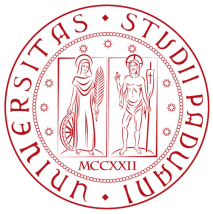

UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Ingegneria dell'informazione

Corso di Laurea Triennale in Ingegneria Elettronica

PROGETTO E REALIZZAZIONE DI UN CONTROLLORE DI VELOCITÀ PER MOTORINO

Relatore:

Prof. Matteo Meneghini

Laureando:

Federico Griso

Matricola n. 1216285

Anno Accademico 2021/22

Sommario

Questo progetto di tesi si pone come obiettivo la realizzazione di un controllore di velocità per motorino gestito tramite Arduino. Verranno analizzati i componenti utilizzati ed i risultati ottenuti.

Il dispositivo permette di impostare la velocità che si vuole che il motorino raggiunga a regime. Successivamente tramite PWM si cerca di arrivare a tale velocità e tramite encoder incrementale rotativo si controlla che il valore effettivo sia quello impostato. Per raggiungere il valore a regime desiderato è necessario utilizzare un controllore PID che, se realizzato in modo adeguato, permette di ottenere un sistema che arrivi a tale valore senza oscillazioni e con tempi di risposta ragionevoli, anche in caso di perturbazioni esterne che possono verificarsi nell'uso comune del dispositivo.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Cruise control

1.1 Storia

La nascita del cruise control risale all'anno 1948 quando Ralph Teetor, originario degli Stati Uniti, ultimò il progetto di un sistema che permette di impostare una precisa velocità dell'auto e di lasciare il pedale dell'acceleratore mantenendo così un'andatura costante. Questo sistema gli valse poi un posto nella Automotive Hall of Fame 6 anni dopo la sua morte.

Essendo non vedente e non potendo guidare aveva la necessità di qualcuno che lo aiutasse in tal senso e lo facesse al posto suo. Disturbato dal tipo di guida dei suoi accompagnatori e dal continuo variare della pressione sull'acceleratore che portava a continui aumenti e diminuzioni di velocità, decise infine di cercare una soluzione per rendere i viaggi più confortevoli.

Il progetto venne ultimato nel 1948. Il sistema assunse poi parecchie denominazioni negli anni, come ad esempio *Speedostat*, *Tempomat* e *Pressomatic*, fu infine Cadillac che decise di chiamarlo *cruise control*.

La prima volta che questo sistema venne applicato su un'automobile fu nel 1958 quando la Chrysler lanciò una nuova linea di lusso (Chrysler Imperial) decidendo di renderlo un optional e di chiamarlo *Auto-pilot*.

Il successo di questo dispositivo è anche dovuto all'embargo sul petrolio del 1978 a danno degli Stati Uniti. Per ovviare a questo problema vennero istituiti dei limiti di velocità che permisero di risparmiare 167000 barili di petrolio al giorno e in tutto ciò il cruise control giocò un ruolo fondamentale.

Verso la fine degli anni '80 il dispositivo passò dall'essere meccanico all'essere elettronico grazie alla Motorola, la quale realizzò il primo circuito integrato dedicato (MC14460 Automotive Speed Control Processor).

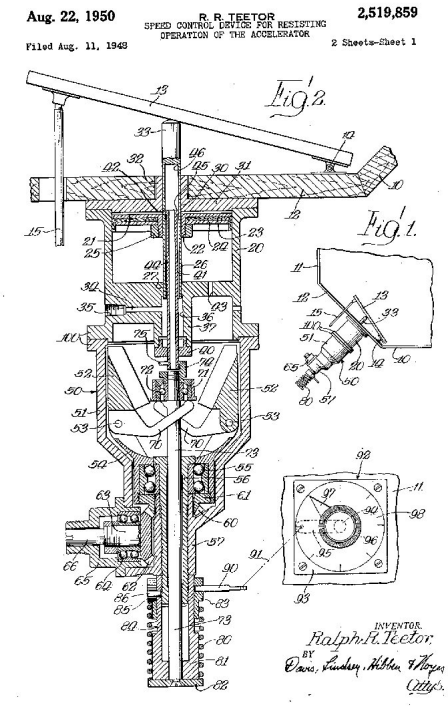

Figura 1.1: Progetto cruise control di Teetor.

1.2 Funzionamento

Il funzionamento si basa sulla possibilità di impostare una velocità e di mantenerla costante senza premere l'acceleratore.

Il cruise control moderno di solito si attiva tramite un comando dedicato che si trova sul volante o sullo sterzo che viene premuto quando si raggiunge la velocità desiderata. Inoltre, in genere, si ha la possibilità di variare l'andatura tramite altri pulsanti (+ o -), oppure accelerando e premendo nuovamente il tasto per l'attivazione.

Il sistema si disattiva automaticamente una volta che viene premuto il pedale del freno, della frizione o quando viene premuto nuovamente il tasto dedicato.

Si trovano principalmente due tipi di cruise control in vendita sul mercato:

- Cruise control
- Adaptive cruise control (ACC)

Di seguito vengono riportate le loro differenze.

Cruise control: è la prima tipologia ed anche la più diffusa; questo sistema gestisce la velocità esclusivamente basandosi sul valore che viene impostato senza interagire con possibili ostacoli esterni.

Adaptive cruise control: è un sistema più evoluto del precedente e permette di regolare automaticamente la velocità in base alla distanza dal veicolo che precede accelerando o rallentando senza che l'utente intervenga sui pedali, mentre in caso di strada libera agisce esattamente

come la versione base del cruise control portando la velocità del veicolo al valore desiderato. Per ottenere questo risultato l'ACC utilizza un radar, posto nella parte anteriore dell'auto, che è in grado di riconoscere gli ostacoli. Nei modelli recenti questo sistema viene anche associato con una telecamera in grado di riconoscere i limiti di velocità e quindi di adeguare quest'ultima a quella rilevata.

Vantaggi:

- Utile per viaggi lunghi riducendo la stanchezza del guidatore
- Minor possibilità di superare i limiti di velocità
- Riduzione consumo di carburante

Svantaggi:

- Rischio velocità elevate su curve pericolose
- Minor attenzione del guidatore e maggior rischio di incidenti

Capitolo 2

Progetto ed analisi componenti

2.1 Introduzione al progetto e componentistica

L'idea per il seguente progetto deriva dall'interesse nei confronti del cruise control. Il dispositivo realizzato consiste nella messa a punto di un sistema che sia in grado di mantenere una velocità costante impostata dall'utente e di mantenerla anche in caso di perturbazioni, con l'obiettivo di riportarsi al valore desiderato in un tempo ragionevole e senza subire troppe oscillazioni rispetto al valore a regime che si vuole raggiungere.

2.1.1 Schema a blocchi

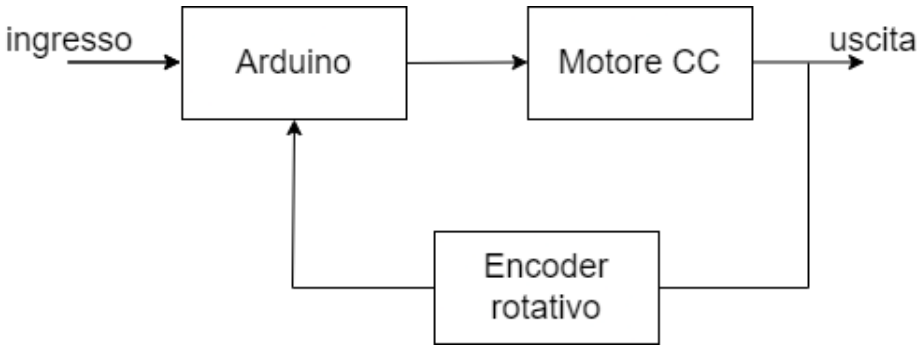

Figura 2.1: Schema a blocchi di progetto.

Dallo schema a blocchi si evince che è necessario avere uno stadio che fornisce l'ingresso desiderato. In pratica consiste in un potenziometro da $10k\Omega$ che permette, tramite la rotazione della manopola, di creare un partitore di tensione. Questo consente di decidere la velocità che si vuole far raggiungere al motorino in uscita. Servirà poi un interruttore che permetta di attivare o meno il sistema.

L'ingresso viene quindi portato ad un microcontrollore, il quale ha il compito di elaborare tutti i segnali che riceve in ingresso e di fornire in uscita un segnale adeguato per soddisfare le richieste dell'utente.

Successivamente si ha un blocco di uscita, che viene attuato dal segnale ricevuto dal microcontrollore.

Infine è necessario avere uno stadio di retroazione, che corrisponde ad un sensore che misura il risultato ottenuto in uscita e lo riporta in ingresso al microcontrollore per permettergli di effettuare controlli ed eventuali modifiche al suo comportamento così da ottenere i risultati desiderati.

Nei prossimi paragrafi è riportata una descrizione teorica della componentistica utilizzata per la realizzazione del progetto.

2.2 Pull-up e pull-down

Nei sistemi digitali deve essere assolutamente evitato che vengano forniti in ingresso dei segnali flottanti. Questo perché possono essere interpretati in modo sbagliato da un eventuale microcontrollore.

Per ovviare a questo problema si utilizzano tecniche di pull-up o pull-down, che consistono nell'utilizzo di un resistore aggiuntivo. Inoltre bisogna cercare di evitare che il pin connesso al microcontrollore possa trovarsi nella situazione in cui sia collegato contemporaneamente al livello alto ed al livello basso del circuito di ingresso.

