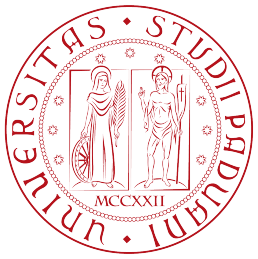


Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in
Ingegneria dell'Informazione

Progettazione e realizzazione del sistema di controllo per skateboard elettrico a motori brushless

Relatore:
Prof. Matteo Meneghini

Laureando:
Giacomo Canevese
1139088

Anno Accademico 2021/2022

Prefazione

In un'era dove i motori elettrici prendono sempre più il sopravvento, la mobilità del singolo soggetto diventa un argomento di fondamentale importanza. Diverse sono le possibilità già presenti in commercio, biciclette elettriche, monopattini, hoverboard e molto altro.
In questa tesi viene trattata la progettazione e la realizzazione di un sistema di controllo per motori brushless dotati di sensori di Hall utilizzati, in questo caso, per mobilitare uno skateboard elettrico tramite un telecomando.

Indice

1 Introduzione **1**

2 Schema Progetto **5**
2.1 Macro blocchi 5
2.2 Conversione Analogico/Digitale 6
2.2.1 Campionamento 6
2.2.2 Quantizzazione 6
2.2.3 Codifica 6
2.3 Modulazione PWM 6
2.4 Funzionamento motore brushless con sensori di Hall 7

3 Hardware **9**
3.1 Elegoo Nano V3.0 9
3.2 Motore brushless con sensori di Hall 10
3.3 modulo nRF24l01 10
3.4 Potenziometro "joystick" 11
3.5 Interruttori 11
3.6 Resistori 12
3.7 Condensatori 12
3.8 Diodi 12
3.9 Mosfet 13
3.10 Mosfet Driver 13
3.11 Regolatore di tensione 14
3.12 L7805 15
3.13 Batterie 18650 15
3.14 Batteria al litio 12V 16
3.15 Circuito di Controllo 17

4 Software **21**
4.1 Trasmettitore 22
4.2 Ricevitore 23
4.3 Controllo Motore 24
4.3.1 Setup 25
4.3.2 ISR (PCINT2_vect) 29
4.3.3 bldc_move() 29
4.3.4 move_clockwise() 30
4.3.5 move_counterclockwise() 30

4.3.6 Eccitamento bobine 31
4.3.7 loop 32
4.3.8 SET_PWM_DUTY(byte duty) 33

5 Osservazioni **35**
5.1 Timer interni 35
5.2 Trasmissione radio 35
5.2.1 Scelta del modulo radio 35
5.2.2 Arduino extra per il ricevitore 36
5.3 Accensione separata per Arduino 36
5.4 Millefori o circuito stampato 37

6 Conclusioni **39**
6.1 Nuove conoscenze acquisite 39
6.2 Possibili miglioramenti futuri 40
6.2.1 Implementazione curva assistita e retromarcia 40
6.2.2 Sistema di frenata 40
6.2.3 Sensore di velocità 40

7 Riferimenti bibliografici **41**

Capitolo 1

Introduzione

La prima macchina elettromagnetica rotante conosciuta, è stata inventata da Michael Faraday nel 1821: consisteva in un filo conduttore tenuto fermo verticalmente alla sua estremità superiore in modo che l'estremità inferiore fosse immersa in un piatto contenente mercurio. Un magnete permanente circolare era sistemato al centro del piatto. Quando una corrente elettrica veniva fatta scorrere nel filo, questo ruotava attorno al magnete mostrando che la corrente generava un campo magnetico attorno al filo.
Sfruttando questa fondamentale proprietà elettromagnetica e conoscendo la legge che descrive le interazioni fra campo elettrico e campo magnetico (forza di Lorentz) è stato possibile capire come, un circuito chiuso percorso da corrente, si comportasse all'interno di un campo magnetico portando così alla realizzazione dei primi motori elettrici.

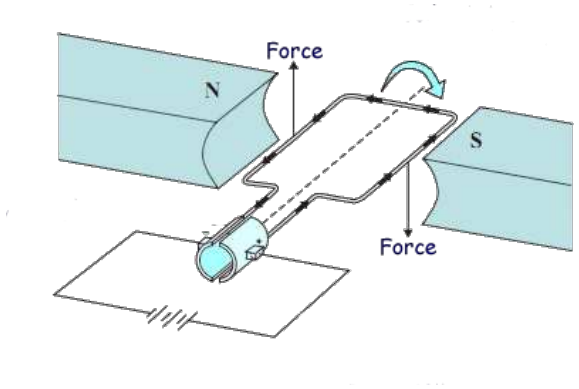
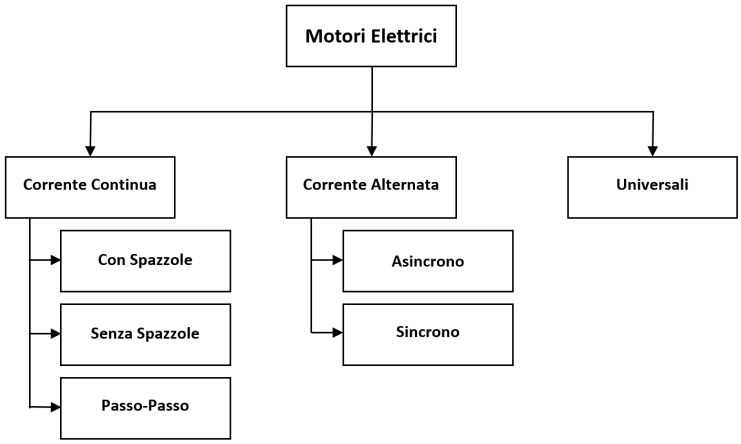


Figura 1.1: Spira percorsa da corrente in un campo magnetico

Al giorno d'oggi i motori elettrici hanno trovato diversi utilizzi: nel trasporto pubblico vengono usati nella locomotive dei treni elettrici e nei treni a levitazione magnetica; nel trasporto privato invece trovano largo impiego nelle auto , nelle biciclette e nei monopattini. Motori elettrici sono anche molto diffusi nell'ambiente industriale per il funzionamento della gran parte delle macchine che costituiscono la catena di montaggio.
La grande diffusione in ambiti così diversi del motore elettrico è dovuta anche al fatto che ne esistono molti tipi con caratteristiche differenti.



I vari tipi di motore si distinguono in base al tipo di alimentazione, dividendosi quindi in motori a corrente continua , motori a corrente alternata e motori universali che, con gli opportuni adattamenti, posso funzionare sia a corrente continua che a corrente alternata.

Figura 1.2: Tipi di motore

In questo elaborato viene studiato l'utilizzo di motori a corrente continua senza spazzole (brushless DC) nel contesto della mobilità privata; in particolare, viene realizzato un sistema di controllo di questi motori per il loro utilizzo in uno skateboard elettrico radiocomandato.

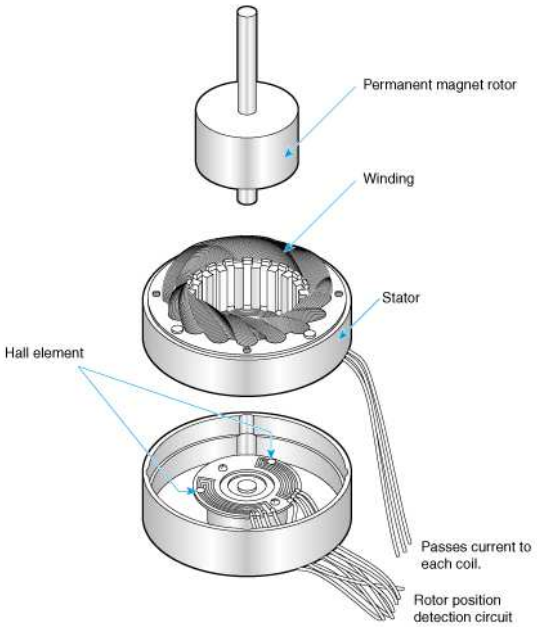


Figura 1.3: Motore Brushless

Analizzando i motori senza spazzole, notiamo che questi, come suggerisce il nome, non utilizzano spazzole per alimentare le spire all'interno del rotore, questo perché il rotore, in questo caso, è dotato di magneti permanenti mentre le spire che generano il campo elettromagnetico sono situate nello statore. Questo permette ai motori brushless di avere dimensioni ridotte inoltre, eliminando le spazzole e non utilizzando slip ring, riduce notevolmente il bisogno di manutenzione ed allunga la vita del motore eliminando praticamente ogni forma di attrito interna.
Nel motore brushless sono spesso inseriti dei sensori ad effetto Hall che, rilevando i campi magnetici, consentono di capire la posizione dei magneti permanenti rispetto alle spire permettendo al sistema di controllo di decidere, di conseguenza, quali spire alimentare.

I sensori di Hall sfruttano la proprietà dei campi magnetici per cui, quando un conduttore percorso da corrente viene immerso in un campo magnetico perpendicolare al verso della corrente si forma una differenza di potenziale attraverso il conduttore. Considerando per esempio una lastra di metallo di larghezza w e spessore d percorsa da una densità di corrente $\vec{j}$ e immergendo la stessa in un campo

magnetico $\vec{B}$ perpendicolare alla corrente, le cariche che percorrono il conduttore subiranno gli effetti della forza di Lorentz.

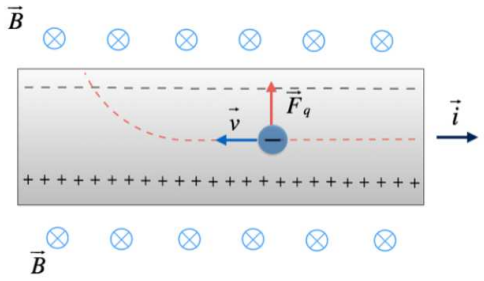


Figura 1.4: Effetto Hall

Le cariche, subendo la forza $\vec{F} = q\vec{v} \times \vec{B}$ si dispongono da un lato o dall'altro del conduttore in base al segno della loro carica, creando all'interno del conduttore un campo elettrico e quindi una differenza di potenziale tra le due facce. Nel caso di perpendicolarità tra corrente e campo magnetico, il campo elettrico vale $E = vB$ e la differenza di potenziale che ne risulta $V_H = Ew$.

Sfruttando questo effetto, i sensori di Hall, consentono di capire se si trovano vicino a un campo magnetico e di conseguenza di conoscere la posizione del rotore.

Capitolo 2

Schema Progetto

In questo capitolo verranno studiati gli schemi a blocchi del progetto completo e delle componenti principali di quest'ultimo con l'obbiettivo di capirne la composizione e l funzionalità.

