POLITECNICO DI TORINO
Corso di Laurea in Ingegneria Meccanica

Tesi di Laurea Magistrale

Strategie di controllo per rover modulare

Relatore
prof. Giuseppe Quaglia

Co-relatori
ing. Andrea Botta
ing. Paride Cavallone
ing. Luca Carbonari

Candidato
Simone Pantanetti

Aprile 2022

Sommario

La famiglia di robot mobili *Epi.q* è stata sviluppata con lo scopo ultimo di una guida autonoma, e per raggiungere tale risultato è necessario che alla base ci sia un framework su cui implementare la sensoristica e il controllo ad alto livello.

La seguente tesi di laurea si pone il compito di sviluppare tale framework, progettando le componenti hardware necessarie e implementando la logica di controllo.

L'interfaccia creata consente quindi di inviare al robot comandi semplici e puntuali che verranno eseguiti dal controllo a basso livello. Tali comandi possono essere generati indistintamente da un computer, il quale può anche essere montato a bordo del robot, o da un controller manuale.

La logica di controllo è stata implementata in C++ mediante l'utilizzo della programmazione a oggetti al fine di facilitare lo sviluppo, la comprensione e il mantenimento del codice sorgente.

Le strategie di controllo implementate prevedono tra l'altro l'utilizzo di un controllore fuzzy. Tale scelta ha permesso di ottenere un sistema che lavora attivamente per mantenere l'assetto del robot durante il moto.

Ringraziamenti

Sono stati due anni abbastanza complicati, però devo dire che li ho vissuti bene. Per questo motivo vorrei ringraziare tutte le persone che sono state con me in questo periodo, non sarebbe stata la stessa cosa senza di loro.

Vorrei ringraziare gli amici del mio paese perché anche se i contatti sono stati pochi siamo comunque rimasti legati.

E infine, ma non per importanza, vorrei ringraziare la mia famiglia. Senza di loro non sarei qui oggi.

Questa volta sono stato un po' sintetico, ma non è quello che conta.

Indice

Elenco delle tabelle

Elenco delle figure

Elenco dei codici

Capitolo 1

Introduzione

Un robot è un'apparecchiatura artificiale che compie determinate azioni in base ai comandi che gli vengono dati e alle sue funzioni, sia in base ad una supervisione diretta dell'uomo, sia autonomamente basandosi su linee guida generali. Questi compiti tipicamente dovrebbero essere eseguiti al fine di sostituire o coadiuvare l'uomo, come ad esempio nella fabbricazione, costruzione, manipolazione di materiali pesanti e pericolosi, o in ambienti proibitivi o non compatibili con la condizione umana o semplicemente per liberare l'uomo da impegni.

Un robot così definito dovrebbe essere dotato di connessioni guidate dalla retroazione tra percezione e azione, e non dal controllo umano diretto. L'azione può prendere la forma di motori elettromagnetici, o attuatori, che muovono un arto, aprono e chiudono una pinza, o fanno deambulare il robot. Il controllo passo-passo e la retroazione sono forniti da un programma che viene eseguito da un computer esterno o interno al robot, o da un microcontrollore. In base a questa definizione, il concetto di robot può comprendere quasi tutti gli apparati automatizzati.

Le discipline coinvolte nella progettazione e realizzazione dei robot sono molteplici: robotica, cibernetica, meccanica, automatica, elettronica, meccatronica, informatica, intelligenza artificiale, ecc.

I robot utilizzati sono di fatto dei sistemi ibridi complessi costituiti da vari sottosistemi quali computer (es. microcontrollori), ovvero da una parte hardware elettronica opportunamente programmata tramite software che regola o controlla una parte meccanica costituita da servomeccanismi per l'esecuzione dei compiti meccanici desiderati. Esistono moltissime tipologie di Robot differenti sviluppate per assolvere i compiti più disparati. Ormai è larghissimo l'impiego dei robot nell'industria metalmeccanica (es. catene di montaggio) e non solo. Si possono catalogare i robot in due macro categorie: "autonomi" e "non autonomi"[1].

robot non autonomi I robot "non autonomi" sono i classici robot utilizzati per adempiere a specifici compiti che riescono ad assolvere in maniera più efficace dell'uomo. Alcuni esempi sono i robot utilizzati nelle fabbriche (robot industriali), i quali hanno l'enorme vantaggio di poter ottenere una produzione più precisa, veloce e a costi ridotti senza o con scarso utilizzo di manodopera umana; oppure i robot utilizzati per lavorare in ambienti ostili (ad esempio su Marte) o con sostanze tossiche. Questi robot sono detti "non autonomi" poiché guidati da un software deterministico che fa

eseguire loro il lavoro in modo ripetitivo; o in alternativa sono direttamente pilotati dall'uomo.

robot autonomi I robot "autonomi" sono invece caratterizzati dal fatto che operano in totale autonomia e indipendenza dall'intervento umano e sono in grado di prendere decisioni anche a fronte di eventi inaspettati. Questi robot sono programmati solitamente con algoritmi che si rifanno a tecniche di intelligenza artificiale: algoritmi genetici, logica fuzzy, apprendimento automatico, reti neurali. I robot autonomi sono adatti a svolgere compiti in ambienti non noti a priori: tipicamente si tratta di robot mobili. Alcuni piccoli robot autonomi vengono utilizzati per il taglio dell'erba nei giardini e nelle pulizie domestiche: essi autonomamente decidono quando partire, dove tagliare/pulire e quando tornare alla base per ricaricarsi.

1.1 Robotica mobile

Viene definito robot mobile una macchina in grado di muoversi nell'ambiente che la circonda, distinguendosi così dalla robotica fissa utilizzata per lo svolgimento di compiti in loco. Ad oggi esistono robot capaci di muoversi ed operare in ogni ambiente. In questa tesi ci occuperemo dei robot mobili terrestri.

Importanti applicazioni per questa ultima tipologia di robot si hanno nel campo della sicurezza, ad esempio per missioni di ricognizione e sorveglianza, in ambito militare o in generale per situazioni pericolose per l'uomo, quali possono essere l'esplorazione e l'intervento in zone o ambienti potenzialmente nocivi. Per poter operare efficacemente negli ambienti più differenti sono state studiate varie modalità di locomozione, ognuna con i suoi pregi e difetti.

Esistono tre categorie cosiddette *pure* di locomozione:

- **Wheeled W** (su ruote);
- **Tracked T** (su cingoli);
- **Legged L** (con arti più o meno articolati);

La locomozione su ruote (categoria **W**) è la soluzione più semplice e più efficiente. I robot dotati di tale sistema di trazione possono raggiungere le velocità più elevate su superfici piane e richiedono un sistema più semplice di controllo rispetto alle altre due categorie, ma a ciò si contrappone spesso una limitata capacità di superamento ostacoli e quindi risultano poco adattabili ai vari tipi di terreni in cui potrebbero trovarsi ad agire.

La presenza di cingoli (categoria **T**) permette al robot, grazie alla maggiore superficie di contatto, una maggiore presa, un miglior movimento in terreni irregolari ed una discreta capacità di superamento ostacoli rispetto ai robot su ruote. Tutto ciò si traduce però in una maggiore resistenza al moto da cui derivano maggiori consumi di energia ed una limitata velocità massima.

La soluzione con arti (categoria **L**) è infine quella che permette al robot la massima mobilità anche in ambienti molto irregolari. Ci sono però elevati dispendi di energia in funzione del tipo di terreno e della complessità dell'architettura degli arti. Il controllo dei vari attuatori può risultare inoltre molto complesso.

(a) *Tracked*

(b) *Wheeled*

(c) *Legged*

Figura 1.1: Categorie *pure* di robot mobili

In figura 1.1 sono mostrate le tipologie sopra citate.

Tali strategie di moto possono essere unite definendo così le categorie di locomozione *ibride* [2] (figura 1.2), studiate per rispondere alle più svariate esigenze di movimento e di efficienza tramite la combinazione dei pregi delle categorie pure da cui derivano. Si hanno quindi le seguenti categorie:

- Wheels - Tracks **WT**;
- Legs - Wheels **LW**;
- Legs - Tracks **LT**;

La classe Legs - Wheels (**LW**), a cui la famiglia dei robot mobili *Epi.q* appartiene, riesce ad unire l'efficienza energetica e la velocità di moto della categoria su ruote (**W**) con la flessibilità e la maggiore capacità di superamento ostacoli permessa dall'utilizzo di arti (**L**) più o meno complessi.

Un interessante esempio di robot **LW** è riportato in figura 1.3b, tale unità è la seconda versione del robot ibrido *Mantis* [3], concepito per scopi di sorveglianza. Dispone di due ruote frontali motrici e di un paio di arti rotanti: la caratteristica che contraddistingue

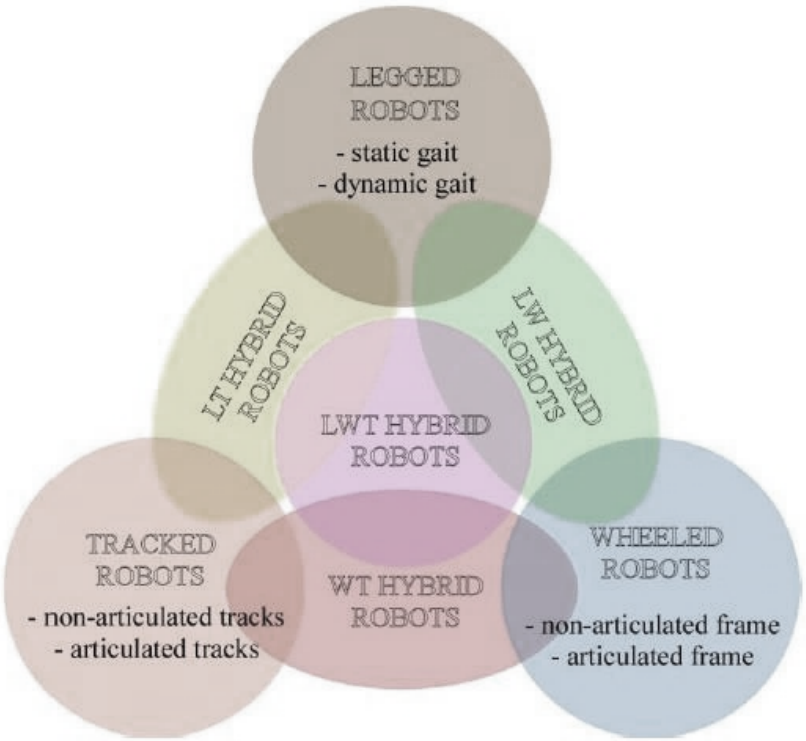


Figura 1.2: Categorie di robot mobili su terra

questa unità. Il moto su superficie piana avviene puramente su ruote mentre gli arti sono chiamati in causa in presenza di un ostacolo da scavalcare.

Robot mobili forniti di arti e cingoli (**LT**) sono la categoria che permette di ottenere le migliori prestazioni su terreni sconnessi, dove si presentano le maggiori difficoltà di movimento. Di contro peccano in efficienza energetica e in velocità, come le due categorie pure da cui derivano. In figura 1.3a è mostrato un esempio.

La combinazione di ruote e cingoli (**WT**) risulta infine la migliore per terreni non eccessivamente irregolari, ad esempio per gli spazi aperti, in quanto si raggiungono buoni livelli di efficienza energetica, velocità.

Il *Daegu Gyeongbuk Institute of Science & Technology* ha sviluppato un robot in questa direzione (figura 1.3c) [4]. L'ostacolo viene affrontato grazie ai cingoli, mentre la possibilità di ripiegarli permette il passaggio al moto su ruote per migliori prestazioni su superfici piane.

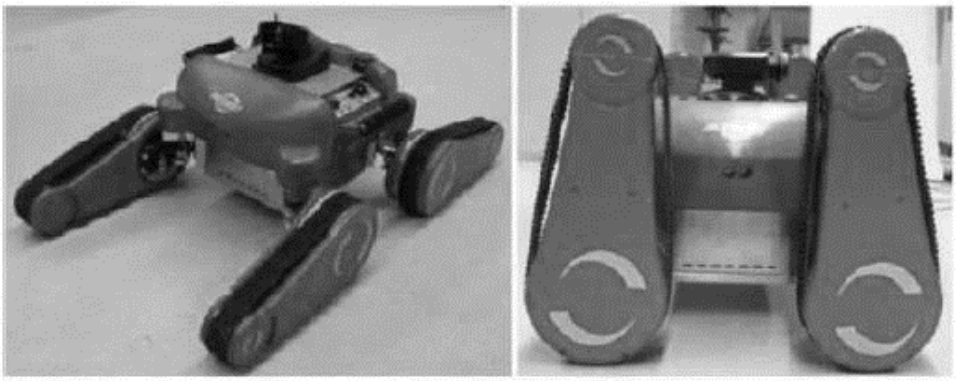
Quando le tre categorie pure sono unite insieme si definisce la soluzione **LWT** la quale permette di affrontare le più svariate condizioni del terreno garantendo alternativamente elevata adattabilità e velocità. Le sue caratteristiche possono portare ad un eccessivo consumo energetico a causa della massa dei vari accessori e gruppi locomotori di cui è costituita l'unità, ed a causa dell'elevata complessità meccanica, la quale può inoltre rendere difficoltosa anche la manutenzione o una variazione dell'architettura.

Azimuth [5] (figura 1.3d) è un buon esempio di un mezzo che riesce ad unire tutte e tre le

(a) *LT*

(b) *LW*

(c) *WT*



(d) *LWT*

Figura 1.3: Categorie *ibride* di robot mobili

tecnologie introdotte: l'unità è costituita da quattro articolazioni leg-track-wheel completamente indipendenti tra loro che possono fornire le più svariate possibilità di superamento ostacoli e movimentazione.

In fase di scelta dell'unità robot migliore, in base alle proprie esigenze, è bene porre particolare attenzione alla complessità meccanica, in quanto ha un peso significativo sull'affidabilità e potrebbe causare lunghi periodi di fermo per eventuali interventi di riparazione in caso di guasto. Anche semplici interventi di modifica ad organi attuatori o di disposizione dei vari gruppi costituenti potrebbero causare lunghi fermi macchina.

Non esiste una strategia migliore in assoluto per la locomozione, ma è scelta di volta in volta la classe da utilizzare in base alle priorità ed agli obiettivi di mobilità ed efficienza.

In figura 1.4 è mostrato un quadro riassuntivo con pregi e difetti delle varie tecnologie introdotte.