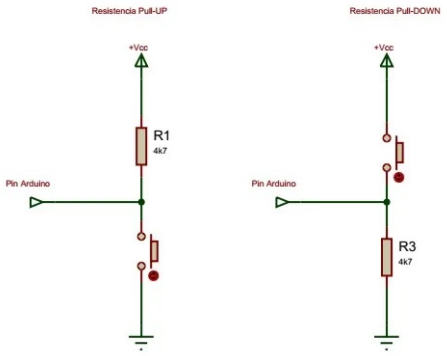

Figura 2.2: Schemi pull-up e pull-down.

Nel caso specifico di questo progetto è stata utilizzata la configurazione pull-down per fornire al microcontrollore il segnale di attivazione o spegnimento del motorino che dunque attiva o meno il processo di lettura da parte del microcontrollore.

È necessario utilizzare il pull-down perché si rischierebbe di trovarsi in una situazione di ingresso flottante nel caso di interruttore aperto.

Per la realizzazione del circuito si utilizza un resistore di valore elevato per evitare scorrimento di correnti elevate su di esso e quindi ridurre al minimo la dissipazione di potenza sulla resistenza.

In questo modo si ottiene in ingresso ad Arduino la tensione fornita dall'alimentazione (5V) nel caso in cui l'interruttore sia chiuso, che corrisponde alla caduta di tensione sulla resistenza. Mentre nel momento in cui l'interuttore è aperto, il pin di ingresso del microcontrollore è collegato a massa direttamente tramite la resistenza.

2.3 Arduino

Il cuore pulsante del progetto è composto da un microcontrollore, nel caso specifico un Arduino UNO, che ha il compito di controllare, elaborare e gestire tutti i segnali che riguardano il sistema e di adottare comportamenti ben precisi per ottenere il risultato che è stato prefissato al paragrafo 2.1.

Arduino è stato ideato e sviluppato nel 2005 come strumento per la prototipazione rapida a scopi principalmente didattici, infatti non è l'ideale per la creazione di sistemi specifici e dedicati. La piattaforma offre un'infinità di possibili utilizzi e ciò rende Arduino molto versatile, ma per applicazioni dedicate lascia inutilizzate moltissime funzioni, creando uno spreco di risorse e componentistica non indifferente. Alcune sue possibili applicazioni possono essere: controllori di luci, di velocità per motori, sensori di temperatura, comunicazioni con altre periferiche e tanto altro ancora.

2.3.1 Architettura

La piattaforma è costituita da: un circuito stampato che integra un microcontrollore che elabora dati e segnali, porte I/O per la comunicazione con l'esterno, un regolatore di tensione per mantenere la tensione costante sulla piattaforma e per non creare sbalzi che potrebbero essere fatali per la scheda o per l'interpretazione dei segnali e un'interfaccia USB per la comunicazione tramite seriale con il computer utilizzato per programmare. All'hardware appena descritto viene affiancato anche un ambiente di sviluppo integrato (IDE). Il software permette anche ai meno esperti di creare semplici script in quanto il linguaggio di programmazione utilizzato è molto intuitivo, derivato da C e da C++.

2.3.2 Hardware

Una scheda Arduino comprende un microcontrollore a 8 bit prodotto da Atmel. Inoltre, come detto anticipatamente, possiede un regolatore di tensione ed un oscillatore a cristallo a 16 MHz. Quest'ultimo è fondamentale per dare il timing necessario per l'esecuzione delle istruzioni per l'ottenimento dei risultati richiesti dal sistema.

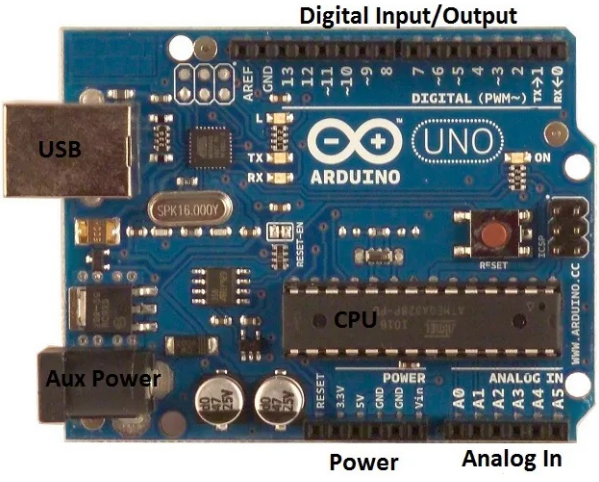

Figura 2.3: Scheda Arduino UNO.

Pin I/0

Arduino è dotato di molti connettori di input/output che sono presenti sulla parte superiore della scheda; questi hanno permesso anche lo sviluppo di molte schede applicative plug-in (*shields*) che sono compatibili con la scheda e direttamente applicabili su di essa.

Un'accortezza che va sempre tenuta in considerazione riguarda i pin di uscita che non sono in grado di erogare correnti elevate. Questo è dovuto al fatto che Arduino preleva circa 500mA dalla porta USB al quale è collegato. Il massimo di corrente erogabile da ogni pin digitale è pari a 40mA, fino a un totale tra tutti i pin di massimo 150mA. Questo rende necessario nella maggior parte delle applicazioni l'utilizzo di transistor collegati ai pin, che permettono di consumare poca corrente e hanno successivamente il compito di pilotare eventuali attuatori collegati a alimentazioni esterne, in base ai segnali forniti dal microcontrollore.

Esistono due tipi di pin sulla piattaforma con caratteristiche e funzioni diverse:

- **I/O digitali:** sono presenti 14 pin digitali che possono essere impostati a piacere come input o output. Ciò è definito da apposite istruzioni presenti nello sketch che viene programmato. Sei di questi pin offrono inoltre la possibilità di generare segnali PWM (Pulse Width Modulation), che verranno affrontati e descritti in un paragrafo successivo. Questi particolari pin permettono di regolare l'uscita tramite la funzione analogWrite(), che offre la possibilità quindi di definire ad esempio la velocità di rotazione di un motorino o l'intensità luminosa di un LED. Mentre in generale per tutti i pin digitali si possono leggere o scrivere 0 e 1, che elettricamente corrispondono a 0V e 5V (alimentazione di Arduino).

- **I/O analogici:** sono presenti 6 pin dedicati esclusivamente ad ingressi di segnali analogici che devono essere compresi nel range $0 \div 5$V. Sono quindi collegati ad un ADC che li converte in segnali digitali composti da 1024 livelli. Questo perché l'ADC lavora a 10 bit, anche se all'occorrenza si può aumentare la sua precisione tramite specifiche tecniche, o eventualmente utilizzando ADC esterni per la conversione dei segnali.

2.3.3 Software

L'ambiente di sviluppo (IDE) di Arduino è concepito per permettere anche ai principianti di iniziare a sviluppare software per progetti fai da te anche molto semplici. Per permettere la stesura del codice sorgente, l'IDE include un editor di testo che è in grado di compilare e caricare il programma sulla scheda. All'interno dell'IDE sono già presenti alcuni sketch di esempio per aiutare l'utente nei primi passi.

Sono inoltre fondamentali le librerie, alcune già presenti di default, che permettono di utilizzare funzioni definite precedentemente, senza doverle scrivere da capo ogni qual volta fossero necessarie per un progetto.

L'IDE offre, inoltre, la possibilità di comunicare tramite seriale con Arduino durante l'esecuzione di un programma. Si tratta di aprire il *Serial monitor*, dove si è in grado di inserire dati o di leggere risultati utili alla comprensione di ciò che il microntrollore sta elaborando. Questa funzione viene spesso utilizzata per il debugging di un programma, in modo tale da verificare che quest'ultimo venga eseguito in modo corretto.

Il compito che svolge Arduino nel progetto è quello di leggere il dato fornito dal potenziometro posto in ingresso; il valore letto corrisponde alla velocità che si desidera venga raggiunta

dal motorino. Da questo dato si ricava dunque il valore di duty cycle che si deve impostare alla PWM che ha il compito di regolare la velocità del motorino a quella desiderata. È quindi fondamentale che tramite un sensore, posto sul motorino, il microcontrollore abbia la possibilità di verificare continuamente che la velocità effettiva a cui si sta andando sia quella desiderata. Perciò, dopo una serie di elaborazioni dei dati ricevuti dal sensore, si utilizza un sistema di controllo specifico che permette di variare il valore di duty cycle nel caso in cui si vengano a verificare perturbazioni sul valore di velocità effettivo o nel caso in cui ci siano differenze tra il valore misurato e quello desiderato. Per fare ciò si implementa un controllo PID, che verrà descritto in dettaglio successivamente. Il PID è un sistema che effettua operazioni sul valore dell'errore tra ingresso e uscita, per fare in modo di portarsi al valore desiderato nel minor tempo possibile e soprattuto senza oscillazioni, che nella maggior parte delle applicazioni possono portare a danni irreparabili.

2.4 PWM

Il PWM (Pulse Width Modulation) è una tecnica che permette di ottenere risultati attribuibili a segnali analogici a partire da segnali digitali. Il segnale digitale preso in considerazione è utilizzato per creare un'onda quadra che varia tra on ed off, dove 'on' corrisponde alla Vcc della scheda Arduino e 'off' al valore 0V. La variazione della porzione di tempo in cui il segnale rimane al valore alto rispetto al valore basso permette di simulare una tensione compresa nel range 0-5V.