2.1 Macro blocchi

Lo skateboard elettrico viene controllato muovendo un potenziometro "joystick" la cui posizione determina il valore di un segnale analogico a tempo continuo che viene poi catturato dal processore di Arduino; questo, tramite un processo di conversione A/D che verrà esaminato più avanti in questo capitolo, converte il segnale analogico in un segnale digitale. Il segnale digitale viene, con gli opportuni aggiustamenti, inviato al trasmettitore che, a sua volta, invia un opportuno segnale radio concorde al segnale ricevuto.
Il segnale radio inviato dal telecomando viene ricevuto dal ricevitore situato sotto lo skateboard che manda quindi un segnale digitale ad Arduino. Arduino riceve questo segnale, lo elabora e, di conseguenza, invia tre opportuni segnali digitali modulati tramite modulazione PWM, esaminata più avanti in questo capitolo, al circuito di controllo che, essenzialmente, riscala questi segnali dandogli la potenza necessaria e li manda alle fasi del motore.
In base alla posizione del motore il circuito di controllo riceve un feedback digitale dai sensori di Hall presenti nel motore, questo feedback viene opportunamente scalato in modo da poter essere elaborato dal processore di Arduino che, in base ai loro valori, deciderà quali fasi del motore eccitare.

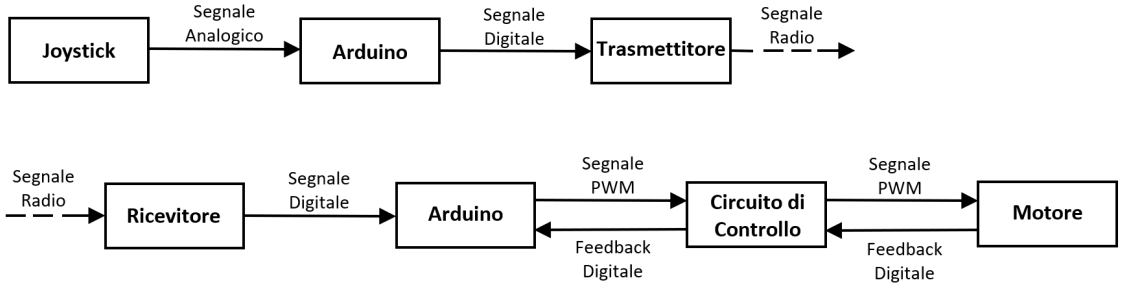


Figura 2.1: Schema a blocchi Skateboard

2.2 Conversione Analogico/Digitale

La scheda Arduino dispone di un Convertitore A/D integrato (ADC - Analog to Digital Converter) a sei canali con una risoluzione di 10 bit. In questo modo i segnali nel range di tensione di $0 \div 5$ V vengono mappati in numeri interi da 0 a 1023, ovvero 4.9 mV per ogni unità.

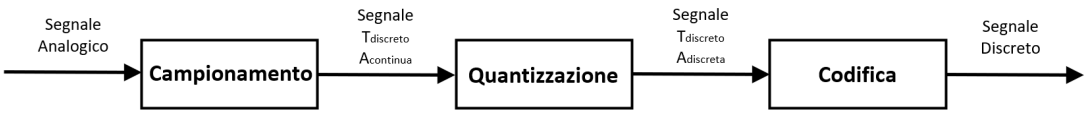


Figura 2.2: Schema a blocchi ADC

2.2.1 Campionamento

Il campionamento serve per discretizzare le variazioni nel tempo della grandezza analogica, cioè per trasformarla in una grandezza che varia solo in corrispondenza di determinati istanti di tempo. Campionare una grandezza analogica significa trasformarla in un'altra grandezza, la quale però non varia più con continuità nel tempo, ma solo in corrispondenza di determinati istanti prefissati. La grandezza campionata non è ancora una grandezza digitale, in quanto essa può ancora assumere infiniti valori.

2.2.2 Quantizzazione

Un quantizzatore approssima ogni valore continuo del segnale campionato con un valore quantizzato appartenente ad un alfabeto finito con L elementi, in questo 1024 possibili valori.

2.2.3 Codifica

Nel processo di codifica viene assegnato, per ogni valore quantizzato, un valore binario. Generando così un segnale binario discreto che rappresenta quindi la conversione digitale del segnale analogico iniziale.

2.3 Modulazione PWM

La modulazione PWM (pulse-width modulation) o modulazione di larghezza di impulso è un tipo di modulazione digitale che permette di ottenere una tensione media variabile dipendente dal rapporto tra la durata dell'impulso positivo e quella dell'intero periodo (duty cycle). Questo tipo di modulazione permette quindi di simulare un segnale analogico partendo da un segnale digitale ed, essendo già implementata

all'interno di Arduino tramite l'utilizzo di 2 clock, è stata immediata la scelta di questo tipo di modulazione per il controllo della velocità dei motori.

2.4 Funzionamento motore brushless con sensori di Hall

Tutte le componenti precedentemente esaminate si uniscono per il corretto funzionamento dei motori brushless; questi, nonostante abbiano molti vantaggi rispetto ai motori con le spazzole, necessitano di un sistema di controllo più complesso. Il commutatore elettronico del motore brushless eccita in sequenza le bobine dello statore generando un campo elettrico rotante che "trascina" con sé il rotore composto da magneti permanenti che, nel caso specifico del motore utilizzato in questo progetto, si trovano sulla circonferenza esterna come si vede in Figura 2.3.

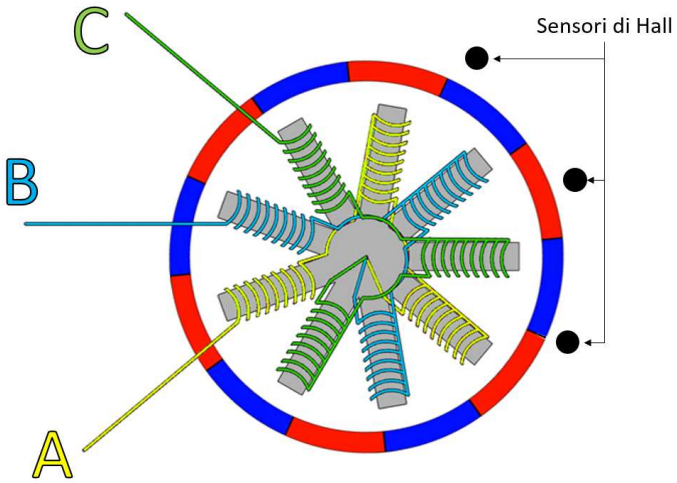


Figura 2.3: Motore brushless del mozzo

In un motore trifase, tre sensori ad effetto Hall incorporati nello statore, indicano al processore le posizioni relative di statore e rotore per far sì che possa eccitare gli avvolgimenti nella sequenza giusta e al momento giusto. Quando i poli magnetici del rotore passano davanti ai sensori Hall, viene generato un segnale alto (per un polo) o basso (per il polo opposto). Come analizzato nel dettaglio più avanti, combinando i segnali dei tre sensori, è possibile determinare l'esatta sequenza di commutazione.

Benché relativamente semplici dal punto di vista meccanico, i motori BLDC richiedono alimentatori stabilizzati e componenti elettronici di controllo sofisticati. La Figura 2.4 mostra una tipica configurazione per il pilotaggio di un motore BLDC con sensori a effetto Hall. Questo sistema mostra le tre bobine del motore disposte in una configurazione a "Y", un microcontrollore, un driver e un inverter trifase comprendente sei transistor MOSFET. L'uscita dal microcontrollore comprende segnali a modulazione della larghezza di impulso (PWM) che determinano i valori medi di tensione e corrente alle bobine (e quindi coppia e velocità del motore). Il motore utilizza tre sensori a effetto Hall per indicare la posizione del rotore. Il rotore

stesso utilizza due coppie di magneti permanenti per generare il flusso magnetico.

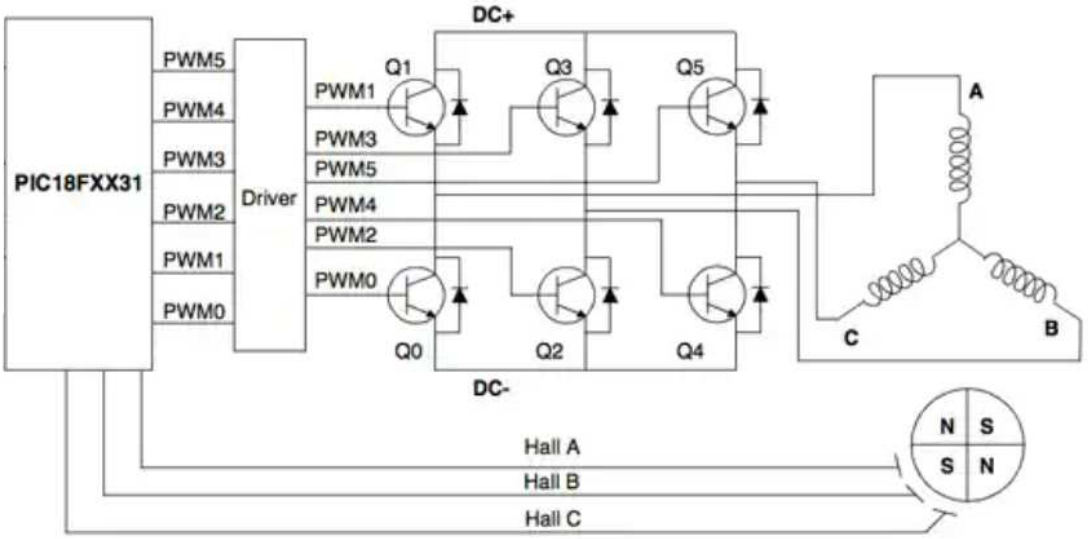


Figura 2.4: Pilotaggio motore brushless

Così facendo, se si misurassero i valori di tensione sugli avvolgimenti e sui sensori di Hall nel tempo, si vedrebbero degli andamenti illustrati in Figura 2.5.

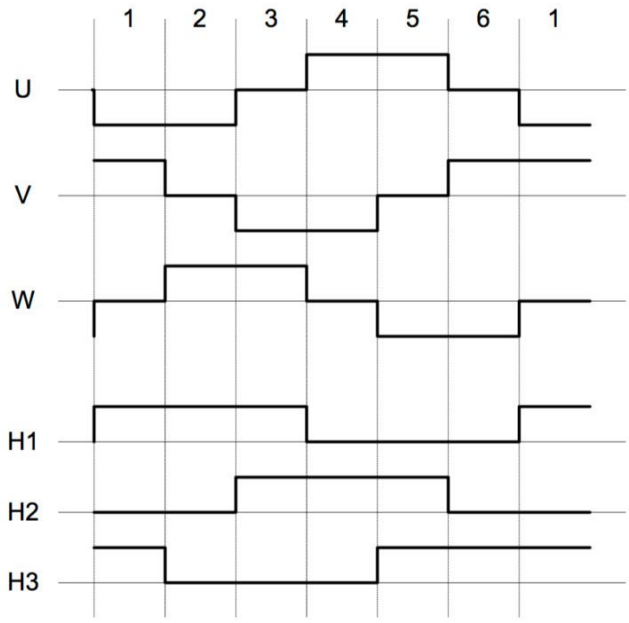


Figura 2.5: Andamento valori di tensione

Capitolo 3

Hardware

In questo capitolo verranno elencati tutti gli elementi hardware utilizzati per la realizzazione del progetto, motivando la scelta di questi ed analizzandone le caratteristiche tecniche.