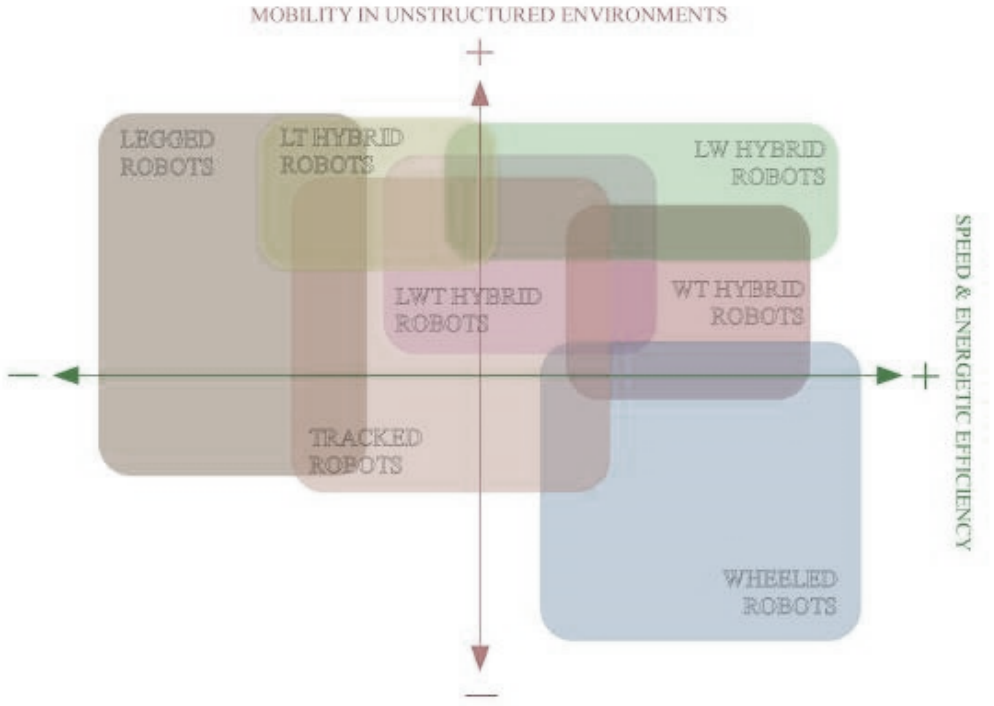


Figura 1.4: Confronto tra le strategie di locomozione in termini di velocità, efficienza energetica e mobilità

1.2 Introduzione alla famiglia *Epi.q*

All'interno della categoria ibrida Legs-Wheels (**LW**) si colloca la famiglia delle unità mobili UGV (Unmanned Ground Vehicle) *Epi.q*, sviluppata da una collaborazione del Politecnico di Torino con l'Università di Genova, che ad oggi conta già diversi prototipi.

I robot *Epi.q* sono progettati per operazioni di sorveglianza, ricognizione, trasporto e in particolare per l'utilizzo in zone pericolose o inaccessibili per l'uomo.

I principali pregi sono l'elevato livello di mobilità e l'efficienza energetica, che derivano dalle classi pure da cui discende la famiglia.

La caratteristica peculiare che contraddistingue tale famiglia, all'interno della classe, è l'unità tripode di locomozione. Essa è basata su un meccanismo epicicloidale che permette al robot il passaggio dal moto su ruote a quello su arti in modo rapido e soprattutto passivo, in funzione delle caratteristiche del terreno e in particolare in funzione della presenza o meno di ostacoli [6].

Tra i prototipi di Epi.q costruiti si hanno:

Epi.q-1 Progettato e costruito dal Dipartimento di Meccanica del *Politecnico di Torino*, è capace di variare la geometria dell'unità tripode da una configurazione chiusa, più compatta, ad una più aperta, migliorando così la sua capacità di superamento ostacoli (figure 1.5a e 1.5b) [7].

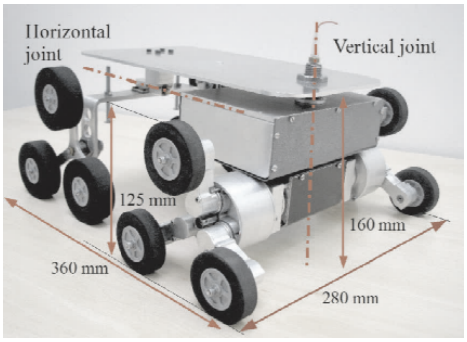
Epi.q-2 Prototipo costruito in collaborazione con l'*Università di Genova*. L'unità di locomozione è stata dotata di un meccanismo epicicloidale più compatto e semplice del modello precedente ed è stata rimossa la possibilità di variare l'apertura dell'unità tripode in quanto si è concluso che i benefici che portava non valevano la complessità meccanica [8].

Epi.q-TG FWD e Epi.q–TG AWD I nomi stanno rispettivamente per *Front Wheel Drive* e *All Wheel Drive*. Essi sono il secondo e il terzo prototipo sviluppati in collaborazione con l'*Università di Genova*, sono caratterizzati da un miglioramento della meccanica generale per ottenere maggiori prestazioni e affidabilità. Il modello **AWD** è inoltre dotato di quattro unità motrici che ne hanno ulteriormente aumentato la capacità di superare ostacoli e l'efficienza di movimento in terreni sconnessi (figure 1.5c e 1.5d) [7].

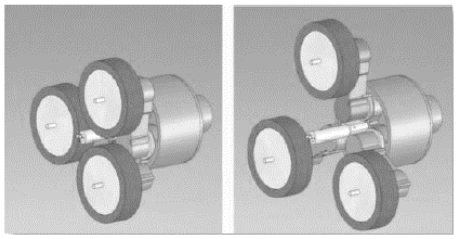
Epi.q-Lizard Modello composto da due telai, collegati tramite un giunto passivo rotoidale che ne permette la rotazione attorno all'asse longitudinale. Con questo prototipo viene introdotto un approccio modulare nella progettazione del telaio. In ogni modulo sono inseriti due motori. La struttura conferisce al robot una grande adattabilità alla conformazione del terreno (figura 1.5e) [9].

Epi.q-Mod1 Con questa versione si applica un approccio modulare più ampio a livello del singolo modulo di trazione. Tale approccio consente quindi di assemblare a piacimento più moduli, anche differenti tra loro, e generare facilmente alternative, permettendone quindi un veloce confronto per cercare di rispondere ai più svariati requisiti funzionali. Come per la versione Epi.q-2 è stata rimossa la possibilità di variare l'apertura del tripode (figura 1.5f) [10].

Epi.q-Mod2 Il modello preso in esame in questa tesi. Composto da due moduli collegati insieme da un giunto rotoidale a due gradi di libertà. Rispetto alla versione precedente si è ampliato lo spazio dedicato al contenimento della componentistica interna (figura 1.6).



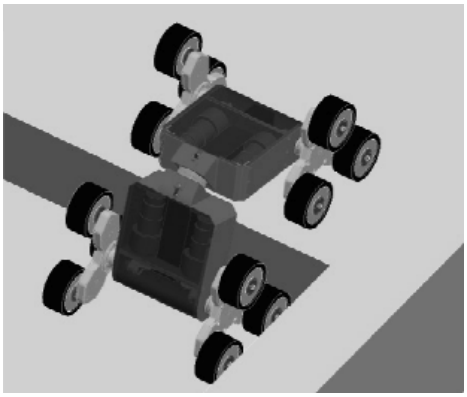
(a) *Epi.q1*



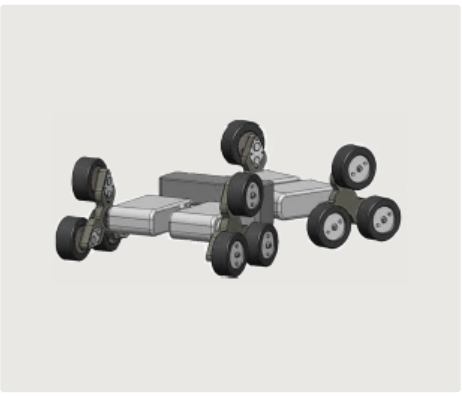
(b) *Tripode ad apertura variabile*

(c) *Epi.q-TG FWD*

(d) *Epi.q-TG AWD*



(e) *Epi.q-Lizard*



(f) *Epi.q-Mod 1*

Figura 1.5: Prototipi *Epi.q*

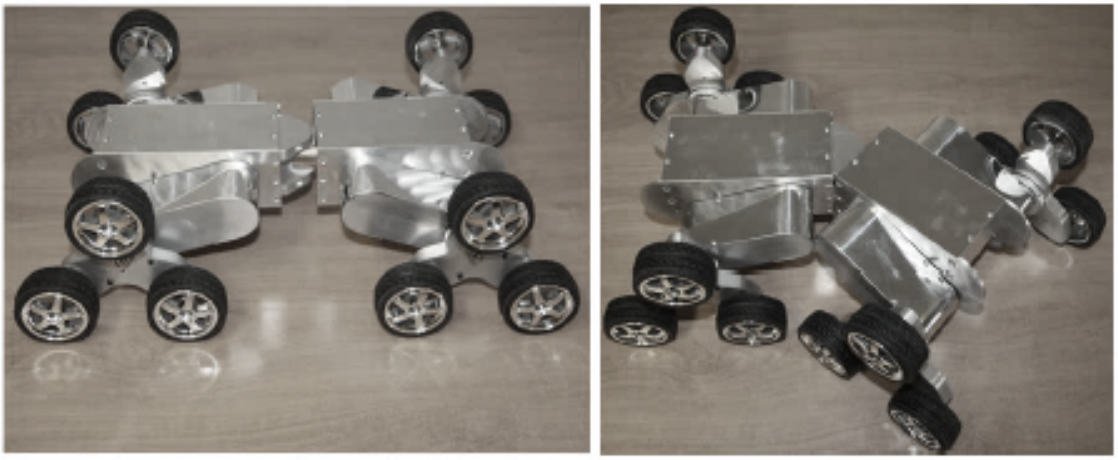


Figura 1.6: Prototipo Epi.q-Mod2

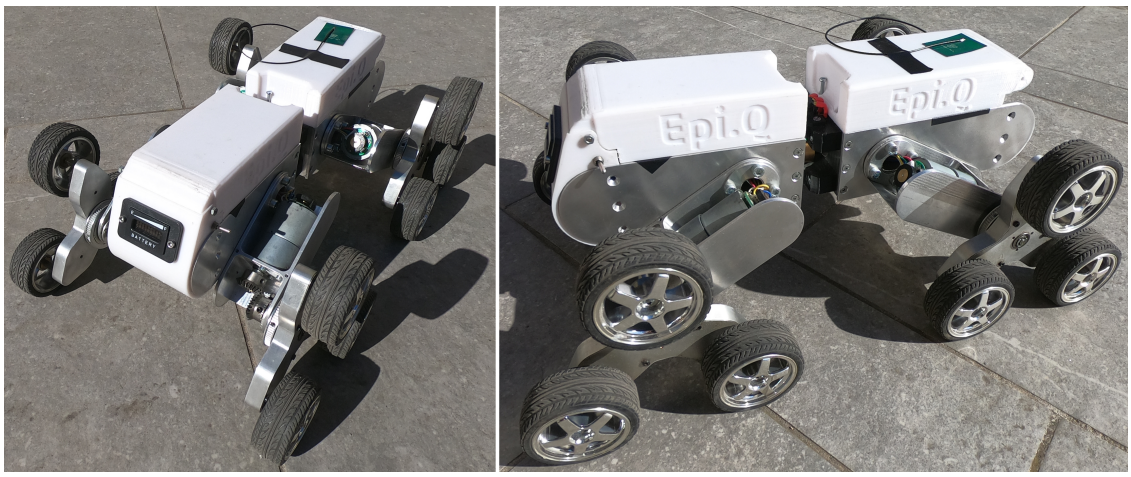


Figura 1.7: Prototipo Epi.q-Mod2 nello stato attuale

Capitolo 2

Struttura meccanica del robot Epi.q-Mod2

Il robot Epi.q-Mod2, da qui in poi chiamato semplicemente Epi.q (figura 2.1), è costituito da due telai, o moduli, contenenti la componentistica elettronica di controllo e la batteria di alimentazione. I due moduli sono collegati tra loro mediante un doppio giunto rotoidale, dentro il quale è allocato parte del cablaggio.

Tale giunto consente la rotazione nel piano orizzontale e nel piano verticale la cui normale è parallela alla direzione di avanzamento. In definitiva il giunto consente l'imbardata e il rollio, ma non consente il beccheggio (figure 2.2 e 2.3).

Epi.q è dotato di quattro unità tripodi, ognuna azionata da un motore indipendente. I tripodi sono collegati ai due telai mediante dei braccetti che hanno anche la funzione di alloggiare i motori, i quali trasferiscono potenza ai primi mediante ruote coniche.

Per come è strutturato, il robot può lavorare sia in modalità **4WD** che **8WD**[1], ed è proprio quest'ultima che viene presa principalmente in esame in questa tesi.

2.1 Unità di locomozione a tripode

Caratteristica peculiare delle unità *UGV Epi.q* è l'unità tripode di locomozione.

Il sistema di trasmissione è basato su un rotismo epicicloidale (figura 2.4) costituito da un ingranaggio solare **B**, tre satelliti primari **C**, che fungono da ruote oziose, tre satelliti secondari **D** solidali con le ruote **E**, e un portatreno **A**, libero di ruotare rispetto al telaio, in cui sono alloggiati gli altri componenti.

I due gradi di libertà, posseduti dal rotismo epicicloidale, consentono al robot il passaggio dalla modalità di avanzamento (**advancing mode**) alla modalità di superamento ostacoli (**climbing mode**) senza un intervento attivo.

[1] WD sta per wheel drive, ovvero ruote motrici: il robot può quindi funzionare con quattro o con otto ruote motrici.

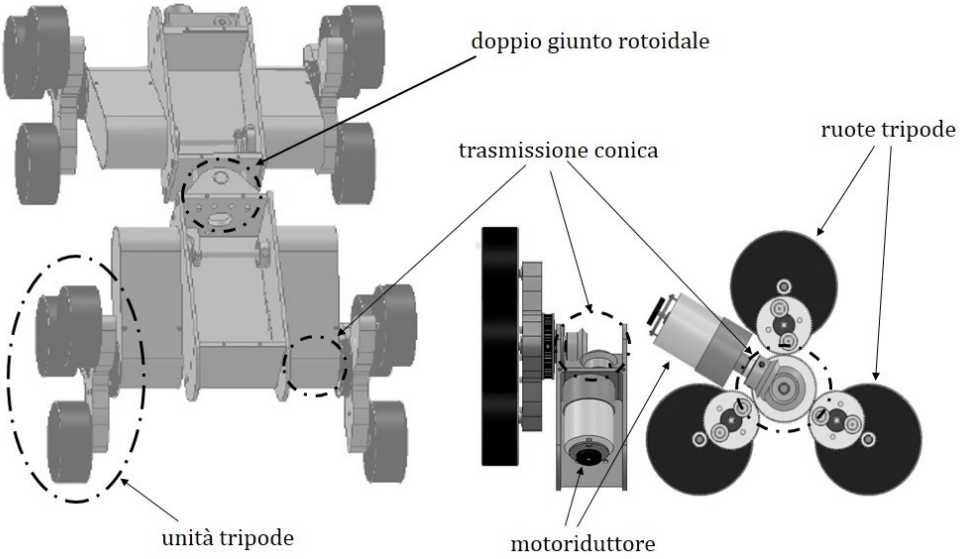


Figura 2.1: Epi.q-Mod2 con dettaglio del tripode e del motoriduttore

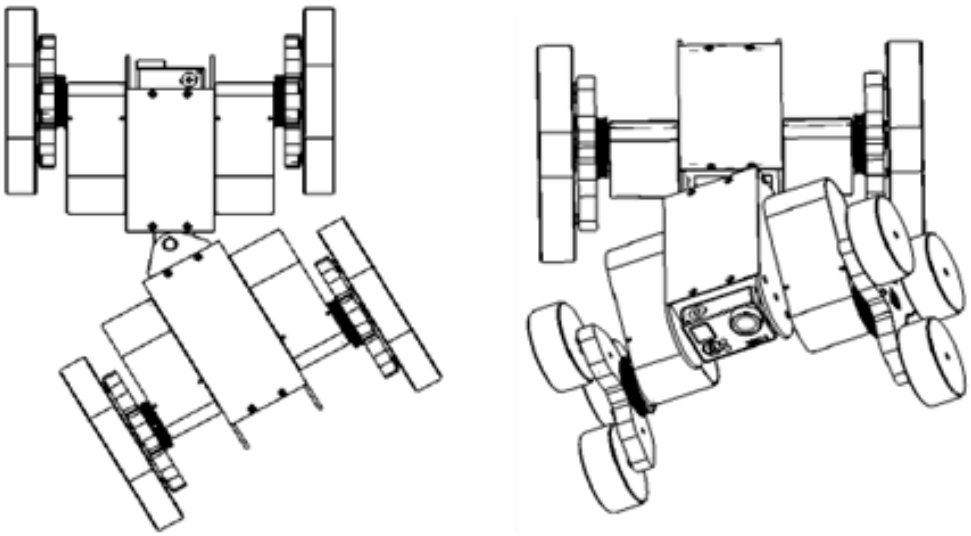


Figura 2.2: Imbardata e rollio del giunto

In funzione della coppia resistente applicata alle ruote e al carico verticale gravante sulle stesse possiamo avere o meno la rotazione del portatreno. Le condizioni limite che consentono la rotazione del portatreno sono due:

1. Basso carico verticale sulle ruote, quindi un'assenza di vincolo per il portatreno;
2. Alta coppia resistente applicata alle ruote, quindi la potenza fluisce tramite la rotazione del portatreno.