Per variare i valori analogici desiderati è quindi necessario modificare il valore di duty cycle del PWM. Una possibile applicazione di questa tecnica, ad esempio, la si può trovare nella modulazione della luminosità di un LED. Infatti impostando un valore di duty cycle pari al 100%, si ottiene il massimo di luminosità, in quanto il LED riceve una tensione continua pari a 5V, mentre nel caso in cui si impostasse un valore molto basso, ad esempio 10%, la luminosità del LED risulterebbe essere molto bassa.

Per ottenere il risultato desiderato bisogna disporre di un periodo adeguatamente piccolo in modo da avere variazioni di impulsi molto rapidi. Nel caso in esame, dove la PWM viene fornita da Arduino, si ha una frequenza di 500 Hz (periodo 2 ms). Su Arduino si hanno, come detto in precedenza, appositi pin che offrono la possibilità di avere la PWM. Per richiamare questa tecnica sullo sketch è necessario utilizzare la funzione analogWrite() su una scala compresa tra 0 e 255, perché Arduino lavora su 8 bit. Nel caso in cui si volesse avere il valore massimo di tensione ottenibile dal microcontrollore, è necessario scrivere analogWrite(255) che corrisponde al 100% di duty cycle, invece scrivendo analogWrite(127) si ha il 50% di duty cycle. Alcuni esempi di ciò che è stato spiegato sono riportati in Figura 2.4.

Nel caso specifico del progetto, la PWM ha il compito di regolare la velocità del motorino, per ottenere quella desiderata impostata dall'utente. Per fare ciò bisogna collegare un mosfet in uscita ad Arduino, che tramite il suo ciclo ripetuto di accensione e spegnimento, causato dalla PWM, porta la velocità del motorino a quella impostata.

È necessario l'utilizzo del mosfet per limitare il flusso di potenza in uscita dal microcontrollore, che nel caso in cui venisse collegato direttamente al motorino porterebbe quasi sicuramente alla rottura, per la richiesta eccessiva di corrente da parte del motorino. Motivo per cui, per fornire la potenza necessaria, bisogna adottare un'alimentazione esterna.

Come detto in precedenza, nel caso in cui venisse impostato il valore massimo di duty cycle, il motorino raggiungerebbe a regime la velocità massima, mentre con una modulazione della

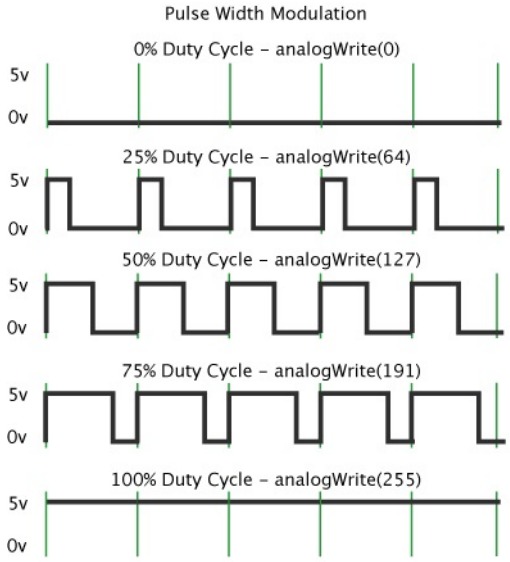

Figura 2.4: Esempi valori di PWM su Arduino.

PWM si ottiene un valore regolabile a piacere.

Il valore della PWM viene regolato automaticamente da un controllo PID, affinché si realizzi un sistema che modifichi autonomamente il valore di duty cycle da fornire alla PWM per raggiungere la velocità desiderata dall'utente, impostata tramite un potenziometro, anche in caso di perturbazioni del sistema.

2.5 Controllo PID

Il controllo PID è un sistema in grado di gestire una grandezza e di tenerla sotto controllo. Il PID riceve in ingresso un valore della grandezza da controllare (set-point) e dei parametri per eseguire la regolazione del sistema.

Tale sistema di controllo è di gran lunga il più utilizzato in ambiti ingegneristici ed è l'acronimo di Proporzionale Integrativo Derivativo.

Lo schema di principio è quello riportato in Figura 2.5.

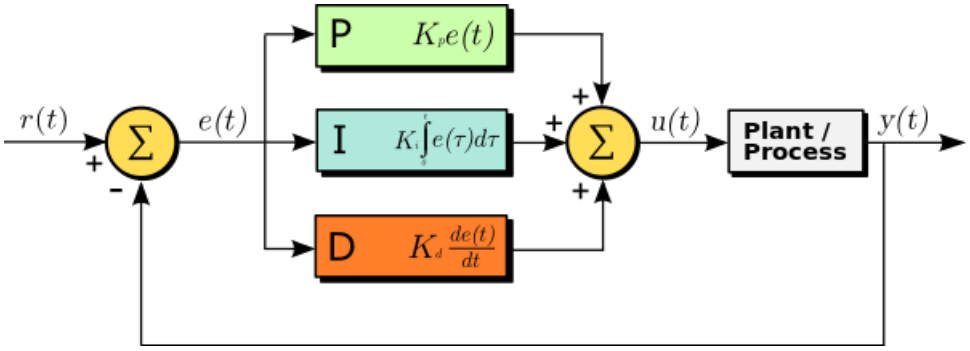

Figura 2.5: Schema a blocchi PID.

In sostanza, in ingresso viene fornito il valore di set-point, ovvero il valore che si vuole ottenere in uscita, il quale viene comparato con il valore di uscita in tempo reale. Dalla

differenza tra i due segnali si ottiene un valore di errore che viene successivamente processato e valutato per cercare di portare, nel minor tempo possibile e con meno oscillazioni possibili, il valore di tale errore ad annullarsi. Per fare ciò è necessario introdurre tre parametri:

- Proporzionale P
- Integrativo I
- Derivativo D

Per realizzare un controllore adeguato è necessario per la maggior parte dei casi utilizzare tutte le componenti menzionate precedentemente, in quanto l'utilizzo di soltanto alcune di esse porta vantaggi, ma anche svantaggi nel modo in cui si raggiunge il valore desiderato in uscita.

La formula che descrive il funzionamento di tale controllo è la seguente:

$$u(t) = K_P \cdot e(t) + K_I \cdot \int_0^t e(\tau)d\tau + K_D \cdot \frac{de(t)}{dt} \tag{2.1}$$

Il primo parametro che viene esaminato è quello Proporzionale K_P. Tale parametro prende in considerazione esclusivamente l'errore che viene calcolato tramite la differenza tra il valore desiderato e quello in uscita in tempo reale. Una volta ottenuto l'errore, esso viene moltiplicato per la costante K_P. Più è elevato il valore del parametro e maggiore sarà il peso di tale componente sul controllo, mentre tale contributo diminuisce man mano che il valore in tempo reale si avvicina al valore di set-point; tale regolazione è immediata e non crea ritardi al sistema.

Dunque la formula in questo caso risulta essere:

$$u(t) = K_P \cdot e(t) \tag{2.2}$$

Nel momento in cui si utilizza il parametro proporzionale, l'inconveniente principale in cui ci si imbatte si verifica perché, nella maggior parte dei casi, si vengono a creare sovraelongazioni ed oscillazioni nella risposta a variazioni di ingressi. Questo perché il parametro K_P viene progettato solitamente con valori molto elevati per ottenere risposte molto rapide, creando così oscillazioni elevate che possono recare danni al sistema.

La determinazione del parametro K_P è arbitraria ed in base a tale valore cambia notevolmente il comportamento del sistema. Se ad esempio si imposta un valore pari a 10, dopo aver applicato un gradino, l'uscita si avvicina lentamente al valore desiderato. Se K_P viene aumentato a 100, il tempo di salita viene ridotto. Tuttavia, se il parametro viene aumentato troppo, anche se il tempo di salita viene ridotto ulteriormente, vengono a crearsi overshoot che possono essere anche di considerevole ampiezza. Un esempio di ciò che è stato appena descritto viene rappresentato in Figura 2.6.

Per questo motivo viene introdotto anche il parametro K_I.

Tale componente somma il termine di errore nel tempo. Per fare ciò è dunque necessario tenere conto della storia pregressa dell'errore.

$$u(t) = K_I \cdot \int_0^t e(\tau)d(\tau) \tag{2.3}$$

Il risultato è che anche un piccolo termine di errore fa aumentare lentamente il termine integrale, a meno che tale errore sia pari a zero. Perciò il compito principale di tale parametro è quello di portare l'errore stazionario a zero, ciò significa che il valore a regime corrisponde perfettamente a quello desiderato.

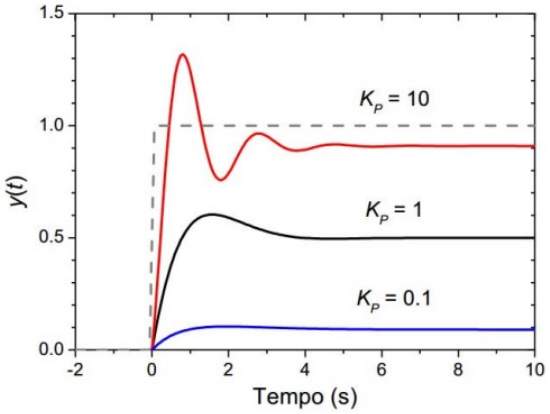

Figura 2.6: Risposta proporzionale con diversi valori di K_P.

Il parametro K_I è definito come $K_I = \frac{K_P}{T_I}$, dove T_I è chiamato 'tempo di reset'; da esso dipende l'effetto di integrazione che risulta essere maggiore quando T_I è piccolo, in quanto il valore dell'integrale sale più rapidamente.

