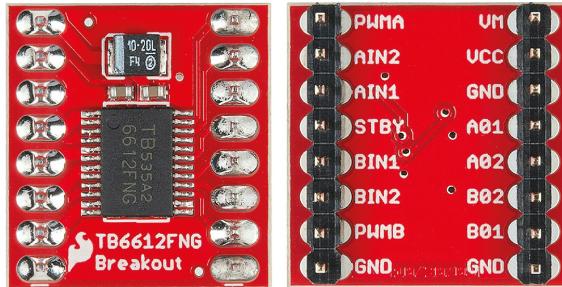


Figura 2.2: *Toshiba TB66112FNG*

La figura 2.2 mostra a sinistra l’insieme dei pin di input per poter controllare il motore *A* e il motore *B*, mentre a destra l’insieme di pin di uscita, compresa alimentazione *Vm* e *GND* del motore.

### 2.1.1 Scheda tecnica

I parametri massimi da rispettare (sia in input che di output) sono resi noti da un datasheet messo a disposizione dall’azienda Toshiba Fig 2.3

Characteristics	Symbol	Min	Typ.	Max	Unit	Remarks
Supply voltage	Vcc	2.7	3	5.5	V	
	VM	4.5	5	13.5	V	
Output current (H-SW)	Iout	---	---	1.0	A	VM ≥ 5V
		---	---	0.4		5V > VM ≥ 4.5V
Switching frequency	fPWM	---	---	100	kHz	

Figura 2.3: *Parametri operativi da rispettare per l’utilizzo del driver [1]*

I limiti massimi imposti sono dunque ben descritti nella tabella, in particolare il motore da pilotare non deve assorbire più di 1.0 A e non deve essere alimentato con più di 13.5 V.

Per capire il corretto funzionamento del driver bisogna analizzarne il circuito elettrico all’interno. Anche in questo caso il datasheet mette a disposizione uno schema elettrico che lo descrive.

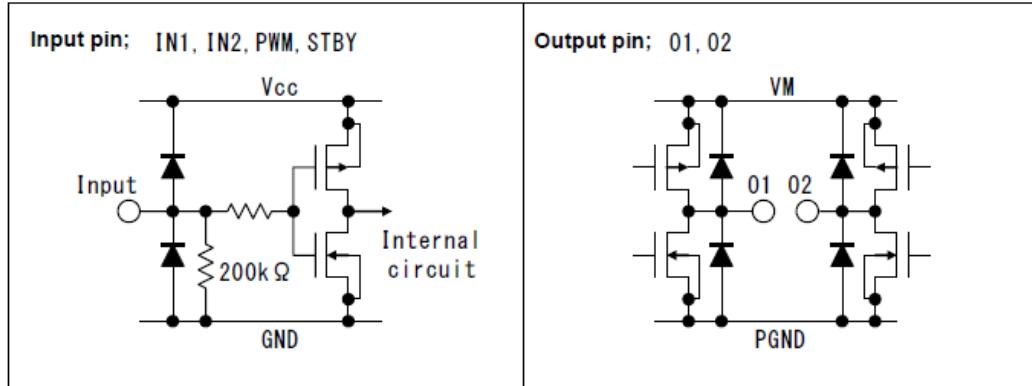


Figura 2.4: Circuito riferito al pilotaggio di un motore [1]

Si può notare che negli stadi alti del circuito in Fig. 2.4 vi è la presenza di P-MOS e mentre negli stadi bassi la presenza di N-MOS. Per pilotare un solo motore c'è bisogno di due ingressi *IN1* e *IN2*. Un input viene collegato ai gate di entrambi i MOS e con la configurazione descritta si possono avere due scenari (dato che l'ingresso è digitale può avere solo due valori, 1 o 0):

1. *IN* ALTO. P-MOS interdetto e N-MOS in conduzione.
2. *IN* BASSO. P-MOS in conduzione e N-MOS interdetto.

La combinazione dei due ingressi genera quindi quattro scenari possibili.

Nella figura 2.5 viene illustrato in rosso il MOS in conduzione e in verde il percorso del circuito equivalente se i morsetti di output fossero collegati al motore. In tal caso è facile intuire che solo in due casi si può selezionare la direzione del motore, quelli in cui c'è differenza di potenziale. Quindi i pin di selezione *IN1* e *IN2* devono essere di livello logico diverso.

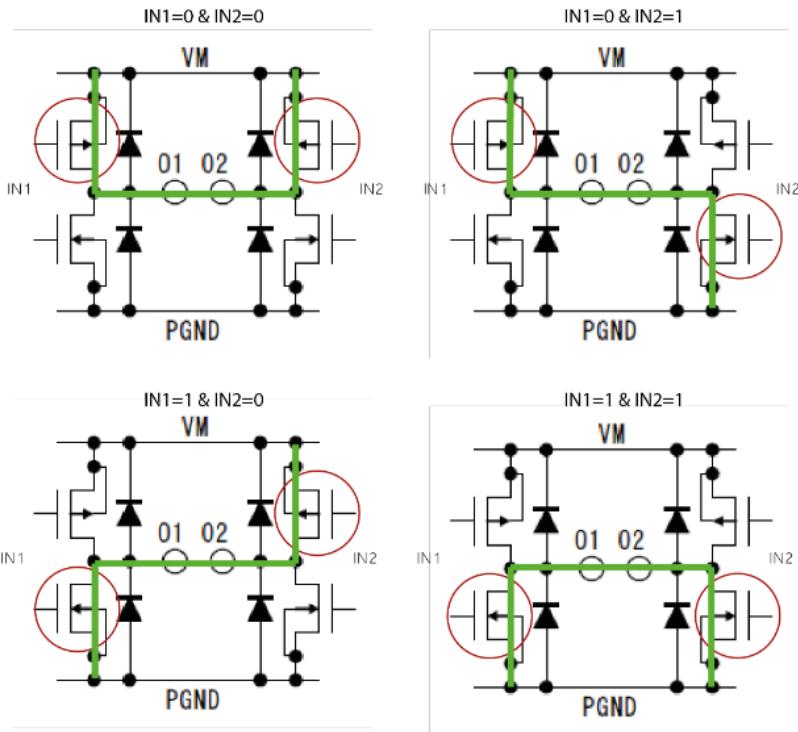


Figura 2.5: Configurazioni per la direzione del motore

### 2.1.2 Pulse-Width Modulation

Il driver precedentemente descritto prevede in ingresso, come segnale di pilotaggio, un segnale modulato a "larghezza di impulso" (dall'inglese Pulse-Width Modulation o PWM).

La modulazione della larghezza di impulso utilizza un'onda quadra la cui durata di impulso e il suo periodo vengono appositamente prefissati, con conseguente variazione del valore medio. Se il periodo è posto in modo che la frequenza del segnale sia abbastanza elevata e l'impianto da pilotare ha un comportamento passa-basso allora l'onda quadra è assimilabile a un segnale costante di valore uguale al valore medio.

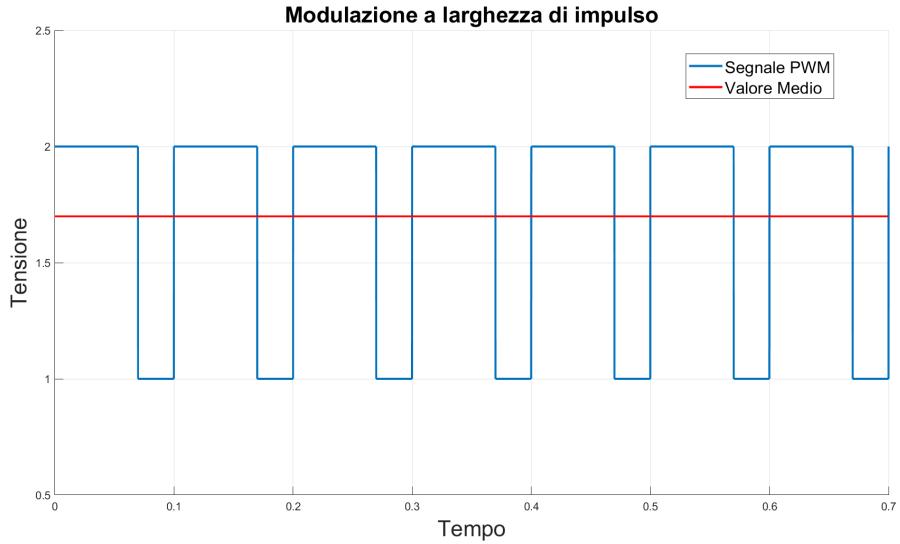


Figura 2.6: Esempio segnale PWM e il suo valor medio

Sia  $x(t)$  un segnale periodico di periodo  $T_0$ . Il valor medio di  $x(t)$  è dato da:

$$\bar{x} = \frac{1}{T_0} \int_0^{T_0} x(t) dt \quad (2.1)$$

L'integrale è esteso a un solo periodo del segnale.

Ponendo  $0 < D < 1$ ,  $DT_0$  rappresenta una frazione del periodo.

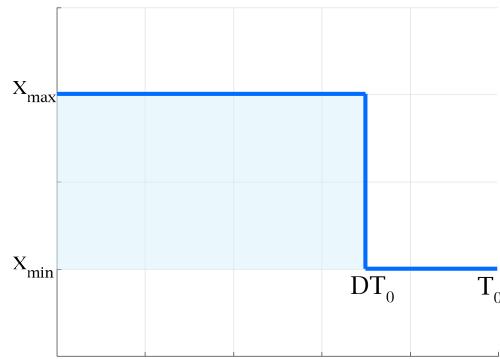


Figura 2.7: Visualizzazione di un periodo di  $x(t)$

La 2.1 diventa:

$$\bar{x} = \frac{1}{T_0} \int_0^{D \cdot T_0} x_{max} dt + \frac{1}{T_0} \int_{D \cdot T_0}^{T_0} x_{min} dt$$

$$\bar{x} = \frac{1}{T_0} (D \cdot T_0 \cdot x_{max}) + \frac{1}{T_0} (T_0 - D \cdot T_0) x_{min}$$

$$\bar{x} = Dx_{max} + (1 - D)x_{min}$$

La scelta di  $D$  è dunque relazionata al valor medio che si vuole ottenere:

$$\bar{x} = D(x_{max} - x_{min}) + x_{min}$$

$x(t)$  è un segnale di tensione e in questo elaborato viene posto  
 $x_{max} \simeq 3V$  e  $x_{min} \simeq 0V$

$$\bar{x} = Dx_{max} \quad (2.2)$$

$$D = \frac{\bar{x}}{x_{max}} \quad (2.3)$$

Il parametro D prende il nome di **duty cycle** [2].

A questo punto si può creare una tabella che descrive pienamente l'utilizzo del driver, almeno per un motore:

Pin	Descrizione
AIN1	Primo ingresso per la selezione della direzione Fig. 2.5
AIN2	Secondo ingresso per la selezione della direzione Fig. 2.5
PWMA	Segnale di pilotaggio Fig. 2.6
VM	Morsetto positivo dell'alimentazione del motore Fig. 2.3
VCC	Morsetto positivo dell'alimentazione del driver Fig. 2.3
GND	Morsetto negativo dell'alimentazione del motore e del driver
AO1	Morsetto del motore
AO2	Morsetto del motore

Il segnale PWM in ingresso è utilizzato per generare una stessa onda quadra in uscita, con tensione di picco diversa pari a VM, ma con lo stesso duty cycle. Il risultato è che si può scegliere la tensione continua per alimentare il motore scegliendo il duty cycle in ingresso, utilizzando la formula 2.3 (ponendo  $x_{max} = 3V$  in ingresso e  $x_{max} = VM$  in uscita) e 2.2 (analogamente).

Dunque per pilotare il motore bisogna agire sul duty-cycle.

## 2.2 Trasduttore

Il trasduttore che si è utilizzato nell'esperimento è un encoder ottico descritto in figura 2.8

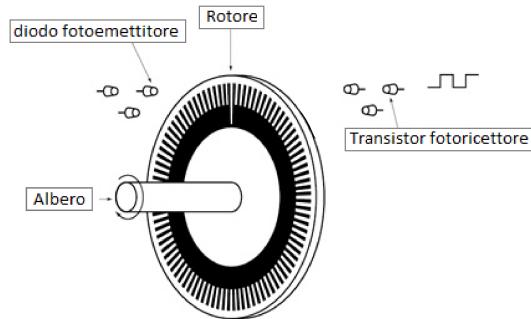


Figura 2.8: *Principio di funzionamento del dispositivo*

Un disco con un numero finito di fessure viene collegato all'albero motore. Durante la rotazione il disco interrompe continuamente il fascio laser e il transistor fotoricettore riceverà il laser ad intermittenza. In questo elaborato si è utilizzato un encoder ottico con a bordo un LM393 (Comparatore elettronico).

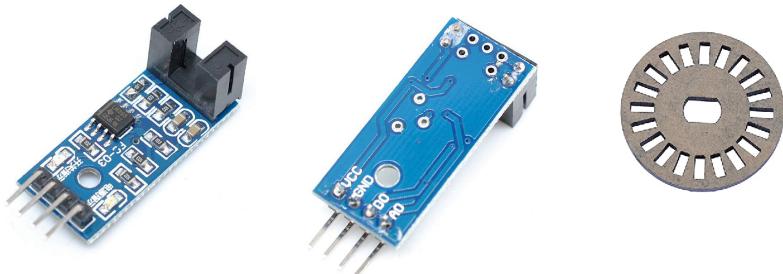


Figura 2.9: *Encoder*

### 2.2.1 Scheda tecnica

Per il dispositivo in questione non esiste un vero e proprio datasheet come con il driver, ma ci si può affidare ai documenti allegati al prodotto acquistato. Innanzitutto per capirne il funzionamento bisogna analizzarne il circuito elettronico.