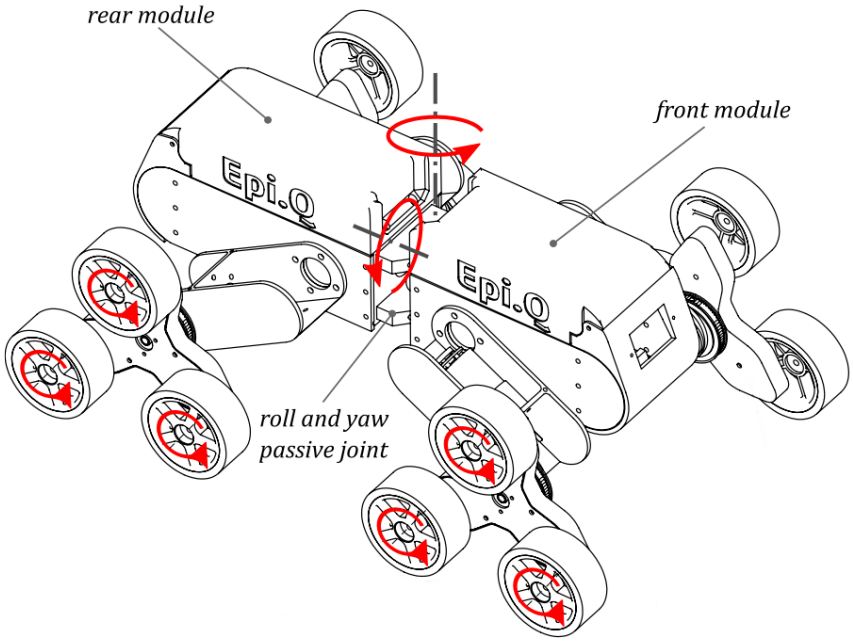


Figura 2.3: Rotazioni del giunto e delle ruote

Quest'ultima condizione incorre quando il robot incontra un ostacolo: la ruota anteriore viene bloccata ed il motore fa ruotare il portatreno in modo tale che l'ostacolo venga superato (figura 2.5).

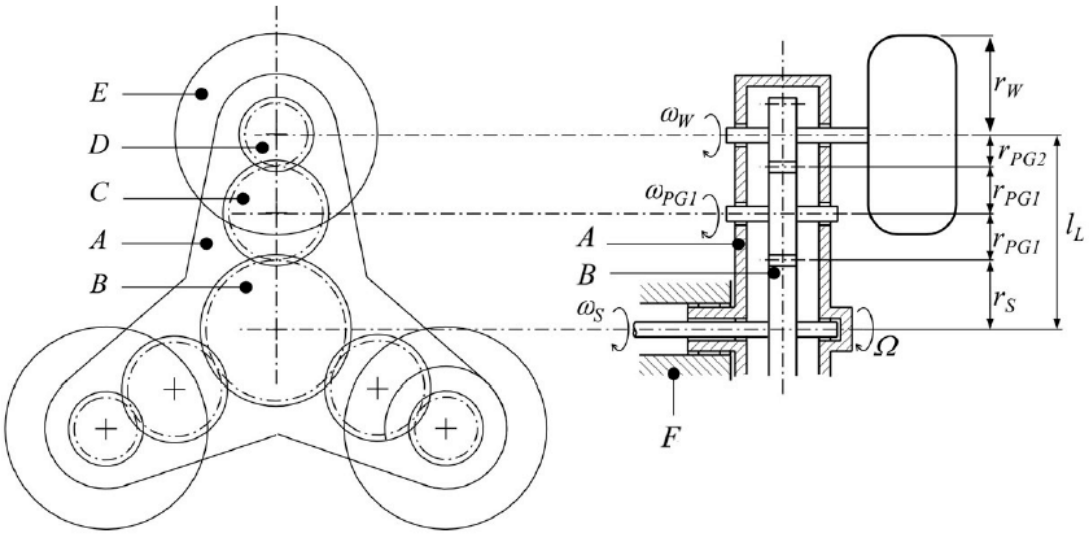


Figura 2.4: Schema funzionale del tripode

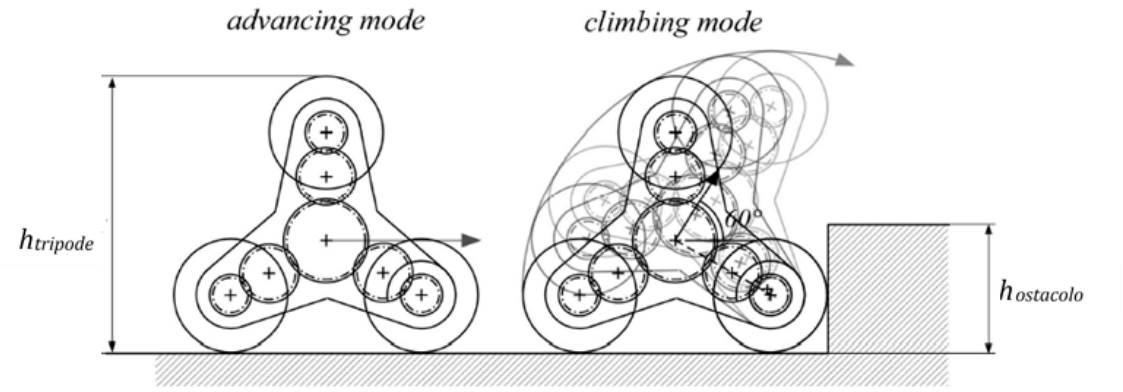


Figura 2.5: Modalità di avanzamento e di superamento ostacoli

Le dimensioni delle ruote (comprese le gomme) e del portatreno sono state opportunamente scelte in modo tale che quest'ultimo ruoti solo in presenza di un ostacolo consistente e non ad ogni minima asperità del terreno, al fine di rendere il movimento più fluido e naturale.

2.1.1 Cinematica del rotismo epicicloidale

La figura 2.6 mostra uno schema basilare di un rotismo epicicloidale in cui troviamo una ruota solare (**1**), una ruota satellite (**2**) e un portatreno (**p**).

Il nostro scopo è mettere in relazione le tre velocità angolari nel sistema di riferimento assoluto: ω_1, ω_2 e ω_p.

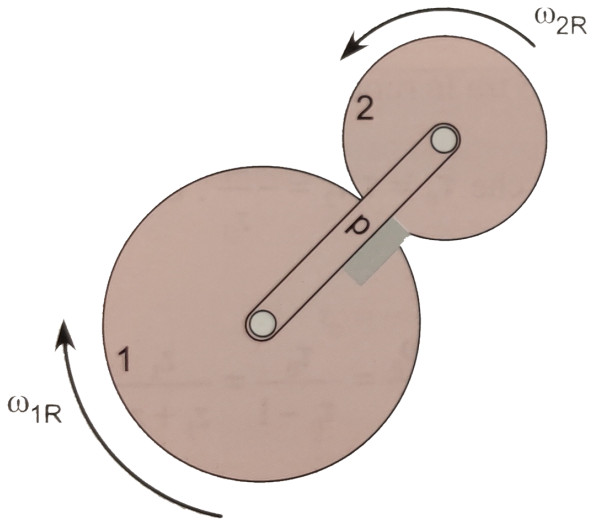


Figura 2.6: Schema rotismo epicicloidale

Al fine di ricavare tali relazioni andiamo ad eseguire un'inversione cinematica e ci posizioniamo solidali al portatreno, il quale sarà fisso nel nuovo sistema di riferimento.

Indicando con ω_{1R} e ω_{2R} le velocità angolari relative, possiamo scrivere:

$$\omega_{1R} = \omega_1 - \omega_p$$
$$\omega_{2R} = \omega_2 - \omega_p$$

in tale sistema di riferimento relativo il rapporto di trasmissione è:

$$\tau_0 = \frac{\omega_{2R}}{\omega_{1R}}$$

da cui avremo:

$$\tau_0 = \frac{\omega_2 - \omega_p}{\omega_1 - \omega_p} \tag{2.1}$$

risolvendo e generalizzando quest'ultima equazione otteniamo:

$$\omega_p = \frac{\tau_0}{\tau_0 - 1}\,\omega_1 + \frac{1}{1 - \tau_0}\,\omega_n \tag{2.2}$$

dove n è il numero che indica l'ultima ruota dell'epicicloidale partendo dal solare.

L'equazione 2.1 è comunemente detta in Italia *formula di Willis* [11].

Nel nostro caso la 2.2 va specializzata a n = 3 (figura 2.7).

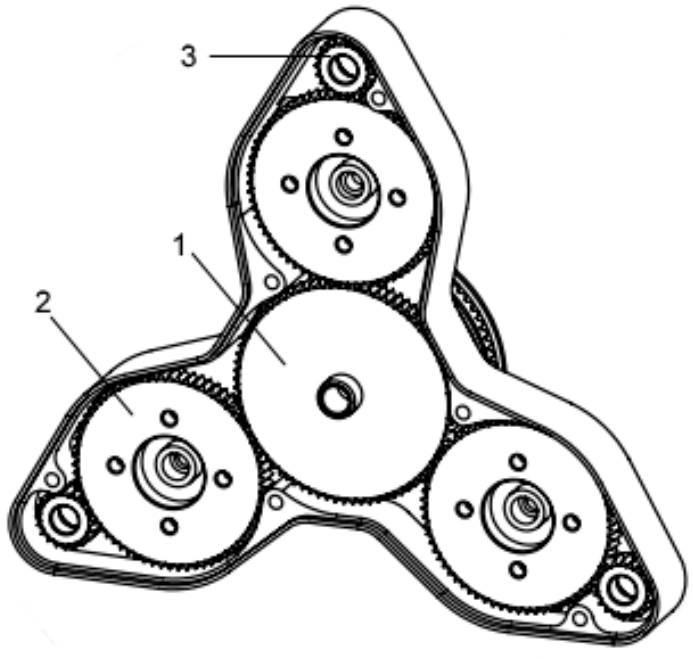


Figura 2.7: Complessivo del tripode con le giuste proporzioni

In *advancing mode* il portatreno non ruota, quindi avremo $\omega_\mathrm{p} = 0$, e di conseguenza la relazione si semplifica in:

$$\omega_3 = \tau_0 \, \omega_1 \tag{2.3}$$

mentre in *climbing mode* ($\omega_3 = 0$) si ha:

$$\omega_\mathrm{p} = \frac{\tau_0}{\tau_0 - 1} \, \omega_1 \tag{2.4}$$

2.1.2 Calcolo dei rapporti di trasmissione

Il rotismo è composto da ruote dentate a denti dritti, i cui dati sono riportati nella tabella seguente:

Tabella 2.1: Dati delle ruote dentate

Ruota	Modulo m (mm)	N° denti z
1	0.5	80
2	0.5	70
3	0.5	20

Essendo la ruota 2 una ruota oziosa, il rapporto di trasmissione τ_0 dipende solo dal numero di denti delle ruote 1 e 3, di conseguenza si ha:

$$\tau_0 = \frac{\omega_{3\mathrm{R}}}{\omega_{1\mathrm{R}}} = \frac{z_1}{z_3} = 4 \tag{2.5}$$

Il τ_0 corrisponde al rapporto di trasmissione che si ha in *advancing mode*, e ciò ci dice che il rotismo così strutturato funge da moltiplicatore di giri.

In *climbing mode* il rapporto di trasmissione diminuisce, così da facilitare il superamento dell'ostacolo:

$$\tau_\mathrm{p} = \frac{\omega_\mathrm{p}}{\omega_1} = \frac{\tau_0}{\tau_0 - 1} = 1.33 \tag{2.6}$$

Il solare ed i satelliti più esterni hanno lo stesso verso di rotazione perché sono presenti le ruote oziose. Il portatreno ha anch'esso lo stesso verso di rotazione del solare perché le ruote sono state dimensionate in modo tale che il τ_0 sia maggiore di 1. In caso contrario avremmo avuto una rotazione di verso opposto, tale per cui il tripode non sarebbe stato in grado di superare gli ostacoli.

Il motore trasmette coppia alla ruota solare mediante due ruote dentate coniche a denti dritti, le quali hanno il solo scopo di variare l'asse di rotazione, in quanto hanno un rapporto di trasmissione pari a 1.

Sulla base del rapporto di trasmissione complessivo, è necessario che il motore elettrico abbia un rapporto di trasmissione interno sufficientemente basso al fine di compensare quello del tripode.

Per questo motivo il robot monta dei motori brushed DC con un rapporto di trasmissione pari a 1/50.

Conoscendo la velocità massima di rotazione dei motori e il raggio delle ruote[2] è possibile calcolare la massima velocità lineare raggiungibile dal robot (vedi 2.3.3).

2.2 Dimensioni del robot

In questa sezione non tratteremo la componentistica elettronica del robot, la quale sarà trattata nel capitolo 4, ma ci occuperemo invece della meccanica.

In figura 2.8 è riportato uno schema con la nomenclatura delle quote, mentre nella tabella 2.2 sono riportati i valori noti[3].

Tabella 2.2: Quote note del robot

l_A (mm)	l_R (mm)	l_F (mm)	l_L (mm)	r_w (mm)	i (mm)	β_R (deg)	β_F (deg)
92.5	58.5	52	60	32	270	30	30

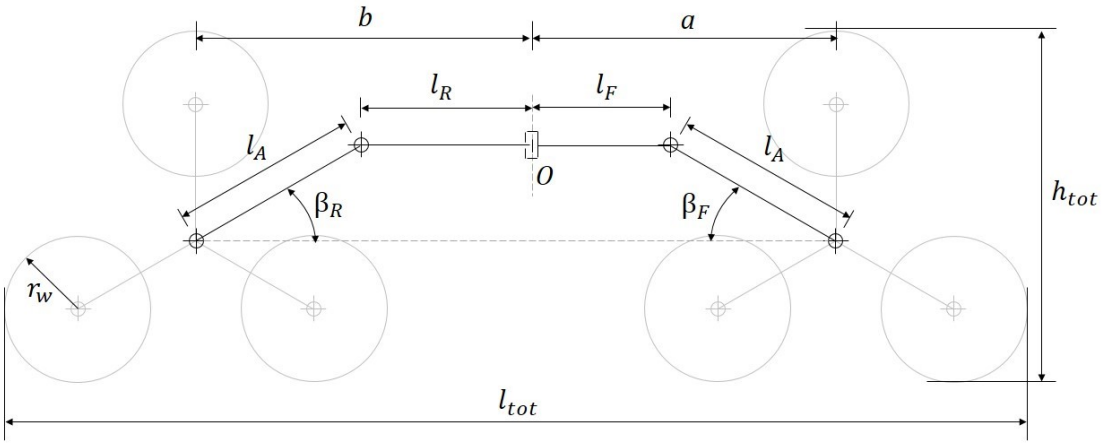


Figura 2.8: Schema laterale del robot

Gli angoli di inclinazione dei braccetti possono essere modificati al fine di cambiare la geometria del robot. Per i nostri calcoli useremo i valori in tabella.

Calcoliamo i valori a e b utili per la successiva analisi cinematica.