In Tabella 2.1 sono riportati gli effetti derivanti da valori differenti di T_I.

Variazioni di T_I	Stabilità	Tempo di risposta
Aumenta	Migliora	Rallenta
Diminuisce	Peggiora	Accelera

Tabella 2.1: Effetti sul sistema in caso di variazioni di T_I.

Un possibile inconveniente derivante dall'utilizzo del controllore PI, ovvero quando vengono combinati gli effetti dei parametri K_P e K_I, è quello che determina talvolta il superamento dell'uscita del controllore e la possibile instabilità del sistema.

In Figura 2.3 sono riportati gli effetti che si possono verificare al variare del parametro integrativo.

Per ovviare al problema dell'instabilità è necessario introdurre un ulteriore parametro, ovvero la componente Derivativa K_D.

Quest'ultima fornisce in uscita la derivata rispetto al tempo dell'errore.

$$u(t) = K_D \cdot \frac{de(t)}{dt} \tag{2.4}$$

dove $K_D = K_P \cdot T_D$. T_D è la costante di tempo dell'azione derivativa e il cui valore determina la velocità di salita del segnale di controllo. In questo caso la stabilià del sistema peggiora sia aumentando sia diminuendo il parametro.

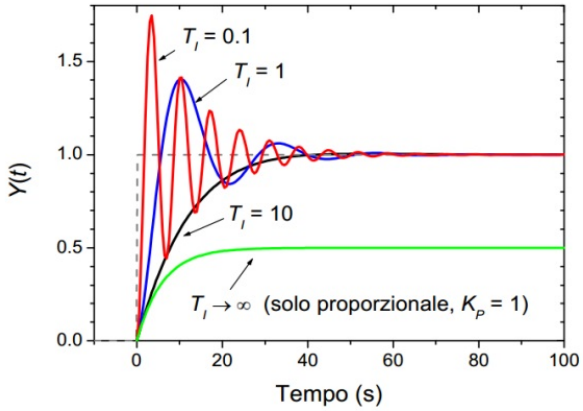

Figura 2.7: Effetto sul sistema con componente integrale al variare di T_I.

In Tabella 2.2 è riportato il comportamento del sistema al variare del parametro.

Variazioni di T_D	Stabilità	Tempo di risposta
Aumenta	Peggiora	Accelera
Diminuisce	Peggiora	Rallenta

Tabella 2.2: Effetti sul sistema in caso di variazioni di T_D.

Il comportamento di tale componente dipende esclusivamente dalla velocità di variazione dell'errore, perciò anticipa la futura evoluzione dell'errore, offrendo al sistema di controllo maggiore prontezza. Il suo intento è quello di contrastare la velocità di variazione della grandezza da controllare. Infatti, se la variazione è molto lenta, la sua influenza si riduce sensibilmente tendendo quasi a zero.

In Figura 2.8 è riportato un esempio di risposta di un controllore esclusivamente Proporzionale (linea rossa) e di risposta dello stesso controllore con l'introduzione dell'azione derivativa (linea blu).

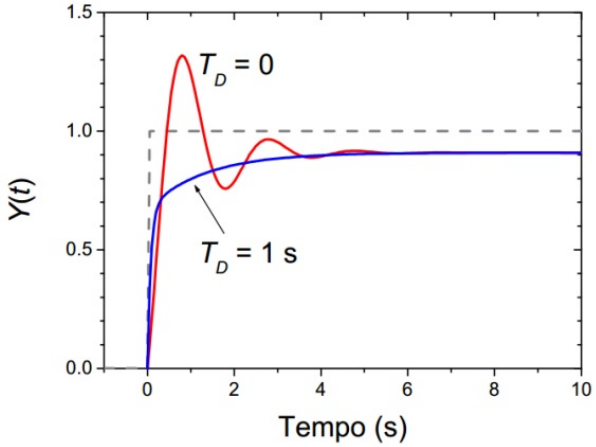

Figura 2.8: Risposta di un controllore P e di un controllore PD.

In Figura 2.8 si può notare un offset rispetto al gradino d'ingresso che si può colmare con l'introduzione dell'azione integrale descritta precedentemente.

Nel progetto questo controllo viene implementato in modo tale che, una volta impostata la velocità che si vuole raggiungere tramite un potenziometro e perciò un determinato valore di duty cycle da fornire alla PWM, il sistema valuti la differenza tra il valore di velocità impostato ed il valore reale in quel momento. Per poi, in funzione di ciò, aumentare o diminuire il valore di duty cycle per portare il numero di giri al minuto del motorino a quello desiderato. I valori dei parametri P, I e D verranno definiti successivamente.

Un metodo spesso utilizzato, per la determinazione dei parametri del PID, è quello definito da Ziegler e Nichols. Tale metodo consiste nell'applicare un controllore esclusivamente Proporzionale e di impostare il parametro K_P ad un valore abbastanza elevato (K_C), in modo tale da poter osservare una risposta che presenti notevoli oscillazioni ripetute nel tempo. Dopodichè si misura il periodo di tali oscillazioni (periodo critico P_C). Infine, per la regolazione dei parametri da fornire al controllore, si utilizza la Tabella 2.3, in base al tipo di controllo che si vuole realizzare. Per poi, all'occorrenza, adattare i parametri al sistema specifico in base ai risultati che si vogliono ottenere.

Tipo	K_P	T_I	T_D
P	$K_C/2$	-	-
PI	$K_C/2.2$	$P_C/1.2$	-
PID	$K_C/1.7$	$P_C/2$	$P_C/8$

Tabella 2.3: Determinazione parametri con metodo Ziegler-Nichols.

2.6 Mosfet

Il mosfet è un componente elettronico molto utilizzato, particolarmente nell'ambito dell'elettronica digitale, ma è diffuso anche in ambito analogico.

Il componente presenta tre terminali: gate, drain e source. L'applicazione sul gate di una tensione genera solitamente un flusso di cariche tra source e drain; esso rappresenta quindi la corrente che attraversa il dispositivo. Questo fa capire che si tratta di un dispositivo controllato in tensione, a differenza del transistor bipolare che è un componente controllato in corrente.

Sul mercato i mosfet si dividono principalmente in due rami in base al tipo di drogaggio del dispositivo, che conferisce al componente modi di funzionamento diversi.

Il funzionamento di un mosfet si può dividere in accumulo, svuotamento, saturazione e conduzione lineare.

Si ha accumulo quando viene fornita al terminale di gate una tensione negativa rispetto al substrato, che generalmente è posto a massa, e come risultato si ottiene che le lacune si accumulano sul substrato non permettendo il flusso di corrente tra source e drain.

Lo svuotamento si ha quando viene impostata una tensione positiva al gate ma più bassa rispetto alla tensione di soglia V_{th}, che permette al mosfet di condurre corrente in quanto si allontanano le lacune dal substrato, ma non abbastanza da creare un canale conduttivo che connette drain e source.

La tensione di soglia dipende dalla costruzione del dispositivo. Qualora V_{th} superi la tensione $V_{DS} + V_{th}$, ci si trova in una situazione di saturazione, in quanto i portatori maggioritari di

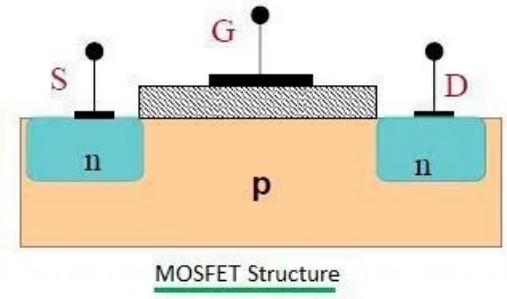

Figura 2.9: Struttura di un mosfet a canale n.

carica vengono attratti sul gate, creando così un canale conduttivo tra source e drain. Nel caso in cui si aumenti V_{DS}, risulta che il canale subisce una strozzatura, per cui, per valori anche elevati della tensione, si arriva al punto in cui la corrente che scorre nel dispositivo non dipende più dalla tensione applicata tra drain e source.

Nel caso in cui la tensione tra drain e source superi $V_{GS} + V_{th}$, viene a crearsi un canale conduttivo e il dispositivo si comporta esattamente come una resistenza, dove quindi la corrente che scorre sul componente aumenta solo all'aumentare della tensione applicata al gate.

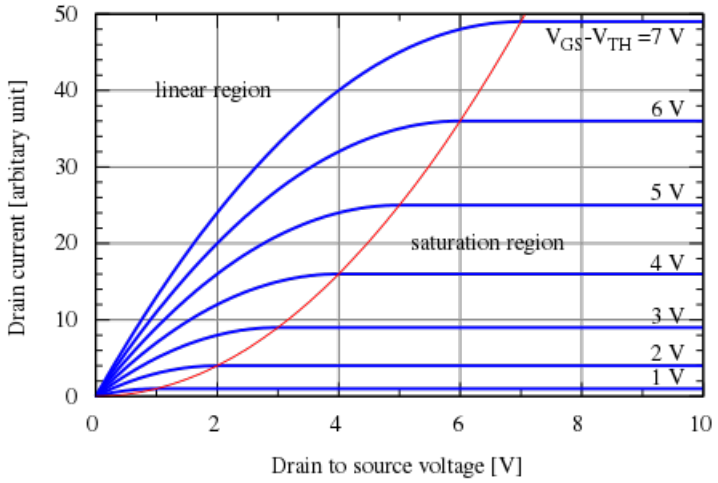

Figura 2.10: Andamento della corrente di drain in funzione della tensione drain-source per vari valori di $V_{GS} - V_{th}$.