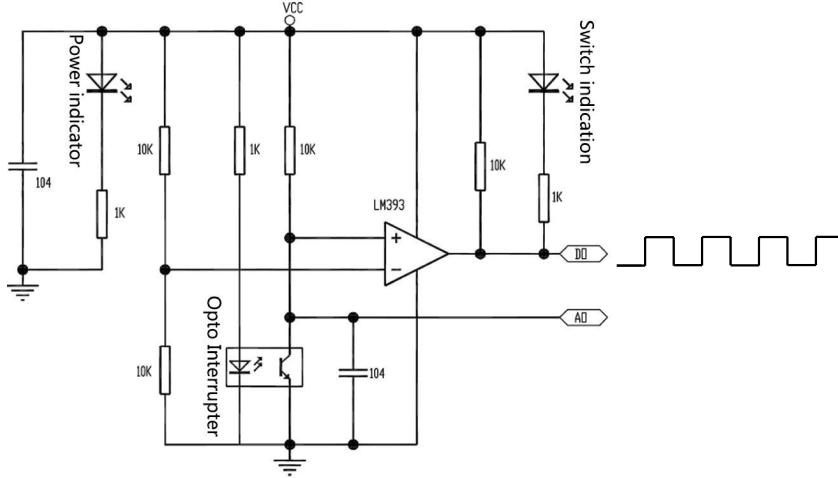


Figura 2.10: *Schema elettrico*

Il fulcro di questo schema è l'amplificatore operazionale in configurazione di comparatore. Quando il fascio laser non raggiunge il transistor fotoricettore, quest'ultimo è in interdizione. Il comparatore confronta quindi una tensione di riferimento pari a  $VCC/2$  (sul morsetto negativo) con  $VCC$  restituendo quindi uscita alta. In caso opposto il confronto avviene tra  $VCC/2$  e la caduta di tensione sul BJT (circa 0) restituendo un'uscita bassa.

Seguendo il ragionamento durante la rotazione della ruota forata l'uscita del sensore è un onda quadra, la cui frequenza è legata alla frequenza di rotazione della ruota (ovvero la velocità del motore).

Con l'encoder in dotazione si hanno 20 fessure. Un periodo dell'onda quadra è legato alla distanza angolare tra una fessura e un'altra, dunque:

$$dx = 2 \cdot \frac{2\pi}{20} = 0.6283 \text{ rad} = 36 \text{ deg} \quad (2.4)$$

La 2.4 rappresenta la **risoluzione** del dispositivo.

## 2.2.2 Comparatore

Durante la sperimentazione si è notato che l'uscita dell'encoder è molto rumorosa, soprattutto in presenza del valore logico alto. Per evitare false misurazioni da parte del microcontrollore si è scelto di porre l'uscita in un comparatore esterno in modo che piccole oscillazioni intorno ai valori stazionari del segnale vengano filtrate.

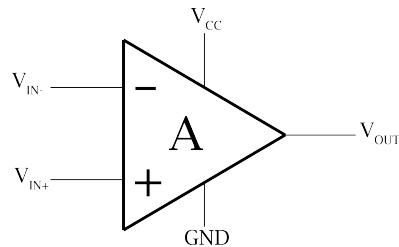


Figura 2.11: *Amplificatore operazionale*

La legge, ideale, che descrive il comportamento dell'amplificatore operazionale, considerando un guadagno  $A$  molto grande [5], è:

$$V_{out} = A(V_{in+} - V_{in-}) \quad (2.5)$$

Se quindi si fissa la  $V_{in-}$  ad un valore  $V_{ref}$  succede che:

$$\begin{cases} V_{out} = V_{cc} & \text{se } V_{in+} > V_{ref} \\ V_{out} = 0 & \text{se } V_{in+} < V_{ref} \end{cases} \quad (2.6)$$

L'uscita è la stessa dell'ingresso a meno di possibili oscillazioni di rumore.  $V_{ref}$  viene impostata come  $V_{cc}/2$ .

Così come con il driver si può costruire una tabella che descrive l'utilizzo dell'encoder:

Pin	Descrizione
Vcc	Morsetto positivo dell'alimentazione dell'encoder 5V
GND	Morsetto negativo dell'alimentazione
D0	Segnale di uscita. In ingresso al comparatore

## 2.3 Microcontrollore

La scheda di controllo utilizzata per gestire i segnali di ingresso all'attuatore e di uscita dall'encoder è la *STM32F303Discovery* della *STMicroelectronics*.

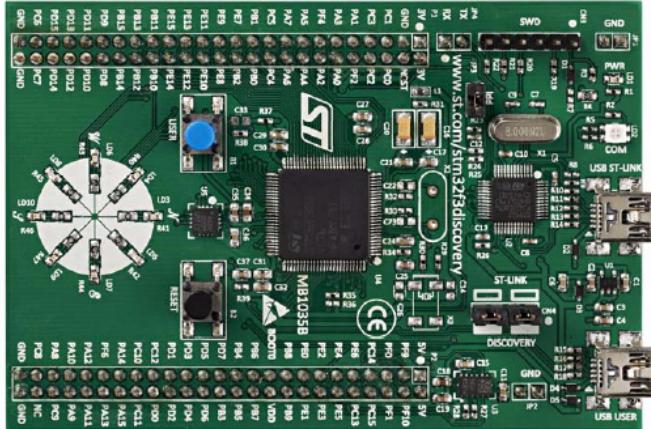


Figura 2.12: *STM32F303Discovery* [4]

La scheda porta a bordo il microcontrollore STM32F303VC a 32bit e rappresenta il "cervello" di tutto l'esperimento. La frequenza di lavoro è scalabile fino a 72MHz, per tutto l'elaborato si terrà in considerazione la massima frequenza di lavoro. Ogni periferica possiede dei registri di configurazione e di utilizzo. La programmazione della scheda prevede quindi l'accesso e la modifica bit a bit dei registri interessati.

Partendo dalla descrizione dell'attuatore e del trasduttore essa si occuperà di:

- Generazione di segnali PWM con duty cycle variabile.
- Interfacciamento con il segnale dell'encoder.
- Implementazione della logica di controllo

In entrambi i casi viene utilizzato un *timer* come periferica. Il timer è una periferica che consiste in un registro di 16bit o 32bit che viene incrementato o sequenzialmente con il clock di sistema o a seguito di un evento. Per il caso di generazione PWM ci si avvale di un incremento sequenziale sfruttando il clock di sistema, mentre per l'interfacciamento con l'encoder ci si avvale di un incremento per evento.

### 2.3.1 Generazione PWM

Per la generazione di segnali PWM si è scelta la periferica TIM4, che fa parte dei *General-purpose Timer* descritti nel datasheet. Ogni pe-

riferita è collegata ad un bus di sistema. Dunque si deve cercare nel datasheet su quale bus è collegato il TIM4.

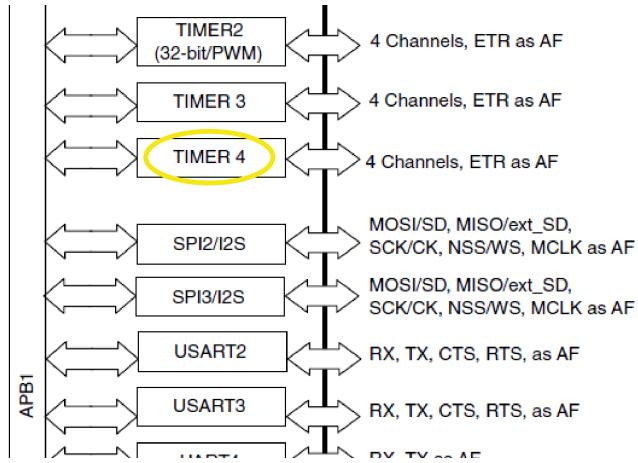


Figura 2.13: Bus APB1 [4]

A questo punto bisogna capirne il funzionamento. Si esamina il circuito logico della periferica.

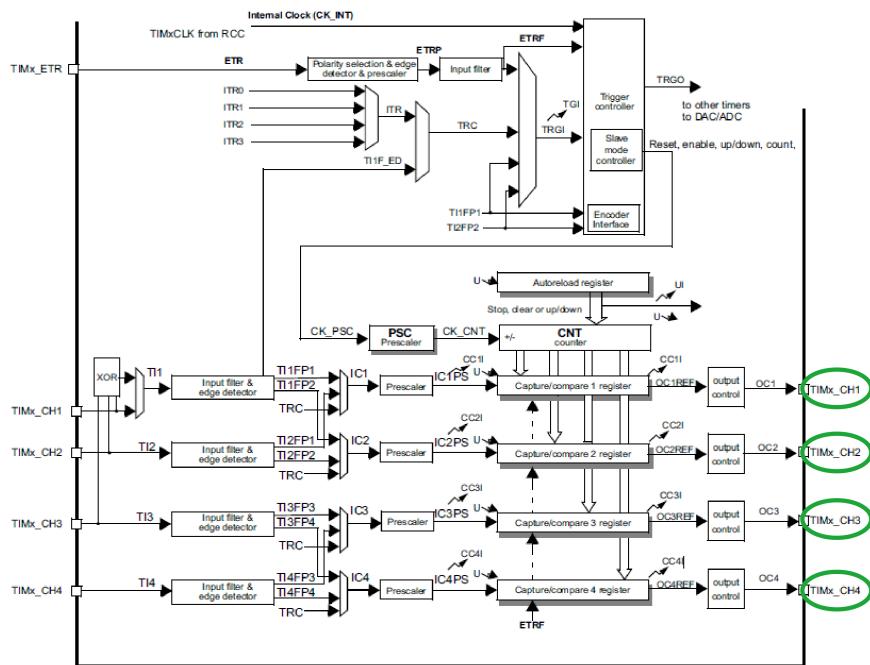


Figura 2.14: Circuito logico della periferica generica TIMx [4]

In verde in Fig. 2.14 sono state evidenziate le uscite fisiche della periferica. Bisogna dunque capire a quali pin effettivi siano realmente collegate. Si è scelto di utilizzare il Canale 1 del TIM4.

Port & Pin Name	AF1	AF2	AF3	AF4	AF5	AF6	AF7
PD0	EVENTOUT						CAN_RX
PD1	EVENTOUT			TIM8_CH4		TIM8_BKIN2	CAN_TX
PD2	EVENTOUT	TIM3_ETR		TIM8_BKIN	UART5_RX		
PD3	EVENTOUT	TIM2_CH1_ETR					USART2_CTS
PD4	EVENTOUT	TIM2_CH2					USART2 RTS
PD5	EVENTOUT						USART2_TX
PD6	EVENTOUT	TIM2_CH4					USART2_RX
PD7	EVENTOUT	TIM2_CH3					USART2 CK
PD8	EVENTOUT						USART3_TX
PD9	EVENTOUT						USART3_RX
PD10	EVENTOUT						USART3 CK
PD11	EVENTOUT						USART3 CTS
PD12	EVENTOUT	TIM4_CH1	TSC_G8_IO1				USART3 RTS
PD13	EVENTOUT	TIM4_CH2	TSC_G8_IO2				
PD14	EVENTOUT	TIM4_CH3	TSC_G8_IO3				
PD15	EVENTOUT	TIM4_CH4	TSC_G8_IO4			SPI2 NSS	

Figura 2.15: Tabella delle Alternate Function del GPIO D [4]

Le prime configurazioni che si possono effettuare sono quelle relative al pin D12.

```
/**TIM4_Ch1-> PD12 (AF2) */
RCC->AHBENR |= RCC_AHBENR_GPIODEN;           //Abilito il clock
GPIOD->MODER |= 1<<25;                      //D12 in modalità Alternate Function
GPIOD->MODER &= ~(1<<24);
GPIOD->AFR[1] = 0;                            //Azzero il registro
GPIOD->AFR[1] |= 1<<17;                      //Configuro AF2
```

In generale il funzionamento del timer è molto semplice. Si avvale di un registro *CNT* che viene incrementato ad ogni fronte di salita (o discesa) del clock. Il conteggio arriva fino ad un valore limite contenuto nel registro *ARR*, dopo di ché parte da 0.

Per la generazione di segnali PMW l'idea è quella di impostare due limiti. Con riferimento al TIM4 sul canale 1, il primo, con il registro *CCR1* imposta l'istante in cui il valore logico del segnale deve commutare (ovvero viene configurato il duty-cycle), il secondo, con il registro *ARR* invece l'istante in cui il conteggio deve partire dall'inizio (ovvero quando finisce un periodo del segnale).

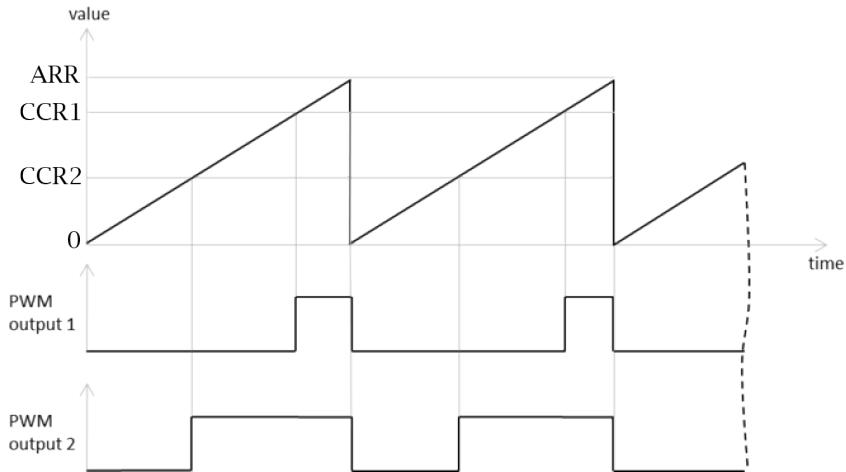


Figura 2.16: Generazione di segnale periodico con Timer

Non rimane altro che impostare il canale 1 come canale di uscita del timer e abilitare il tutto. Tale configurazione viene effettuata solo dal registro *CCMR1*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC2M [3]	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC1M [3]
															Res.
															RW
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]	OC2PE	OC2FE	CC2S[1:0]	OC1CE	OC1M[2:0]	OC1PE	OC1FE	CC1S[1:0]						
	IC2F[3:0]		IC2PSC[1:0]			IC1F[3:0]		IC1PSC[1:0]							
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Figura 2.17: Capture/Compare Mode Register (gestisce i canali 1 e 2) [4]

I bit evidenziati sono quelli che interessano la configurazione del canale 1 come output PWM.