[2]Per ruote intendo il cerchio con sopra montato lo pneumatico, anche se in realtà sono ruote giocattolo.

[3]La quota l_L è visibile in figura 2.4, mentre i è visibile in figura 2.9

$$a = l_{\mathrm{A}} \cos \beta_{\mathrm{F}} + l_{\mathrm{F}} \sqrt{1 - \left[\frac{l_{\mathrm{A}}}{l_{\mathrm{F}} + l_{\mathrm{R}}} (\sin \beta_{\mathrm{F}} - \sin \beta_{\mathrm{R}}) \right]^2} \tag{2.7}$$

$$b = l_{\mathrm{A}} \cos \beta_{\mathrm{R}} + l_{R} \sqrt{1 - \left[\frac{l_{\mathrm{A}}}{l_{\mathrm{F}} + l_{\mathrm{R}}} (\sin \beta_{\mathrm{F}} - \sin \beta_{\mathrm{R}}) \right]^2} \tag{2.8}$$

Avendo imposto al robot un'inclinazione dei braccetti simmetrica le due relazioni si semplificano in:

$$a = l_{\mathrm{A}} \cos \beta_{\mathrm{F}} + l_{\mathrm{F}} \simeq 132\,\mathrm{mm} \tag{2.9}$$

$$b = l_{\mathrm{A}} \cos \beta_{\mathrm{R}} + l_{\mathrm{R}} \simeq 139\,\mathrm{mm} \tag{2.10}$$

Gli ingombri del robot sono:

$$h_{\mathrm{tot}} = 2r_{\mathrm{w}} + l_{\mathrm{L}}(1 + \sin \pi/6) = 154\,\mathrm{mm}$$
$$l_{\mathrm{tot}} = a + b + 2r_{\mathrm{w}} + 2l_{\mathrm{L}} \cos \pi/6 \simeq 439\,\mathrm{mm}$$
$$w_{\mathrm{tot}} = 291\,\mathrm{mm}$$

w_{tot} indica la larghezza complessiva del singolo modulo, la cui quota non è visibile in figura 2.8.

2.3 Analisi cinematica

Diversamente dalle classiche auto, Epi.q possiede uno sterzo differenziale, ciò significa che riesce a percorrere una curva imponendo una differenza di velocità tra le ruote interne e le ruote esterne, e non mediante una rotazione delle stesse.

A questo fatto si aggiunge un problema strutturale del robot che lo rende sottosterzante, ovvero la presenza dei tripodi.

Come si vede in figura 2.9 le ruote hanno una componente di velocità che tende a creare uno slittamento verso l'esterno, e questo è tanto maggiore, quanto più piccolo è il raggio della curva che si vuole percorrere.

Per semplificare la trattazione analitica ogni tripode viene sostituito da una ruota virtuale (figura 2.10), la quale non ha componenti di velocità laterale, e di conseguenza nessuno slittamento [12].

2.3.1 Raggio di curvatura

Data la geometria del robot, il raggio di curvatura del modulo frontale ρ (figura 2.11) è funzione delle velocità imposte all'anteriore.

Andiamo ad individuare un angolo fittizio α^* compreso tra la retta passante per il centro di rotazione e il centro delle ruote anteriori, e la retta che congiunge le punte dei vettori velocità con il centro di rotazione.

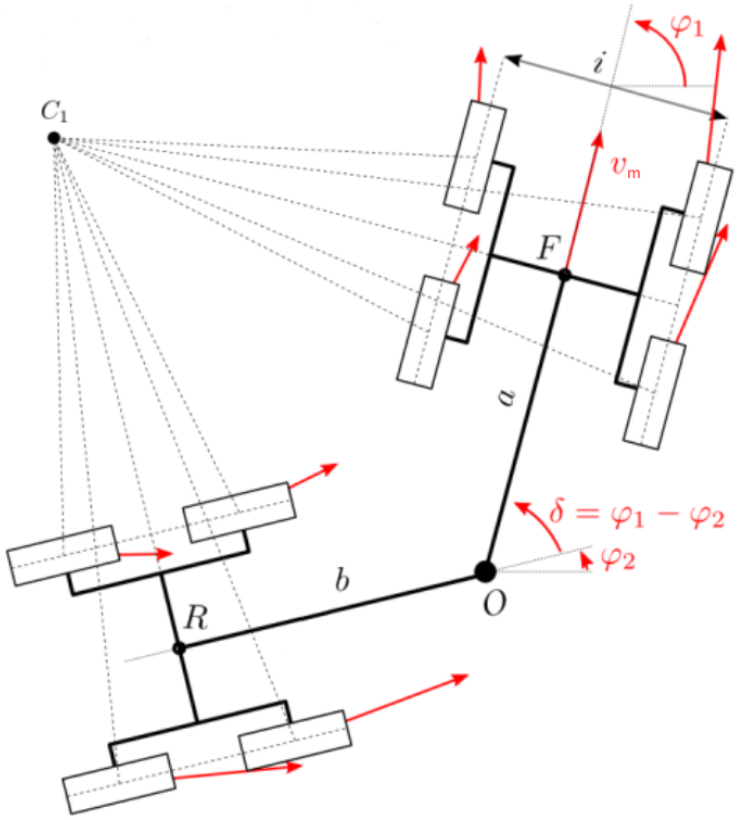


Figura 2.9: Configurazione generica di sterzata

La tangente di questo angolo può essere calcolata in due modi:

$$\tan \alpha^* = \frac{v_m}{\rho}$$
$$\tan \alpha^* = \frac{\Delta v}{i}$$

di conseguenza otteniamo una prima relazione:

$$\rho = \frac{v_m}{\Delta v}\, i \tag{2.11}$$

dove v_m indica la velocità media e Δv la differenza di velocità tra esterno ed interno.

Lo stesso risultato poteva essere ottenuto applicando le proprietà di similitudine dei triangoli.

Andiamo ora a esplicitare una seconda relazione che lega il raggio di curvatura all'angolo di imbardata δ. Tale relazione è valida solo in condizioni stazionarie, in particolare quando

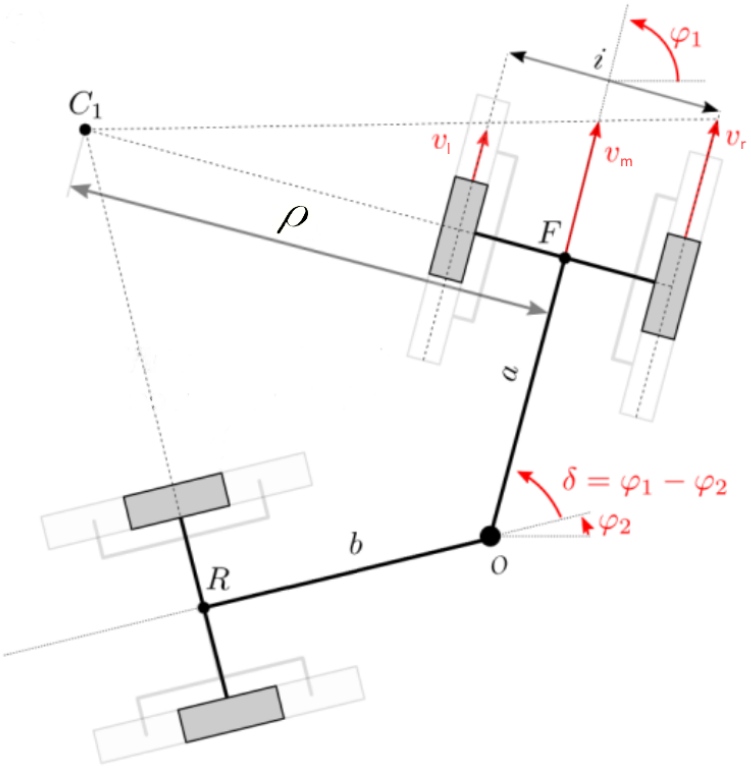


Figura 2.10: Ruote virtuali

δ è costante. Abbiamo che:

$$z = a + b\cos\delta$$
$$t = \frac{z}{\sin\delta}$$
$$\rho = t\cos\delta + b\sin\delta$$

da cui otteniamo:

$$\rho = \frac{a\cos\delta + b}{\sin\delta} \tag{2.12}$$

Eguagliando le relazioni 2.11 e 2.12 otteniamo una terza relazione che lega le velocità all'angolo di imbardata:

$$i\, v_m \sin\delta = \Delta v(a\cos\delta + b) \tag{2.13}$$

Questa relazione è scritta mediante l'utilizzo di velocità lineari, ma nulla ci vieta di scriverla in funzione dei numeri di giri dei motori[4], i quali possono essere facilmente acquisiti mediante l'utilizzo degli encoder presenti. Avremo quindi:

[4]L'importante è rimanere coerenti con le unità di misura scelte.

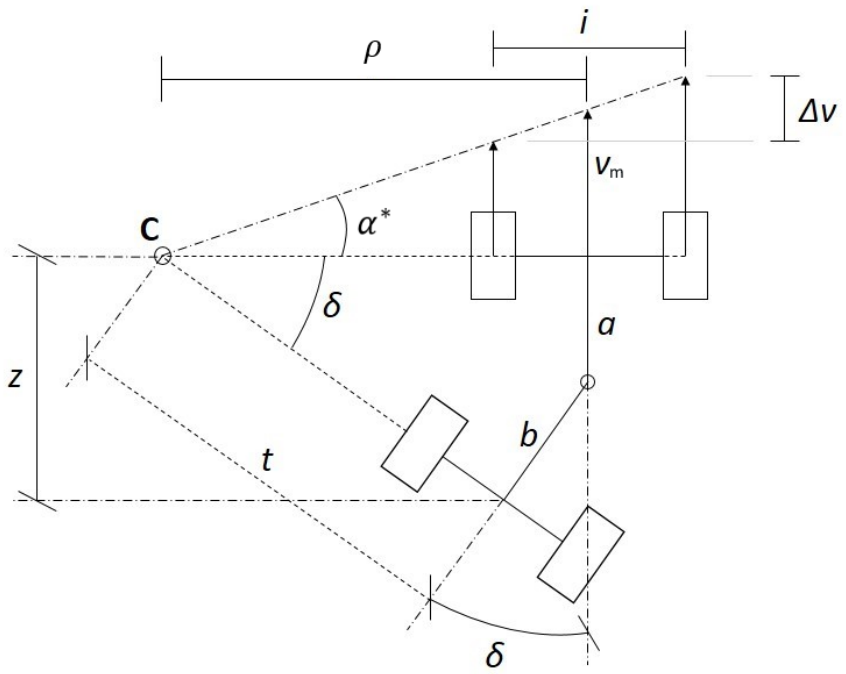


Figura 2.11: Analisi di una generica configurazione di sterzata

$$i\, n_{\mathrm{m}} \sin\delta = \Delta n(a\cos\delta + b) \tag{2.14}$$

2.3.2 Coefficiente di curvatura

Indicando con n_{l} e n_{r} i numeri di giri, rispettivamente, del motore sinistro e del motore destro, abbiamo che:

$$n_{\mathrm{m}} = \frac{n_{\mathrm{r}} + n_{\mathrm{l}}}{2}$$
$$\Delta n = n_{\mathrm{r}} - n_{\mathrm{l}}$$

Di conseguenza si ha che le curve antiorarie hanno sia un Δn che un angolo δ positivi (figura 2.10).

Sostituendo nella 2.14 e svolgendo qualche passaggio, abbiamo che:

$$n_{\mathrm{r}} = n_{\mathrm{l}} \frac{1 + \dfrac{i\sin\delta}{2(a\cos\delta + b)}}{1 - \dfrac{i\sin\delta}{2(a\cos\delta + b)}} \tag{2.15}$$

Ponendo:

$$K = \frac{1 + \dfrac{i \sin\delta}{2(a\cos\delta + b)}}{1 - \dfrac{i \sin\delta}{2(a\cos\delta + b)}} \tag{2.16}$$

la 2.15 si semplifica in:

$$\frac{n_r}{n_l} = K(\delta) \tag{2.17}$$

la quale ci dice che il rapporto tra le velocità dei motori è funzione dell'angolo di imbardata, e tramite la 2.12, di conseguenza, è funzione del raggio della curva che vogliamo percorrere.

Da qui in avanti K verrà chiamato *coefficiente di curvatura* (esso ci tornerà utile nel capitolo 5, in cui tratteremo la logica di funzionamento).

La 2.17 ci dice inoltre che se stiamo percorrendo una curva di raggio costante, il rapporto tra le velocità si mantiene costante, indipendentemente dalla velocità con cui stiamo percorrendo la curva stessa, a meno di slittamenti eccessivi.

Il coefficiente di curvatura è una funzione periodica, ma il robot ha un angolo di imbardata limitato nell'intervallo [-40°, 40°], dovuto al contatto fra le ruote, che rende tale funzione monotona crescente.

In figura 2.12 è riportato il grafico della funzione in suddetto intervallo. Possiamo notare che per $\delta = 0$, K vale 1, quindi avremo eguali velocità tra il lato destro e il sinistro. Per valori estremi dell'angolo, il coefficiente passa da circa 0.5 a un valore poco superiore 2.

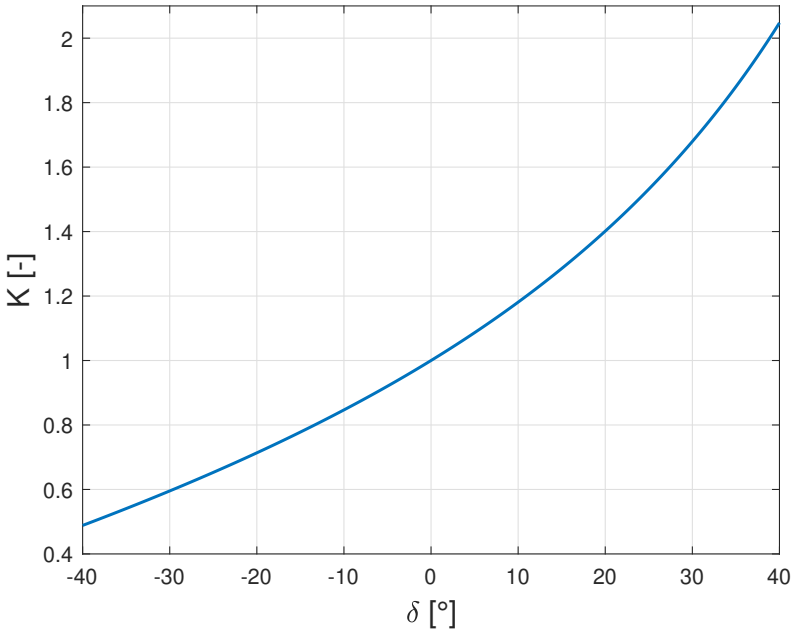


Figura 2.12: Coefficiente di curvatura K

Riportiamo anche il grafico del raggio di curvatura in funzione dell'angolo di imbardata (figura 2.13) e il grafico del suo reciproco (figura 2.14), il quale ci mostra che c'è una diretta

proporzionalità tra l'angolo e la curvatura. Possiamo infine calcolare il raggio minimo di curva in assenza di slittamenti, che è pari a:

$$\rho(\delta_{\max}) = 374\,\text{mm} \tag{2.18}$$

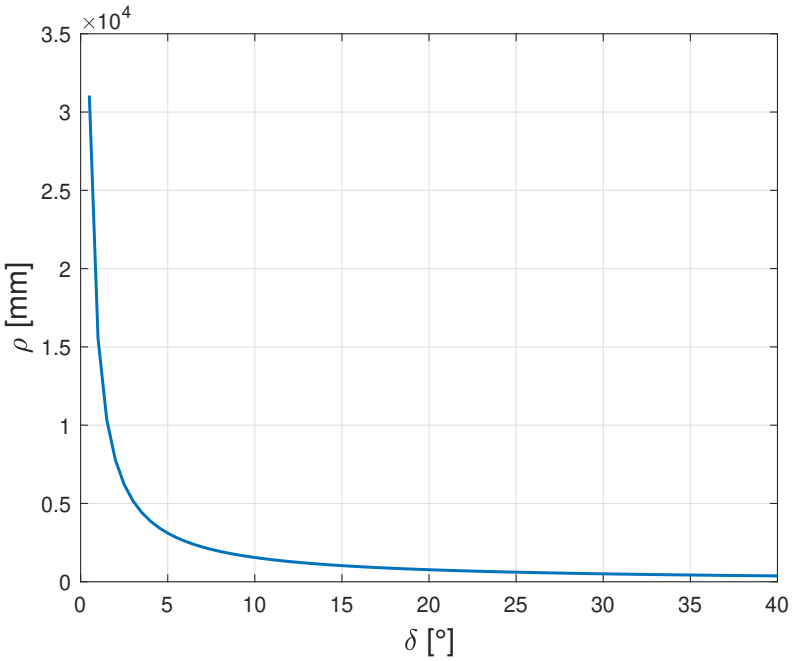


Figura 2.13: Raggio di curvatura in funzione dell'angolo

2.3.3 Velocità massima in curva

Sulla base di quanto riportato in 2.1.2, e sapendo che i motori hanno una velocità di rotazione massima pari a 10 000 rpm, possiamo calcolare la velocità massima raggiungibile dal robot in *advancing mode*:

$$v_{\text{m}} = \frac{2\pi}{60}\left(\frac{n_{\text{r}} + n_{\text{l}}}{2}\right)\tau_{\text{m}}\,\tau_0\,\frac{r_{\text{w}}}{1000} \tag{2.19}$$

La velocità è funzione delle singole velocità dei motori, quindi è logico pensare che più la curva da percorrere ha un raggio piccolo più lentamente saremo costretti a procedere.