Nel progetto il mosfet viene utilizzato come dispositivo switching, in modo tale da separare la parte di potenza composta dal motorino e la parte di controllo composta da Arduino. Il mosfet è necessario in quanto Arduino non sarebbe in grado di fornire la potenza adatta al motorino per girare. Il mosfet è quindi molto utile, in quanto permette ad Arduino di erogare la minor quantità di corrente possibile. Grazie alla PWM è in grado di regolare la corrente che scorre sul motorino e dunque permette di regolarne la velocità con il minimo consumo di potenza. La scelta è ricaduta sul mosfet in quanto, a differenza del BJT, il consumo di corrente sul gate è molto minore rispetto alla corrente richiesta dal transistor; questo rende il mosfet un dispositivo comandato in tensione e quindi perfetto per l'applicazione di interesse del progetto.

2.7 Motore CC

Un motore in corrente continua è un tipo di macchina elettrica che converte energia elettrica in energia meccanica. All'interno è formato da avvolgimenti che una volta alimentati da corrente creano campi magnetici che mettono in moto dei magneti collegati al rotore. Il rotore successivamente converte e trasmette il movimento dei magneti all'albero motore.

La velocità dipende dunque sia dall'input che dal design del motore.

Il motore è composto da due parti: uno statore e un'armatura. Il primo è la parte fissa del dispositivo, mentre la seconda è la parte mobile che ruota. Dunque, come anticipato prima, lo statore ha il compito di fornire all'armatura il campo rotante in cui può muoversi.

Gli avvolgimenti di filo di rame sono collegati al commutatore che trasmette loro energia elettrica. Il commutatore ha il compito di eccitare a turno ciascuna bobina dell'armatura, creando così una forza rotante costante.

Sul mercato si trovano vari tipi di motori CC, i più usati sono i motori senza spazzole e quelli con le spazzole.

- **Motore senza spazzole:** rispetto ai motori con spazzole non hanno il commutatore, che viene sostituito con un servomeccanismo che è in grado di rilevare e regolare l'angolo del rotore. Il vantaggio principale è che non avendo le spazzole questo tipo di motori ha durata maggiore, in quanto le spazzole subiscono usura nel tempo.
- **Motore con spazzole:** come detto in precedenza questo tipo di motori presenta un commutatore che ha il compito di invertire la corrente dopo ogni giro. Le spazzole sono collegamenti morbidi che permettono al motore di ruotare in ogni direzione.

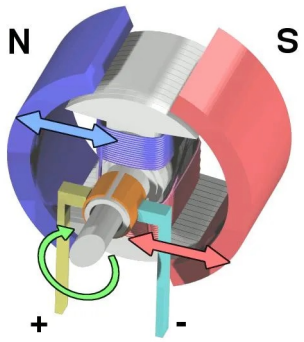

Figura 2.11: Vista interna di un motore CC composto da statore ed armatura.

Per la realizzazione del progetto viene usato un motore brushless, in quanto è il tipo di motore più utilizzato attualmente, questo perché, come detto prima, non si va incontro ad usura delle spazzole nel tempo, il che risulta essere un vantaggio non trascurabile nelle applicazioni che comprendono l'utilizzo di motori CC.

2.8 Encoder rotativo incrementale

Gli encoder rotativi sono dispositivi elettromeccanici che convertono il movimento angolare di rotazione in segnali generalmente digitali.

Esistono encoder rotativi assoluti ed incrementali (relativi).

Gli encoder rotativi assoluti sono dotati di funzioni di tracciamento per determinare punti di posizionamento specifici. Mantengono i dati anche quando l'alimentazione viene interrotta e, quando il dispositivo viene riacceso, il funzionamento parte dal punto di interruzione.

Al contrario, gli encoder rotativi incrementali registrano le deviazioni nel corso di determinati intervalli di tempo o di rotazione angolare, fornendo un segnale ogni qualvolta l'albero di rotazione percorra tale distanza.

Gli encoder rotativi ottici sono costruiti usando un disco con segmenti trasludici per lasciar passare la luce in determinate aree. Usando dei led e dei fotodiodi sui lati opposti del disco, si permette a questi ultimi di rilevare la luce che attraversa il disco ed emettere forme d'onda impulsive che corrispondono alla sequenza dei segmenti trasludici del disco.

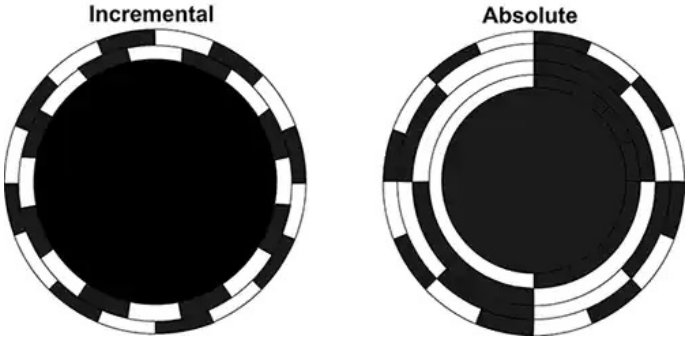

Figura 2.12: Esempi di encoder rotativo incrementale ed assoluto.

In Figura 2.12 si può notare che nel caso dell'encoder rotativo incrementale sono presenti due dischi esterni. Questo è dovuto al fatto che nella quasi totalità dei casi i segnali forniti dal sensore sono due e sono in quadratura tra loro (90°), ciò permette di conoscere oltre alla posizione del motore anche il verso di rotazione. Questo perché nel caso in cui il motore ruoti in senso orario in uscita si riceve un impulso A e subito dopo un impulso B, mentre nel caso in cui il motore ruoti in senso antiorario si ottiene B e successivamente A. In Figura 2.13 è riportato un esempio pratico di ciò che è stato appena descritto.

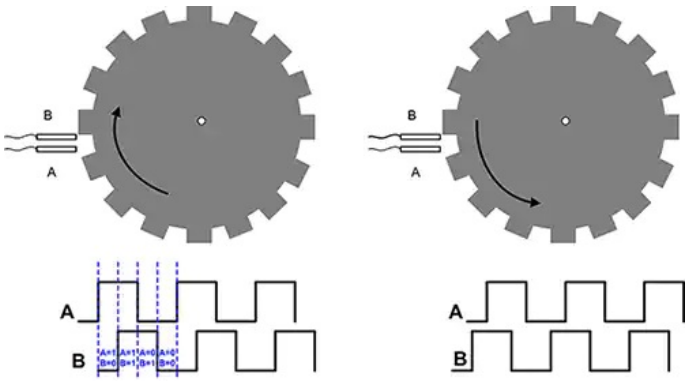

Figura 2.13: Esempio di segnali A e B in funzione del senso di rotazione.

Grazie all'encoder rotativo incrementale si è quindi in grado di determinare la velocità di rotazione di un motore. Ciò avviene grazie alla misurazione del numero di impulsi che il sensore genera in un secondo. Successivamente lo si moltiplica per 60 e lo si divide per il numero di impulsi che il sensore è in grado di generare per una rivoluzione.

$$RPM = \frac{(\text{n}^\circ \text{ impulsi ricevuti}) \cdot 60}{\text{n}^\circ \text{ impulsi per rivoluzione}} \tag{2.5}$$

Capitolo 3

Realizzazione pratica

In questo capitolo verranno specificati i componenti utilizzati per la realizzazione del progetto, il programma implementato e le forme d'onda di interesse per la verifica del funzionamento del circuito.

3.1 Componentistica

Componente	**Valore / Codice costruttore**
Arduino UNO	
Batteria	9V
Resistori	$1k\Omega$
Potenziometro	$10k\Omega$
Pulsante	
nMosfet	IRFZ44N
Diodo	
Motorino CC	
Motoriduttore	Fattore riduzione: 4.4
Encoder rotativo incrementale	

Tabella 3.1: Componentistica utilizzata.

Per la realizzazione del progetto sono necessarie minimo due alimentazioni, tra cui 9V per alimentare il motorino in modo tale da limitare il consumo di potenza richiesto ad Arduino, dopodichè è necessario utilizzare l'alimentazione a 5V fornita direttamente dal microcontrollore per l'utilizzo del potenziometro, il quale permette di impostare la velocità che si desidera raggiungere. Inoltre vengono utilizzati i 3.3V per alimentare l'encoder rotativo che permette di misurare la velocità di rotazione del motorino. Da specifiche, per l'alimentazione è possibile utilizzare anche i 5V a piacimento.