- OC1PE (Output Compare 1 Preload Enable) [1] abilita le operazioni di accesso (in lettura e scrittura) del registro CCR1 (Capture/Compare Register) per determinare la commutazione del segnale
- OC1M (Output Compare 1 Mode) identifica la modalità di PWM che si vuole generare. *PWM mode 1* [110] indica un segnale che rimane alto per la durata  $CNT < CCR1$  e commuta in basso per la durata  $CNT > CCR1$ . *PWM mode 2* [111] indica un segnale che rimane basso per la durata  $CNT < CCR1$  e commuta in alto per la durata  $CNT > CCR1$  (come in Fig. 2.16).

Con la Fig. 2.13 e alle considerazioni appena fatte:

```
RCC->APB1ENR |= RCC_APB1ENR_TIM4EN; //Abilito TIM4

TIM4->CCMR1 |= 1<<3; //Abilito l'accesso al CCR1
TIM4->CCMR1 |= 1<<6; //PWM Mode 1 per il canale 1
TIM4->CCMR1 |= 1<<5;
TIM4->CCMR1 &= ~(1<<4);
```

Dopo aver configurato l'uscita bisogna abilitare il confronto con il registro CCR1 non essendo abilitato di default. Per farlo ci si serve del registro CCER e si setta a 1 il bit di abilitazione

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CC4NP	Res.	CC4P	CC4E	CC3NP	Res.	CC3P	CC3E	CC2NP	Res.	CC2P	CC2E	CC1NP	Res.	CC1P	CC1E
rw		rw	rw												

Figura 2.18: *Capture/Compare Enable Register [4]*

```
TIM4->CCER |= 1; //Abilitazione del confronto
```

Un'ultima configurazione da fare è la scelta del periodo del segnale e del duty-cycle.

Nell'esperimento si è scelto di utilizzare una frequenza di 50kHz per il segnale PWM. Per calcolare il valore del registro *ARR* bisogna capire a che valore deve arrivare il conteggio per avere un periodo di 1/50kHz in uscita. Visto che alla frequenza di 72MHz in un secondo si effettuano 72.000.000 conteggi, si può impostare la seguente proporzione:

$$72.000.000 \text{ conteggi} : 1s = c_1 : \frac{1}{50.000Hz}$$

$$c_1 = \frac{72.000.000 \text{ conteggi}}{50.000s^{-1}} 1s = 1440 \text{ conteggi}$$

Il duty-cycle rappresenta una frazione del periodo del segnale (in questo caso una frazione dei conteggi). Se si imposta il duty-cycle *D* in percentuale si può scrivere, in termini di conteggi:

$$c_2 = \frac{1440}{100} D$$

In generale dunque il calcolo conteggio per il periodo e per il duty-cycle si può ricondurre a due formule.

$$c_1 = \frac{f_c}{f_s} \quad (2.7)$$

$$c_2 = \frac{c_1}{100} D \quad (2.8)$$

con  $f_c$  frequenza di sistema,  $f_s$  frequenza desiderata del segnale e  $D$  duty-cycle percentuale desiderato.

```
TIM4->ARR = 1440; //Periodo del PWM
TIM4->CCR1 = (int) (1440*duty)/100; //Duty-Cycle
```

### 2.3.2 Interfacciamento con l'encoder

Il timer è una periferica che permette di interfacciare l'uscita di un encoder con lo scopo di misurarne la posizione. L'idea è quella di utilizzare l'uscita del sensore come ingresso del timer per incrementare il registro  $CNT$  ad ogni fronte di salita/discesa del segnale. Ogni conteggio significa che l'albero motore ha effettuato una rotazione di  $dx$  (2.4).

Nell'esperimento si è scelto di utilizzare il TIM2 per l'interfacciamento con l'encoder. Anche in questo caso bisogna scegliere uno dei canali di input/output della periferica e abilitare il pin corrispondente in modalità "alternate function".

In Fig. 2.14 (che vale anche per il TIM2) si è scelto di utilizzare il canale 2.

Port & Pin Name	AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF14	AF15
PA0	TIM2_CH1_ETR	TSC_G1_IO1					USART2_CTS	COMP1_OUT	TIM8_BKIN	TIM8_ETR					EVENT OUT
PA1	RTC_REFIN	TIM2_CH2	TSC_G1_IO2				USART2_RTS		TIM15_CH1N						EVENT OUT
PA2	TIM2_CH3	TSC_G1_IO3					USART2_TX	COMP2_OUT	TIM15_CH1						EVENT OUT
PA3	TIM2_CH4	TSC_G1_IO4					USART2_RX		TIM15_CH2						EVENT OUT
PA4		TIM3_CH2	TSC_G2_IO1	SPI1_NSS	SPI3 NSS	USART2_CK									EVENT OUT
PA5	TIM2_CH1_ETR	TSC_G2_IO2		SPI1_SCK											EVENT OUT
PA6	TIM16_CH1	TIM3_CH1	TSC_G2_IO3	TIM8_BKIN	SPI1_MISO	TIM1_BKIN	COMP1_OUT								EVENT OUT
PA7	TIM17_CH1	TIM3_CH2	TSC_G2_IO4	TIM8_SPI1	SPI1_MOSI	TIM1_CH1N	COMP2_OUT								EVENT OUT
PA8	MCO			I2C2_SMB	I2S2_MCK	TIM1_CH1	USART1_CK	COMP3_OUT		TIM4_ETR					EVENT OUT
PA9			TSC_G4_IO1	I2C2_SCL	I2S3_MCK	TIM1_CH2	USART1_TX	COMP5_OUT	TIM15_BKIN	TIM2_CH3					EVENT OUT
PA10		TIM17_BKIN	TSC_G4_IO2	I2C2_SDA		TIM1_CH3	USART1_RX	COMP6_OUT		TIM2_CH4	TIM8_BKIN				EVENT OUT
PA11						TIM1_CH1N	USART1_CTS	COMP1_OUT	CAN_RX	TIM4_CH1	TIM1_CH4	TIM1_BKIN2	USB_DM	EVENT OUT	

Figura 2.19: Tabella delle Alternate Function del GPIO A [4]

Si possono effettuare le configurazioni relative al pin A1.

```

/** TIM2_Ch2 -> PA1 (AF1) */
RCC->AHBENR |= RCC_AHBENR_GPIOAEN;           //Abilito il clock per la periferica GPIOA

GPIOA->MODER |= 1<<3;                         //PA1 in modalità Alternate Function
GPIOA->MODER &= ~(1<<2);                      //Configuro AF1
GPIOA->AFR[0] |= 1<<4;

```

A questo punto occorre configurare il TIM2 per impostare come input e sorgente esterna di aggiornamento il canale 2. Il registro principale per le configurazioni è il registro *SMCR*:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SMS[3]
															rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ETP	ECE	ETPS[1:0]		ETF[3:0]		MSM		TS[2:0]		OCCS		SMS[2:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figura 2.20: *Slave Mode Control Register [4]*

I bit evidenziati sono quelli che interessano la configurazione per la selezione del trigger, ovvero la selezione della sorgente che permette l'aggiornamento del *CNT*

- SMS (Slave Mode Selection) [111] Modalità con clock esterno sul fronte di salita. In pratica il segnale dell'encoder fungerà da clock per il timer (essendo un segnale digitale).
- TS (Trigger Selection) [110] Permette di selezionare l'input in cui entrerà il clock esterno. Con questa scelta di bit si è scelto l'ingresso sul canale 2.

Il timer è sensibile sul fronte di salita del segnale, quindi eventuali rumori possono influenzare il valore del conteggio. Per tale motivo si è preferito aggiungere un comparatore in cascata all'encoder.

Un'ultima configurazione è da fare sul registro *CCMR1* per impostare che il canale 2 fungerà da input.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC2M[3]	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC1M[3]
							Res.								Res.
							rw								rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]	OC2PE	OC2FE	CC2S[1:0]	OC1CE	OC1M[2:0]	OC1PE	OC1FE	CC1S[1:0]						
	IC2F[3:0]		IC2PSC[1:0]			IC1F[3:0]		IC1PSC[1:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figura 2.21: *Capture/Compare Mode Register [4]*

- CC2S (Capture Compare 2 Select) [01] Configura il canale 2 come input.

A questo punto il codice è abbastanza semplice da scrivere, avendo esperienza di quello precedente.

```
RCC->APB1ENR |= RCC_APB1ENR_TIM2EN; //Abilito TIM2

TIM2->SMCR |= 7; //Modalità con clock esterno
TIM2->CCMR1 &= ~(1<<9); //Ch2 usato come input
TIM2->CCMR1 |= (1<<8);
TIM2->SMCR |= 1<<6; //Input sul ch2
TIM2->SMCR |= 1<<5;
TIM2->SMCR &= ~(1<<4);
```

Per vie sperimentali si è notato che l'encoder non riesce a gestire velocità maggiori di  $500\text{rad/s}$  (l'uscita non è più attendibile). Quindi è importante non richiedere al sistema di gestire velocità maggiori di quest'ultima. In una tabella si possono rappresentare i parametri chiave di questo dispositivo.

Velocità massima	500 rad/s
Risoluzione	36°

## 2.4 Motore elettrico

Il motore elettrico utilizzato è una macchina a corrente continua. La tensione di alimentazione nominale è di 12V, quindi è consigliabile di non alimentarlo con una tensione maggiore.



Figura 2.22: Motore CC a 12V

Per quanto riguarda la corrente assorbita, essa dipende molto dal carico collegato alla macchina. Più il carico ha un momento di inerzia elevato più c'è bisogno di corrente per metterla in moto.

Il motore è stato utilizzato solo in scopo didattico, quindi senza nessun carico. La corrente che si può misurare al massimo della alimentazione è di circa  $500mA$ .

Il driver che è adibito a pilotarlo rientra perfettamente in tali specifiche.

## 2.5 Configurazione completa

Si hanno tutti gli strumenti necessari per il collegamento elettrico dei componenti. Manca solo la configurazione dei pin del microcontrollore per gestire la direzione del motore.

Si è scelto di utilizzare i pin PB0 e PB2 come rispettivamente AIN1 e AIN2 e PB1 come STBY (rappresenta lo stato di stand-by per il risparmio energetico del driver, nell'esperimento non starà mai in questo stato) (Fig. 2.2).

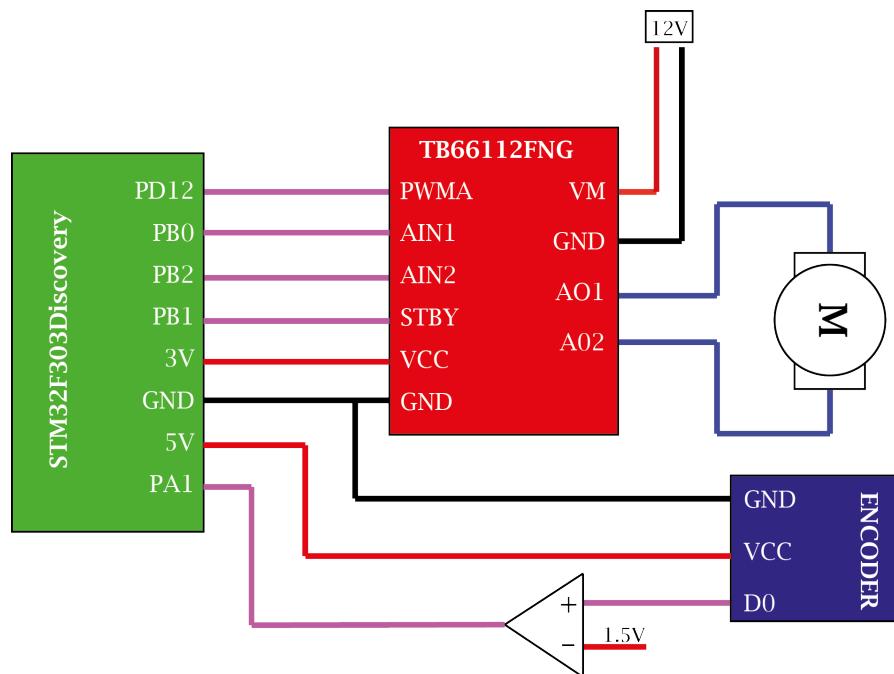


Figura 2.23: Collegamento logico della struttura

Le funzioni necessarie per l'utilizzo dei componenti e da implementare sul microcontrollore possono essere racchiuse in una libreria.

ria. Esse rappresentano la base del controllo e verranno richiamate frequentemente nell'elaborato.

```

void init_encoder() {
    /** TIM2_Ch2 -> PA1 (AF1) */
    RCC->AHBENR |= RCC_AHBENR_GPIOAEN;           //Abilito GPIOA
    GPIOA->MODER |= 1<<3;                      //PA1 Alternate Function
    GPIOA->MODER &= ~(1<<2);
    GPIOA->AFR[0] |= 1<<4;                      //Configuro AF1

    RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;          //Abilito TIM2
    TIM2->SMCR |= 7;                            //Clock esterno
    TIM2->CCMR1 &= ~(1<<9);                  //Ch2 usato come input
    TIM2->CCMR1 |= (1<<8);
    TIM2->SMCR |= 1<<6;                      //Input sul ch2
    TIM2->SMCR |= 1<<5;
    TIM2->SMCR &= ~(1<<4);
}

void start_encoder() {
    TIM2->CR1 |= TIM_CR1_CEN;                   //Abilito il conteggio
}

void setDirection(int dir) {
    RCC->AHBENR |= RCC_AHBENR_GPIOBEN;          //Abilito GPIOB
    GPIOB->MODER |= 1;                          //PB0 come output
    GPIOB->MODER &= ~(1<<1);
    GPIOB->MODER |= 1<<2;                    //PB1 come output
    GPIOB->MODER &= ~(1<<3);
    GPIOB->ODR |= 1<<1;                      //PB1 uscita alta
    GPIOB->MODER |= 1<<4;                    //PB2 come output
    GPIOB->MODER &= ~(1<<5);

    switch(dir) {
        case 1:
            GPIOB->ODR |= 1;
            GPIOB->ODR &= ~(1<<2);
            break;
        case 0:
            GPIOB->ODR &= ~1;
            GPIOB->ODR |= (1<<2);
            break;
    }
}

void init_pwm() {
    /**TIM4_Ch1-> PD12 (AF2) */
    RCC->AHBENR |= RCC_AHBENR_GPIODEN;          //Abilito GPIOD
    GPIOD->MODER |= 1<<25;                   //D12 Alternate Function
    GPIOD->MODER &= ~(1<<24);
    GPIOD->AFR[1] = 0;                         //Azzero il registro
    GPIOD->AFR[1] |= 1<<17;                  //Configuro AF2

    setDirection(1);

    RCC->APB1ENR |= RCC_APB1ENR_TIM4EN;        //Abilito TIM4
    TIM4->CCMR1 |= 1<<3;                     //Accesso al CCR1
    TIM4->CCMR1 |= 1<<6;                     //PWM Mode 1
}

```

```
    TIM4->CCMR1 |= 1<<5;
    TIM4->CCMR1 &= ~(1<<4);
    TIM4->CCER |= 1;                                //Abilito il confronto

    TIM4->ARR = 1440;                             //Freq 50kHz
    TIM4->CR1 |= 1;                                //Abilito il conteggio
}

void set_pwm(double duty) {
    TIM4->CCR1 = (int) (1440*duty)/100;      //duty-cycle
}
```

# Capitolo 3

## Identificazione del modello matematico

Gli strumenti descritti nei capitoli precedenti possono essere utilizzati per ricavare un modello matematico del motore sperimentalmente. L'obiettivo è quello di porre in ingresso un gradino e misurarne la risposta stimandone un'approssimazione matematica. Lavorando in digitale si deve scegliere un periodo di campionamento in relazione alle dinamiche del sistema.