Ricordando l'equazione 2.17 possiamo scrivere che:

$$n_{\text{m}} = \frac{n_{\text{r}}}{2}\left(\frac{K+1}{K}\right) \tag{2.20}$$

ed ipotizzando che n_{r} sia massimo e di percorrere una curva antioraria ($\delta > 0$), possiamo mettere in relazione la velocità massima all'angolo di imbardata (figura 2.15).

Come si può notare, la velocità massima raggiungibile è pari a 2.681 m/s, ma più l'angolo aumenta più questa diminuisce fino ad arrivare ad un valore di 1.969 m/s.

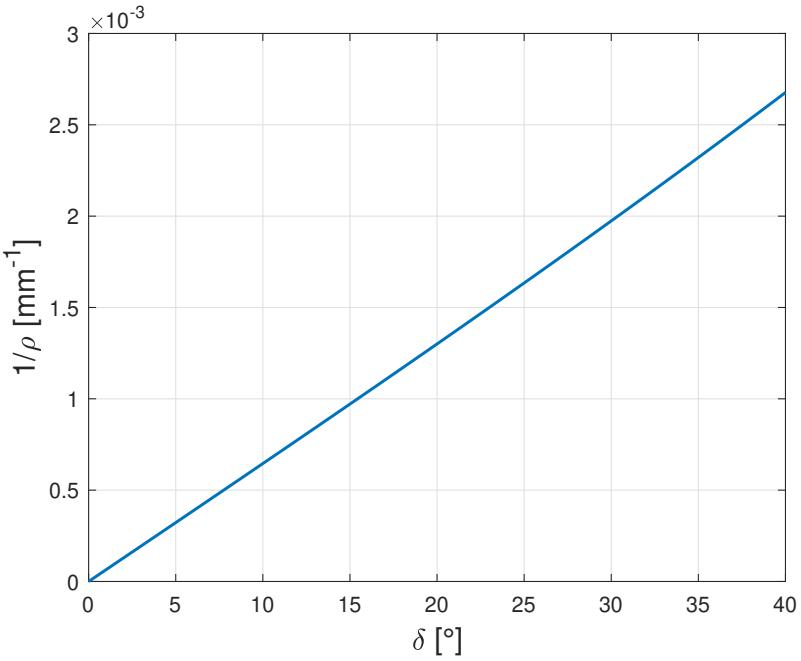


Figura 2.14: Curvatura in funzione dell'angolo

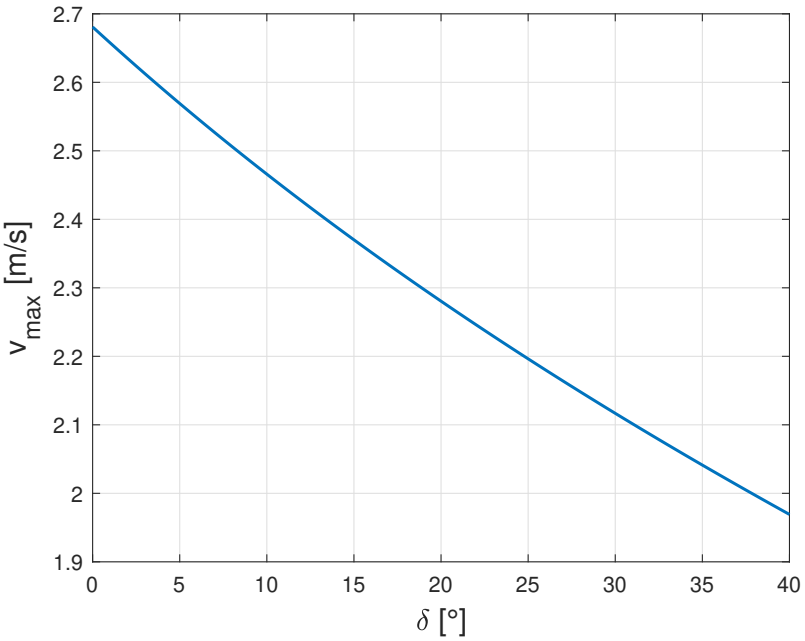


Figura 2.15: Velocità massima raggiungibile in funzione dell'angolo di imbardata

Capitolo 3

Sensorizzazione del giunto

In questo capitolo analizzeremo nel dettaglio il giunto di imbardata: la struttura iniziale, il processo e le scelte che hanno portato alla sua sensorizzazione.

3.1 Stato iniziale del giunto

Componente fondamentale del robot *Epi.q* è il giunto che collega i due moduli da cui è composto. Tale giunto è stato progettato in modo tale da avere due gradi di libertà, come mostrato in figura 2.3.

In figura 3.1 è mostrato un esploso dello stesso. Abbiamo due piastre, la (2) e la (5), le quali fanno parte rispettivamente del modulo posteriore e del modulo anteriore, che sono gli estremi del giunto. Le due piastre (1) sono avvitate alla piastra (2) in modo da creare un alloggiamento per il perno (3), il quale sarà quindi libero di ruotare. La boccola (4) è libera di ruotare attorno al perno, ed essa è cava per consentire il passaggio dei cavi tra i due moduli. Per lo stesso motivo il perno è stato realizzato con un diametro minore nella zona centrale.

La rotazione relativa tra la boccola ed il perno permette quindi al robot di sterzare (angolo di imbardata).

L'altro grado di libertà, ovvero il rollio, lo si ottiene dal fatto che le piastre (5) e (6), solidali tra loro, possono ruotare lungo l'asse della boccola, mentre la traslazione è vincolata dagli anelli seeger.

Nel normale funzionamento in modalità **4WD**, in cui il retrotreno viene semplicemente trascinato, non è fondamentale conoscere il valore dell'angolo di imbardata δ, anche se potrebbe essere utile per successive analisi di traiettoria. Cosa ben diversa è invece la modalità **8WD** in cui il retrotreno svolge una parte attiva nella trazione del veicolo, di conseguenza potrebbe essere rilevante acquisire tale valore al fine di sviluppare un algoritmo di controllo che ne tenga conto.

3.2 Scelta del sensore

Il primo passo consiste nello scegliere quale tipologia di sensore utilizzare, tenendo conto delle caratteristiche di misura, dell'affidabilità e della semplicità di montaggio.

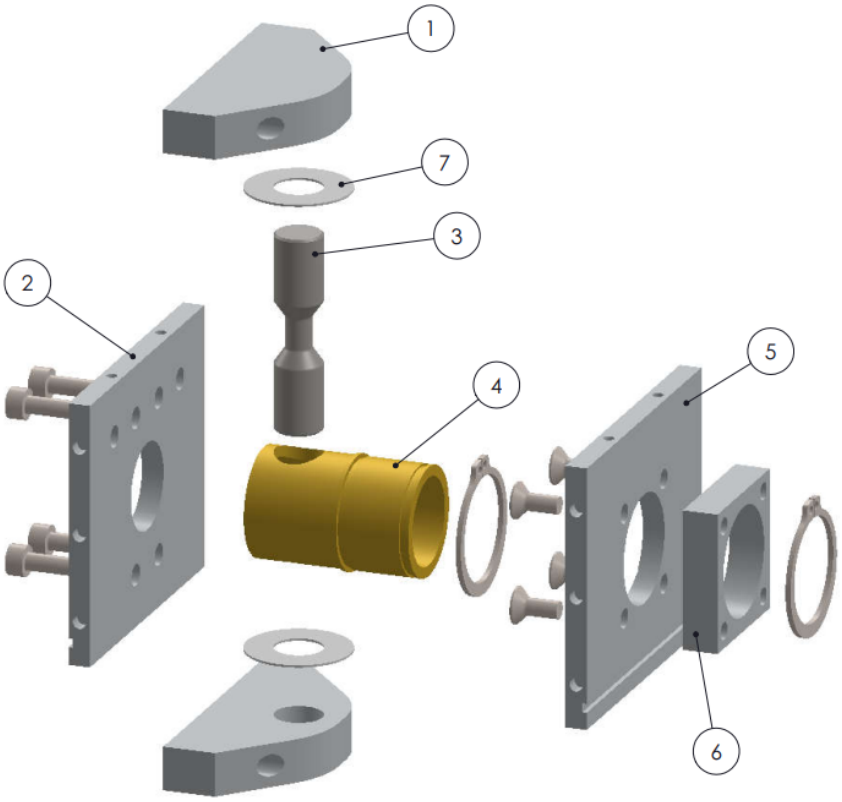


Figura 3.1: Vista esplosa del giunto non sensorizzato

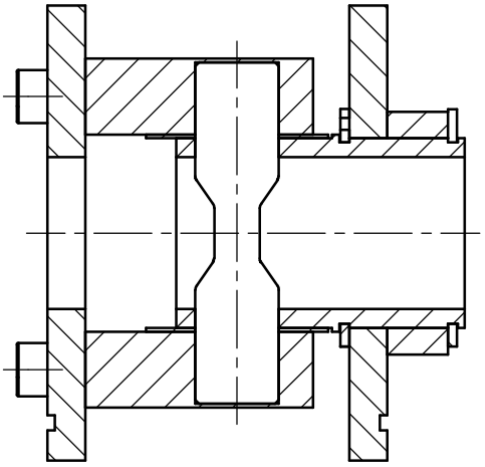


Figura 3.2: Vista in sezione del giunto non sensorizzato

Le due principali tipologie prese in esame sono state gli encoder assoluti e i potenziometri rotativi. Per via del minore costo, della maggiore semplicità costruttiva, della migliore robustezza e della maggiore compattezza questi ultimi sono stati preferiti ai primi.

Il principio su cui si basano è la lettura di una tensione, la quale varia in funzione dell'angolo, che in seguito verrà convertita.

In figura 3.3 è mostrato lo schema classico di un potenziometro. Applicando una tensione nota agli estremi (1) e (3), l'estremo (2) fornirà una tensione variabile in uscita. La resistenza R è di valore noto e solitamente è lineare, anche se esistono potenziometri logaritmici.

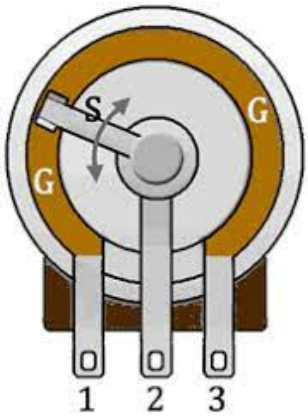


Figura 3.3: Potenziometro rotativo

Indicando con V_{in} e V_{out} la tensione in ingresso e in uscita, e con θ e θ_{tot} la posizione angolare e l'angolo massimo della resistenza, possiamo scrivere che:

$$V_{\text{out}} = \frac{\theta}{\theta_{\text{tot}}} V_{\text{in}} \tag{3.1}$$

Il potenziometro scelto è un CTS serie 284 (figura 3.4), il quale garantisce un'elevata compattezza (diametro di 16 mm) e un'elevata affidabilità, visti il grado di protezione IP67 e i 3 milioni di cicli rotazionali garantiti. Il valore della sua resistenza è pari a 10 kΩ con una tolleranza di ±10 % e una linearità di ±2 %.

3.3 Giunto sensorizzato

Definito il sensore da utilizzare è necessario modificare il giunto affinché sia possibile calettarlo sullo stesso.

In primo luogo sono stati definiti quali componenti devono essere solidali fra loro. Si è scelto dunque di bloccare la carcassa del potenziometro alla piastra superiore e calettare l'elemento rotante all'alberino interno del giunto.

Come detto nel paragrafo 3.1, l'albero può ruotare liberamente sia rispetto alle piastre, sia rispetto alla boccola, quindi è necessario vincolarlo in qualche modo.

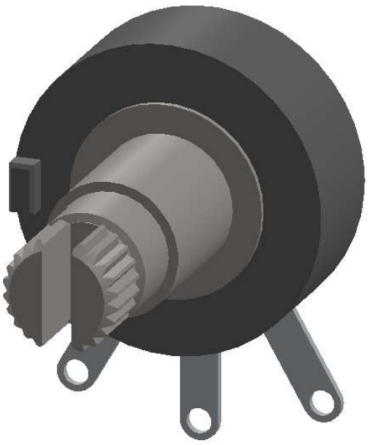


Figura 3.4: Vista assonometrica del potenziometro

La scelta è ricaduta nell'accoppiamento perno-boccola, in modo che la rotazione imposta dalla piastra del modulo frontale sia trasmessa tramite la boccola al perno, sul quale è calettato l'elemento rotante del potenziometro.

Tenuto conto del fatto che le piastre, il perno e la boccola necessitavano di lavorazioni alle macchine utensili, alcune delle quali con una precisione abbastanza elevata, si è deciso di stampare i pezzi in 3D[1], cosicché vi fosse una maggiore flessibilità nella forma dei nuovi componenti. Ed essendo la plastica più facilmente deformabile, la precisione richiesta negli accoppiamenti è diminuita.

In figura 3.5 si può notare un esploso del nuovo giunto.

Le piastre (1) e (2) non sono state modificate, così come il rollio del giunto.

Le piastre (3) e (4) sono invece state realizzate con una geometria leggermente differente e sono state inspessite al fine di garantire solidità strutturale. L'accoppiamento con la piastra (1) è comunque realizzato tramite viti, ma invece della filettatura sono state scavate due sedi in cui incastrare i dadi.

Tra il perno (6) e le piastre, per limitare gli attriti, sono state inserite delle bronzine autolubrificanti (7), le quali sono montate per interferenza. Quindi è il perno che striscia su queste ultime.

La piastra superiore (3) presenta due fori, uno per consentire il passaggio dell'alberino del potenziometro, e l'altro per consentire il passaggio dei cavi. Gli altri due più piccoli sono invece dei fori ciechi filettati su cui viene avvitato il supporto del potenziometro (8).

Il potenziometro (9) è vincolato al suo supporto perché è avvitato tramite un dado e perché presenta una piccola sporgenza[2] che ne impedisce la rotazione.