3.1 Elegoo Nano V3.0

- Microcontrollore: ATmega328
- Tensione logica: 5 V
- Tensione di alimentazione: $6 \div 20$ V
- Pin I/O Digitali: 14
- Pin di Input Analogici: 8
- Corrente pin I/O: 40 mA
- Memoria Flash: 32 kB
- SRAM: 2 kB
- Frequenza di Clock: 16 MHz

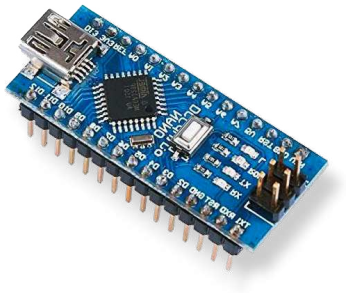


Figura 3.1: Elegoo Nano V3.0

Elegoo Nano è una scheda clone di quella della famiglia "Arduino" basata sui processori ATmega, in questo caso un ATmega328. Su questa scheda è presente una porta micro-B USB, che può essere utilizzata per alimentare la scheda e per programmarla, un pulsante di reset , il processore ATmega328 e tutti i componenti necessari a consentire il corretto funzionamento di quest'ultimo. Elegoo Nano dispone di 22 pin I/O digitali e analogici (utili per la lettura di sensori), di cui 6 PWM (indispensabile per il controllo di motori), è inoltre compatibile con l'IDE (Integrated Development Enviroment) di Arduino basato sul linguaggio di programmazione C che rende semplice ed immediata la programmazione della scheda. È stata scelta questa scheda per il progetto in quanto le sue dimensione sono inferiori rispetto ad altre che vengono normalmente utilizzate per scopi simili (Arduino Uno, Raspberry PI, etc...) ed è più conveniente dal punto di vista economico.

3.2 Motore brushless con sensori di Hall

- Potenza: 240W
- Tensione: 24 V
- Velocità: 1850 rpm
- Corrente massima: 10 A
- Diametro: 75 mm

Figura 3.2: Motore Brushless

I due motori brushless utilizzati per questo progetto sono due motori del mozzo (hub motor) da 240W alimentati a 24V. Sono stati scelti questi motori per la loro semplicità di assemblaggio in quanto rendono direttamente la ruota motrice senza bisogno di sistemi di trasmissione, per la presenza dei sensori di Hall che rendono più semplice il controllo delle fasi del motore e per la loro potenza più che sufficiente per il loro scopo.

3.3 modulo nRF24l01

- Tensione di alimentazione: 3,3 V
- Tensione logica: 5 V
- Frequenza: $2400 \div 2525 MHz$
- Corrente massima: 10 A
- Diametro: 75 mm

Figura 3.3: Modulo nRF24l01

Il modulo nRF24l01 è un modulo radio a basso consumo energetico che riesce a stabilire connessioni bidirezionali comunicando sulla banda di frequenza 2.4GHz. Per essere gestito da Arduino, il modulo nRF24L01 utilizza l'interfaccia SPI (Serial Peripheral Interface) ed è un interfaccia seriale che consente la comunicazione tra il microcontrollore dell'Arduino e altri circuiti integrati esterni. La comunicazione avviene tra un dispositivo master e dispositivi detti slave. La scelta è ricaduta su questo modulo di trasmissione per il suo rapporto qualità prezzo e per la sua predisposizione alla comunicazione con Arduino.

3.4 Potenziometro "joystick"

- Tensione di alimentazione: 5 V
- Resistenza interna: 10 KΩ
- Terminali: 5
- Grado di rotazione: 120°

Figura 3.4: Potenziometro "joystick"

Questo particolare potenziometro è stato scelto per diversi motivi. L'alimentazione a 5V lo rende facilmente utilizzabile con Arduino e la sua forma, simile quella di un joystick, rende semplice il suo utilizzo manuale con una buona precisione.

3.5 Interruttori

Nel progetto sono stati utilizzati 3 tipi diversi di interruttori: un interruttore a bilanciere per alimentare i circuiti stampati, un deviatore a bilanciere per alimentari le schede Arduino Nano presenti sotto lo skateboard ed un deviatore a scorrimento per l'accensione della scheda Arduino Nano presente sul telecomando.

Figura 3.5: Interruttori

Gli interruttori utilizzati in questo progetto sono stati scelti in base a diversi criteri. Il deviatore a scorrimento utilizzato nel telecomando è stato scelto per le sue dimensioni ridotte e le sue caratteristiche tecniche sufficienti a soddisfare la modesta richiesta di energia necessaria ad alimentare un Arduino. Lo stesso criterio è stato utilizzato per gli interruttori presenti sullo skateboard, entrambi soddisfano le caratteristiche tecniche necessarie, in particolare l'interruttore a bilanciere che alimenta i circuiti stampanti consente il passaggio di un massimo di 25A, maggiori della corrente di picchio massima consumata dai due motori che si aggira sui 20A.

3.6 Resistori

Per questo progetto sono stati usati diversi resistori nella realizzazione della scheda di controllo dei motori.
In dettaglio sono stati utilizzati resistori da 100Ω, 1KΩ, 2,2KΩ e 7,5KΩ. Tutte le resistenze utilizzate hanno un potenza di 1/2Watt ed una tolleranza del 5% tranne per le resistenze da 7,5KΩ che hanno una tolleranza del 1%. La scelta è ricaduta su questi resistori per le caratteristiche tecniche più che sufficienti per il loro scopo all'interno del circuito e per la semplicità di reperibilità.

Figura 3.6: Resistore

3.7 Condensatori

Nella progettazione del circuito di controllo dei motori è risultato indispensabile l'utilizzo di condensatori, sono stati scelti condensatori elettrolitici per le dimensioni ridotte rispetto ai condensatori ceramici e per il loro prezzo ridotto. I valori dei condensatori utilizzati in questo progetto sono 2,2μF, 10μF, 100μF e 220μF, tutti con una tensione nominale di 35V più che sufficiente per un circuito a 24V.

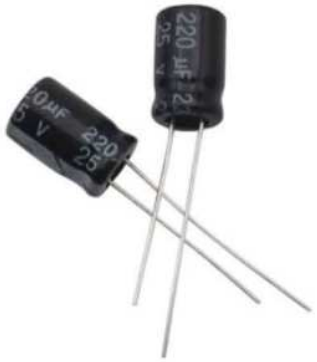


Figura 3.7: Condensatori elettrolitici

3.8 Diodi

- Picco di tensione contraria V_{RRM}: 100 V
- Corrente diretta media $I_{F(AV)}$: 1 A
- Corrente diretta di picco I_{FSM}: 30 A

Figura 3.8: Diodo

Per il corretto funzionamento del Driver 3.10 è necessario l'utilizzo di un diodo tra la tensione di alimentazione V_{CC} e la tensione di floating high-side V_B per evitare

sovraccarichi. A questo scopo sono stati utilizzati diodi 1N4002 che garantiscono caratteristiche sufficiente allo scopo che devono svolgere nel circuito, sono economici e facilmente reperibili.

3.9 Mosfet

- Tensione di rottura drain-to-source V_{DSS}: 55 V
- Tensione gate-to-source massima V_{GS} : $\pm 20V$
- Resistenza minima in conduzione $R_{DS(on)}$: 17,5 mΩ
- Corrente di drain massima I_D: 49 A, V_{GS} @ 10 V

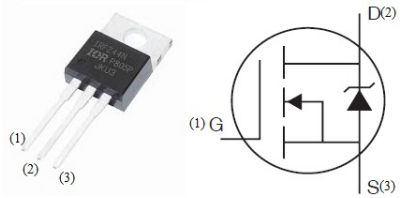


Figura 3.9: Mosfet irfz44n

L'inverter trifase necessario per l'alimentazione dei motori è realizzato utilizzando 6 mosfet di tipo N con diodo zener abbinato. Per questo scopo sono stati utilizzati mosfet IRZ44N per la comodità di avere già il diodo zener integrato e per le caratteristiche tecniche che soddisfano i requisiti per il circuito, in particolare la corrente massima di drain è più del doppio della corrente massima che assorbe il motore così da evitare qualsiasi problema al mosfet.

3.10 Mosfet Driver

- Corrente di uscita H_O: 130 mA
- Corrente di uscita L_O: 270 mA
- Tensione "1" logico V_{IH} minima: 2,5 V
- Tensione "0" logicoV_{IL} massima: 0,8 V
- Tensione di alimentazione: -0,3 $\div$25 V

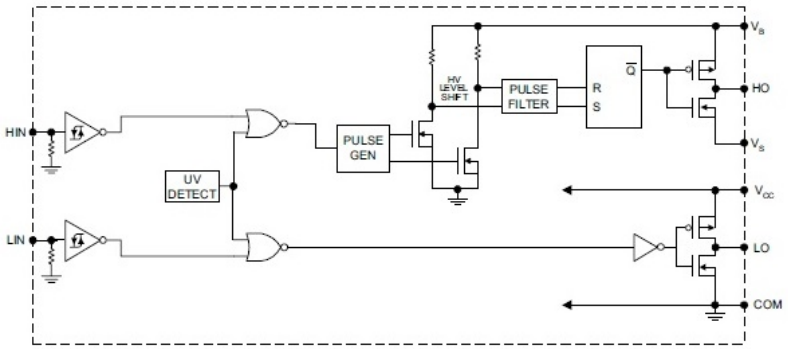


Figura 3.10: IRS2101

Essendo la tensione tipica V_{GS} = 10V per garantire la resistenza R_{DS} minima nei mosfet IRFZ44N, ed essendo la tensione logica di Arduino 5V, risulta necessaria

l'introduzione di un driver per controllare i transistor. Sono stati scelti per questo scopo i driver IRS2101 per i prezzi contenuti e per la predisposizione al controllo di "high and low side mosfet" come in questo progetto. Dato anche l'utilizzo della modulazione PWM risulta fondamentale una risposta il più veloce possibile da parte del driver, in questo caso possiamo vedere dalla Figura 3.11 che si ha un ritardo in accensione che vale $t_{on} + t_r$ che tipicamente assume un valore di 230ns mentre in spegnimento $t_{off} + t_f$ vale tipicamente 185ns.

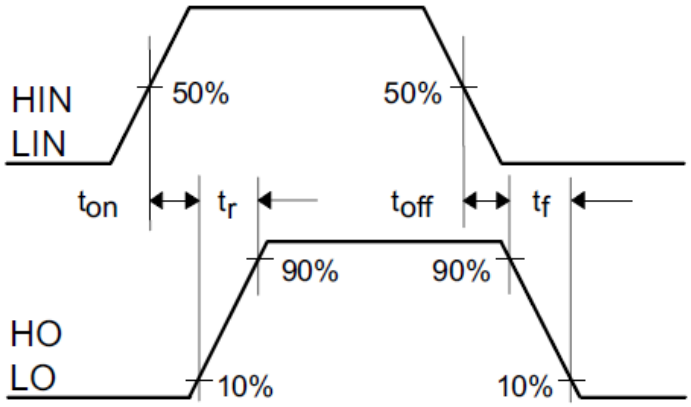


Figura 3.11: ritardo di propagazione

3.11 Regolatore di tensione

- Differenza di tensione input-output V_{i-o}: 40 V
- Corrente al pin ADJ I_{ADJ}: 100 μA
- Massima corrente di carico $I_{o(max)}$: 2,2 A

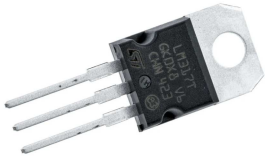


Figura 3.12: LM317T

Per alimentare il Driver 3.10 in modo tale da consentire la corretta tensione ai mosfet è necessario ottenere una tensione vicina ai 10 V. Per fare ciò si è ricorso all'utilizzo di un LM317T, un regolatore di tensione che, data una V_i di alimentazione può, con gli opportuni valori di resistenze, avere in uscita una V_o fino a 40V più bassa. In questo caso usando un LM317T come in Figura 3.13, per ottenere una tensione vicina ai 10V con una V_i di circa 22V, si è usata R_1 di 1kΩ e una R_2 di 7,5kΩ.