### **Teorema 1 (Teorema di campionamento di Nyquist-Shannon)**

*Dato un segnale analogico  $s(t)$  la cui banda di frequenze sia praticamente limitata dalla frequenza  $f_M$ , e dato  $n \in \mathbb{Z}$ , il segnale  $s(t)$  può essere univocamente ricostruito a partire dai suoi campioni  $s(n\Delta t)$  presi a frequenza  $f_s = \frac{1}{\Delta t}$  se e solo se  $f_s > 2f_M$  [6]*

Visto che il sistema da controllare è sconosciuto si inizia a campionare con un periodo di campionamento molto alto per poi affinare il valore successivamente.

All'interno della *STM32F303Discovery* significa realizzare una ISR temporale che viene chiamata ogni  $\Delta t$ . L'idea è quella di utilizzare un timer e settarlo in modo tale che ogni  $\Delta t$  generi un'interruzione software. Il timer che si è scelto di utilizzare è il TIM3 (dato che il TIM2 è per l'encoder e il TIM4 per la generazione PWM). Le configurazioni sono le stesse viste al Capitolo 2, con la differenza di abilitare le interruzioni tramite il registro *DIER*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TDE	Res.	CC4DE	CC3DE	CC2DE	CC1DE	UDE	Res.	TIE	Res.	CC4IE	CC3IE	CC2IE	CC1IE	UIE
	rw		rw	rw	rw	rw	rw		rw		rw	rw	rw	rw	rw

Figura 3.1: DMA/Interrupt Enable Register [4]

Il bit evidenziato abilita le interruzioni. Quindi il codice da implementare è il seguente.

```
int init_timer(int frequenza_campionamento) {
    /* Frequenza di campionamento */
    int i = 1;
    uint32_t frequenza_timer;
    uint32_t arr;
    RCC->APB1ENR |= RCC_APB1ENR_TIM3EN; //Abilito TIM3
    do {
        frequenza_timer = FRQ/i;
        arr = (uint32_t) frequenza_timer/frequenza_campionamento;
        i++;
    } while(arr > 65535);
    TIM3->ARR = (uint16_t) arr;
    TIM3->PSC = (uint16_t) i-1; //Scalatura del clock
    TIM3->DIER |= 1; //Abilito le interruzioni
    NVIC->ISER[0] |= 1<<29; //ISR per il TIM3
    return arr;
}

void start_timer() {
    TIM3->CR1 |= TIM_CR1_CEN; //Abilito il conteggio
}
```

Nell'esperimento si è scelto di campionare con un periodo iniziale di campionamento  $\Delta t = 1ms$ .

### 3.1 Posizione angolare

Inizialmente si sceglie di considerare il motore come un sistema con ingresso una tensione modulata con un duty-cycle e in uscita la posizione angolare, poiché l'encoder è in grado di misurare una posizione anziché una velocità. Di conseguenza ci si aspetta, con una corretta misurazione, una funzione di trasferimento  $W(s)$  (1.4) con l'aggiunta di un polo nell'origine.

$$G(s) = \frac{1}{s}W(s) \quad (3.1)$$

Dato che  $W(s)$  è modellata con uscita la velocità angolare, mentre la misurazione è in posizione. Un primo programma di misurazione con-

siste nel campionare il registro  $CNT$  del TIM3 ogni  $\Delta t$  ed effettuare un grafico nel tempo.

```
unsigned int new_step;
int main() {
    init_encoder();           // Sensore
    init_timer(1000);         // Campionamento - 1kHz
    init_pwm();               // Segnale di controllo

    start_encoder();          // Abilitazione dell'encoder
    start_timer();             // Avvio del controllore
    set_pwm(50);               // Duty-cycle del 50%
    while(1);
}

void TIM3_IRQHandler() {
    TIM3->SR &= ~1;           //Reset del bit di interruzione
    new_step = TIM2->CNT;      //Posizione corrente
}
```

Il software utilizzato per caricare il programma sulla scheda è *IAR Embedded Workbench* permette di memorizzare i valori delle variabili in un file di testo. Si memorizza dunque una sequenza della variabile *newstep* su un file in modo può essere letto da *Matlab*.

---

```
clc, close all, clear all;

%% Estrazione ed elaborazione dati
tab = readtable('Spazio_50_50kHz.txt');
Vcc = 12;
duty_cicle = 50;
Vin = duty_cicle;
dt = 1e-3;                                % [s]
dx = 36*pi/180;                            % [rad]
count = tab.Var2;                           %Valori di new_step

t = 0:dt:(numel(count)-1)*dt;              %Asse dei tempi [s]
theta = count*dx;                          %Asse della posizione angolare [rad]

%% Posizione angolare
plot(t,theta, 'Color', 'Black');
grid;
title('Posizione angolare [rad]');
xlabel('Secondi [s]');
ylabel('Radiani [rad]');
axis([0,0.15,0,30]);
```

---

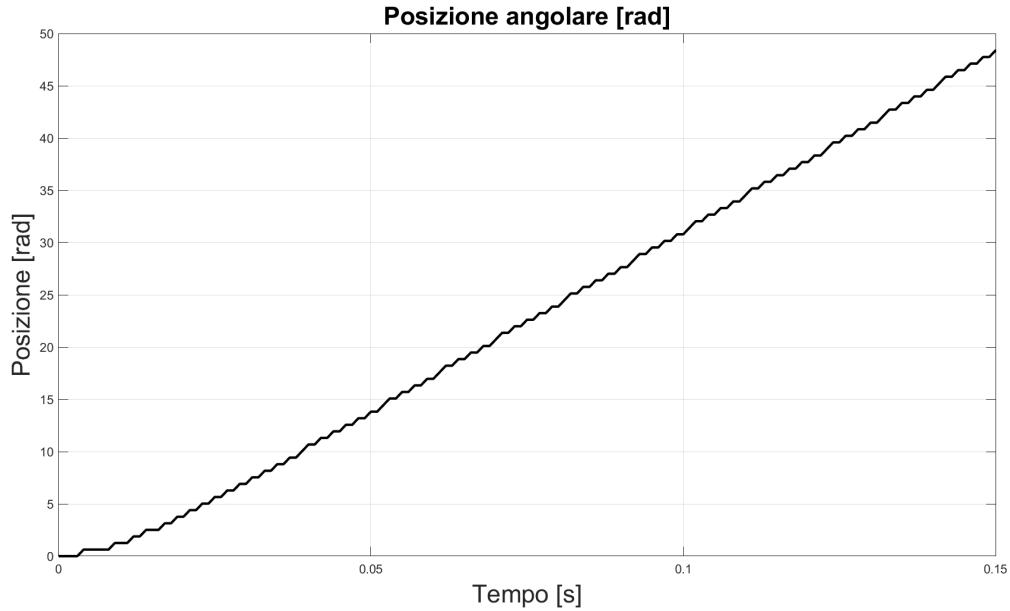


Figura 3.2: Misura della posizione angolare con duty-cycle del 50%

Nell'esecuzione dell'esperimento si fa riferimento ad un duty-cycle del 50% in ingresso al driver.

Per trovare una FdT la cui risposta al gradino è una buona approssimazione a quella misurata, si può ricorrere a degli algoritmi numerici. In particolare si è scelto di identificare una funzione di trasferimento del primo ordine (in accordo con l'approssimazione a poli dominanti discussa nel Capitolo 1), con un polo nell'origine, che rispetto alla misurazione minimizzi lo **scarto quadratico medio**.

$$G(s) = \frac{1}{s} \frac{dcg}{(1 + s\tau)} \quad (3.2)$$

Con  $dcg$  costante.

L'algoritmo numerico deve iterare i valori dei parametri finché non si trovi una combinazione di essi che generi una  $G(s)$  tale che la sua risposta al gradino che abbia il minimo scarto quadratico medio rispetto alla risposta misurata.

La funzione di Matlab *fminsearch* prevede in ingresso una funzione parametrica con il valore iniziale dei parametri e ne restituisce il minimo locale. Si definisce dunque in Matlab una la funzione di risposta al gradino della 3.2 con parametri  $dcg$  e  $\tau$  che restituisce lo scarto quadratico medio con la misura.

---

```
function sqm = ideal_tf_primo_ordine(par, t, w, in)
```

---

```

tau = par(1); %Costante di tempo del sistema
dcg = par(2); %Guadagno della FdT
s = tf('s');
G = dcg/(s*(1+s*tau)); %FdT Obiettivo
[y,ty] = step(G,t); %Risposta al gradino
y = y*in; %Normalizzazione
y = interp1(ty,y,t,'linear','extrap'); %Interpolazione lineare
sqm = mean((y-w).^2); %Scarto quadratico medio
end

```

Utilizzando successivamente la funzione *fminsearch* secondo la documentazione:

```

polo = 0.2; gain = 20;
set = [polo,gain];
parametri = fminsearch(@(x)ideal_tf_primo_ordine(x,t,theta,Vin),set);
tau = parametri(1); dcg=parametri(2);
%Funzione di trasferimento V->Theta
G = dcg/(s*(1+s*tau));
[y,ty] = step(G,t);
y = y*Vin;
hold on, plot(ty,y, 'Color', 'Green', 'LineWidth',2);
legend('Posizione misurata', 'Posizione stimata');
axis([0,0.07,0,20]);

```

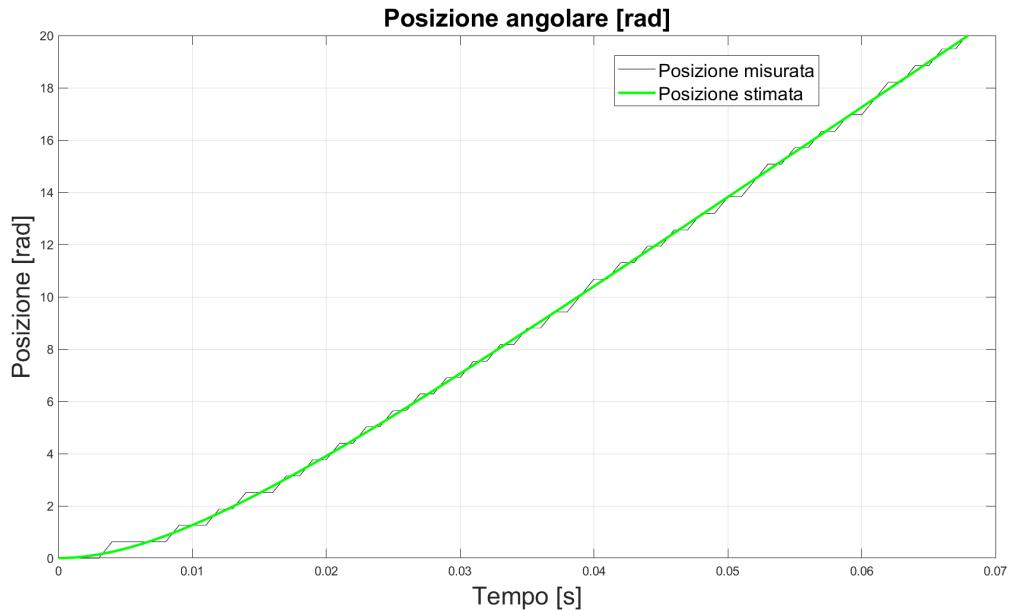


Figura 3.3: Funzione di trasferimento approssimata al primo ordine

Da cui risulta:

$$G(s) = \frac{1}{s} \frac{6.702}{(1 + 0.011s)}$$

Nel caso specifico si può anche identificare una FdT del secondo ordine in accordo con il modello reale.

$$G(s) = \frac{1}{s} \frac{\alpha}{(1+s\tau_1)(1+s\tau_2) + \alpha\beta} \quad (3.3)$$

$$\text{con } \alpha = \frac{K_a \phi}{LJ} \quad \beta = K_e \phi \quad \tau_1 = \frac{R}{L} \quad \tau_2 = \frac{b}{J}$$

Si definisce quindi, in Matlab, la funzione di risposta al gradino della 3.3 con parametri  $\alpha$ ,  $\beta$ ,  $\tau_1$  e  $\tau_2$  che restituisce lo scarto quadratico medio con la misura.