Nel circuito, per selezionare la velocità che il sistema deve raggiungere a regime, viene utilizzato un potenziometro da $10k\Omega$, che viene collegato ad un ingresso analogico del microcontrollore. Per far partire la lettura del potenziometro è stato inserito un pulsante, che permette di mantenere la lettura a zero quando non viene premuto, nonostante il potenziometro non sia impostato ad un valore nullo. Per evitare che la lettura del pin analogico, collegato al pulsante,

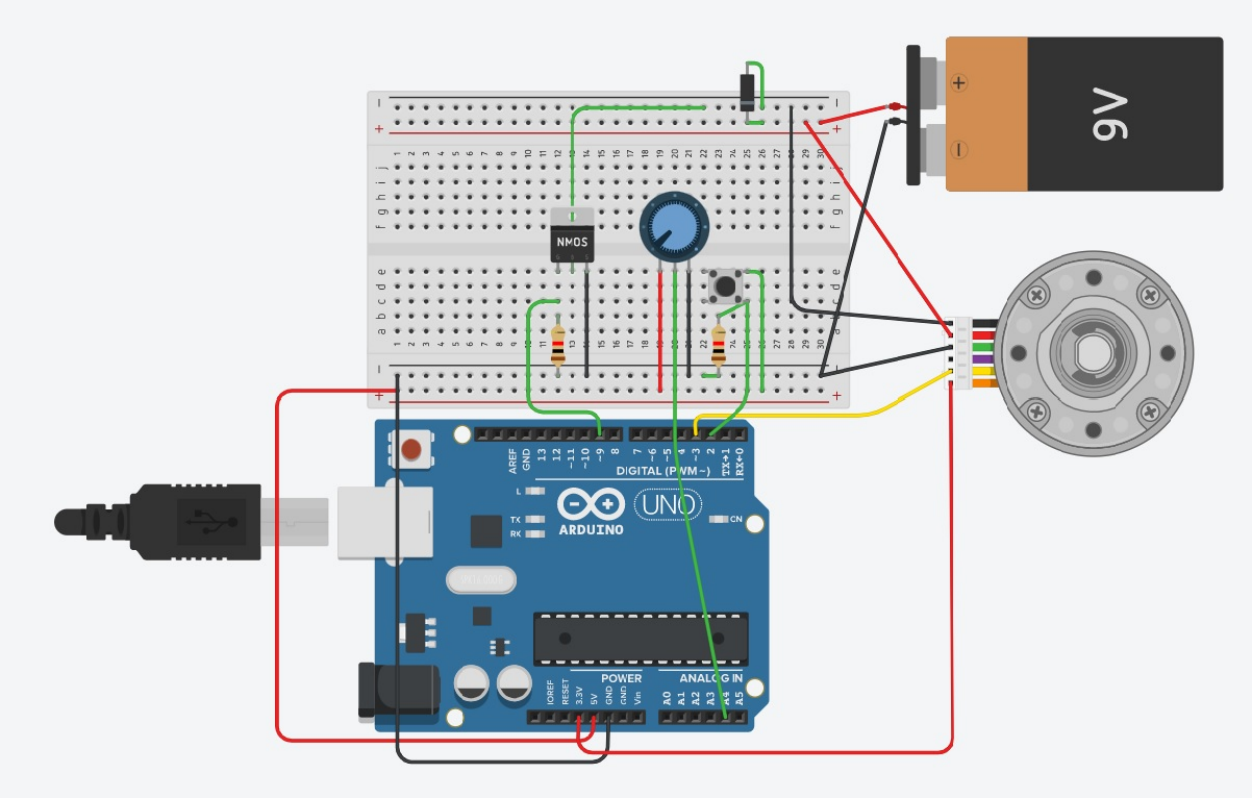

Figura 3.1: Schema circuito realizzato tramite Tinker Cad.

possa essere flottante e quindi leggere valori randomici, è stata introdotta una resistenza di pull-down che permette di avere il pin analogico sempre connesso a massa, nel caso in cui il pulsante non venga premuto.

Viene posto un diodo in parallelo al motorino, in modo tale che possano fluire attraverso il diodo e tornino nel motore le extracorrenti, che si vengono a creare dall'immagazzinamento di energia nel carico induttivo del motore, quando il mosfet non è in conduzione.

Per separare la parte di controllo dalla parte di potenza, viene inserito un mosfet che ha il compito di pilotare il motorino e di limitare al massimo la potenza in uscita dal microcontrollore. Infatti il gate del mosfet non richiede un consumo di corrente considerevole, che in alternativa sarebbe troppo elevato nel caso in cui si dovesse alimentare direttamente il motorino.

Il motore che viene usato è completo di encoder rotativo incrementale e di motoriduttore per la riduzione dei giri dell'albero di un fattore 4.4. In Figura 3.1 si può notare che i cavi in uscita corrispondono a:

- **Cavo nero:** pin negativo del motorino, collegato al drain del mosfet;
- **Cavo rosso:** pin positivo del motorino, collegato direttamente al polo positivo della batteria a 9V;
- **Cavo verde:** massa dell'encoder;
- **Cavo viola:** canale B dell'encoder, non collegato in quanto non viene utilizzato;
- **Cavo giallo:** canale A dell'encoder, collegato al pin 3 del microcontrollore, dal quale esce un impulso ogni qualvolta l'albero del motorino ruoti di un angolo specifico;

- **Cavo arancione:** alimentazione positiva dell'encoder, collegato ai 3.3V del microcontrollore.

Tutto ciò che è stato appena descritto viene gestito e controllato da un microcontrollore, nel nostro caso Arduino. Questo ha il compito di verificare il valore impostato tramite il potenziometro (valore di velocità desiderato), confrontarlo con il valore effettivo letto dall'encoder dopo aver effettuato conti specifici per determinarne la velocità. Dato che in uscita dall'encoder si ottiene unicamente una serie di impulsi, è necessario utilizzare una formula che permetta di ricavare il numero di giri per minuto, a partire dal numero di impulsi ricevuti in 100 ms. La formula è molto simile alla 2.5 presentata nel capitolo precedente. Nel nostro caso la formula utilizzata è la seguente:

$$RPM = \frac{encoderValue \cdot 600}{11 \cdot 4.4} \tag{3.1}$$

dove *encoderValue* è la variabile utilizzata nel programma Arduino per contare il numero di impulsi ricevuti in 100 ms. La variabile viene multiplicata per 600 in modo da ottenere il numero di impulsi teorico in un minuto. Successivamente il prodotto viene diviso per 11, che indica il numero di impulsi che l'encoder genera in un giro dell'albero. Infine si divide per un ulteriore fattore 4.4, che corrisponde al fattore di riduzione introdotto dal motoriduttore implementato nel motorino, il quale limita il numero massimo di giri per minuto a 1000.

Dopo aver effettuato il controllo di errore tra questi valori, il microcontrollore implementa un sistema di controllo PID, che ha l'obiettivo di raggiungere il valore desiderato nel minor tempo e con il minor numero di oscillazioni possibile.

Nel paragrafo successivo verrà riportato e commentato nello specifico il programma che è stato usato per ottenere i risultati prestabiliti.

3.2 Programma Arduino

```
float error, error_prev, error_sum=0;
long encoderValue=0;
long previousMillis, currentMillis=0;
int rpm, thrpm=0;
int motorPwm=0;
int run=0;
int pid_rpm=0;

#define ENC_IN 3
#define START 2
#define PWM 9
#define Kp 0.2      //Proporzionale
#define Ki 0.5      //Integrativo
#define Kd 0.01     //Derivativo

void setup()
{
  pinMode(ENC_IN, INPUT_PULLUP);
```

```
  pinMode(START, INPUT_PULLUP);
  pinMode(PWM, OUTPUT);
  Serial.begin(9600);
  attachInterrupt(digitalPinToInterrupt(ENC_IN), updateEncoder, RISING);
  attachInterrupt(digitalPinToInterrupt(START), start, RISING);
  previousMillis=millis();

}

void loop()
{
  currentMillis = millis();
  if (currentMillis - previousMillis > 100)
  {
    previousMillis = currentMillis;

    rpm = (float)(encoderValue * 600 / 11 /4.4);  //calculate RPM

    Serial.println( (float) motorPwm);
    Serial.print('\t');
    Serial.println((float) rpm);

    encoderValue=0;   //reset counter
  }

  if(run==1)
  {
    thrpm= map(analogRead(A4), 0, 1023, 0, 1000);  //speed target

    error = thrpm - rpm;
    error_sum += error;                            //integral part
    if (error_sum >1000) error_sum = 1000;
    if (error_sum <-1000) error_sum = -1000;

    pid_rpm = error*Kp + error_sum*Ki + (error - error_prev)*Kd;  //PID control

    error_prev = error;

    if (pid_rpm <1000 & pid_rpm >0)
    {
      motorPwm=map(pid_rpm, 0, 1000, 0, 255);
      analogWrite(PWM, motorPwm);                 //set motor speed
    }
    else
    {
      if (pid_rpm>1000)
      {
```

```
        analogWrite(PWM, 255);
      }
      else
      {
        analogWrite(PWM, 0);
      }
    }
  }
  else
  {
    analogWrite(PWM, 0);
  }
}

void updateEncoder()
{
  encoderValue++;
}

void start()
{
  if(run==0)
  {
    run=1;
  }
  else
  {
    run=0;
  }
}
```

Il codice riportato è stato utilizzato per ottenere i risultati inseriti nel paragrafo successivo.

Un programma Arduino è formato sostanzialmente da 3 parti: definizione di variabili e costanti, impostazione di porte I/O e main del programma con le istruzioni da eseguire. Oltre a tutto ciò sono state inserite anche le funzioni *updateEncoder* e *start* che tramite interrupt entrano in azione quando il microcontrollore rileva fronti di salita sui pin adibiti a tale scopo.

Nello specifico, la funzione *updateEncoder* ha il compito di aumentare la variabile counter ogni qualvolta l'encoder rotativo emetta un segnale in seguito alla rotazione dell'albero motore. La variabile counter è necessaria per la determinazione del numero di giri al minuto del motore. Essa viene azzerata ogni 100 ms in modo tale da poter ricalcolare la velocità teorica del motorino, nel caso in cui le condizioni persistessero per un minuto.