L'accoppiamento tra la boccola (5) e il perno (6) è realizzato mediante l'uso di una linguetta, la quale è stata direttamente stampata sul perno.

[1] Il materiale utilizzato è il tought PLA, che sarebbe un PLA rinforzato tramite fibre.

[2] La sporgenza è ben visibile in figura 3.4.

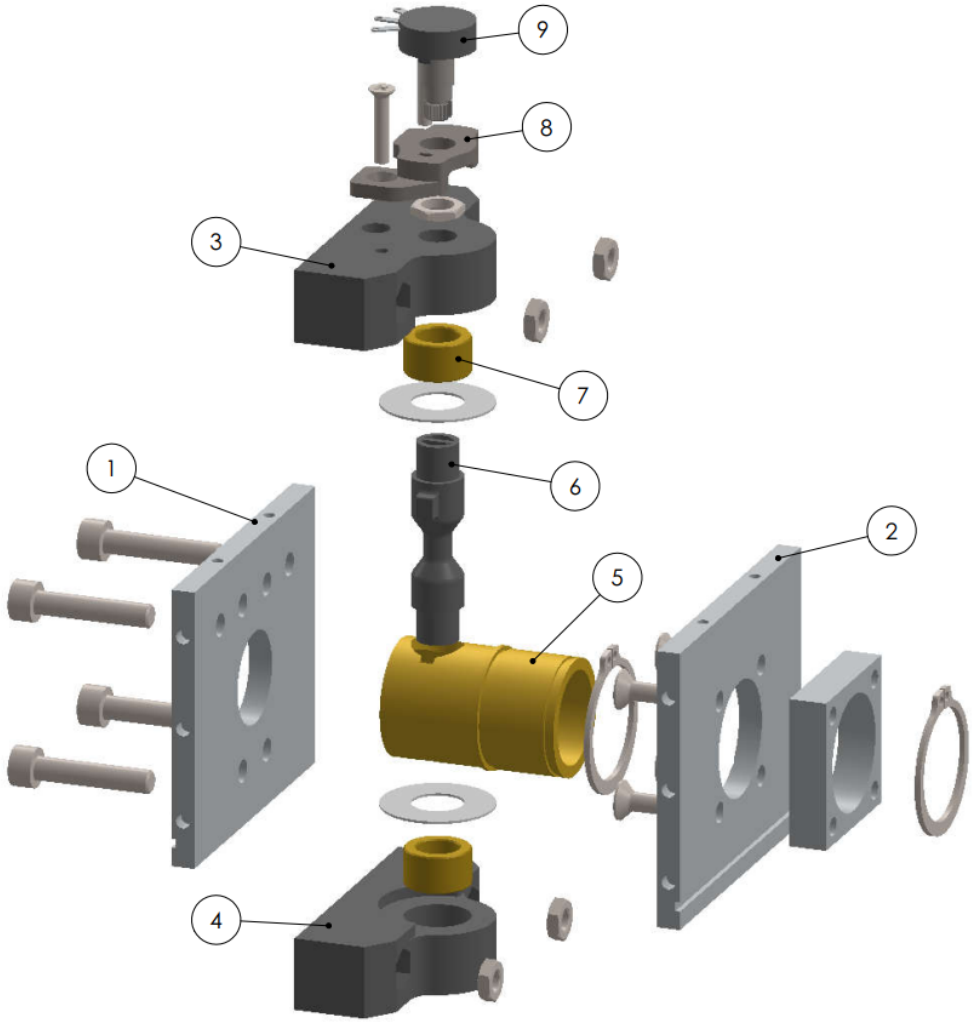


Figura 3.5: Vista esplosa del giunto sensorizzato

L'ultimo accoppiamento, ovvero quello tra il potenziometro (9) e il perno (6), è anch'esso un accoppiamento di forma. Nel perno è stato realizzato un negativo dell'estremità del potenziometro, come mostrato in figura 3.6, e le dimensioni sono state scelte in modo tale che i due siano forzati, così da non avere giochi[3].

Per ultimo sono stati modificati anche i bordi delle piastre (3) e (4) in modo da creare una battuta per la piastra (2) ed evitare così che le ruote possano entrare in contatto. Si è scelto quindi di limitare l'angolo δ nell'intervallo [-38°, 38°]. Questa battuta ci permette di conoscere con relativa certezza quale sia il valore dell'angolo agli estremi del campo di mobilità, e ciò ci tornerà utile in fase di taratura.

[3]La stessa cosa è stata fatta nell'accoppiamento perno-boccola.

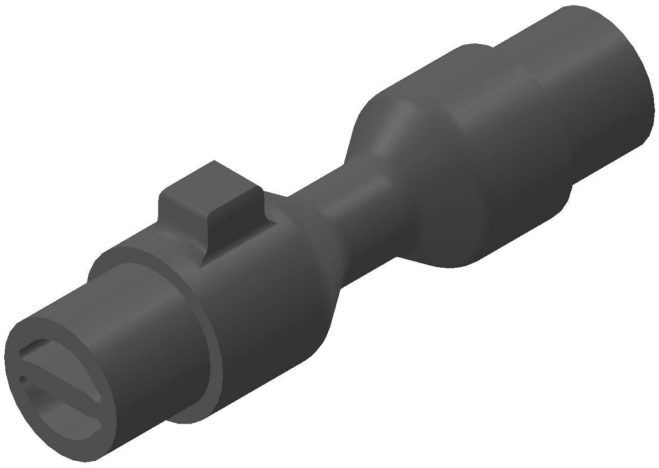


Figura 3.6: Perno stampato

3.3.1 Taratura del sensore

Il potenziometro ha un angolo totale di circa 280°, ma il campo di lavoro del giunto è di 76°. Essendo noti i valori dell'angolo agli estremi, è sufficiente misurare i corrispettivi valori di tensione ed applicare una mappatura lineare.

Nel nostro caso la tensione applicata ai capi è pari a 3.3 V, mentre le acquisizione vengono fatte tramite un convertitore analogico digitale con una risoluzione pari a 12 bit[4].

I dati in tensione vengono acquisiti in una scala che dipende dalla risoluzione che nel nostro caso va da 0 a 4095, quindi avremo $2^{12}-1$ livelli di tensione. Tali livelli sono espressi in funzione di una tensione interna di riferimento che nel nostro caso coincide con quella di alimentazione.

Indicando con V la tensione in uscita dal potenziometro e con $\overline{V}$ la tensione acquisita nella scala dell'ADC possiamo scrivere che:

$$V = \frac{\overline{V}}{4095} V_{\text{ref}} \tag{3.2}$$

Effettuata la misura, eseguiamo una mappatura lineare come segue:

$$\delta = (\overline{V} - \overline{V}_{\text{min}}) \frac{\delta_{\text{max}} - \delta_{\text{min}}}{\overline{V}_{\text{max}} - \overline{V}_{\text{min}}} + \delta_{\text{min}} \tag{3.3}$$

in cui δ_{max} e δ_{min} corrispondono ai valori massimi e minimi dell'angolo δ, e quindi noti; $\overline{V}_{\text{max}}$ e $\overline{V}_{\text{min}}$ sono invece i valori massimi e minimi letti dall'ADC che corrispondono ai valori noti dell'angolo.

Applicando quindi la 3.3 passiamo facilmente dalla lettura della tensione $\overline{V}$ all'angolo di imbardata δ.

Sapendo inoltre che:

[4] Viene usato il microcontrollore stesso.

$$\overline{V}_{\max} = 2836 \tag{3.4}$$

$$\overline{V}_{\min} = 1507 \tag{3.5}$$

possiamo calcolare la *sensibilità dello strumento*[5]. Essa è solitamente definita come segue:

$$S = \frac{\partial R}{\partial E} \tag{3.6}$$

dove ∂R rappresenta la variazione infinitesima dell'uscita a fronte della variazione infinitesima dell'ingresso ∂E.

Molto più interessante per noi è calcolare la *sensibilità della scala*, la quale ci fornisce il valore di una divisione, ed altro non è che l'inverso della sensibilità. Trovandoci in presenza di uno strumento discreto (ADC) passiamo inoltre dalle differenze infinitesime alle differenze finite. Avremo quindi che:

$$\frac{1}{S} = \frac{\Delta E}{\Delta R} \tag{3.7}$$

Ipotizzando che la sensibilità sia costante in tutto il nostro campo di misura possiamo calcolarla tramite i valori estremi a noi noti. Avremo quindi che:

$$\frac{1}{S} = \frac{\delta_{\max} - \delta_{\min}}{\overline{V}_{\max} - \overline{V}_{\min}} = 0.057 \, \frac{\text{deg}}{\text{div}} \tag{3.8}$$

Osservando i dati sperimentali si è visto che la misura di tensione oscilla in un intorno di ampiezza circa pari a 10. Per questo motivo è lecito supporre che l'incertezza sia circa pari a ± 5 e quindi riportata all'angolo è pari a $\pm\, 0.285°$, che possiamo tranquillamente arrotondare a $\pm\, 0.3°$. Tale incertezza dipende dai disturbi elettromagnetici essendo i cavi non schermati, ma per i nostri scopi non crea alcun problema.

[5] Per strumento si intenda la catena di misura composta da potenziometro e convertitore analogico digitale.

Capitolo 4

Struttura elettrica ed elettronica

In questo capitolo andremo ad analizzare la componentistica elettrica ed elettronica presente all'interno del robot.

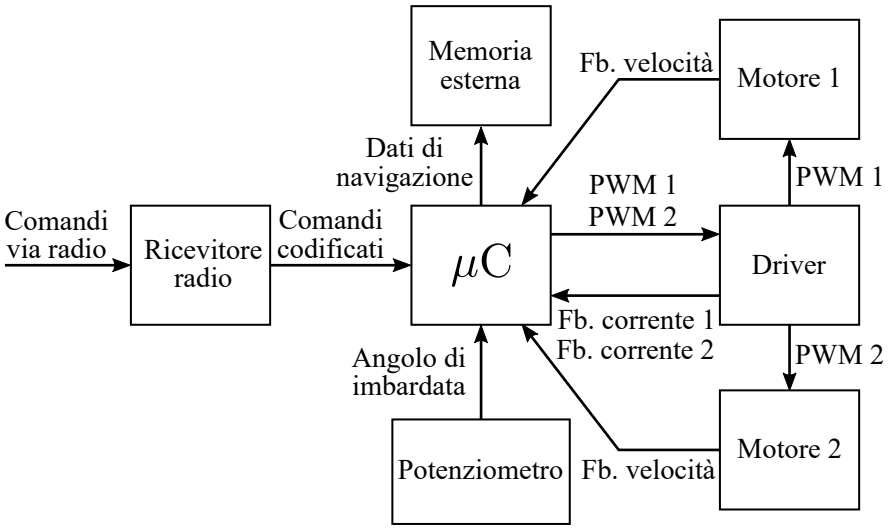


Figura 4.1: Schema componentistica *Epi.q*

In figura 4.1 è riportato lo schema dei componenti presenti in *Epi.q*. Troviamo un microcontrollore che riceve i comandi da un trasmettitore radio, li elabora e genera dei segnali PWM che invia al driver, il quale può controllare due motori elettrici indipendenti[1]. Il driver invia dei feedback in tensione proporzionali alla corrente circolante all'interno dei motori. Questi ultimi, invece, tramite un encoder calettato sul proprio albero, inviano dei feedback di velocità. Tramite il potenziometro si esegue la lettura dell'angolo di imbardata

[1] I driver all'interno del robot in realtà sono due, con un totale di quattro motori elettrici indipendenti.

δ. In ultimo, viene eseguito un log di tutti i dati di navigazione in una memoria esterna, la quale è alloggiata direttamente sul microcontrollore.

I comandi da inviare al robot possono essere generati in due modi, come mostrato in figura 4.2.

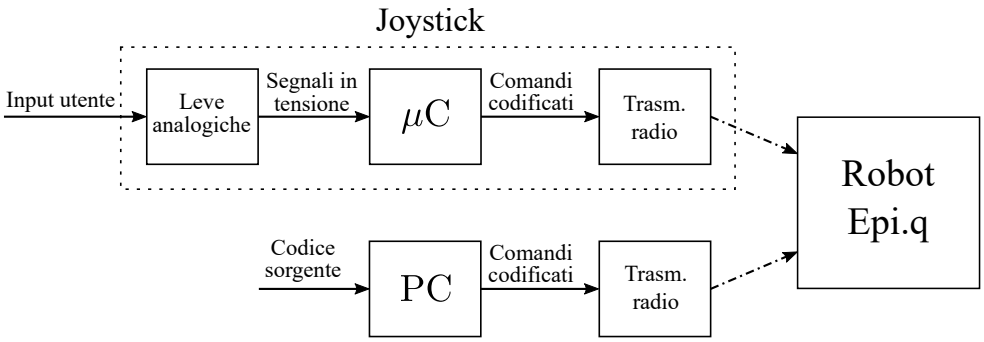


Figura 4.2: Schema generazione dei comandi

Il primo di questi, che è quello utilizzato, consente il controllo del robot mediante l'utilizzo di un joystick programmabile. In questo caso è l'utente che, muovendo gli analogici, genera dei segnali in tensione che vengono elaborati del microcontrollore, codificati e inviati via radio.

Alternativamente i comandi possono essere generati da un computer a seguito dell'esecuzione di un codice sorgente e inviati mediante il trasmettitore radio con la stessa codifica. In questo caso però si perde il feedback dell'operatore, visto che non sono implementate trasmissioni dati verso il PC.

4.1 Componentistica del joystick

Il joystick utilizzato (figura 4.3) è della marca "DFrobot"[2]. Al suo interno è presente un Arduino Leonardo[3], microcontrollore programmabile che consente di definire il funzionamento del joystick mediante la scrittura di un codice sorgente. In aggiunta è presente anche un trasmettitore radio (XBee), lo stesso presente anche nel robot. Il pinout e la mappa dei pulsanti verranno definiti nel prossimo capitolo.

Essendo programmabile, potrebbe essere utilizzato come un PC, con lo scopo di inviare comandi preimpostati senza un controllo attivo da parte dell'operatore. Il microcontrollore al suo interno ha comunque una capacità computazionale limitata, e molto inferiore rispetto al micro presente nel robot.

[2]Link di riferimento: https://www.dfrobot.com/product-858.html.

[3]Link di riferimento: https://www.arduino.cc/en/Main/Arduino_BoardLeonardo.

Figura 4.3: Joystick

4.2 Componentistica di Epi.q

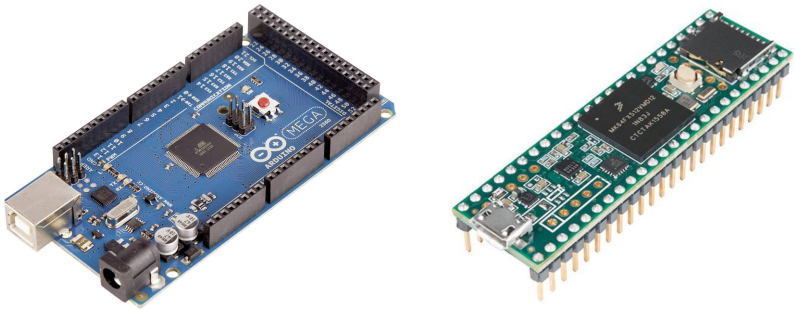
Epi.q è composto da:

- un microcontrollore, che si occupa di gestire tutta la logica di funzionamento e invia o riceve segnali dai vari componenti;
- due driver, che fungono da interfaccia bassa-alta potenza per il controllo dei motori;
- quattro motori, che sono i veri e propri attuatori presenti nel robot;
- un ricevitore radio, che permette la comunicazione wireless (modulo XBee);
- un potenziometro, che tramite un segnale in tensione permette la misura dell'angolo di imbardata;
- una memoria esterna che immagazzina i dati di navigazione in tempo reale.