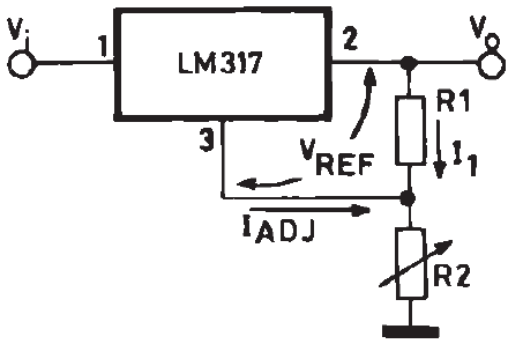


Figura 3.13: Connessione tipica LM317T

3.12 L7805

- Corrente di output I_O: 1,5 A
- Tensione di output V_O: 5 V
- Tensione di input max V_I: 35 V

Figura 3.14: L7805

Durante i primi test del progetto è risultato evidente che, se alimentati a circa 10V il sistema di comunicazione tra Arduino e il ricevitore radio, non funzionava adeguatamente; mentre il disturbo non persisteva quando veniva alimentato tramite USB. Per risolvere il problema si è ricorsi all'utilizzo di un L7805 che permette, a partire da una qualsiasi tensione inferiore di 35V, di ottenere una tensione costante a 5V che, una volta usata per alimentare Arduino ha risolto il problema.

3.13 Batterie 18650

- Capacità: 2500mAh
- Tensione nominale: 3,7 V
- Tensione massima: 4,20 ±0,05 V
- Tensione minima di scarica: 3 V
- Impedenza interna: $\leq 70m\Omega$
- Corrente standard di scaricamento : 0,52 A
- Corrente di scaricamento rapido: 1,3 A
- Corrente di scaricamento massima: 2,6 A

Figura 3.15: Batteria 18650

Per alimentare il telecomando dello skateboard è necessaria una fonte di energia capace di fornire una tensione compresa tra i 6,5 e i 12V per alimentare Arduino dal pin V_{IN} che fosse sufficientemente piccola da poter essere tenuta facilmente in mano. La scelta è ricaduta nelle batterie 18650. Usandone solo 2 in serie, si riesce facilmente a raggiungere la tensione di alimentazione voluta; sono di una dimensione contenuta e con una capacità tale da permette di dover essere ricaricate raramente in un utilizzo di questo tipo.

3.14 Batteria al litio 12V

- Tecnologia: LiPo
- Tensione nominale: 11,1 V
- Capacità: 4500 mAh
- Tasso di scarico: 60 C

Figura 3.16: Batterie LiPo 12V

L'alimentazione dei motori dello skateboard richiede una tensione di 24V e la capacità di fornire almeno 20A, inoltre è necessario che la batteria abbia un peso ridotto per non impattare sulle prestazioni dei motori e una dimensione non ingombrante per poter essere fissata sotto lo skateboard. Viste le necessità la scelta è ricaduta su batterie LiPo (a polimeri di litio), con due batterie 3S1P cioè composte da 3 celle in serie, messe a loro volta in serie, si ottiene una tensione nominale di 22,2V sufficiente per il funzionamento dei motori.

3.15 Circuito di Controllo

Come spiegato precedentemente in 2.4, per avere un corretto funzionamento dei motori brushless è necessario un sistema di controllo non implementabile tramite il solo utilizzo di Arduino. Per la realizzazione di questo sistema è stato prima progettato il circuito tramite un apposito programma (KiCad) rispettando le specifiche dei vari componenti scelti per la realizzazione di quest'ultimo. Nel sistema di controllo, oltre ai driver 3.10 (U1, U2, U3) e ai mosfet 3.9 (Q1-6) necessari alla realizzazione del driver trifase, si nota anche una parte del circuito dedicata all'utilizzo del regolatore di tensione 3.11 per alimentare i driver e un'altra parte dove si vedono le resistenza di pull-up utilizzate per i segnali dei sensori di Hall.

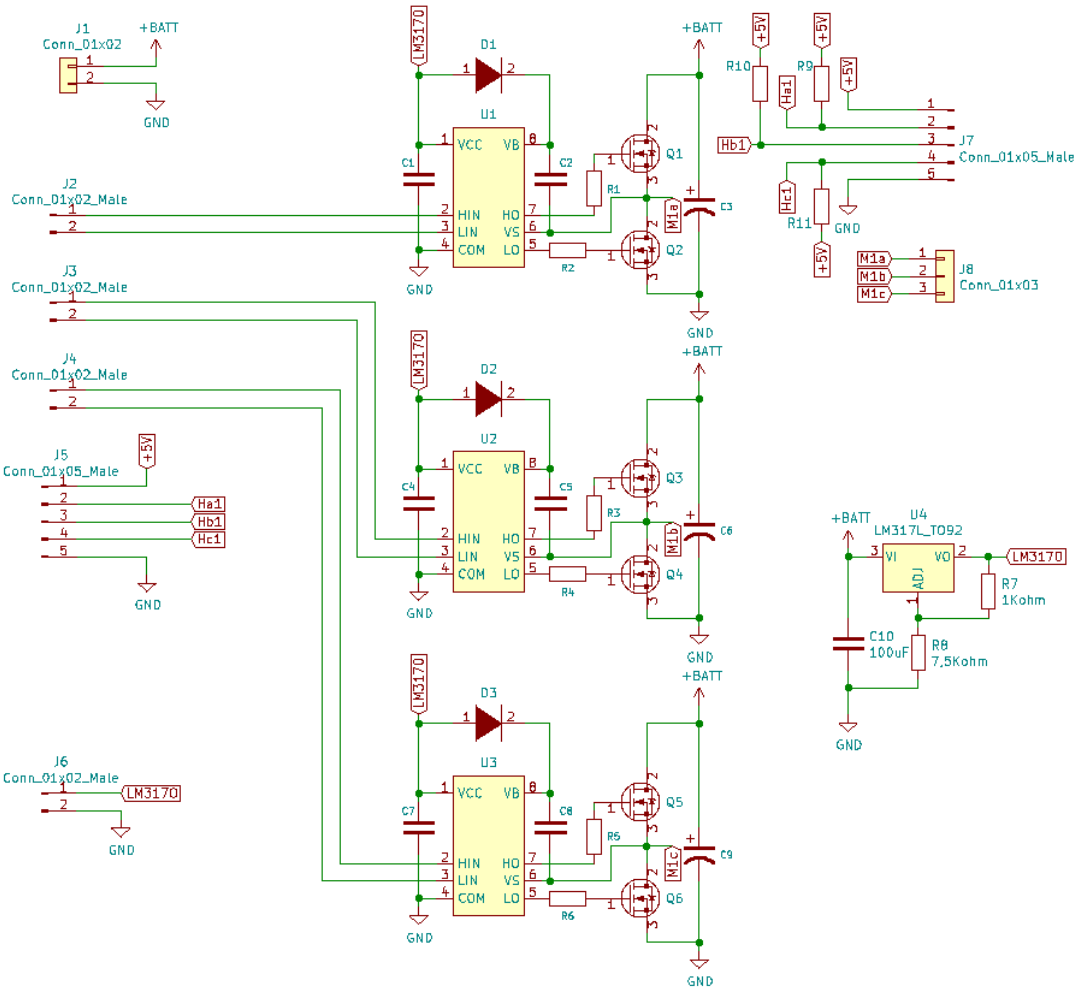


Figura 3.17: Circuito

Successivamente alla progettazione del circuito, si è prima testato il corretto funzionamento su breadboard utilizzando una batteria a 12V invece che 24V per diminuire le potenze in gioco così da non rischiare di danneggiare la breadboard. Una volta verificato il corretto funzionamento, si è passati alla realizzazione del circuito su una scheda "millefori" che però non è risultata adatta allo scopo in quanto, per realizzare il circuito evitando stagnature troppo vicine con rischio di

cortocircuiti sarebbe servita una scheda di dimensioni eccessive. Si è quindi deciso di realizzare un vera e propria scheda a circuito stampato (PCB) tramite un'azienda alla quale è stato mandato il progetto del circuito stampato.

In questo modo si è riusciti a realizzare il circuito in uno spazio molto più contenuto eliminando il rischio di cortocircuiti indesiderati dovuti a piste prodotte manualmente.

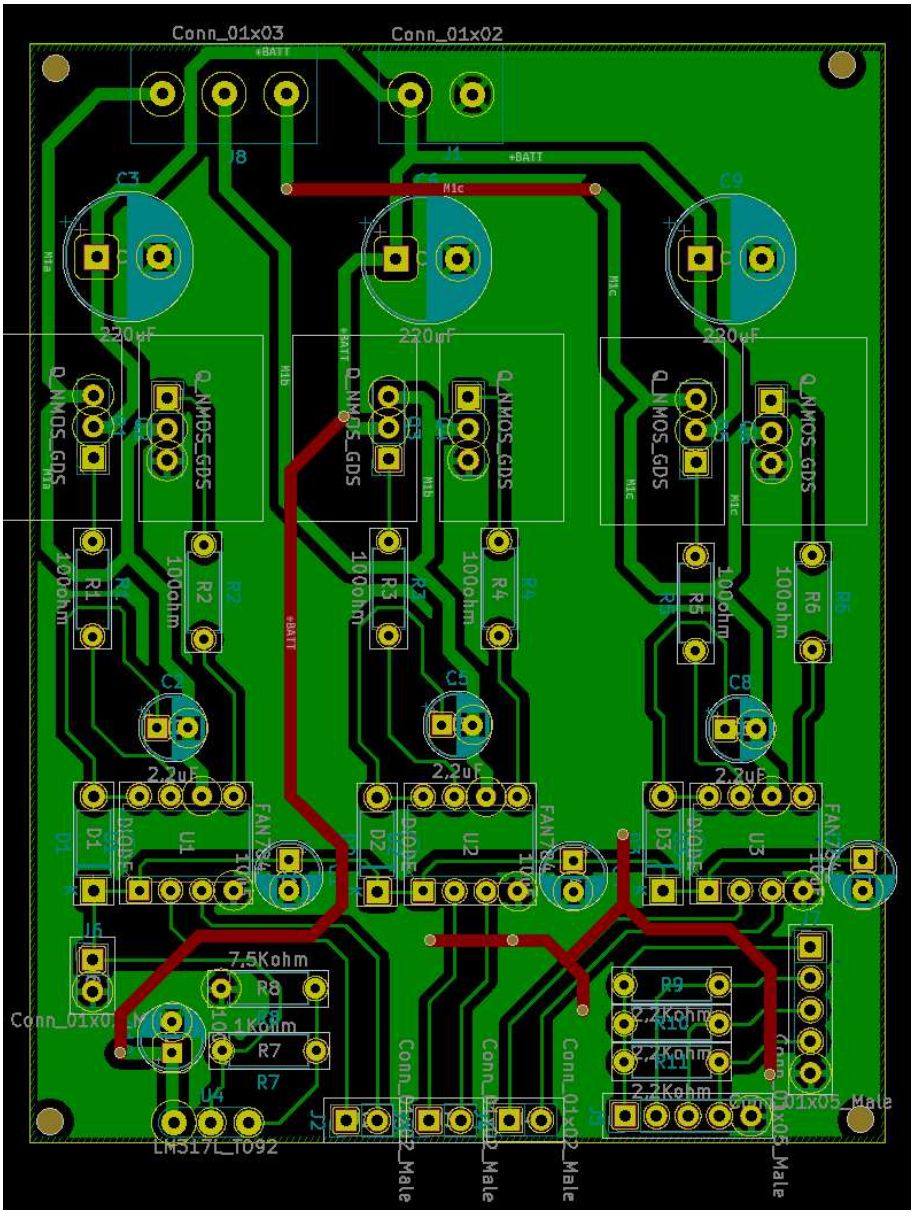


Figura 3.18: Progetto circuito stampato

Una volta ricevuto le schede a circuito stampato è stato sufficiente stagnare i componenti nella posizione corretta, collegare le schede ad Arduino correttamente programmato e ai motori e il progetto si è potuto considerare concluso.