La funzione *start* invece ha semplicemente il compito di modificare, ad ogni fronte di salita del pin di controllo, il valore della variabile *run* in modo tale da far avviare il motore alla pressione di un pulsante o di fermarlo nello stesso modo.

Come anticipato, all'interno del main, è stato introdotto un modo per leggere il numero di giri al minuto ogni 100 ms. Si elabora successivamente il numero totale di impulsi forniti

dall'encoder in un minuto, si divide poi il risultato per il numero di segnali inviati al giro e si introduce un divisore di valore pari al fattore di riduzione introdotto da parte del motoriduttore.

A questo punto viene mappato il valore letto dal potenziometro da un valore basato su 10 bit ad uno che si basa su 8 bit dato che, nel caso di Arduino, i pin analogici lavorano a 10 bit e quelli digitali invece a 8 bit. Il valore convertito corrisponde al numero di giri che il motorino deve raggiungere a regime. Tale lettura viene mappata, tramite una proporzione, su un valore compreso tra 0 e 1000 (rpm massimo dichiarato dal costruttore). Il risultato viene poi attribuito a *thrpm*, il quale viene in seguito utilizzato per il calcolo dell'errore.

Il passo successivo comprende l'implementazione del sistema PID a partire dall'errore rilevato tra il numero di giri impostato dall'utente e quello reale del motorino.

Inizialmente si somma l'errore appena calcolato allo storico di tutti gli errori verificati nel tempo, il quale poi è componente fondamentale della parte integrativa del PID. Tale somma viene limitata tra -1000 e +1000 in modo tale da evitare che si accumulino errori fino a raggiungere un valore che produrrebbe continue oscillazioni indesiderate.

Dopodichè viene creata una variabile *pid_rpm*, che viene utilizzata per fornire il valore di velocità necessario per raggiungere l'obiettivo prefissato, in funzione delle correzioni sull'errore introdotte dal PID.

Tale calcolo comprende le componenti proporzionali, derivative ed integrative del sistema. Per fare ciò viene quindi introdotta la componente proporzionale che effettua un controllo solamente sul valore dell'errore. Successivamente è necessario introdurre la variabile *error_sum* che tiene conto della somma di errore che viene a crearsi nel tempo durante il funzionamento del dispositivo. Inoltre si introduce la variabile *error_prev* che invece serve a tenere in memoria il valore di errore attuale per essere poi utilizzato nel ciclo successivo per il calcolo della componente derivativa. Perché il sistema dia risultati accettabili, è ovviamente necessario moltiplicare ogni componente con il parametro dedicato ed impostato in modo adeguato.

I parametri per ottenere un funzionamento ottimale sono stati impostati in modo tale da avere una componente proporzionale non esageratamente elevata, la quale potrebbe causare sovraelongazioni ed oscillazioni elevate. La costante K_I è stata scelta superiore rispetto a quella proporzionale di un fattore 2.5, che come visto nel capitolo precedente corrisponde ad impostare T_I pari a 0.4. Questo permette di ottenere una stabilità ed un tempo di risposta accettabili. Tale parametro viene scelto superiore per ottenere una risposta precisa ed uguale al valore desiderato, come riportato in seguito si nota che scegliendo un K_P maggiore rispetto a K_I si ottiene un sistema con sovraelongazioni importanti e che fatica a portarsi al valore di input a regime. Mentre K_D viene impostato più piccolo rispetto a K_P di un fattore 2, che corrisponde a T_D, il quale permette di ottenere un tempo di risposta molto ridotto. Tutti i parametri non sono stati inizializzati a valori elevati, dato che si lavora con numeri compresi tra 0 e 1000 e le variazioni di errore sono molto frequenti a causa del calcolo non perfetto della velocità effettiva del motorino tramite l'encoder.

In seguito, il valore calcolato tramite il PID, viene mappato su una scala compresa tra 0 e 255, che corrisponde al duty cycle da fornire al motorino per raggiungere la velocità desiderata.

Nel prossimo paragrafo verranno analizzate le forme d'onda ottenute e verranno confrontati i risultati acquisiti in seguito alla variazione dei parametri del PID.

3.3 Analisi forme d'onda

Di seguito sono riportati i grafici dei risultati più interessanti riferiti al progetto. Ovvero le forme d'onda riguardanti il numero di giri del motorino ed il duty cycle fornito al PWM, per portarsi alla velocità desiderata.

Per l'acquisizione dei grafici è stato utilizzato il *Serial plotter* messo a disposizione da Arduino, che permette di riportare i valori stampati sulla seriale in forma di grafici.

Il caso preso in esame in seguito è caratterizzato dai seguenti parametri del controllo PID: $K_P = 1$, $K_I = 0.1$ e $K_D = 0.01$. Ciò comporta, secondo la teoria, un sistema che presenta una sovraelongazione importante ed ulteriori oscillazioni prima di portare l'uscita ad un valore più o meno costante.

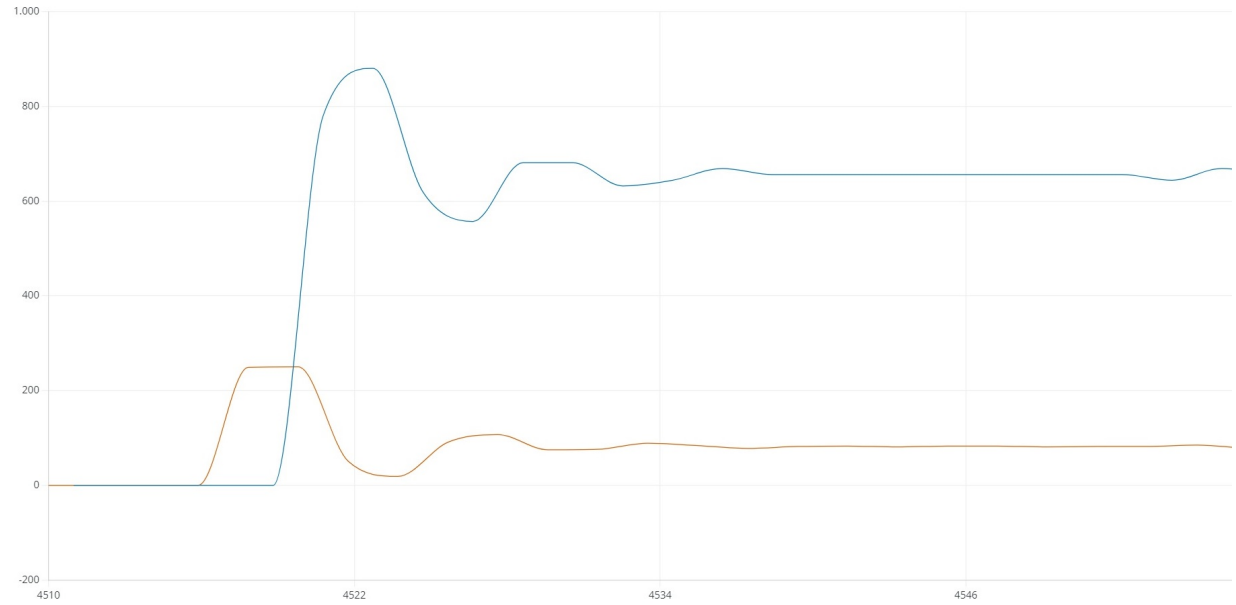

Figura 3.2: Andamento di velocità e duty cycle con $K_P = 1$, $K_I = 0.1$, $K_D = 0.01$.

In Figura 3.2 la linea blu rappresenta il numero di giri al minuto, mentre quella arancione rappresenta il duty cycle fornito al motorino. Si può notare come sia presente una sovraelongazione molto accentuata, seguita da alcune oscillazioni, prima di riuscire a portare il sistema in equilibrio. Tale sovraelongazione è dovuta al fatto che la componente K_P ha un ruolo alquanto importante nel calcolo effettuato dal programma. Fornisce un valore che, poi sommato alle altre componenti, crea un valore in uscita ben maggiore rispetto a quello di riferimento. Anche il tempo di risposta risulta essere abbastanza lento, questo è dovuto alla componente K_D, dato che è stata impostata ad un valore molto ridotto rispetto a K_P. Il principale problema per questo sistema risulta essere la sovraelongazione che si viene a creare prima di riuscire a stabilizzare il sistema, che in alcuni casi potrebbe causare problemi in quanto molto superiore al valore desiderato.

Da notare che il valore a regime non risulta essere pari a 830 (valore impostato tramite potenziometro), questo perché la costante K_I è stata impostata a un valore talmente basso che fatica a portare l'uscita al valore desiderato. Per ottenere un valore che si avvicina ad 830 bisognerebbe che la variabile *error_sum* raggiungesse valori intorno a 8000. Questo è impossibile in quanto tale variabile è stata ridotta ad un valore massimo pari a 1000.

Un altro effetto che si può notare dall'analisi del grafico è una continua oscillazione del numero di giri anche a sistema stabile. Tale oscillazione non è dovuta al sistema PID, bensì alla imprecisione del calcolo che fornisce il numero di giri effettivo del motorino, dovuto alla elaborazione non ottimale dei dati forniti dall'encoder. In questo caso specifico tale effetto è trascurabile, in quanto oscillazioni minime della velocità a regime non influenzano in modo evidente il funzionamento del sistema.