---

```
function sqm = ideal_tf_secondo_ordine(par, t, w, in)
    tau1 = par(1); %Costante di tempo elettrica
    tau2 = par(2); %Costante di tempo meccanica
    a = par(3);
    b = par(4);
    s = tf('s');
    G = a/(s*((tau1+s)*(tau2+s)+a*b)); %FdT Obiettivo
    [y,ty] = step(G,t); %Risposta al gradino
    y = y*in; %Normalizzazione
    y = interp1(ty,y,t,'linear','extrap'); %Interpolazione lineare
    sqm = mean((y-w).^2); %Scarto quadratico medio
end
```

---

Utilizzando poi analogamente *fminsearch*:

---

```
t1 = 100; t2 = 1000; alfa = 1e6; beta = 0;
set = [t1, t2, alfa, beta];
parametri = fminsearch(@(x)ideal_tf_secondo_ordine(x,t,theta,Vin),set);
tau1 = parametri(1); tau2=parametri(2); a = parametri(3); b=parametri(4);
%Funzione di trasferimento V->Theta
G = a/(s*((1+s*tau1)*(1+s*tau2)+a*b));
[y,ty] = step(G,t);
y = y*Vin;
hold on, plot(ty,y, 'Color', 'Green', 'LineWidth', 2);
legend('Posizione misurata', 'Posizione stimata');
axis([0,0.07,0,20]);
```

---

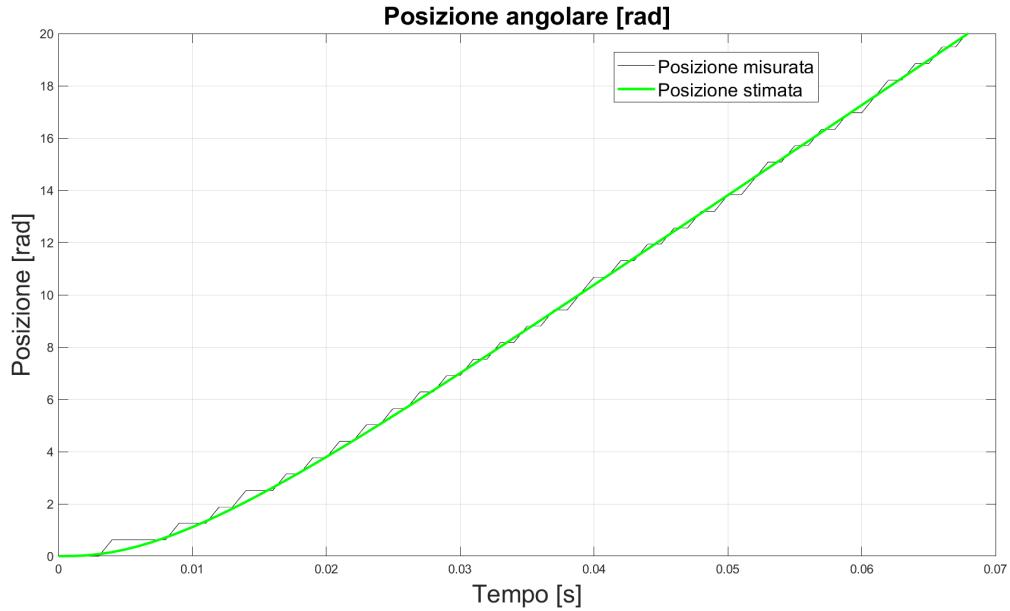


Figura 3.4: *Funzione di trasferimento secondo il modello matematico*

Da cui risulta:

$$G(s) = \frac{5.4198 \times 10^5}{s(102.4287 + s)(788.6668 + s) + 90.6875}$$

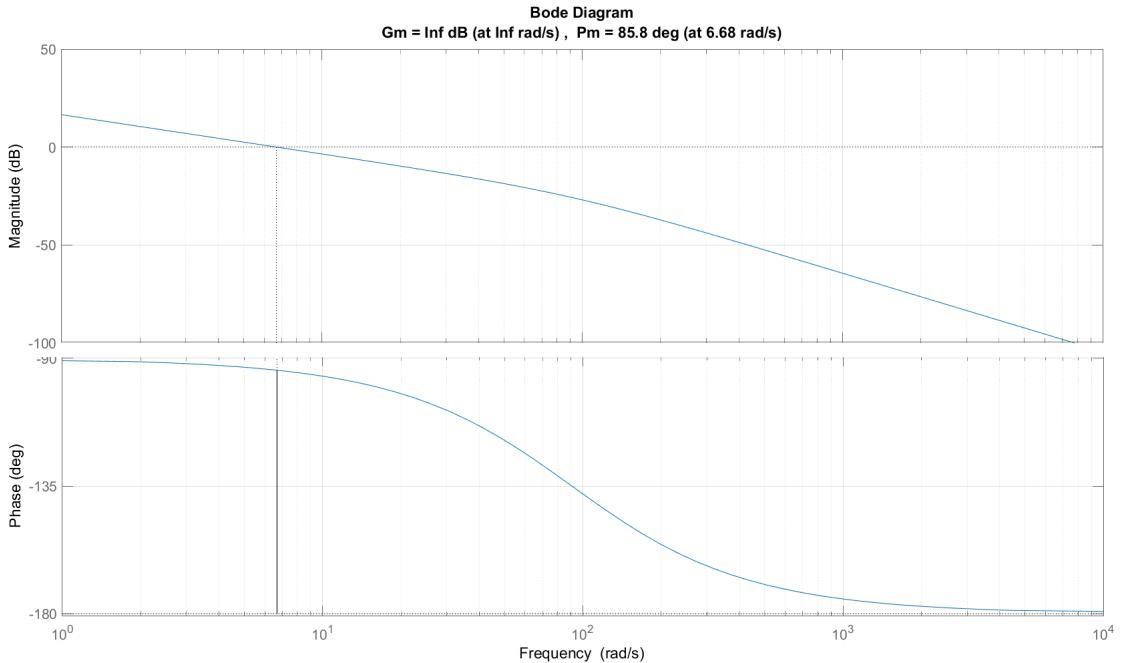
La costante di tempo risulta essere circa uguale alla costante di tempo della parte meccanica dell'impianto. In genere accade perché la dinamica elettrica è molto più veloce della dinamica meccanica, per cui può essere trascurata in un'approssimazione a poli dominanti.

## 3.2 Velocità angolare

$G(s)$  è stata identificata, sulla base degli strumenti disponibili, in modo che in ingresso si abbia il duty-cycle e in uscita la posizione angolare. Per ottenere la velocità angolare bisogna dunque applicare un derivatore in cascata. Il derivatore ideale non è fisicamente realizzabile. Alternativamente applicando un derivatore reale non solo si può ottenere in uscita la velocità angolare, ma si può anche smorzare il rumore additivo introdotto dall'encoder, grazie al polo per la fisica realizzabilità.

$$D(s) = \frac{s}{1 + sT} \tag{3.4}$$

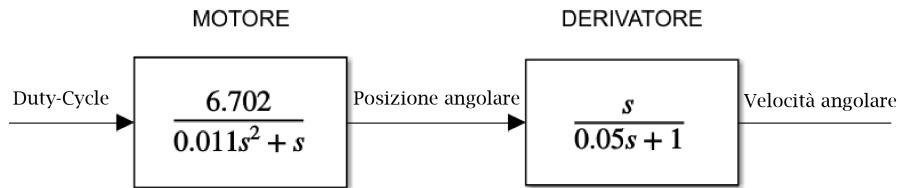
$T$  deve essere tale che non influenzi eccessivamente la banda del sistema. I diagrammi di Bode di  $G(s)$  mettono in evidenza la banda.



Approssimativamente la banda del sistema si aggira intorno ai 10 rad/s. Il valore di  $T$  deve essere tale che  $\frac{1}{T} > 10$ . E' corretto scegliere dunque:

$$T = 0.05 \quad \Rightarrow \quad D(s) = \frac{s}{1 + 0.05s}$$

Le considerazioni fatte fin ora identificano quindi il sistema complessivo seguente.



Per effettuare un confronto con l'impianto reale bisogna digitalizzare il derivatore e implementarlo nella scheda di controllo. Precedentemente si era effettuato solo un campionamento della posizione angolare, la velocità richiede dunque questo passaggio matematico indispensabile.

### 3.2.1 Digitalizzazione

La digitalizzazione è un processo di conversione che determina il passaggio dal campo dei valori continui a quello dei valori discreti. Il periodo di campionamento è già noto, fissato a 1 ms.

Nel campo dei sistemi di controllo per digitalizzare un sistema in forma di funzione di trasferimento si effettua la sostituzione:

$$s = \frac{1}{\Delta t} \frac{z - 1}{az + 1 - \alpha} \quad \text{con } \alpha \in [0, 1]$$

Scegliendo  $\alpha = \frac{1}{2}$  si ottiene la sostituzione di Tustin, ottenendo:

$$s = \frac{2}{\Delta t} \frac{z - 1}{z + 1} \quad (3.5)$$

Sostituendo dunque la 3.5 nella 3.4 si ottiene il sistema tempo-discreto:

$$D(z) = \frac{\alpha(z - 1)}{z - \beta} \quad \text{con } \alpha = 19.8 \beta = 0.9802 \quad (3.6)$$

Per implementare dunque la precedente FdT all'interno del microcontrollore, essa deve essere espressa nel dominio del tempo discreto. L'antitrasformata- $\mathcal{Z}$  della 3.6 determina la risposta all'impulso del derivatore discretizzato.

$$\mathcal{Z}^{-1}[D(z)](n) = d(n)$$

Si considera:

$$F(z) = \frac{1}{z} D(z) = \frac{1}{z} \frac{\alpha(z - 1)}{z - \beta}$$

$F(z)$  può essere scomposta in fratti semplici sfruttando la tecnica dei residui.

$$F(z) = \frac{R_1}{z} + \frac{R_2}{z - \beta}$$

Ogni residuo  $R_n$  vale per definizione:

$$R_1 = \lim_{z \rightarrow 0} zF(z) = \lim_{z \rightarrow 0} \frac{\alpha(z - 1)}{z - \beta} = \frac{\alpha}{\beta}$$

$$R_2 = \lim_{z \rightarrow 0} (z - \beta)F(z) = \lim_{z \rightarrow 0} \frac{\alpha(z - 1)}{z} = \frac{\alpha(\beta - 1)}{\beta}$$

La  $D(z)$  si può quindi scrivere:

$$D(z) = zF(z) = \frac{\alpha}{\beta} + \frac{\alpha(\beta - 1)}{\beta} \frac{z}{z - \beta} \quad (3.7)$$

Ricordando le trasformate notevoli

$$\mathcal{Z}[\delta(n)] = 1 \quad \mathcal{Z}[\beta^n u(n)] = \frac{z}{z - \beta}$$

e applicando la proprietà di linearità della  $\mathcal{Z}$ -Trasformata la antitrasformata di 3.7 equivale:

$$d(n) = \frac{\alpha}{\beta} \delta(n) + \frac{\alpha(\beta - 1)}{\beta} \beta^n \quad n > 0 \quad (3.8)$$

La precedente rappresenta la risposta all'impulso del derivatore reale. Dato che in uscita all'impianto si ha la posizione angolare per come è modellato, la velocità angolare si può calcolare attraverso un prodotto di convoluzione discreta.

$$\omega(n) = \theta(n) * d(n) = \frac{\alpha}{\beta} \theta(n) + \frac{\alpha(\beta - 1)}{\beta} [\beta^n * \theta(n)] \quad (3.9)$$

in cui  $\theta(n)$  rappresenta l'andamento temporale della posizione angolare. Ponendo  $h(n) = \beta^n$  la convoluzione discreta con  $\theta(n)$  è per definizione [6]:

$$\theta(n) * h(n) \triangleq \sum_{k=0}^n \theta(k)h(n - k)$$

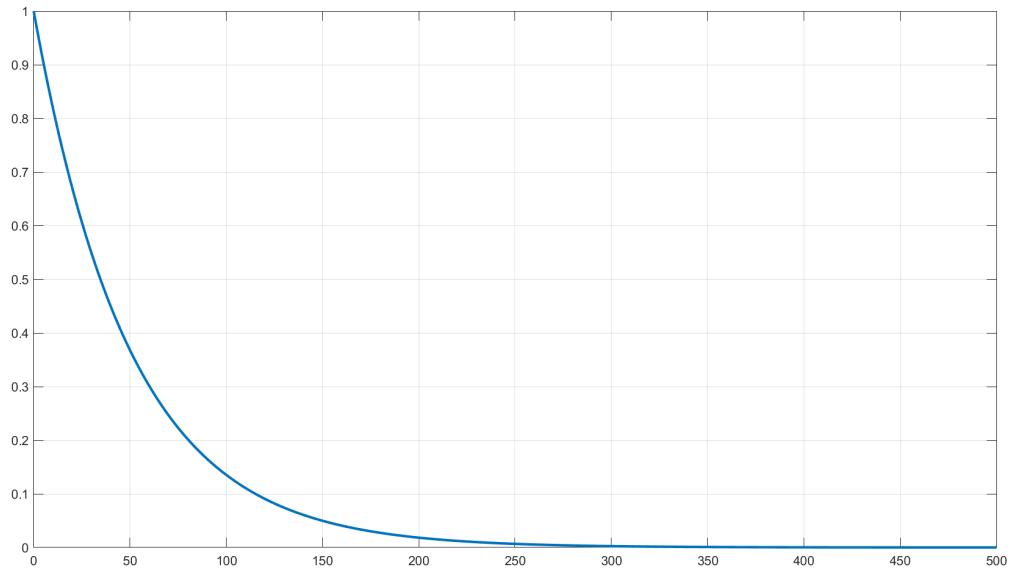


Figura 3.5: Andamento di  $\beta^n$  in funzione di  $n$

Per implementarla su microcontrollore si deve fare innanzitutto la considerazione che

$$\lim_{n \rightarrow \infty} \beta^n = 0 \quad \text{con } 0 < \beta < 1$$

In particolare dopo  $h(500) = 4.5 \times 10^{-5}$  il prodotto tra  $\theta(n)$  e  $h(n)$  nella sommatoria è circa zero. Il derivatore 3.9 quindi può essere implementato nel seguente modo. Ponendo:

$$a = \frac{\alpha}{\beta} \quad b = \frac{\alpha(\beta - 1)}{\beta} \quad c = \beta$$

```
#define CONV 500           //Dimensione massima dell'array dato che
                           //dopo CONV i valori del vettore di convoluzione
                           //sono trascurabili
double theta[CONV];
double vettore_convoluzione[CONV];
double a=20.2, b=-0.4, c=0.9802; //Parametri del derivatore

void init_derivatore() {
    //Inizializzazione del vettore di convoluzione
    for(int i=0; i<CONV; i++) {
        vettore_convoluzione[i] = exp(gamma, i);
        theta[i] = 0;
    }
}

void inserisci(double* vettore, double pos, int dim) {
    for(int i=0; i<dim-1; i++) {
        vettore[i] = vettore[i+1];
    }
}
```

```

        }
        vettore[dim-1] = pos;
    }

double derivatore(double theta_new) {
    double somma_differenziale=0;
    double prod_conv = 1;
    inserisci(theta,theta_new,CONV);
    //Convoltione discreta
    for(int i=0, k=CONV-1; i<CONV && k>=0; i++, k--) {
        prod_conv = theta[k]*vettore_convoluzione[i];
        somma_differenziale = somma_differenziale+prod_conv;
    }
    velocita = (a*theta_new)+(b*somma_differenziale);
    return velocita;
}

```

A questo punto si deve confrontare la risposta del sistema a tempo continuo approssimato e la misurazione discreta effettuata con l'aggiunta del filtro derivativo discreto.