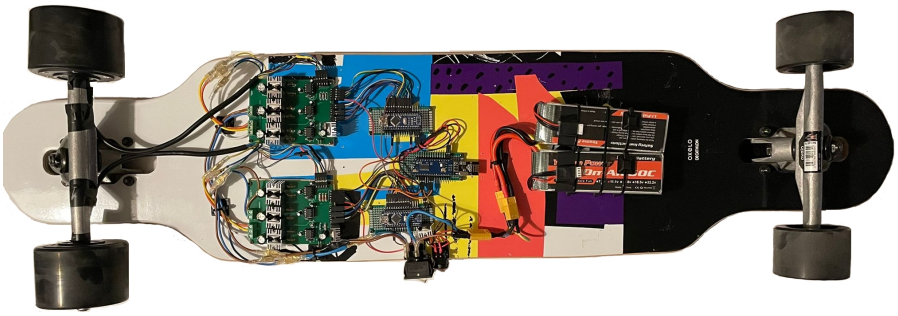


Figura 3.19: Progetto concluso

Capitolo 4

Software

In questo capitolo viene studiato il software sviluppato per il funzionamento del progetto. Nel sistema vengono utilizzati in totale quattro processori, uno usato nel telecomando e tre situati sotto lo skateboard. Di questi tre uno si occupa di riceve il segnale mentre gli altri due, programmati allo stesso identico modo, servono a controllare attivamente i motori. In totale avremo quindi tre programmi differenti, i tre programmi sono stati chiamati:

-Trasmettitore 4.1, si occupa di rilevare il valore del segnale dal potenziometro, elaborarlo ed interfacciarsi con il modulo nRF24l01 per trasmettere un segnale proporzionale al valore del potenziometro;

-Ricevitore 4.2, si interfaccia con il modulo nRF24l01 per leggere il segnale trasmesso dal telecomando ed inviare ai due Arduini che controllano i motori, tramite modulazione PWM, un segnale concorde al segnale ricevuto;

-Controllo Motore 4.3, il programma più complesso dei tre, oltre a leggere il segnale il PWM mandata dall'altro Arduino si occupa di rilevare, tramite interrupt i valori dei sensori di Hall e, in base a questi ultimi, decide a quali pin inoltrare il segnale PWM ricevuto in base a quale bobine del motore bisogna eccitare.

4.1 Trasmettitore

```
1  #include <SPI.h>
2  #include <nRF24L01.h>
3  #include <RF24.h>
4
5  RF24 radio(7, 8);
6
7  const uint8_t address[6] = "Nodo1";
8
9  int joystick;
10
11 void setup() {
12
13   radio.begin();
14   radio.setPALevel(RF24_PA_LOW);
15   radio.setChannel(108);
16   radio.stopListening();
17   radio.openWritingPipe(address);
18
19 }
20
21 void loop() {
22
23   joystick = analogRead(A0) - 512;
24   radio.write(&joystick, sizeof(joystick));
25
26 }
```

Figura 4.1: Codice trasmettitore

Il programma chiamato Trasmettitore è il programma caricato sull'Arduino presente nel telecomando e svolge 2 funzioni, legge il valore del resistore "joystick" e manda tramite il modulo nRF24l01 un segnale radio. Per fare ciò utilizza tre librerie, SPI.h, nRF24L01.h, RF24.h, le prime due librerie principalmente mappano la memoria e inizializzano la comunicazione SPI del processore. Più interessante è invece la libreria RF24.h dove di fatto viene regolata la comunicazione tra il processore e il modulo radio.
Esaminando i prototipi delle funzioni notiamo che, per iniziare una comunicazione con il modulo radio è necessario creare un oggetto di tipo RF24, come è stato fatto alla riga 5, immettendo come parametri i pin adibiti a "chip enable" e "SPI chip select", che in questo caso sono rispettivamente i pin 7 e 8.
Serve poi creare una variabile che contiene un valore che simula l'indirizzo o il "pipe" su cui i due moduli vogliono comunicare, in questo caso, come si vede a riga 7, si è voluto chiamare l'indirizzo "Nodo1".

Una volta create tutte le variabili e gli oggetti necessari, si passa al setup del programma dove, per prima cosa va chiamata la funzione begin() che accende il processore del modulo radio, successivamente viene settato il livello di dell'amplificatore di potenza (Power Amplifier) tramite la funzione setPALevel, in questo caso è stato settato a un livello basso perché la distanza di ricezione richiesta per questo progetto è bassa e in questo modo non si rischia di amplificare eventuali rumori.

Resta da settare il canale radio sul quale trasmettere, come visto in 3.3 infatti il modulo trasmette in una frequenza tra $2400 \div 2525$ MHz ammettendo 125 possibili canali diversi, per questo programma è stato scelto il canale 108 ed è stato impostato alla riga 15 chiamando la funzione setChannel passando direttamente il valore del canale desiderato come parametro.

Essendo questo il programma del trasmettitore, serve settare il modulo radio in modo da trasmettere un segnale; per fare ciò vengono chiamate le funzioni stopListening, che impedisce al modulo di funzionare da ricevitore, e openWritingPipe che inizializza il "pipe" di comunicazione.

La trasmissione radio vera e propria avviene all'interno del loop del programma dove, tramite la funzione write che, per inviare un dato, ha necessità di ricevere come parametri un puntatore al dato che si vuole inviare e il numero di byte che formano il dato, come si può vedere alla riga 24.

Per quanto riguarda la lettura del valore del potenziometro questa è molto più

semplice, essendo Arduino già dotato di convertitore analogico-digitale, il suo "sistema operativo" è già predisposto alla lettura di valori analogici, in questo caso, dato che il potenziometro è collegato ad Arduino tramite il pin A0, è sufficiente chiamare la funzione analogRead(A0) per ottenere un valore da 0 a 1023 a partire dal valore analogico del potenziometro. Per come è costruito, il valore del potenziometro senza alcuna pressione in ogni direzione varia tra 511 e 512, per il progetto è interessante conoscere il valore del potenziometro quanto questo viene "spinto" in avanti mentre i valori che assume quando viene "tirato" vengono ignorati, per questo è stato deciso di trasmettere un valore che varia tra 0 a 512 quando il potenziometro è fermo o viene spinto e assume valori negativi (che verranno poi scartati) quando il potenziometro viene tirato.

4.2 Ricevitore

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

RF24 radio(7, 8);

int mot_dx_direction = 5;
int mot_sx_direction = 6;
int mot_dx_PWM = 9;
int mot_sx_PWM = 10;

int vel = 0;
int motor_speed = 0;

const uint8_t address[6] = "Nodo1";
```

Figura 4.2: Impostazioni ricevitore

Le impostazioni per il programma ricevitore sono molto simili a quelle del trasmettitore 4.1 in quanto, per entrambi i programmi, buona parte del software è dedicato alla comunicazione con il modulo radio che però in questo caso viene usato come ricevitore. Si può notare in Figura 4.2 che le librerie utilizzate sono le stesse, anche i pin "chip enable" e "SPI chip select" sono gli stessi del trasmettitore ma soprattutto, è indispensabile che sia uguale l'indirizzo sul quale si vuole far comunicare i due moduli, in questo caso l'indirizzo è "Nodo1" come sul trasmettitore.

Oltre a comunicare con il modulo radio questo programma deve anche interagire con altri due Arduini, per dire loro a che velocità far girare i motori e se farli ruotare in senso orario o in senso antiorario. Per fare ciò si utilizzano il pin 5 e il pin 6 per il controllo del verso di rotazione, mentre il pin 9 e 10, dotati di PWM, vengono utilizzati per comunicare la velocità di rotazione.

Nella parte di setup del programma vengono settati come output i pin utilizzati per la comunicazione con i due Arduini, e viene settata la comunicazione con il modulo radio come nel programma del trasmettitore.

```
17 void setup() {
18
19   pinMode(mot_dx_PWM, OUTPUT);
20   pinMode(mot_sx_PWM, OUTPUT);
21   pinMode(mot_dx_direction, OUTPUT);
22   pinMode(mot_sx_direction, OUTPUT);
23   radio.begin();
24   radio.setPALevel(RF24_PA_LOW);
25   radio.setChannel(108);
26   radio.openReadingPipe(0, address);
27   radio.startListening();
28
29 }
```

Figura 4.3: Setup ricevitore

Come si vede in figura 4.3 per inizializzare la comunicazione con il modulo radio si utilizzano sempre le funzioni begin, setPALevel e setChannel come per il trasmettitore, questa volta però, per settare il modulo da ricevitore invece che da trasmettitore si apre prima il canale di comunicazione tramite la funzione openReadingPipe e successivamente la funzione startListening che da inizio alla comunicazione tra il trasmettitore e il ricevitore.

Nella sezione ciclica del programma il ricevitore verifica costantemente se ci sono byte da ricevere grazie alla funzione available, funzione di tipo bool che ritorna true se ci sono byte da ricevere nel canale di comunicazione, false altrimenti. Se c'è qualcosa da ricevere tramite la funzione read viene assegnato il valore letto alla variabile il cui puntatore è inserito come parametro della funzione insieme alle sue dimensioni.