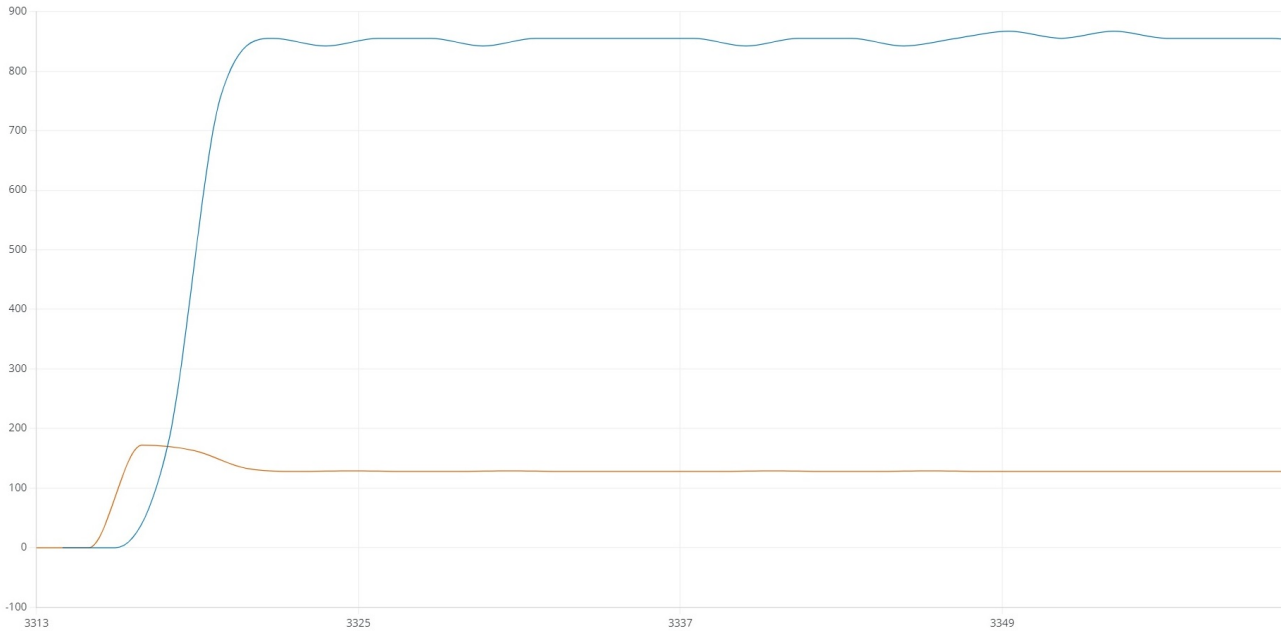

Figura 3.3: Andamento di velocità e duty cycle con $K_P = 0.2$, $K_I = 0.5$, $K_D = 0.01$.

In Figura 3.3 è riportato l'andamento di velocità e duty cycle nel caso in cui K_I abbia un valore superiore rispetto a quello precedente di un fattore 5, allo stesso modo è stato ridotto K_P a 0.2 in modo tale da limitare la sovraelongazione che si verrebbe a creare nel caso in cui lo si mantenesse invariato. Si può notare come si abbia un picco nel momento in cui si attiva il sistema, che risulta essere molto ridotto rispetto a Figura 3.2, per il motivo spiegato precedentemente. In questo caso risultano esserci inoltre oscillazioni molto minori prima di raggiungere il valore a regime, tale effetto deriva principalmente dalla riduzione di K_P ed alla conseguente riduzione della sovraelongazione, che comporta quindi un errore successivo molto minore da dover compensare. Inoltre tale sistema riesce a portarsi al valore desiderato a regime grazie all'incremento di K_I.

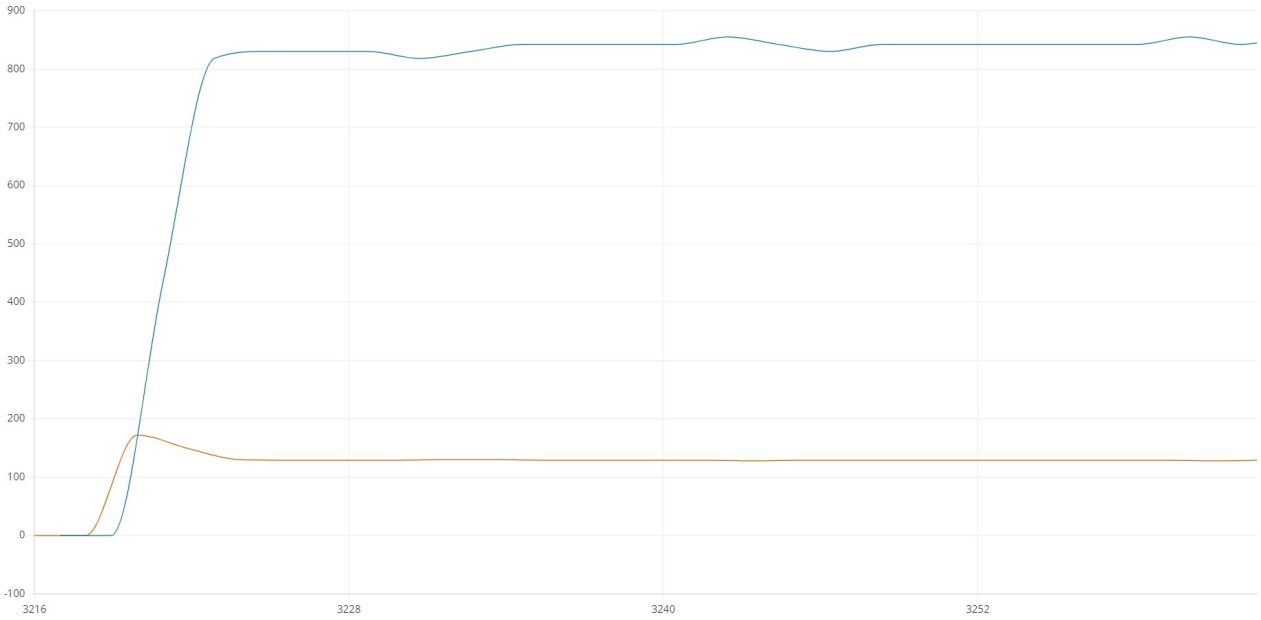

Figura 3.4: Andamento di velocità e duty cycle con $K_P = 0.2$, $K_I = 0.5$, $K_D = 0.1$.

Infine è stato modificato il parametro K_D di un fattore 10. Ciò, secondo la teoria, comporta un incremento nella velocità di risposta del sistema. Infatti come si evince da Figura 3.4 la risposta del sistema al duty cycle, fornito al motorino, è nettamente più veloce rispetto ai casi presi in esame precedentemente. Questo effetto è dovuto al fatto che, nel calcolo iniziale, la derivata dell'errore risulta essere un valore molto elevato che poi si va a sommare alle altre componenti, causando quindi una riduzione sostanziale del tempo di risposta del sistema.

Capitolo 4

Conclusioni

4.1 Commenti sul progetto

Questo progetto di tesi ha permesso di realizzare un controllore di velocità per motorino tramite l'utilizzo di un controllo PID. Con i risultati ottenuti, è stato possibile verificare i diversi tipi di comportamento del sistema, al variare semplicemente di tre parametri. Inoltre è stato raggiunto l'obiettivo di attuare un sistema, che modificasse il valore di duty cycle in modo tale da cercare di portare la velocità a un determinato valore nonostante eventuali disturbi esterni.

Inizialmente, per la determinazione dei valori da attribuire ai parametri del PID, è stato utilizzato il metodo Ziegler-Nichols descritto al Paragrafo 2.5. Quindi è stato impostato un valore di K_P abbastanza elevato, che determinasse in uscita una sovraelongazione considerevole e presentasse ripetute oscillazioni, mantenendo i restanti parametri nulli. Sono stati determinati poi i parametri del PID in modo tale da ottenere dei risultati in linea con i requisiti stabiliti dal metodo in questione. In seguito, sono state effettuate alcune prove che hanno prodotto risultati non adeguati, rispetto agli obiettivi prestabiliti. Infatti la risposta del sistema era caratterizzata ancora da sovraelongazione e oscillazioni non indifferenti, motivo per cui è stato necessario utilizzare il metodo *trial and error*, a partire dai parametri ottenuti tramite il metodo Ziegler-Nichols. Questo metodo consiste nel modificare i parametri, uno alla volta, fino a ottenere i risultati più adatti al sistema specifico.

Come presentato nel capitolo precedente, è stato possibile verificare come un sistema possa presentare risposte diverse modificando anche di poco una o più costanti.

Personalmente mi ritengo soddisfatto del risultato finale ottenuto, in quanto il sistema realizzato rispecchia tutte le caratteristiche che mi ero prefissato a inizio progetto. Ovvero rapidità di risposta, precisione e stabilità.

4.2 Miglioramenti futuri

Possibili modifiche, che possono essere apportate al sistema per ottenere risultati migliori, riguardano l'utilizzo di un encoder che permetta di ottenere una lettura del numero di giri del motorino più precisa. Tale scopo può essere raggiunto tramite l'utilizzo di un encoder che presenti più fori sulla corona rotante, in modo tale da ricevere un numero maggiore di impulsi, riducendo così l'errore di lettura sul giro.

Al sistema può essere inoltre aggiunto un display, che permetta di visualizzare i dati in tempo reale.

Può essere ulteriormente aggiunto un ponte ad H (L298) che permetterebbe al motorino di girare in entrambi i sensi di rotazione, utilizzando semplicemente dei segnali, provenienti dal microcontrollore.