---

```

%% Filtro Derivativo
T = 0.05;
D = s/(1+s*T);
Dd = c2d(D,0.001,'tustin');

%% Velocita angolare
W = G*D; %Funzione di trasferimento V->Omega
[y,ty] = step(W,t);
y = y*Vin;
[w,tw] = lsim(Dd,theta,t,0);
hold on, plot(tw,w, 'Color', 'Black'); %Velocita angolare misurata
plot(ty,y, 'Color', 'Green', 'LineWidth',2); %Velocita angolare stimata

w_temp = interp1(tw,w,0.4*dt:0.6);
reg = mean(w_temp); %Valore di regime
reg_95perc = reg-((reg*5)/100);
tw_95perc = tw(find(w > reg_95perc, 1)); %Tempo di assestamento al 5%
w_reg95perc = ones(1, length(tw)).*reg_95perc;
hold on, plot(tw,w_reg95perc, 'Color', 'Blue');
legend('Velocita misurata', 'Velocita stimata','Regime al 95%');
grid;
title('Velocita angolare [rad/s]');
xlabel('Secondi [s]');
ylabel('Radianti [rad/s]');
axis([0,0.5,0,400]);

```

---

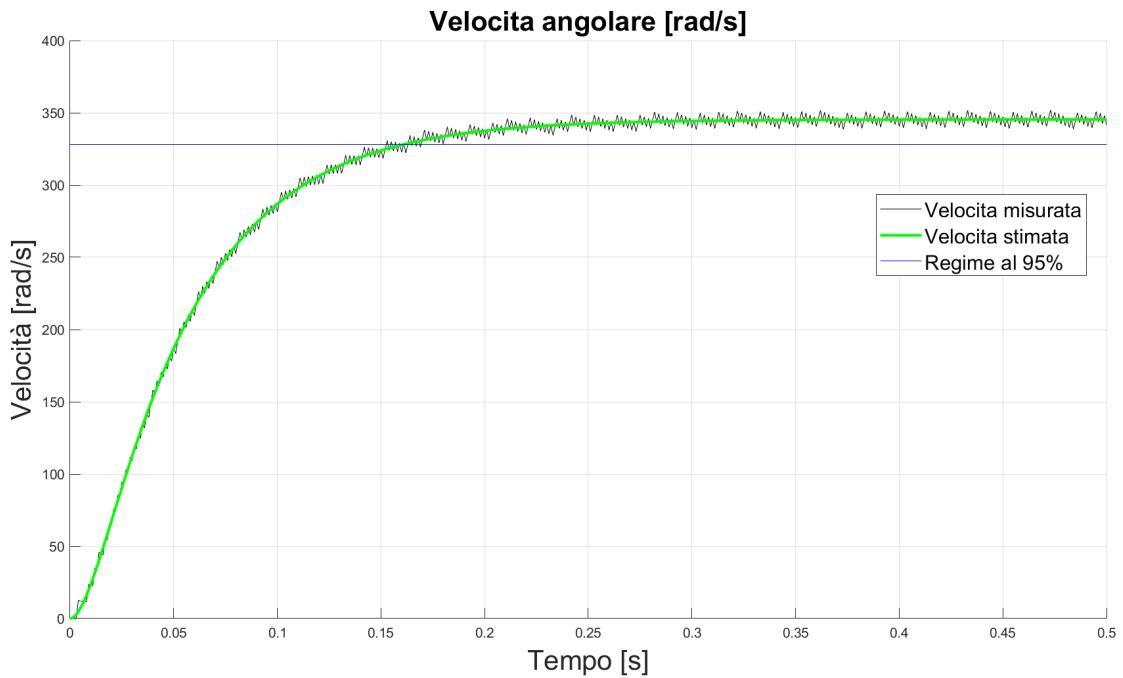


Figura 3.6: Risposta della funzione di trasferimento approssimata al primo ordine con derivatore reale

L'andamento coincide con le aspettative, prova del fatto che l'impianto è stato modellato correttamente

# Capitolo 4

# Controllo digitale

Nel capitolo precedente si è effettuata una modellazione sperimentare per il caso di risposta al gradino. In particolare il gradino era di ampiezza pari al duty-cycle e pari al 50%. L'obbiettivo è di scovare eventuali non linearità dovute dai componenti hardware dell'esperimento. Si effettuano dunque vari test sperimentali impostando il gradino a valori diversi. Si è scelto di effettuare test per i seguenti valori percentuali:

Duty-cycle del 20%	Duty-cycle del 25%	Duty-cycle del 30%
Duty-cycle del 35%	Duty-cycle del 40%	Duty-cycle del 45%
Duty-cycle del 50%	Duty-cycle del 55%	Duty-cycle del 60%
Duty-cycle del 65%	Duty-cycle del 70%	

I risultati di ogni test sono stati salvati in appositi file di testo denominati "*Spazio\_DC\_50kHz.txt*" in cui *DC* indica il duty-cycle. La funzione di trasferimento che approssima il comportamento del motore

$$G(s) = \frac{1}{s} \frac{dcg}{1 + s\tau}$$

deve essere uguale, o simile, per ogni test. Il confronto dunque varia solo sul guadagno e sulla costante di tempo.

---

```
name1 = 'Spazio_';
name2 = '_50kHz.txt';

%% Estrazione ed elaborazione dati
Vcc = 12;
dt = 1e-3; % [s]
dx = 36*pi/180; % [rad]
N = (70-20)/5 +1; % Numero di esperimenti
```

```
s = tf('s'); % Variabile per Fdt Continua
j=1;
for i=20:5:70
    name = [name1, int2str(i),name2];
    tab = readtable(name);
    theta(j,:) = tab.Var2*dx;
    duty_cicle(j) = i;
    in(j) = duty_cicle(j);
    j = j+1;
end
t = 0:dt:(numel(theta(1,:))-1)*dt;

%% FdT
polo = 0.2;
gain = 20;
cost0 = [polo, gain];
for i=1:N
    temp = theta(i,:);
    cost(i,:) = fminsearch(@(x)ideal_tf(x,t,temp'),in(i)),cost0);
end
dc = 20:5:70;

subplot(1,2,1);
plot(dc,cost(:,1),'*');
grid;
axis([20,70,0,0.02]);
title('Variazione della costante di tempo');

subplot(1,2,2);
plot(dc,cost(:,2),'*');
grid;
axis([20,70,0,10]);
title('Variazione del guadagno');
```

---

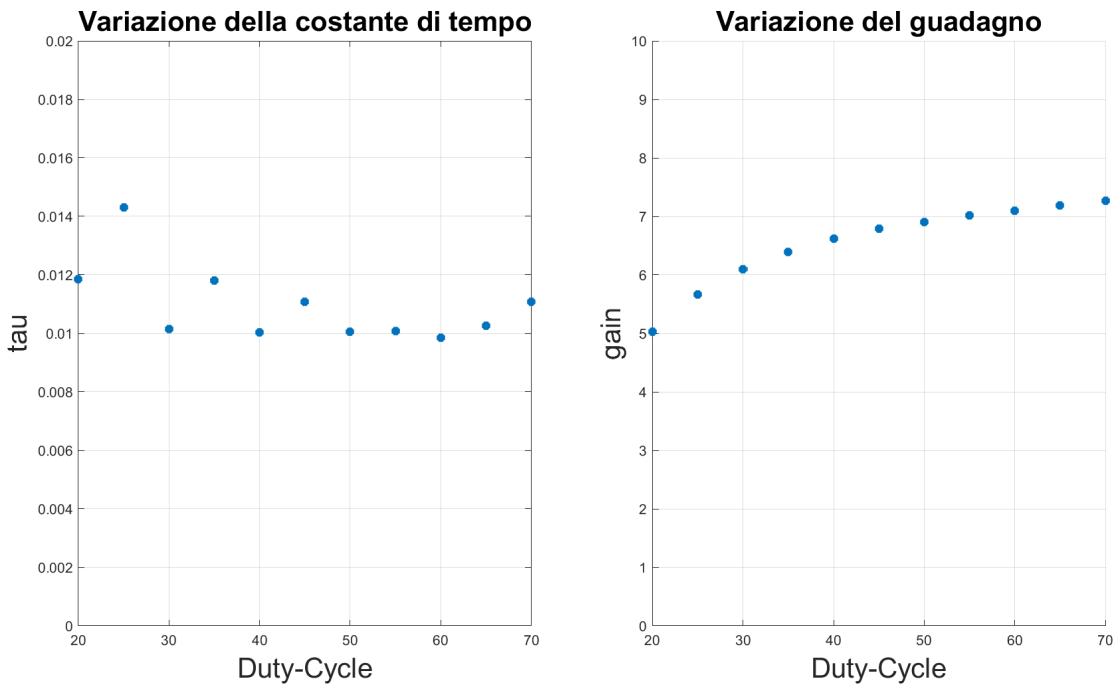


Figura 4.1: Variazione della costante di tempo e del guadagno in relazione al duty-cycle

Sia la costante di tempo che il guadagno variano in relazione all'ingresso. Nonostante ciò appartengono entrambi a fasce di valori ben definite. Se si considera una media aritmetica di entrambi i parametri si ottiene la seguente FdT.

$$G(s) = \frac{1}{s} \frac{6.55}{1 + 0.011s} \quad (4.1)$$

La 4.1 viene presa in esame per la realizzazione di un regolatore PI basato sulle formule di inversione.

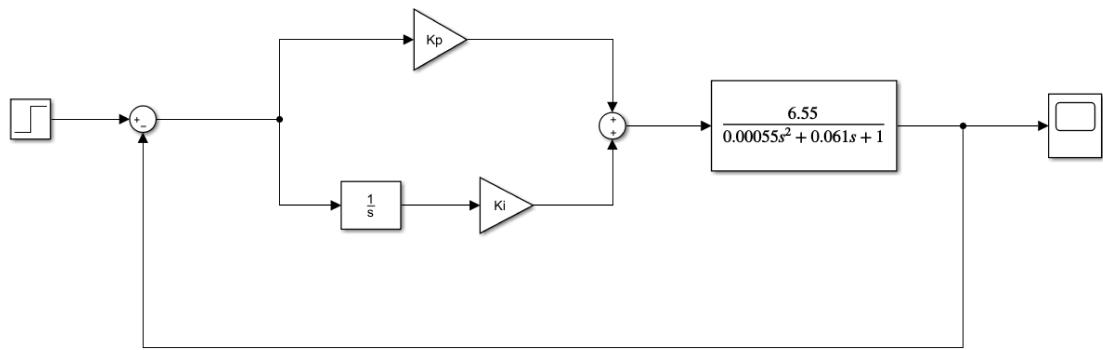
## 4.1 Regolatore PI

Nell'ipotesi che si voglia effettuare un controllo in velocità, la funzione di trasferimento che rappresenta l'impianto da controllare è dunque la  $G(s)$  ricavata precedentemente con l'aggiunta del derivatore reale.

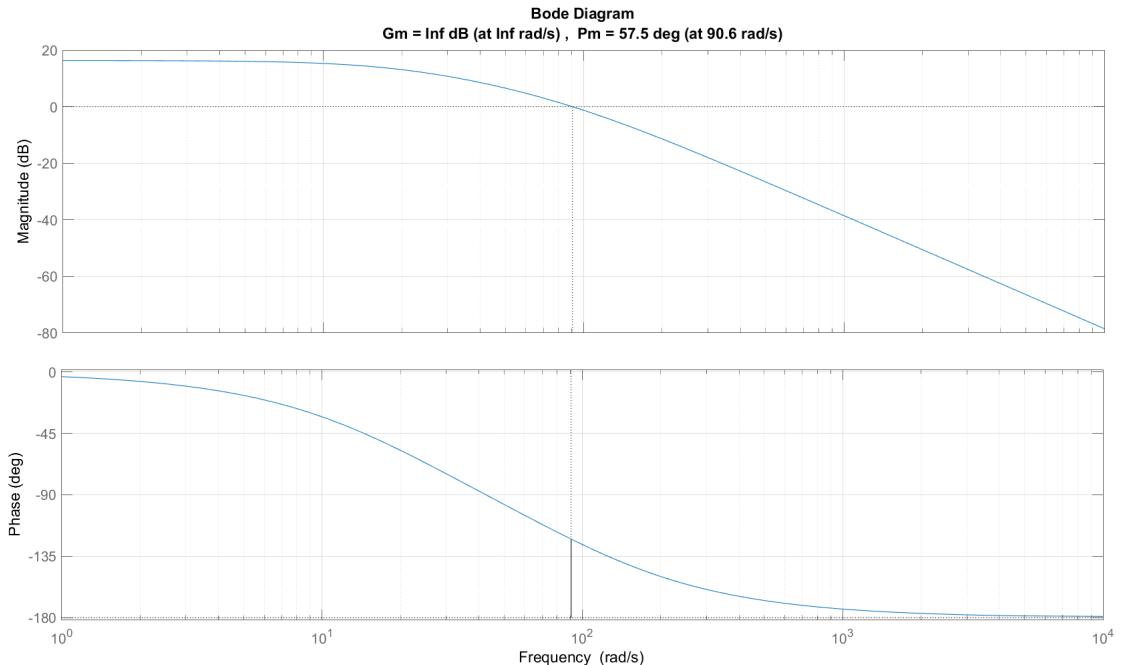
$$F(s) = G(s)D(s) = \frac{6.55}{(1 + 0.011s)(1 + 0.05s)} \quad (4.2)$$

$F(s)$  prende il nome di FdT a ciclo aperto.