4.2.1 Microcontrollore

Originariamente il microcontrollore installato sul robot era un *Arduino Mega* (figura 4.4a). A causa della non adeguata capacità computazionale, si è deciso di sostituirla con una famiglia dalle prestazioni superiori. La scelta è ricaduta sulla schede *Teensy*, e in particolare

sulla *Teensy 3.5* (figura 4.4b). La particolarità di queste ultime è che supportano le librerie native di Arduino, motivo per il quale, a meno di piccole differenze, il codice sorgente è interscambiabile.



(a) *Arduino Mega* (b) *Teensy 3.5*

Figura 4.4: Microcontrollori

Tabella 4.1: Confronto Arduino Mega e Teensy 3.5

	Arduino Mega	**Teensy 3.5**
Microcontroller	ATmega2560 @ 16 MHz	ARM Cortex-M4 @ 120 MHz
Operating Voltage	5V	3.3V
Input Voltage	7-12V	3.3-5V
Digital I/O Pins	54 (15 PWM output)	64 (20 PWM output)
Analog Input Pins	16	27 + 2 analog output
Flash Memory	256 KB	512 KB
SRAM	8 KB	256 KB
EEPROM	4 KB	4 KB
Length	101.52 mm	61.66 mm
Width	53.3 mm	17.78 mm

La frequenza di clock del *Teensy* è molto superiore rispetto all'Arduino, caratteristica che lo rende decisamente più performante, in aggiunta al fatto che ha a disposizione una memoria superiore. La differenza non riguarda tanto la flash, quanto più la ram, la quale consente l'implementazione di codici più complessi e che devono gestire una maggiore quantità di dati.

Altra caratteristica fondamentale è l'ingombro. Il *Teensy* è molto più compatto e si presta meglio ad essere montato su un circuito stampato, al contrario dell'Arduino, la cui superficie è di poco inferiore a quella disponibile.

Infine, il *Teensy* possiede, oltre alle linee seriali, ai bus I^2C e SPI presenti anche sulle schede Arduino, due bus CAN e uno slot nativo per l'inserimento di una micro sd. Quest'ultima caratteristica consente una velocità di scrittura e lettura dati superiore rispetto ad uno slot esterno[4], il che permette di aumentare la frequenza di campionamento dati.

[4]Premesso che la memoria inserita sia di una classe adeguata.

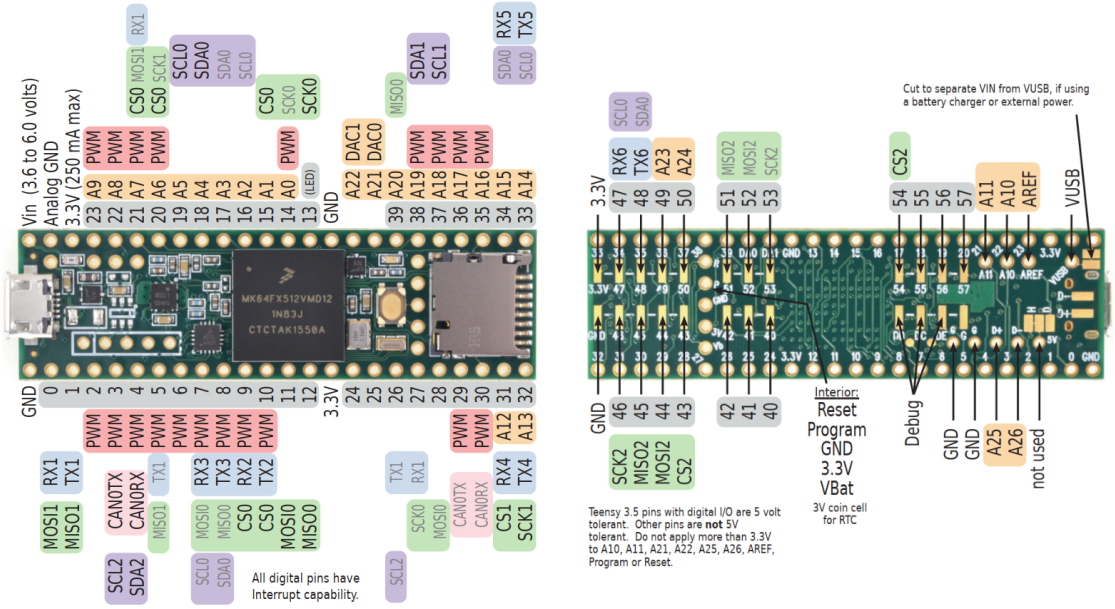


Figura 4.5: Pinout della scheda teensy 3.5

4.2.2 Driver

La scheda utilizzata è la **Dual MC33926 Motor Driver Carrier**[5]. Essa è in grado di alimentare contemporaneamente due motori DC con tensioni di uscita dai 5 ai 28 V e correnti continuative di 3 A, con picchi di 5 A.

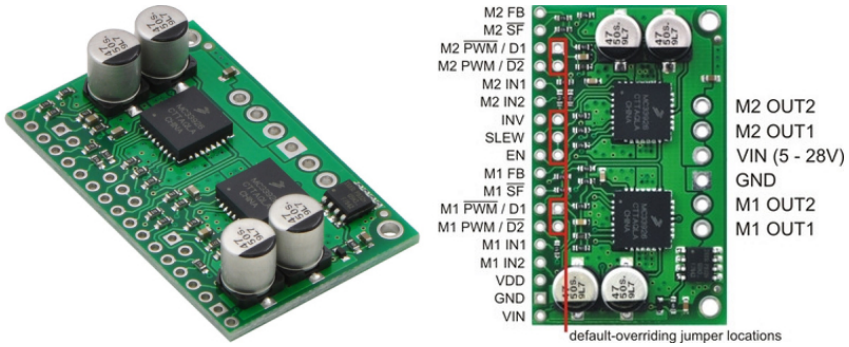


Figura 4.6: Dual MC33926 Motor Driver Carrier

In figura 4.6 è visibile la scheda con il relativo pinout. Il controllo dei motori avviene tramite un segnale PWM che il microcontrollore invia al pin PWM o $\overline{\text{PWM}}$ (il secondo inverte in segnale). Di conseguenza il driver applica ai motori tale onda quadra amplificata in base alla tensione di alimentazione della scheda stessa (pin VIN).

[5]Sito di riferimento: https://www.pololu.com/product/1213

Tramite i pin IN1 e IN2 si sceglie invece la polarità, ovvero il verso di rotazione. Affinché ciò avvenga è necessario imporre una tensione HIGH su un pin e una tensione LOW sull'altro, se entrambi ricevono lo stesso livello di tensione il motore non viene alimentato.

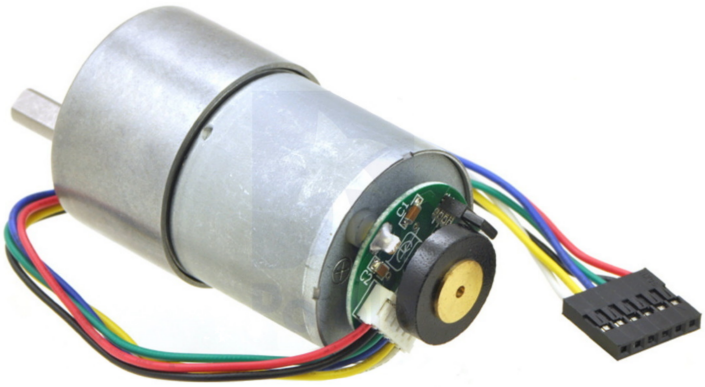
Tramite il pin FB si possono acquisire informazioni sulla corrente istantanea circolante nei motori. A tale scopo è sufficiente eseguire una lettura analogica ed in funzione di tale tensione si può calcolare la corrente, sapendo che il guadagno è pari a 525 mV/A.

Il pin EN ha invece la funzione di abilitare o disabilitare l'intero driver.

Il driver possiede inoltre dei sistemi di protezioni, in particolare si ha una protezione contro le sotto tensioni, contro le sovracorrenti e contro il surriscaldamento. In caso una o più dovessero entrare in azione, la scheda si spegnerebbe immediatamente.

4.2.3 Motoriduttore

Il motore scelto per movimentare il robot è un motore elettrico in corrente continua con spazzole e tensione nominale di 12 V. Questa famiglia (figura 4.7) contiene al suo interno un riduttore di velocità, il cui rapporto di trasmissione varia a seconda del modello scelto.



Color	Function
Red	motor power
Black	motor power
Green	encoder GND
Blue	encoder Vcc
Yellow	encoder A
White	encoder B

Figura 4.7: Motoriduttore

Al fine di ottenere un buon compromesso tra velocità massima e coppia massima si è optato per la versione con rapporto di trasmissione pari a $\tau_m = 1/50$, denominato "50:1 Metal Gearmotor 37Dx54L mm 12V with 64 CPR Encoder (No End Cap)"[6].

Nella seguente tabella 4.2 sono riportate le principali caratteristiche.

Le velocità di fuga e le coppie sono riferite all'albero di uscita, e tramite il rapporto di trasmissione è possibile calcolare i valori riferiti all'albero motore.

Il valore che ci interessa maggiormente è la velocità di fuga alla tensione nominale riferita all'albero motore, la quale è pari a 10 000 rpm. Tale velocità verrà utilizzata per definire il duty cycle nella progettazione del controllo.

Sull'albero motore è inoltre calettato un encoder rotativo ad effetto hall, il quale ha due canali sfasati di 90° tra di loro con una risoluzione complessiva pari a 64 CPR (counts per revolution). Lo sfasamento è necessario se si vuole conoscere il verso di rotazione.

[6]Sito di riferimento: https://www.pololu.com/product/1444.

Tabella 4.2: Specifiche generali motore DC

Specifications	**@ 12 V**	**@ 6 V**
No-load speed (rpm)	200	100
No-load current (mA)	300	250
Stall current (mA)	5000	2500
Stall torque (kg · cm)	12	6

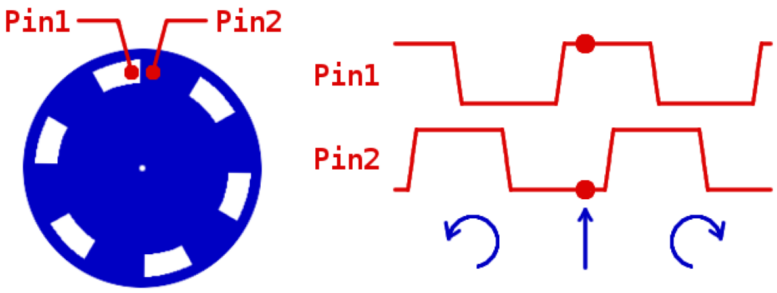


Figura 4.8: Schema dei segnali dell'encoder

Come si può notare in figura 4.8 la rotazione dell'encoder genera due segnali ad onda quadra. Tramite le interrupt[7] si possono rilevare i fronti di salita e di discesa e incrementare o decrementare di conseguenza una variabile[8].

Sapendo che al tempo $t = t_1$ la variabile contatore vale $C = C_1$ si ha che al tempo $t = t_2$, $C = C_2$. Di conseguenza si può calcolare la velocità angolare media in quell'intervallo di tempo, che è data da:

$$n = \frac{60 \cdot \Delta C}{64 \cdot \Delta t} \text{ [rpm]} \tag{4.1}$$

in cui 64 è la risoluzione dell'encoder e il tempo è espresso in secondi.

Questa modalità di calcolo della velocità ha delle implicazioni importanti sul funzionamento del robot e sul suo controllo. In primo luogo è necessario eseguire il calcolo con una certa frequenza al fine di avere un andamento quanto più reale possibile. In secondo luogo, invece, eseguire il calcolo troppo spesso può provocare errori per via del fatto che andiamo a disabilitare le interrupt frequentemente[9], e ciò non consente un normale incremento della variabile, oltre alla normale incertezza sull'ultima cifra.

[7] Al verificarsi di un'interrupt la CPU interrompe il programma in esecuzione ed esegue un sottoprogramma, chiamato routine di interrupt, allocato ad uno specifico indirizzo di memoria. Terminata la routine d'interrupt, la CPU torna al programma precedentemente interrotto e ne prosegue l'esecuzione.

[8] Tale compito viene svolto nel programma mediante l'utilizzo di una libreria esterna: https://www.pjrc.com/teensy/td_libs_Encoder.html

[9] Nel momento in cui andiamo ad accedere alla memoria dove è contenuta la variabile è necessario disabilitare le interrupt per evitare che la routine acceda allo stesso indirizzo.

Per fare un esempio proprio su quest'ultimo punto, ipotizziamo di avere la nostra variabile C che può valere C_0 o $C_0 + 1$, quindi avremo che l'intervallo di incertezza è pari ad 1. Utilizzando l'equazione 4.1 possiamo scrivere l'intervallo in termini di velocità calcolata in funzione dell'intervallo di campionamento scelto.

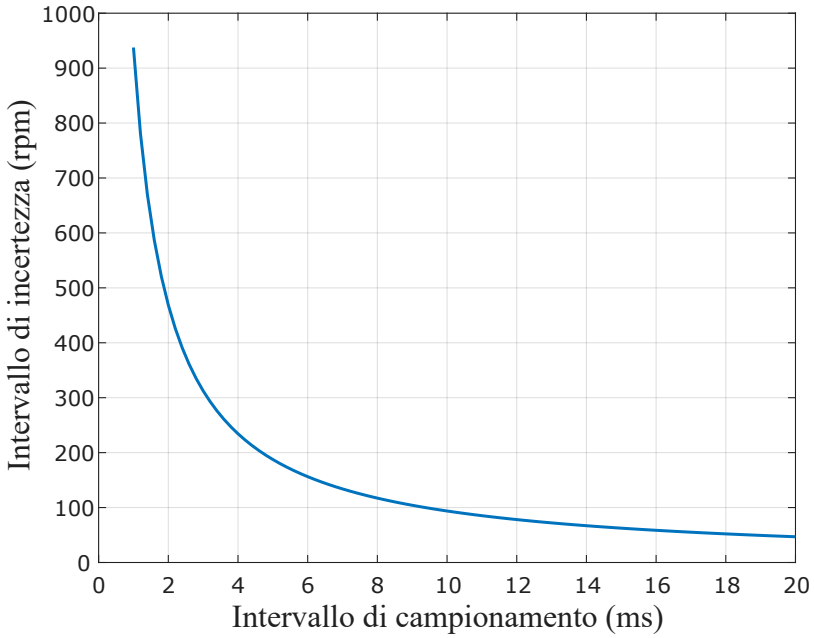


Figura 4.9: Intervallo di incertezza dell'encoder

Come si può notare in figura 4.9 un intervallo di campionamento basso rende l'intervallo di incertezza elevato. Essendo quest'ultimo in valore assoluto, avrà un peso notevole nel caso in cui la velocità di avanzamento del robot fosse bassa.

In figura 4.10 riportiamo in ordinata l'intervallo di incertezza diviso per la velocità di rotazione dell'albero motore, al fine di ottenere un'incertezza relativa.

Come si può osservare, il tempo di campionamento ha un maggior effetto per basse velocità del robot, ovvero quando i colpi registrati nell'intervallo temporale sono pochi.

Questo aspetto non è il solo che va ad influenzare l'errore nel calcolo della velocità e, come detto in precedenza, ci possono essere altri fattori che interferiscono con la variabile contatore. Per questo motivo si è deciso di tenere il tempo di campionamento pari a 10 ms e di utilizzare un filtraggio numerico per limitare le oscillazioni.

Sulla base della scelta di questo tempo ciclo sono stati definiti anche i tempi del controllo e del programma stesso.