```
31 void loop() {
32   while(radio.available(0)){
33
34     radio.read(&vel, sizeof(vel));
35   }
36
37   if(vel < 5) motor_speed = 0;
38   else motor_speed = map(vel, 0, 511, 0, 255);
39
40   digitalWrite(mot_sx_direction, LOW);
41   digitalWrite(mot_dx_direction, HIGH);
42   analogWrite(mot_dx_PWM, motor_speed);
43   analogWrite(mot_sx_PWM, motor_speed);
44 }
```

Figura 4.4: Loop ricevitore

Per quanto riguarda la comunicazione con i due Arduini di controllo dei motori si elabora il segnale ricevuto, che come visto in 4.1 avrà un valore variabile tra -511 e 511. Per i valori negativi e inferiori a 5, si comunica ad Arduino di tenere fermo il motore, questo perché anche quando non spostato, il potenziometro tende a non trovarsi perfettamente a 0 ma a valori solitamente compresi tra -3 e 5.

Quando invece vengono rilevati valori più elevati, tramite la funzione map di Arduino, si riscalano i valori da 0 a 511 in una scala di valori da 0 a 255 ideale per la trasmissione tramite PWM che nel caso di Arduino varia in valori da 0 a 255. Infine, tramite la funzione analogWrite si invia il segnale modulato.

4.3 Controllo Motore

Per il programma di controllo si è deciso di utilizzare, invece che usare il solito linguaggio di programmazione simile al linguaggio C, un linguaggio orientato ai registri del processore, per aver un maggior controllo soprattutto per quanto riguarda la modulazione PWM e il rilevamento del cambiamento di stato dei pin di interrupt.

4.3.1 Setup

```
1  byte bldc_step, motor_speed;
2
3  void setup() {
4    DDRD  |= 0x1C;
5    PORTD  = 0x00;
6    DDRB  |= 0x0E;
7    PORTB  = 0x31;
8
9    TCCR1A = 0;
10   TCCR1B = 0x01;
11
12   TCCR2A = 0;
13   TCCR2B = 0x01;
14
15   ADMUX  = 0x60;
16   ADCSRA = 0x84;
17
18   PCICR  = 4;
19   PCMSK2 = 0xE0;
20
21   bldc_step = (PIND >> 5) & 7;
22   bldc_move();
23 }
```

Figura 4.5: Setup controllo

Come si vede in Figura 4.5, come prima cosa, alla riga 1, vengono create due variabili, bldc_step dove verrà memorizzata in quale delle sei fasi del motore ci si trova, e motor_speed che indica invece un valore da 0 a 255 che rappresenta la velocità desiderata del motore tramite modulazione PWM.
Nel setup del programma si inizia a intervenire sui registri del processore per configurare i pin e gli interrupt, settare i timer interni e configurare il modulo ADC. In particolare, nelle righe da 5 a 8 vengono configurati come output i pin 2, 3, 4, 9, 10 e 11, nelle righe 9 e 10 viene settato il clock del Timer1 mentre nelle righe 12 e 13 viene settato il clock del Timer2, settare i due timer interni di Arduino è necessario per utilizzare poi la modulazione PWM. I comandi di riga 18 e 19 abilitano il pin change interrupt sui pin 5, 6 e 7 dove verranno ricevuti i segnali dai sensori di Hall del motore.

Le ultime due righe del setup, servono a far fare il "primo passo" al motore, assegnando alla variabile bldc_step il valore della fase del motore in base ai valori dei sensori di Hall, e invocando la funzione bldc_move() 4.3.3 che da inizio alla rotazione del motore.

Si studia ora più nel dettaglio cosa fanno effettivamente i comandi utilizzati nel setup.

13.4.11 PORTD – Port D Data Register

Bit	7	6	5	4	3	2	1	0	
0x0B (0x2B)	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

13.4.12 DDRD – Port D Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x0A (0x2A)	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 4.6: PORTD e DDRD

Il registro DDRD gestisce la direzione dei dati, decide quindi se i pin associati a Port D fungeranno da input se il valore corrispettivo in DDRD vale 0 o da output se vale 1. In questo caso i pin che vogliamo utilizzare come output associati a Port D

sono i pin 2, 3 e 4 associati rispettivamente alle posizioni PD3, PD4 e PD5 in Port D come si vede in Figura 4.7, per settare questi pin come output bisogna quindi portare a 1 i bit in posizione 3, 4 e 5. Per fare ciò si è deciso di compiere un'operazione di or logico tra il valore iniziale di DDRD (che vale 0) con il valore esadecimale 1C che in binario vale 00011100, successivamente si assegna il valore logico 0 a tutti i pin di Port D tramite il comando PORTD = 0x00.

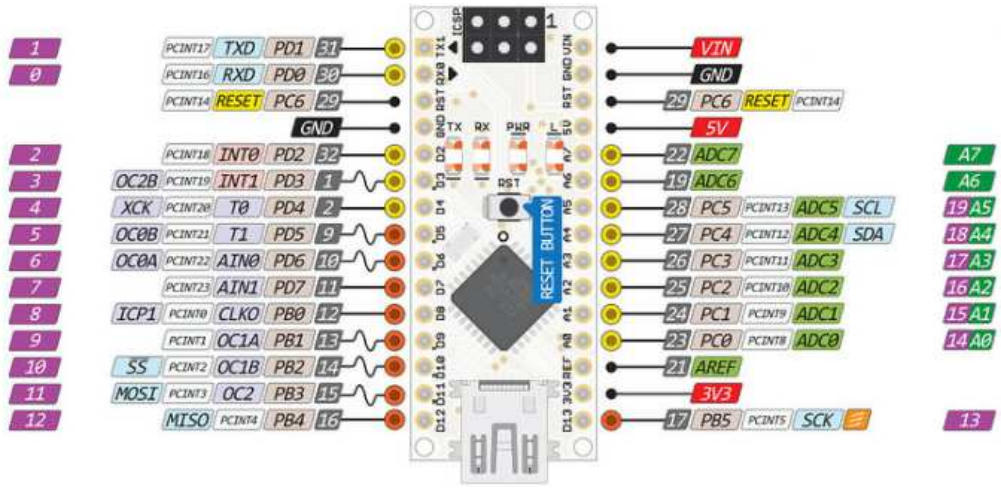
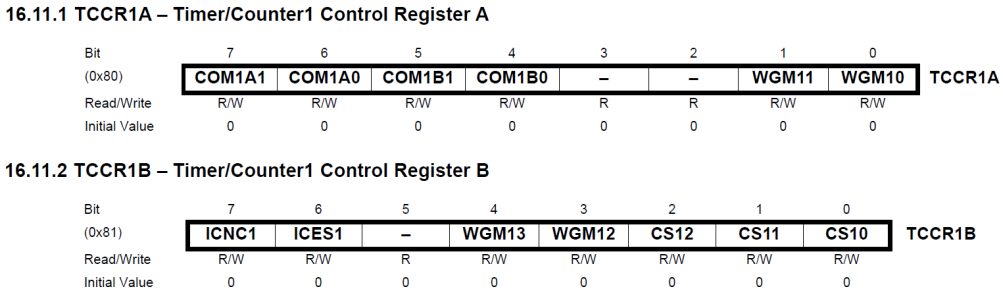


Figura 4.7: Pinout Arduino Nano

Lo stesso ragionamento è stato utilizzato per i comandi DDRB |= 0x0E e PORTB = 0x31 che settano invece i pin 9, 10 e 11 che invece appartengo al Port B come output.

I comandi TCCR1A = 0 e TCCR1B = 0x01 servono a selezionare in che modo verranno utilizzati i pin OC1A e OC1B che , come si vede in Figura 4.7 corrispondono ai pin 10 e 11, e che velocità di clock vogliamo utilizzare per il Timer1 che consente a questi due pin di essere utilizzati come uscite PWM.



16.11.1 TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

16.11.2 TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 4.8: Registri TCCR1A e TCCR1B

I bit 7:4 del registro TCCR1A regolano il Compare Output Mode per il canale A i bit 7 e 6, e per il canale B i bit 5 e 4, per ora, lasciando tutti questi bit a 0 OC1A e OC1B risultano "sconnessi" e continuano le loro operazioni normali regolare dal registro DDRB. Nel caso uno di questi bit fosse diverso da 0 il comportamento di OC1A e OC1B varia in base ai valori assegnati ai bit WGM13:0 come verrà analizzato in seguito. Con i due comandi presenti nel setup del programma tutti i

bit dei due registri restano a 0 tranne per il bit CS10 del registro TCCR1B, come si evince dalla tabella in Figura 4.9, impostare a 1 questo bit significa utilizzare il clock al massimo della sua velocità che per il clock interno di Arduino vuol dire avere una frequenza di 16MHz.

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Figura 4.9: Tabella Clock

Lo stesso ragionamento si ripete per i comandi TCCR2A = 0 e TCCR2B = 0x01, questi però inizializzano il Timer2 di Arduino che regola la modulazione PWM di OC2A, cioè il pin 11.

I comandi ADMUX = 0x60 e ADCSRA = 0x84 configurano il modulo ADC per la lettura del segnale PWM che arriva dal Ricevitore 4.2.

24.9.1 ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

24.9.2 ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 4.10: Registri ADMUX e ADCSRA

Osservando i due registri si nota che, dando ad ADMUX il valore 0x60 che equivale a 0110000 in binario, si imposta a 1 il valore di REFS0 e ADLAR mentre gli altri valori restano a 0. Quando REFS0 vale 1 e REFS1 vale 0 Arduino imposta come tensione di riferimento per la conversione analogico-digitale la tensione AV_CC, settare ADLAR a 1 invece "shifta" il risultato della conversione a sinistra all'interno del registro in cui verrà depositato. In realtà i registri utilizzati per questo scopo sono due, ADCH e ADCL, con ADLAR = 1 il risultato della conversione verrà registrato come in Figura 4.11. I bit 3:0 di ADMUX servono invece per selezione quale pin analogico sarà utilizzato come ingresso, in questo caso, volendo usare il pin A0, è stato sufficiente lasciare tutti questi bit a 0.

24.9.3.2 ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
(0x79)	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
(0x78)	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Figura 4.11: Registri ADCH e ADCL

Assegnando al registro ADCSRA il valore 0x84 che equivale a 10000100 in binario invece si vanno a modificare i valori di ADEN e ADPS2, ADEN è semplicemente un bit di abilitazione del modulo ADC, impostandolo quindi a 1 si rende possibile la conversione, i bit ADPS2:0 invece determinano il fattore di divisione tra la frequenza di clock di sistema e il clock di input al modulo ADC. Impostando questi 3 bit al valore 100 si selezione un fattore di divisione di 16, impostando quindi il clock del modulo ADC a 1MHz. Il bit ADSC è un bit che, quando impostato a 1 inizia una singola conversione ADC, ADATE abilita invece l'auto triggering dell'ADC consentendo la conversione ogni qualvolta ci sia un ingresso ai pin analogici abilitati dal registro ADMUX, ADIF è un flag interrupt, il suo valore diventa 1 quando la conversione è termita e lancia un interrupt se ADIE, cioè l'ADC Interrupt Enable bit, vale 1.

I comandi PCICR = 4 e PCMSK2 = 0xE0 servono invece ad abilitare gli interrupt sui pin 5:7 i quali riceveranno i segnali provenienti dai sensori di Hall del motore, il valore di questi segnali è indispensabile per determinare la fase del motore corrente e di conseguenza cosa sia necessario fare per passare alla corretta fase successiva.

13.2.4 PCICR – Pin Change Interrupt Control Register

Bit	7	6	5	4	3	2	1	0	
(0x68)	–	–	–	–	–	PCIE2	PCIE1	PCIE0	PCICR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

13.2.6 PCMSK2 – Pin Change Mask Register 2

Bit	7	6	5	4	3	2	1	0	
(0x6D)	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	PCMSK2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 4.12: Registri PCICR e PCMSK2

Tramite il comando PCICR = 4 che equivale ad assegnare al registro il valore binario 00000100 si assegna, a bit PCIE2:0 il valore binario 100, così facendo si abilitano i pin di interrupt PCINT[23:16] mentre restano non abilitati i pin PCINT[14:0]. Questa decisione è stata presa in quanto, volendo utilizzare i pin 5, 6 e 7 per leggere i sensori di Hall, è necessario abilitare i rispettivi PCINT21, PCINT22 e PCINT23.

Il comando PCMSK2 = 0xE0 invece assegna al registro il valore binario 11100000 portando quindi a 1 il valore dei primi 3 bit abilitando l'interrupt su cambio di stato dei tre pin PCINT[21:23].

Infine nel setup del programma si esegue il comando bldc_step = (PIND >> 5)

& 7 che va a fare una operazione di and logico tra il valore dei pin di portD shiftato di 5 posizione a destra e 7, cioè va a prendere il valore di PD7, PD6, PD5 ignorando i valori di PD4:0 e li mette in and con 00000111. Il risultato di questa operazione viene poi memorizzano nella variabile bldc_step per conoscere la fase del motore attuale grazie ai valori dei sensori di Hall.

Viene infine chiamata la funzione bldc_move() che verrà esaminata successivamente.

4.3.2 ISR (PCINT2_vect)