Il regolatore scelto per il controllo è il regolatore PI. Quindi il sistema completo risulta essere il seguente.



In cui i diagrammi di Bode della 4.2 sono i seguenti.



Per la taratura dei parametri si applicano le formule di inversione studiate al Capitolo 1. In generale dunque si sceglie una fase  $P$  in modo che il sistema a ciclo chiuso sia approssimabile ad un sistema del primo ordine o del secondo ordine (con poli complessi e coniugati), e successivamente si sceglie una crossing frequency  $\omega_c$  che ne determini il tempo di assestamento.

### 4.1.1 Implementazione su microcontrollore

Il regolatore PI ha, nel dominio della frequenza, l'espressione 1.7. Per implementare il controllore nella STM32 si deve effettuare un procedimento di digitalizzazione e per poi ricavare un'espressione tempo-discreto. Dunque utilizzando la sostituzione di Tustin 3.5 tenendo conto che il periodo di campionamento è  $\Delta t = 0.001s$ :

$$K(s) = K_p + \frac{Ki}{s} \Rightarrow K(z) = K_p + \frac{Ki}{2000} \frac{z+1}{z-1}$$

Nel tempo discreto il coefficiente  $K_i$  viene moltiplicato per il valore 0.0005. L'espressione tempo-discreto del controllore è dunque:

$$u(n) = K_p e(n) + \frac{Ki}{2000} \sum_{k=0}^n e(k)$$

La sua realizzazione in codice C è la seguente.

```
double Kp = 0.1143;           //Azione proporzionale PID
double Ki = 0.0012;           //Azione integrale PID
double errore;
double pwm;
double somma_integrale;
double REF = 200;             //Velocità di riferimento
double dx = 36*3.14/180;       //Risoluzione

void TIM3_IRQHandler() {
    TIM3->SR &= ~1;
    new_step = TIM2->CNT;
    new_step = (double) new_step*dx;

    velocita = derivatore(new_step); //Derivata numerica

    /**RETROAZIONE DEL SISTEMA*/
    errore = REF-velocita; //Calcolo dell'errore

    /**CONTROLLO PI*/
    somma_integrale = somma_integrale + errore;
    pwm = (double) (Kp*errore + Ki*somma_integrale);
    if(pwm>70)
        pwm=70;
    if(pwm<0)
        pwm=0;
    set_pwm(pwm); //Applicazione dell'uscita
}
```

In tal caso, visto che il modello dell'impianto è stato modellato a partire dal duty-cycle, l'uscita del regolatore è anch'essa espressa in termini

di duty-cycle.

## 4.2 Approssimazione al secondo ordine

Per avere un sistema che abbia una risposta più veloce in termini tempo di salita (ma con la presenza di eventuali oscillazioni) bisogna scegliere:

$$P < -120^\circ$$

Nell'ipotesi di scegliere  $P = -135^\circ$  e supponendo di desiderare un tempo di assestamento al 5% di circa 0.1s,  $t_{a5\%} = 0.1s$ , applicando la 1.15:

$$\begin{cases} \omega_c = \frac{3}{\zeta t_{a5\%}} \Rightarrow \omega_c \simeq 66 \text{rad/s} \\ \zeta \simeq \frac{\phi_m}{100} \Rightarrow \zeta \simeq 0.45 \end{cases}$$

In Matlab si può definire uno script che generi i valori di  $K_p$  e  $K_i$  basandosi sulle formule 1.18.

---

```
%Modello;
s = tf('s');
G = 6.55/((1+0.05*s)*(1+0.011*s));
dt = 0.001;
ref = 200;

%% Controllo PI
Wc = 66;
Pc = -135;
mod = abs(evalfr(G, 1j*Wc));
phs = rad2deg(angle(evalfr(G, 1j*Wc)));
Kp = cosd(Pc-phs)/mod;
Ki = -Wc*sind(Pc-phs)/mod;
K = (Kp+Ki/s);
Ki=Ki*0.0005;           %Digitalizzazione Tustin
speed = readtable('Controllo.txt');
ts = 0:dt:(numel(speed.Var2)-1)*dt;

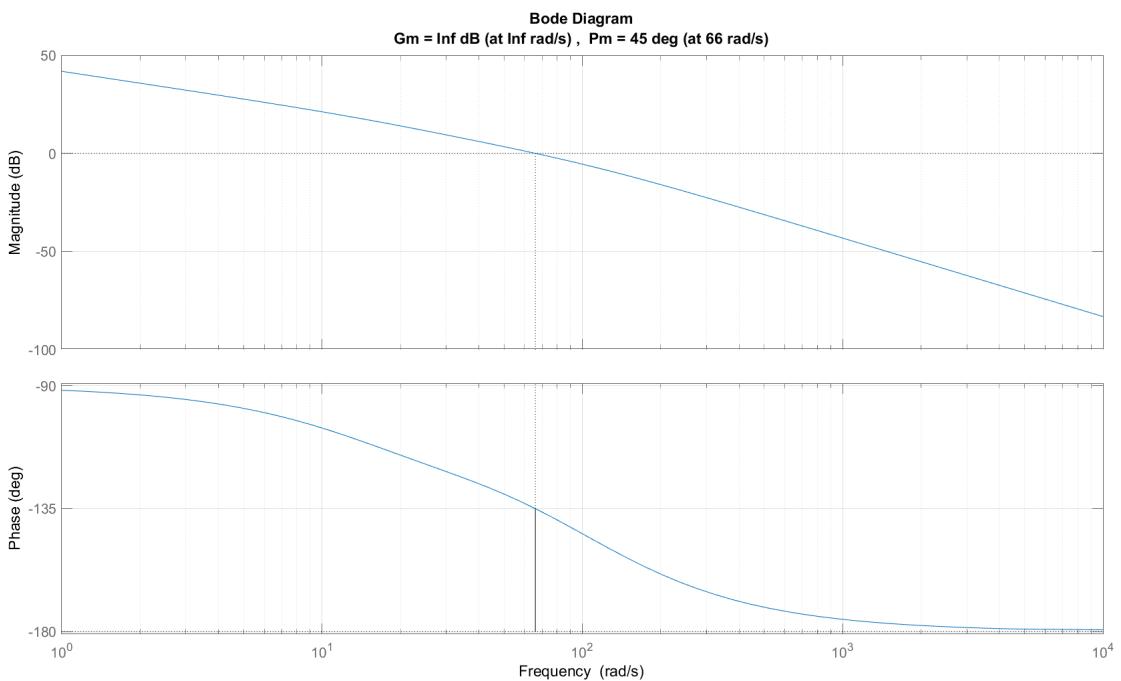
W = feedback(F*K,1);
[y,ty] = step(W,ts);
title('Regolatore PI');
xlabel('Secondi [s]'); ylabel('Velocità angolare [rad/s]');
grid;
hold on;
plot(ty,y*ref,'LineWidth',2);
plot(ts,speed.Var2);
legend('Risposta ideale', 'Risposta misurata');
axis([0,0.5,0,300]);
```

---

Definito quindi

$$K(s) = 0.5853 + \frac{18.7403}{s} \Rightarrow K(z) = 0.5853 + 0.0094 \frac{z+1}{z-1}$$

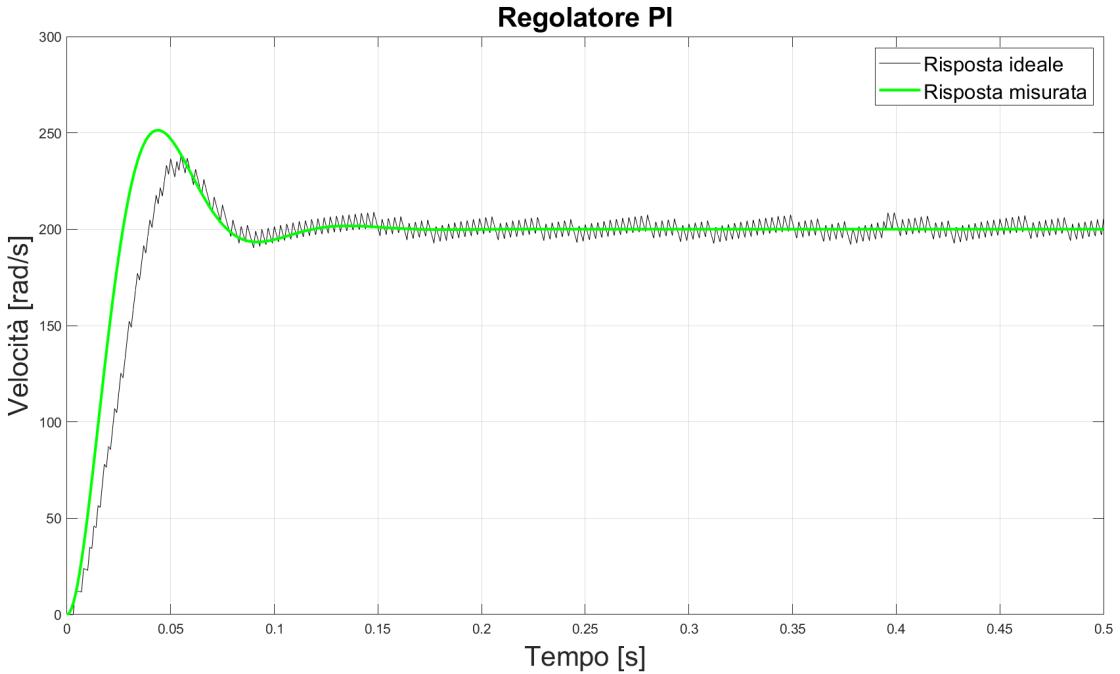
I diagrammi di Bode della funzione a ciclo aperto  $K(s)F(s)$  risultano essere:



In cui il margine di fase e la frequenza di attraversamento corrispondono con quelli desiderati a priori.

La risposta del sistema a ciclo chiuso ideale e misurata possono essere messe a confronto.

Andando a sostituire dunque i valori dei coefficienti  $K_p$  e  $K_i$  discretizzati nel codice C precedentemente descritto e analizzando la misura in Matlab con la velocità di riferimento di 200 rad/s:



Nonostante la misura sia molto rumorosa, il tempo di assestamento è circa 0.1s come previsto e la sovraelongazione percentuale si può calcolare come:

$$s \simeq e^{\frac{-\pi\zeta}{\sqrt{1-\zeta^2}}} \simeq 0.20 \quad \Rightarrow \quad s\% = 20\%$$

in accordo con la modellistica il valore di picco è circa 240rad/s.

### 4.3 Approssimazione del primo ordine

Per avere una risposta approssimabile ad un sistema del primo ordine, si deve scegliere

$$P > -120^\circ$$

Nell'ipotesi di scegliere  $P = -90$  e supponendo di desiderare un tempo di assestamento al 5% di circa 0.2s,  $t_{a5\%} = 0.2s$  applicando la 1.17

$$\omega_c = \frac{3}{t_{a5\%}} = 15 \text{ rad/s}$$

Lo stesso script di Matlab utilizzato precedentemente va a tarare i parametri del PI.