```
24
25 ISR (PCINT2_vect){
26   bldc_step = (PIND >> 5) & 7;
27   bldc_move();
28 }
29
```

Figura 4.13: Funzione ISR (PCINT2_vect)

La funzione ISR (PCINT2_vect) è la funzione che viene autonomamente invocata all'attivazione di uno degli interrupt. La funzione in questo caso si limita ad aggiornare la fase attuale del motore rilevando i nuovi valori dei sensori di Hall e chiama la funzione bldc_move().

4.3.3 bldc_move()

```
30 void bldc_move(){
31    if(PINB & 1 == 1){
32      move_clockwise();
33    }else{
34      move_counter_clockwise();
35    }
36
37 }
38
```

Figura 4.14: Funzione bldc_move()

La funzione bldc_move() si limita a controllare, tramite un'operazione di and logico, se il pin 8 che corrisponde a PB0, vale 0 o 1, in base a questo determina se il motore dovrà ruotare in senso orario o in senso antiorario e chiamare quindi la funzione move_clockwise() 4.3.4 se si vuole far ruotare il motore in senso orario o move_counterclockwise() 4.3.5 se lo si vuole far ruotare in senso antiorario.

Come visto in 4.2 la scelta della direzione di rotazione del motore avviene precedentemente, questo consente allo stesso programma di essere caricato in entrambi gli Arduini, per avere poi la rotazione corretta è sufficiente cablare correttamente il tutto in modo che il processore che controlla il motore di destra abbia il pin 8 connesso al pin 5 del ricevitore mentre quello che controlla il motore di sinistra deve avere il pin 8 connesso al pin 6.

4.3.4 move_clockwise()

```
39 void move_clockwise(){
40
41   switch(bldc_step){
42     case 1:
43       AH_CL();
44       break;
45     case 2:
46       BH_AL();
47       break;
48     case 3:
49       BH_CL();
50       break;
51     case 4:
52       CH_BL();
53       break;
54     case 5:
55       AH_BL();
56       break;
57     case 6:
58       CH_AL();
59       break;
60     default:
61       PORTD = 0;
62       break;
63   }
64 }
65
```

Figura 4.15: Funzione move_clockwise()

La funzione move_clockwise() decide, tramite la funzione switch dove come parametro viene usata la variabile bldc_step che rappresenta la fase in cui si trova il motore, quali bobine del motore eccitare e di conseguenza come far ruotare il motore. Se per esempio i valori logici dei sensori di Hall sono 001 (case 1), viene invocata la funzione AH_CL() che farà si di attivare il mosfet high della bobina A e il mosfet low della bobina C lasciando disattivati entrambi i mosfet della bobina B che si troverà quindi in uno stato "flottante". Come si vede in Figura 2.5, in ogni fase del motore è presente una bobina col mosfet high attivo, una col mosfet low attivo ed un'altra con entrambi i mosfet "spenti".
Una volta eseguita la funzione AH_CL il motore ruoterà, cambiando la posizione dei magneti permanenti cambieranno anche i valori logici dei sensori di Hall, come visto in Figura 2.5, richiamando così la funzione ISR (PCINT2_vect) 4.3.2 che aggiornerà il valore di bldc_step permettendo di continuare la rotazione del motore.
Si nota che, di default, viene eseguito il comando PORTD = 0 che assicura che tutti i mosfet siano disattivati.

4.3.5 move_counterclockwise()

La funzione move_counterclockwise() svolge la stessa funzione di move_clockwise() 4.3.4 ma con una leggera differenza, invertendo infatti le bobine eccitate si ottiene una rotazione con il verso contrario.

Questo è indispensabile per questo progetto in quante per portare avanti lo skateboard le due ruote devono girare nel verso opposto

```
66 void move_counter_clockwise(){
67
68   switch(bldc_step){
69     case 1:
70       CH_AL();
71       break;
72     case 2:
73       AH_BL();
74       break;
75     case 3:
76       CH_BL();
77       break;
78     case 4:
79       BH_CL();
80       break;
81     case 5:
82       BH_AL();
83       break;
84     case 6:
85       AH_CL();
86       break;
87     default:
88       PORTD = 0;
89       break;
90   }
91 }
92
```

Figura 4.16: Funzione move_counterclockwise()

.

4.3.6 Eccitamento bobine

Le funzioni invocate in move_clockwise() 4.3.4 e move_counterclockwise() 4.3.5 per eccitare le bobine del motore svolgono tutte una funzione simile con minime differenze. In questo sottocapitolo viene esaminata la funzione AH_BL() specificando quali differenze vengono apportate nelle altre funzioni.

```
93 void AH_BL(){
94   PORTD &=  0xEB;
95   PORTD |=  0x08;
96   TCCR2A =  0;
97   TCCR1A =  0x81;
98 }
```

Figura 4.17: Funzione AH_BL()

Il progetto è stato pensato in modo da controllare i mosfet high che alimentano le bobine tramite pwm, mentre i mosfet low vengono controllati semplicemente tramite un segnale logico. Per fare ciò, ricordato quali pin logici e quali pin pwm sono stati impostanti come output in Setup 4.3.5, è stato deciso di connettere, per quanto riguarda la bobina A, il controllo

del mosfet high al pin 9 e quello del mosfet low al pin 2, per la bobina B il controllo del mosfet high è collegato al pin 10 e quello del mosfet low al pin 3 mentre per la bobina C il mosfet high è controllato dal pin 11 e quello low dal pin 4. Studiando i comandi della funzione AH_BL(), possiamo vedere come, tramite il comando PORTD &= 0xEB si va a compiere un'operazione di and logico tra il valore di PORTD e 0xEB che equivale a 11101011, cioè si vanno a portare a 0 i valore di PD2 e PD4 lasciando invariati gli altri per poi successivamente compiere un'operazione di or logico questa volta tra PORTD e 0x08 che equivale a 00001000 che va quindi a portare a 1 il valore di PD3 lasciando invariati gli altri. Con questi due comandi ci assicura quindi di portare a 0 i valori di PD2 e PD4 che corrispondono al pin 2 e al pin 4 e di portare a 1 invece PD3 che corrisponde al pin 3, in questo modo abbiamo attivato il mosfet low della bobina B.

Similmente con i comandi TCCR2A = 0 e TCCR1A = 0x81 si attiva il mosfet high della bobina A tramite segnale PWM. Più precisamente, con TCCR2A = 0 si "scollega" OC2A cioè il pin 11, mentre con TCCR1A = 0x81, ricordando la Figura 4.8 si va ad attivare l'uscita PWM di OC1A cioè il pin 9 che controlla il mosfet high della bobina A.

Per le altre combinazioni di eccitazioni di bobine il ragionamento si ripete variando i mosfet che si vogliono attivare o disattivare.

4.3.7 loop

Nella sezione ciclica del programma di controllo si effettua costantemente la conversione analogico-digitale del segnale ricevuto dal ricevitore e si utilizza il risultato come valore del duty cycle dei segnali PWM in uscita tramite la funzione SET_PWM_DUTY().

```
136 void loop() {
137   ADCSRA |= 1 << ADSC;
138   while(ADCSRA & 0x40);
139   motor_speed = ADCH;
140   SET_PWM_DUTY(motor_speed);
141 }
```

Figura 4.18: Funzione loop()

Dato che il segnale che arriva dal ricevitore è già un segnale PWM e data la scelta di registrare il valore della conversione analogico-digitale allineandola a sinistra fatta in setup 4.3.1, leggendo solamente il valore del registro ADCH si può ottenere il risultato della conversione con buona approssimazione in un valore che varia già da 0 a 255, ottimo quindi per essere utilizzato come duty cycle.

4.3.8 SET_PWM_DUTY(byte duty)

```
130 void SET_PWM_DUTY(byte duty){
131   OCR1A  = duty;
132   OCR1B  = duty;
133   OCR2A  = duty;
134   }
135
```

Figura 4.19: Funzione SET_PWM_DUTY(byte duty)

La funzione SET_PWM_DUTY(byte duty) si limita ad impostare il duty cyclo di OC1A, OC1B e OC2A al valore del parametro duty, per fare ciò assegna ai registri a 16 bit OCR1A, OCR1B, OCR2A il valore del parametro.

Capitolo 5

Osservazioni

Nello sviluppo di questo progetto sono state riscontrate diverse difficoltà, in questo capitolo si analizzano i problemi incontrati e le soluzioni che si sono scelte per superarli.

5.1 Timer interni

Fin dall'inizio della progettazione dello skateboard è apparso evidente che sarebbe stato necessario l'utilizzo della modulazione PWM per un controllo preciso dei motori, i pin predisposti a fungere da uscita PWM sono sei in Arduino Nano.

Per funzionare, la modulazione PWM, necessita di un segnale di clock che viene ricavato dai timer interni di Arduino, in totale questi timer sono tre, Timer0, Timer1 e Timer2, ad ogni timer sono associate 2 uscite PWM, rispettivamente OC0A→D6, OC0B→D5, OC1A→D9, OC1B→D10, OC2A→D12, OC2B→D3.

Teoricamente quindi sarebbe stato possibile, utilizzando soltanto due Arduini, controllare entrambi i motori controllando sia i mosfet high che i mosfet low tramite PWM. Procedendo nello sviluppo però e nello studio dei datasheet del processore è apparso evidente che questo non sarebbe stato possibile. Il Timer0 di Arduino è infatti utilizzato dal processore per compiere diverse funzioni interne tra cui le operazioni di delay(), millis() e micros(), per non modificare quindi le funzioni interne della scheda si è scelto di non utilizzare Timer0, limitando però le possibili uscite PWM a quattro ed essendo quindi obbligati a controllare solo i mosfet high tramite modulazione PWM e controllare i mosfet low con normali segnali digitali.

5.2 Trasmissione radio

L'implementazione della trasmissione radio dal telecomando allo skateboard ha portato due principali problemi allo sviluppo del progetto.

5.2.1 Scelta del modulo radio

Inizialmente il modulo scelto per la comunicazione radio è stato un modulo RF a 433MHz, a differenza del modulo nRF24l01 3.3 utilizzato nel progetto finale, questo

era più economico, composto da un ricevitore ed un trasmettitore diversi tra loro, la comunicazione non avveniva tramite SPI e la connessione con Arduino era più semplice.

Purtroppo però il costo inferiore e il metodo di comunicazione più semplice hanno come lato negativo una qualità inferiore che comporta una perdita di dati non indifferente nella trasmissione. Svolgendo i primi test con questo modulo infatti capitava spesso che, una volta smesso di accelerare tramite il joystick, ci volessero diversi secondi prima che il ricevitore rilevasse la variazione del segnale e le ruote del motore cominciassero a decelerare.

Questo problema è stato risolto investendo nel modulo nRF24l01 3.3, leggermente più costoso e più complesso da connettere ad Arduino ma molto più affidabile e versatile ad altre possibili applicazioni future in quanto, un singolo modulo, può essere utilizzato sia come trasmettitore che come ricevitore ed un singolo trasmettitore può comunicare con più ricevitori contemporaneamente.

5.2.2 Arduino extra per il ricevitore

Inizialmente il progetto prevedeva l'utilizzo, sullo skateboard, di due schede Arduino, una per ogni motore, che condividessero il collegamento con il ricevitore radio che controllasse la velocità dei motori. Studiando però le librerie utilizzate dai moduli di trasmissione radio è apparso evidente che questo non sarebbe stato possibile. Per inviare o leggere un segnale infatti, questi moduli utilizzano uno dei timer di Arduino e, tramite la modifica di alcuni parametri all'interno delle librerie, si può scegliere se utilizzare Timer1 o Timer2.

Per utilizzare i moduli radio quindi si deve rinunciare all'utilizzo di uno dei due timer riducendo ulteriormente la quantità di uscite PWM utilizzate, già limitate a quattro come visto in 5.1. Per ovviare a questo problema si è aggiunto un altro Arduino con il solo scopo di ricevere il segnale radio ed elaborarlo prima di inoltrarlo agli altri due Arduini di controllo dei motori, questa soluzione ha anche permesso di poter controllare il verso di rotazione dei motori senza dover utilizzare due programmi diversi per i processori di controllo.

5.3 Accensione separata per Arduino

Nel progetto iniziale gli Arduini dello skateboard, che non posso essere alimentati direttamente con i 24V delle batterie, dovevano essere alimentati, tramite due appositi pin di uscita presenti nel circuito stampato, con la tensione ottenuta tramite il regolatore di tensione 3.11.

In uno dei primi test però si è verificato un problema molto particolare, non appena si è chiuso il circuito di alimentazione delle schede tramite l'apposito interruttore uno dei motori ha provato a ruotare e la sua scheda a circuito stampato ha emesso del fumo. Avendo il sistema funzionato fino a poco prima e non essendoci cortocircuiti presenti, l'unica causa possibile del guasto era che, al momento della riaccensione, i processori non erano ripartiti subito dal setup ma avevano, per una frazione di secondo, cercato di riprendere lo svolgimento del programma da dove si era interrotto quando si è tolta l'alimentazione al circuito dopo il test precedente.

Per risolvere il problema si è cambiato il sistema di alimentazione dei processori, si è deciso di aggiungere un interruttore che li connettesse direttamente alle batterie tramite l'utilizzo di un L7805 3.12 che regola la tensione al valore di 5V sufficiente per la loro alimentazione. Da quando questa soluzione è stata adottata, accendendo i processori prima di alimentare le schede di controllo, non si sono più verificati guasti di questo tipo ma, per eliminare qualsiasi rischio, nei test successivi si sono utilizzati dei fusibili così da proteggere le schede ed i loro componenti da correnti troppo elevate.

5.4 Millefori o circuito stampato

Per la realizzazione del circuito di controllo 3.15 si era inizialmente pensato di realizzarlo su una scheda millefori, questo tipo di schede dal costo contenuto consentono di realizzare qualsiasi circuito desiderato ma le connessioni tra i componenti del circuito devono essere realizzate manualmente tramite piccoli cavi isolati il che richiede molto tempo ed una buona precisione.

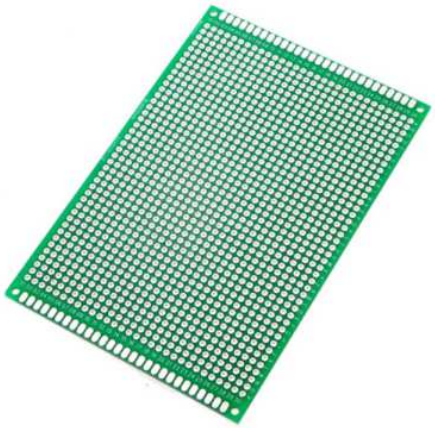


Figura 5.1: Scheda millefori

Inoltre in questo tipo di schede è facile che due stagnature vicine non siano perfettamente isolate tra di loro il che comporta un rischio di presenza di collegamenti indesiderati, per ovviare a ciò si potrebbero eseguire stagnature più distanti ma in questo tipo di progetto le dimensioni sono un parametro importante. Dopo aver effettuato una prova di circuito su una scheda millefori e aver attestato che, per poter eseguire tutte le stagnature a distanza di sicurezza sarebbe stata necessaria una scheda troppo grande si è deciso di passare a una scheda a circuito stampato.

Una scheda a circuito stampato si compone di un substrato solido, piano e di spessore costante costituito da materiali isolanti e da un laminato di rame fatto aderire ad entrambe le facce. La piastra così ottenuta viene forata per consentire il futuro passaggio dei terminali passanti dei componenti elettronici e soprattutto per realizzare il collegamento elettrico tra i piani superiori ed inferiori. Per ricavare dalla superficie in rame la rete di collegamenti necessari, si esegue l'asportazione chimica selettiva del rame in eccesso. Questo avviene mediante deposizione di fotoresist, esposizione fotografica, sviluppo e attacco acido.

Per eseguire tutti questi complessi passaggi di produzione della scheda si è dovuti ricorrere ad una azienda esterna alla quale sono stati inviati i disegni progettuali dei collegamenti elettrici desiderati. Il prezzo della realizzazione di schede a circuito stampato è molto più elevato di quello di una scheda millefori ma, in un progetto con un circuito così complicato, con correnti relativamente elevate e con la necessità di avere dimensioni contenute è l'opzione migliore.

Capitolo 6

Conclusioni

Questo capitolo esamina le conclusioni tratte dopo la realizzazione di questo progetto, si discute quindi delle nuove conoscenze acquisite e vecchie conoscenze consolidate nella realizzazione di questo skateboard elettrico, si valuteranno inoltre i miglioramenti che si potrebbero in futuro implementare a quest'ultimo.

6.1 Nuove conoscenze acquisite

Per realizzare questo progetto e la tesi che ne consegue, sono stati utilizzati diversi strumenti e conoscenze. Alcuni di questi erano già conosciuti ma in questa esperienza sono stati approfonditi, altri invece erano completamente sconosciuti. Tra le nuove conoscenze acquisite va sicuramente inserito il "linguaggio" LaTeXusato per la scritture di questo documento, questo modo innovativo di scrivere testi consente di avere il controllo totale di ogni elemento del testo, dalla formattazione alla posizione di ogni paragrafo oltre a fornire sistemi molto più semplici rispetto ad altri editor per il posizionamento di immagini, la scrittura di formule matematiche e la rappresentazione di grafici e tabelle.

Restando nell'ambito della programmazione, nello svolgere questo progetto si sono approfondite le conoscenze dell'IDE di Arduino, già utilizzato in passato ma per applicazioni molto più semplici, in questo caso invece è stato necessario approfondire tutte le potenzialità di questo processore studiandone le caratteristiche dal datasheet. In particolare per il programma Controllo Motore 4.3 dove si è ricorsi all'utilizzo di interrupt e ad un controllo più approfondito del modulo ADC e della modulazione PWM.

A livello tecnico invece si sono rinsaldate conoscenze già acquisite, per esempio la stagnatura o l'utilizzo di mezzi di misura per verificare il corretto funzionamento del circuito di controllo.

Tramite questa esperienza si sono anche approfondite le conoscenze sul funzionamento del motore brushless e sulle tecniche di controllo di quest'ultimo che non sempre sono semplici da implementare.

6.2 Possibili miglioramenti futuri

6.2.1 Implementazione curva assistita e retromarcia

Nel progetto viene utilizzato un potenziometro joystick che ha la possibilità di rilevare i cambiamenti di posizione positivi e negativi sia sull'asse X che sull'asse Y. Il joystick viene però utilizzato solo ad un quarto delle sue possibilità in quanto, per la marcia dello skateboard, è sufficiente considerare le variazioni sulla metà positiva dell'asse Y. Grazie al fatto che viene usato un Arduino "ricevitore" che controlla poi i due Arduini di controllo, potrebbe essere possibile implementare un sistema che invece tiene conto delle variazioni in ogni direzione permettendo la manovra di retromarcia quando il joystick si sposta su valori negativi dell'asse Y e fungendo da differenziale quando il joystick si sposta lungo l'asse X.

6.2.2 Sistema di frenata

A differenza di molti altri sistemi di mobilità elettrica, questo skateboard non è fornito di sistemi di frenata, il che può essere pericoloso. Si potrebbe in futuro implementare un sistema di frenata meccanica sulle ruote motrici; la frenata "elettrica" rigenerativa non è consigliabile nel sistema realizzato in quanto, le batterie LiPo 3.14 utilizzate, per quanto piccole e leggere, necessitano di un sistema di ricarica equilibrata complesso da realizzare ed in generale come sistema di frenata non è molto efficiente.

6.2.3 Sensore di velocità

Con le regolamentazioni sui mezzi elettrici di trasporto personale sempre più rigide, sarebbe stato interessante inserire un sensore di velocità che, connesso all'Arduino "ricevitore" potesse essere utilizzato per sviluppare un software che, una volta raggiunta una velocità massima, variasse il segnale trasmesso agli Arduini di controllo per far sì che lo skateboard smettesse di accelerare e mantenesse una velocità costante fin tanto che non si smettesse di accelerare tramite il joystick.

Capitolo 7

Riferimenti bibliografici

Benvenuti N, Zorzi M., *Principle of Communications Networks and Systems*, Chichester (UK), John Wiley & Sons, 2011.

Sitografia

https://it.emcelettronica.com/controllo-di-motori-brushless

https://it.emcelettronica.com/controllo-motori-brushless

https://www.arduino.cc/

https://it.wikipedia.org/wiki/Circuito_stampato

https://www.newsauto.it/guide/motore-auto-elettrica-2020-251237/

https://www.elprocus.com/brushless-dc-motor-advantages-applications-and-control/

https://www.lombardoandrea.com/arduino-nrf24l01-comunicazione-base/

https://www.automate.org/blogs/types-of-electric-motors