---

```
%Modello;
s = tf('s');
```

---

```

G = 6.55/((1+0.05*s)*(1+0.011*s));
dt = 0.001;
ref = 200;

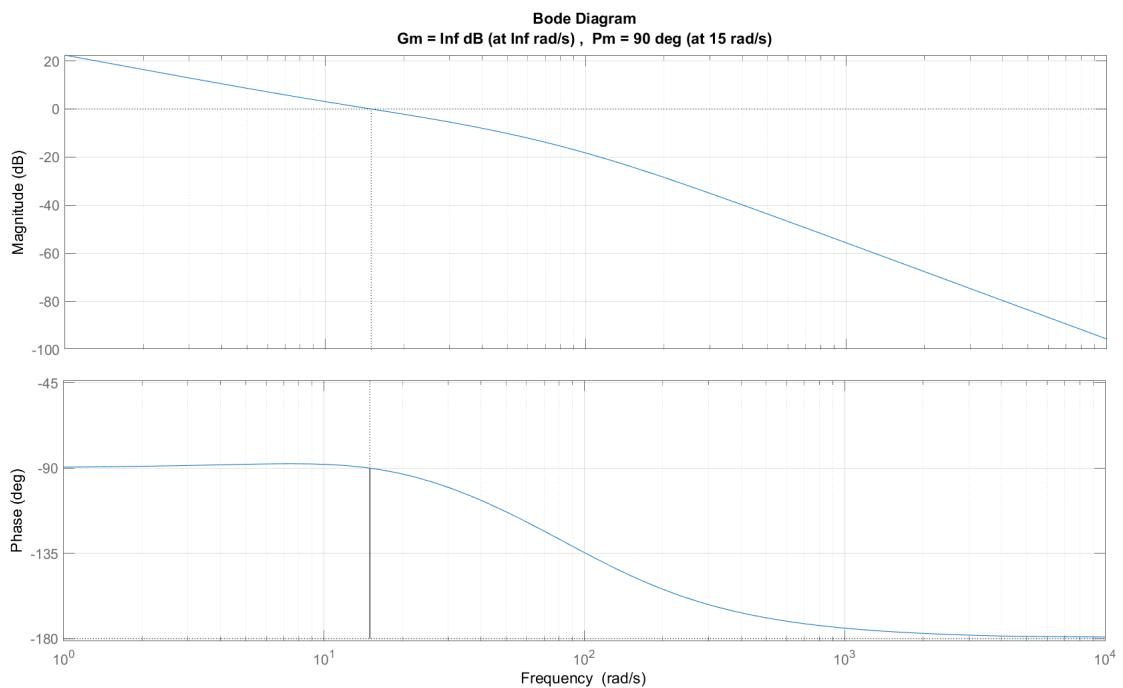
%% Controllo PI
Wc = 15;
Pc = -90;
mod = abs(evalfr(G, 1i*Wc));
phs = rad2deg(angle(evalfr(G, 1i*Wc)));
Kp = cosd(Pc-phs)/mod;
Ki = -Wc*sind(Pc-phs)/mod;
K = (Kp+Ki/s);
Ki=Ki*0.005;           %Digitalizzazione Tustin
speed = readtable('Controllo.txt');
ts = 0:dt:(numel(speed.Var2)-1)*dt;

W = feedback(F*K,1);
[y,ty] = step(W,ts);
title('Regolatore PI');
xlabel('Secondi [s]'); ylabel('Velocita angolare [rad/s]');
grid;
hold on;
plot(ty,y*ref,'LineWidth',2);
plot(ts,speed.Var2);
legend('Risposta ideale', 'Risposta misurata');
axis([0,0.5,0,300]);

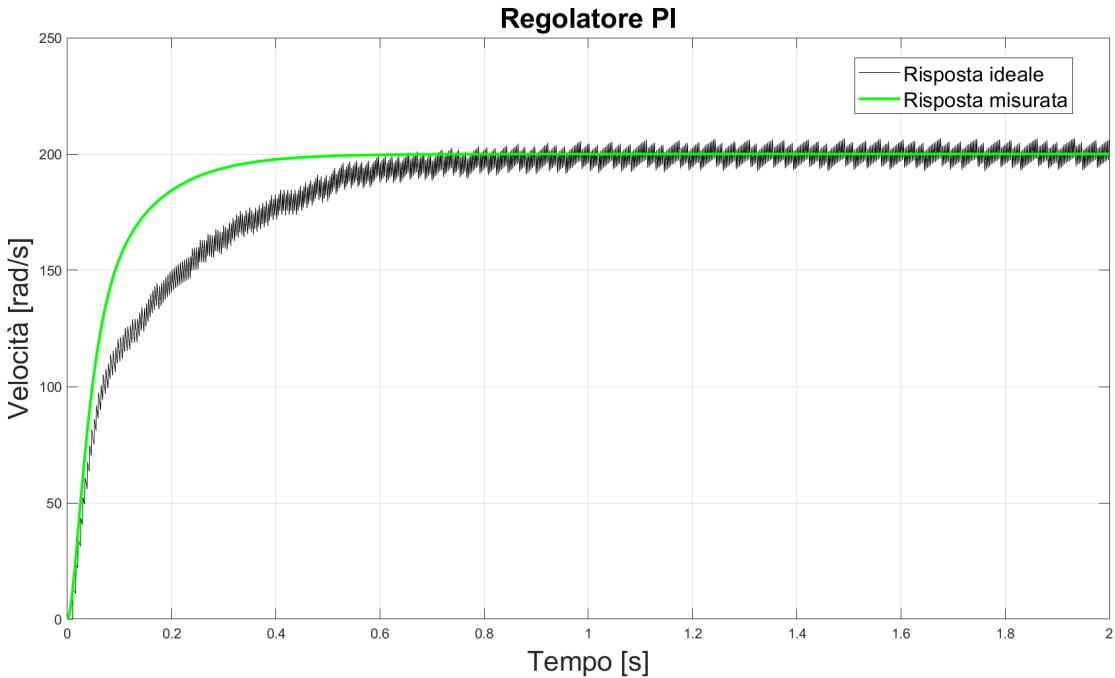
```

$$K(s) = 0.1397 + \frac{2.0067}{s} \Rightarrow K(z) = 0.1397 + 0.0010 \frac{z+1}{z-1}$$

I diagrammi di Bode della funzione a ciclo aperto sono i seguenti.



E il confronto tra le risposta ideale e misurata è il seguente.



Il tempo di assestamento è più lungo di quello previsto, questo è dovuto a vari fattori, quali l'approssimazione del modello dell'impianto e la non linearità generata dal driver e dall'encoder.

Il codice completo per la realizzazione del regolatore è il seguente.

"Encoder.h"

```
#ifndef ENCODER_H
#define ENCODER_H
#include <stm32f30x.h>
#define FRQ 72000000

void init_encoder();
void init_pwm();
int init_timer(int);

void set_pwm(double);
void setDirection(int);

void start_timer();
void start_encoder();

#endif
```

"Encoder.c"

```
void init_encoder() {
    /* TIM2_Ch2 -> PA1 (AF1) */
```

```

RCC->AHBENR |= RCC_AHBENR_GPIOAEN;           //Abilito GPIOA
GPIOA->MODER |= 1<<3;                      //PA1 Alternate Function
GPIOA->MODER &= ~(1<<2);
GPIOA->AFR[0] |= 1<<4;                       //Configuro AF1

RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;          //Abilito TIM2
TIM2->SMCR |= 7;                            //Clock esterno
TIM2->CCMR1 &= ~(1<<9);                   //Ch2 usato come input
TIM2->CCMR1 |= (1<<8);
TIM2->SMCR |= 1<<6;
TIM2->SMCR |= 1<<5;
TIM2->SMCR &= ~(1<<4);
}

void start_encoder() {
    TIM2->CR1 |= TIM_CR1_CEN;                  //Abilito il conteggio
}

void setDirection(int dir) {
    RCC->AHBENR |= RCC_AHBENR_GPIOBEN;         //Abilito GPIOB
    GPIOB->MODER |= 1;                         //PB0 come output
    GPIOB->MODER &= ~(1<<1);
    GPIOB->MODER |= 1<<2;                   //PB1 come output
    GPIOB->MODER &= ~(1<<3);
    GPIOB->ODR |= 1<<1;                      //PB1 uscita alta
    GPIOB->MODER |= 1<<4;                   //PB2 come output
    GPIOB->MODER &= ~(1<<5);

    switch(dir) {
        case 1:
            GPIOB->ODR |= 1;
            GPIOB->ODR &= ~(1<<2);
            break;
        case 0:
            GPIOB->ODR &= ~1;
            GPIOB->ODR |= (1<<2);
            break;
    }
}

void init_pwm() {
/*TIM4_Ch1-> PD12 (AF2) */
    RCC->AHBENR |= RCC_AHBENR_GPIODEN;         //Abilito GPIOD
    GPIOD->MODER |= 1<<25;                   //D12 Alternate Function
    GPIOD->MODER &= ~(1<<24);
    GPIOD->AFR[1] = 0;                        //Azzero il registro
    GPIOD->AFR[1] |= 1<<17;                  //Configuro AF2

    setDirection(1);

    RCC->APB1ENR |= RCC_APB1ENR_TIM4EN;        //Abilito TIM4
    TIM4->CCMR1 |= 1<<3;                     //Accesso al CCR1
    TIM4->CCMR1 |= 1<<6;                     //PWM Mode 1
    TIM4->CCMR1 |= 1<<5;
    TIM4->CCMR1 &= ~(1<<4);
    TIM4->CCER |= 1;                          //Abilito il confronto

    TIM4->ARR = 1440;                         //Frq 50kHz
}

```

```

        TIM4->CR1 |= 1;                                //Abilito il conteggio
    }

void set_pwm(double duty) {
    TIM4->CCR1 = (int) (1440*duty)/100;      //duty-cycle
}

int init_timer(int frequenza_attivazione) {
/**Frequenza di campionamento */
    int i = 1;
    uint32_t frequenza_timer;
    uint32_t arr;
    RCC->APB1ENR |= RCC_APB1ENR_TIM3EN;      //Abilito TIM3
    do{
        frequenza_timer = FRQ/i;
        arr = (uint32_t) frequenza_timer/frequenza_attivazione;
        i++;
    }while(arr > 65535);
    TIM3->ARR = (uint16_t) arr;                //Auto-reload register
    TIM3->PSC = (uint16_t) i-1;                  //Scalatura del clock
    TIM3->DIER |= 1;                            //Abilito le interruzioni
    NVIC->ISER[0] |= 1<<29;                    //ISR per il TIM3
    return arr;
}

void start_timer() {
    TIM3->CR1 |= TIM_CR1_CEN;                  //Abilito il conteggio
}

"main.c"

#include <stm32f30x.h>
#include <Encoder.h>
#define CONV 500          //Valore massimo del vettore di convoluzione
#define MAX 2000

int fc = 1000;                      //Frequenza di campionamento

double matlab[MAX];           //Array da passare a matlab
int index_matlab;
double dx;                      //Risoluzione dell'encoder
int index_pos;

double theta[CONV];            //Vettore della posizione angolare
double vettore_convoluzione[CONV]; //Vettore per la convoluzione
int index_theta;
double velocita;

double alfa=20.2, beta=-0.4, gamma=0.9802; //Parametri del derivatore

double Kp = 0.1397;             //Azione proporzionale PID
double Ki = 0.0010;             //Azione integrale PID
double new_step;               //Posizione corrente
double errore;
double pwm;

```

```

double somma_integrale; //Variabile di appoggio per l'integrazione

double REF = 200; //Velocita di riferimento

double exp(double base, int esp) {
    double prodotto=1;
    if(esp == 0)
        return 1;
    else{
        for(int i=0;i<esp;i++)
            prodotto = (double) prodotto* base;
        return prodotto;
    }
}

void init_derivatore() {
    for(int i=0; i<CONV; i++) {
        vettore_convoluzione[i] = exp(gamma, i);
        theta[i] = 0;
    }
}

void inserisci(double* vettore, double pos ,int dim) {
    for(int i=0; i<dim-1; i++) {
        vettore[i] = vettore[i+1];
    }
    vettore[dim-1] = pos;
}

double derivatore(double theta_new) {
    double somma_differenziale=0;
    double prod_conv = 1;
    inserisci(theta,theta_new,CONV);
    //Convoluzione discreta
    for(int i=0, k=CONV-1; i<CONV && k>=0; i++, k--) {
        prod_conv = theta[k]*vettore_convoluzione[i];
        somma_differenziale = somma_differenziale+prod_conv;
    }
    velocita = (alfa*theta_new)+(beta*somma_differenziale);
    return velocita;
}

int main(){
    init_encoder();
    init_timer(fc);
    init_pwm();

    start_encoder();
    init_derivatore();
    somma_integrale = 0;
    errore = 0;
    dx = 36*3.14/180;
    start_timer();

    set_pwm(0);

    while(1);
}

```

```
void TIM3_IRQHandler() {
    TIM3->SR &= ~1;
    new_step = TIM2->CNT;
    new_step = (double) new_step*dx;
    velocita = derivatore(new_step);

    /**RETROAZIONE DEL SISTEMA*/
    errore = REF-velocita;

    /**CONTROLLO PID*/
    somma_integrale = somma_integrale + errore;
    pwm = (double) (Kp*errore + Ki*somma_integrale);
    if(pwm>70)
        pwm=70;
    if(pwm<0)
        pwm=0;
    set_pwm(pwm);

    /**VETTORE DI PASSAGGIO PER MATLAB*/
    if(index_matlab<MAX) {
        matlab[index_matlab] = velocita_filtrata;
        index_matlab++;
    }
}
```

# Conclusioni

Questo lavoro di tesi presenta la realizzazione digitale di un prototipo di un controllore PI applicabile ad una qualunque macchina a corrente continua, purché si effettuino correttamente le fasi descritte al capitolo tre e al capitolo quattro. Durante le sperimentazioni è stata messa in risalto la scarsa qualità dell'encoder utilizzato, che ha prodotto misurazioni alquanto rumorose ma che rispettano le specifiche teoriche. Gli strumenti che si sono avuti a disposizione non sono dunque strumenti di alta qualità. Lo scopo dell'elaborato è di fatti puramente didattico. In relazione alle aspettative il controllo è funzionante. La particolarità sta nell'aver realizzato un algoritmo di controllo parametrico e modulare, permettendo un'alta manutenibilità del codice. Con la stessa scheda infatti si può controllare qualunque altra macchina a corrente continua cambiando solo i parametri necessari: periodo di campionamento, coefficienti del PI e coefficienti del derivatore reale. La modularità consente una rapida implementazione, in futuro, di filtri per il rumore introdotto dagli strumenti hardware col fine di migliorarne le prestazioni.

In ambito industriale si devono utilizzare strumenti di alta precisione, sia per quanto riguarda l'encoder (che permette un controllo molto preciso) sia per quanto riguarda il driver. Quest'ultimo infatti deve essere ben dimensionato in relazione alle caratteristiche del motore da pilotare, ne vale il danneggiamento fisico dello stesso.

# Bibliografia

- [1] *Datasheet - TB66112FNG.*
- [2] [https://en.wikipedia.org/wiki/pulse-width\\_modulation](https://en.wikipedia.org/wiki/pulse-width_modulation)
- [3] <https://it.emcelettronica.com/controllo-di-motore-corrente-continua-analisi-e-progetto-con-matlabsimulink>
- [4] *Reference Manual - STM32F303VC.*
- [5] Kenneth C. Smith Adel S. Sedra. *Circuiti per la microelettronica.*
- [6] Francesco Verde Giacinto Gelli. *Segnali e Sistemi.*
- [7] Nicola Schiavoni Paolo Bolzern, Riccardo Scattolini. *Fondamenti di controlli automatici.*
- [8] Malcolm J. Skove W. Edward Gettys, Frederick J. Keller. *Fisica 2 - Elettromagnetismo e Onde.*