Per fare un esempio, viene naturale che un eventuale controllo in velocità abbia lo stesso tempo di ciclo e, di conseguenza, si ha che il loop di invio/ricezione dei comandi dovrà essere un multiplo.

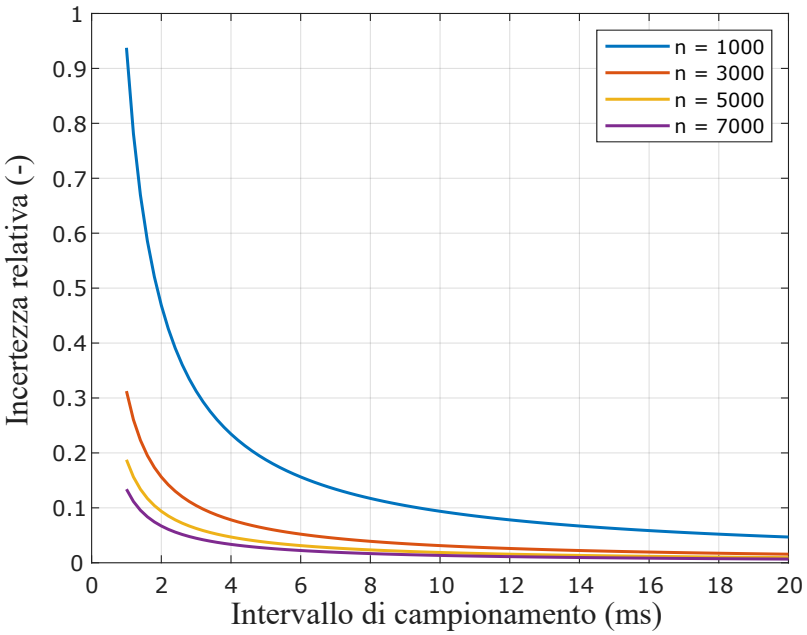


Figura 4.10: Intervallo relativo di incertezza dell'encoder

4.2.4 Trasmettitore radio

I comandi vengono trasmessi al robot mediante l'utilizzo di moduli a radio frequenza, nello specifico la scheda utilizzata è un XBee PRO S1 (figura 4.11), basata sullo standard IEEE 802.15.4 e operante alla frequenza di 2.4 GHz (la stessa del wi-fi ma con un protocollo diverso).

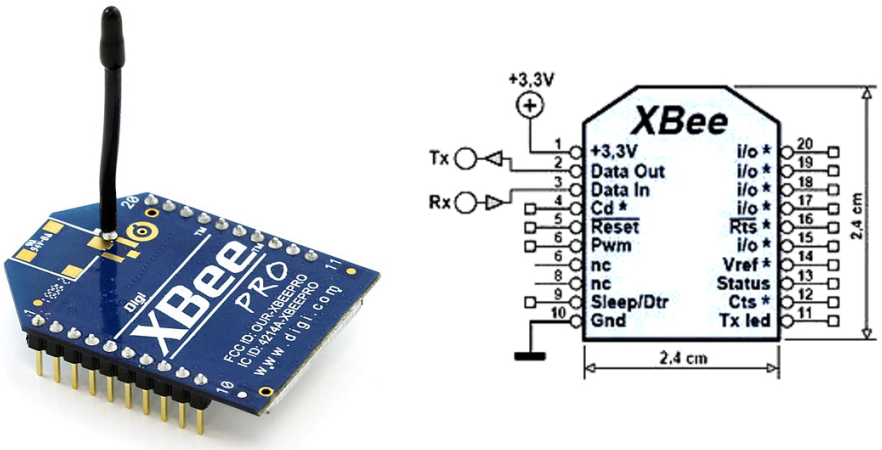


Figura 4.11: Modulo XBee e pinout

La scheda XBee è molto versatile e può svolgere diverse funzioni in autonomia, tra cui operazioni di input e output quali la lettura dei sensori. Per i nostri scopi non sfrutteremo appieno tutte le sue caratteristiche, ma ci limiteremo a utilizzarlo come estensione del cavo seriale (figura 4.12). Per lo scopo utilizziamo infatti solo i pin di alimentazione e i pin della porta UART.

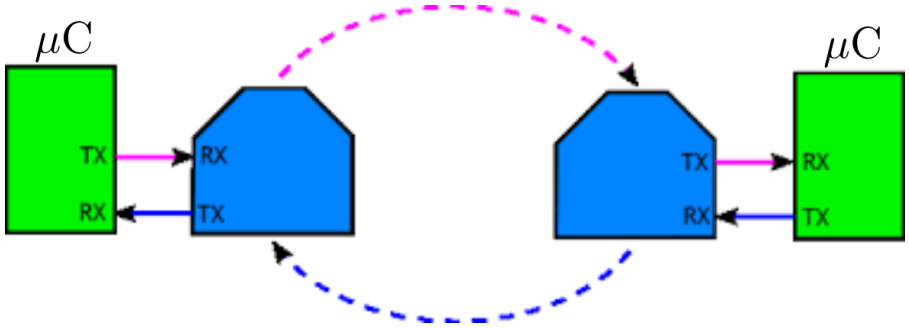


Figura 4.12: Schema di comunicazione tramite XBee

L'attuale configurazione prevede che i comandi generati e codificati dall'Arduino presente nel joystick vengano inviati mediante protocollo seriale al modulo XBee, il quale provvede, tramite la comunicazione radio, al trasferimento dei dati. Il modulo ricevente inoltra tali informazioni alla porta seriale della scheda Teensy.

La comunicazione radio avviene ad un baud rate[10] costante pari a 250 000 b/s, mentre la trasmissione dati con il microcontrollore tramite cavi può essere impostata al valore che più si preferisce, e deve essere uguale per entrambi i moduli. Nello specifico i valori standardizzati sono i seguenti: 1200, 2400, 4800, 9600 (default), 19 200, 38 400, 57 600, 115 200, 230 400 b/s.

Impostare un baud rate elevato potrebbe portare ad errori nella trasmissione dei dati, al contrario un valore basso non riuscirebbe a soddisfare le nostre esigenze creando un collo di bottiglia.

La velocità massima consigliata dai produttori della scheda per non incorrere in errori è pari a 38 400, ed è proprio quella scelta. Inoltre il tempo ciclo tra l'invio di un comando ed il successivo è pari 20 ms.

La quantità di dati che viene inviata ad ogni ciclo è pari a 6 byte, di cui 3 sono informazioni utili al controllo, e 3 sono necessari per il corretto funzionamento della libreria[11].

Ogni ciclo si impiegano quindi circa 1,25 ms per il trasferimento dei dati tramite seriale[12].

Altro dato importante del modulo XBee è il suo range di trasmissione. In un contesto indoor o urbano arriviamo a circa 90 m, mentre in campo libero si arriva anche oltre 1 km (questi dati sono riferiti alla versione pro, la base ha prestazioni decisamente inferiori).

---

[10] Il termine baud, in telecomunicazioni e informatica, rappresenta il numero di simboli (dati) che viene trasmesso in un secondo.

[11] Libreria "EasyTransfer", link: https://github.com/madsci1016/Arduino-EasyTransfer.

[12] Questo valore non include l'intera catena, ma unicamente il passaggio dal microcontrollore a XBee o viceversa sulla base del buad rate scelto.

4.2.5 Alimentazione

Il robot dispone di una batteria da 12 V con una capacità di 1700 mAh. Tale batteria alimenta direttamente i driver, i quali si occupano dell'alimentazione dei motori.

Sono presenti inoltre due regolatori di tensione. Il primo è un L7805 che fornisce una linea a 5 V con cui vengono alimentati il teensy e gli encoder dei motori. L'altro regolatore è collegato alla linea dei 5 V e l'abbassa fino ai 3.3 V in modo da poter alimentare anche il modulo XBee. Quest'ultimo richiede una corrente decisamente superiore a quella che può fornire il Teensy, motivo per il quale è stata necessaria una linea a parte.

Il potenziometro invece è alimentato da direttamente dal microcontrollore perché le correnti assorbite sono estremamente basse, minori di 1 mA.

4.3 Circuito stampato

Considerata la necessità di cambiare il microcontrollore, si è colta l'occasione per riprogettare il circuito elettrico al fine di renderlo più compatto e più facilmente mantenibile. Si è realizzato perciò un PCB su cui allocare tutti i vari componenti, in modo da minimizzare i cablaggi all'interno del robot stesso.

In figura 4.13 è visibile la struttura del circuito elettrico prima della realizzazione del PCB. Si può notare come l'elevata densità di cavi renda complicato qualsiasi tipo di intervento, e l'elevata dimensione dell'Arduino Mega obblighi a collegare i componenti impilandoli uno sopra l'altro.

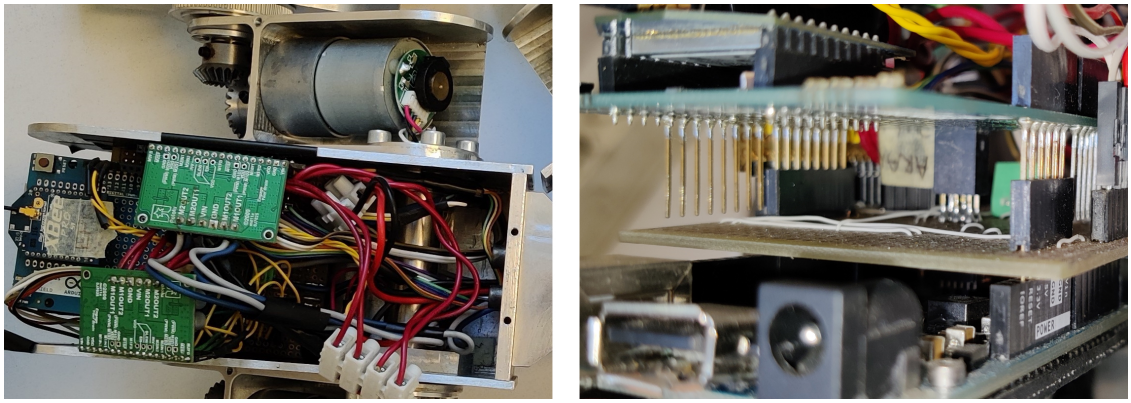


Figura 4.13: Circuito elettrico precedente

Il primo passo nella realizzazione del PCB è stato disegnare uno schema circuitale al fine di definire tutti i vari collegamenti elettrici, ed in seguito si è passato allo sbroglio e alla tracciatura delle piste.

In figura 4.14 è riportato tale schema, nel quale sono contenuti unicamente i collegamenti diretti con il PCB.

Il collegamento dei componenti avviene mediante connettori e non per saldatura diretta, ad eccezione dei regolatori di tensione. Nello specifico sono stati utilizzati dei *Pin Socket* per il *Teensy*, il modulo XBee e i driver, mentre sono stati utilizzati dei connettori JST PH 2.0 per i cavi.

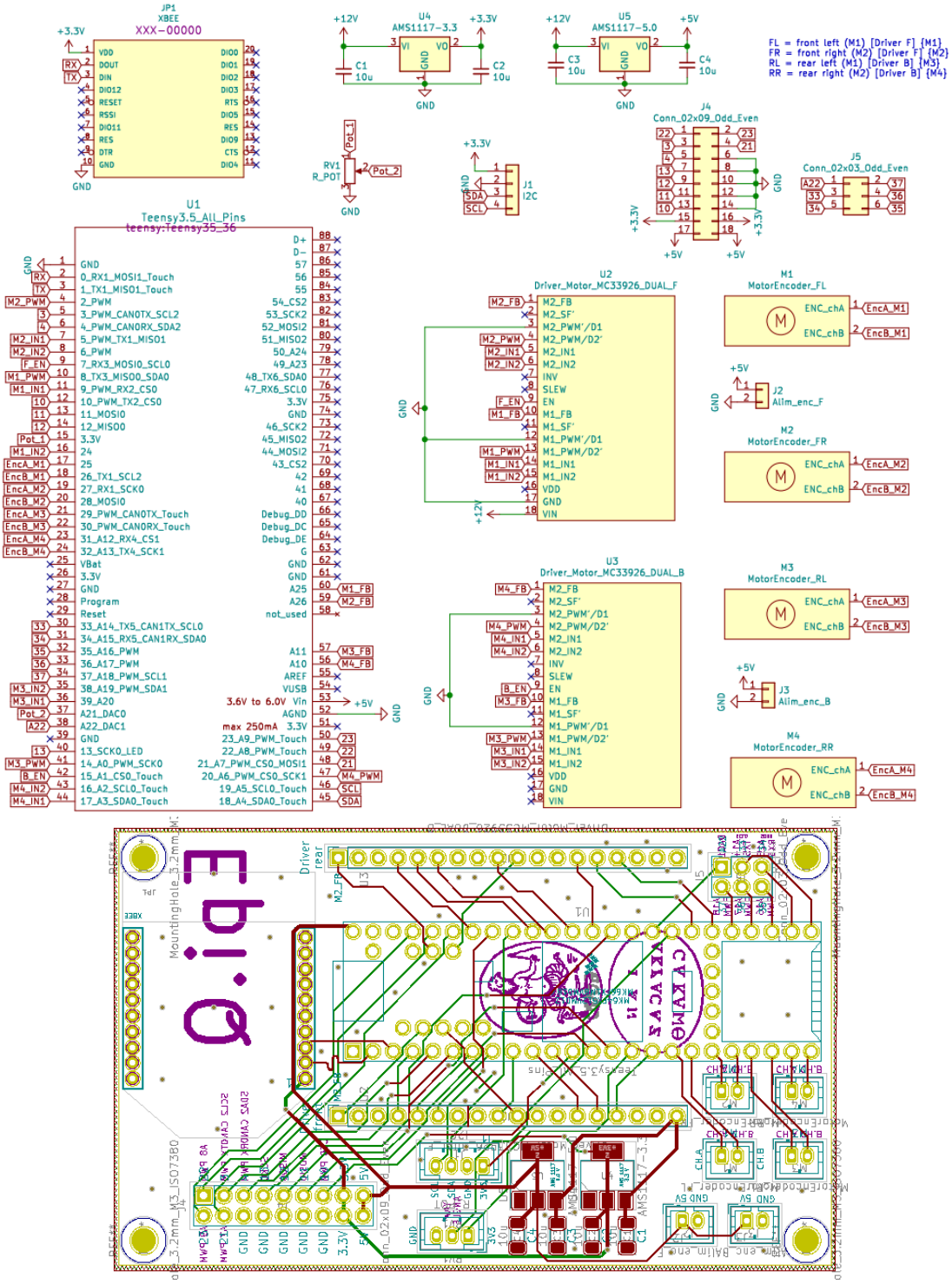


Figura 4.14: Schema circuitale e sbroglio delle piste

I pin non utilizzati sono stati resi accessibili mediante dei *Pin Header* al fine di semplificare gli sviluppi futuri.

Nella parte inferiore della stessa figura è riportata l'intera tracciatura delle piste. Non sono presenti i GND per una questione visiva: essi coprono gran parte della superficie libera della scheda al fine di compensare meglio i disturbi elettromagnetici e non renderebbero visibile il layer inferiore.

Nella realizzazione del PCB si è cercato innanzitutto di limitare la sua superficie e di sfruttare lo spazio verticale come si può ben vedere in figura 4.16. Tale scelta ha consentito di rendere il tutto più compatto e organizzato possibile, al fine di agevolare i disassemblaggi.

I connettori bianchi (JST PH) a due poli alimentano e ricevono i segnali degli encoder, il connettore a tre poli è invece collegato al potenziometro. L'ultimo rimanente, per il momento scollegato, è un bus I^2C, comprensivo di alimentazione, che potrebbe essere utilizzato per collegare altri sensori, ad esempio un *IMU*. Stesso discorso vale per i pin header.

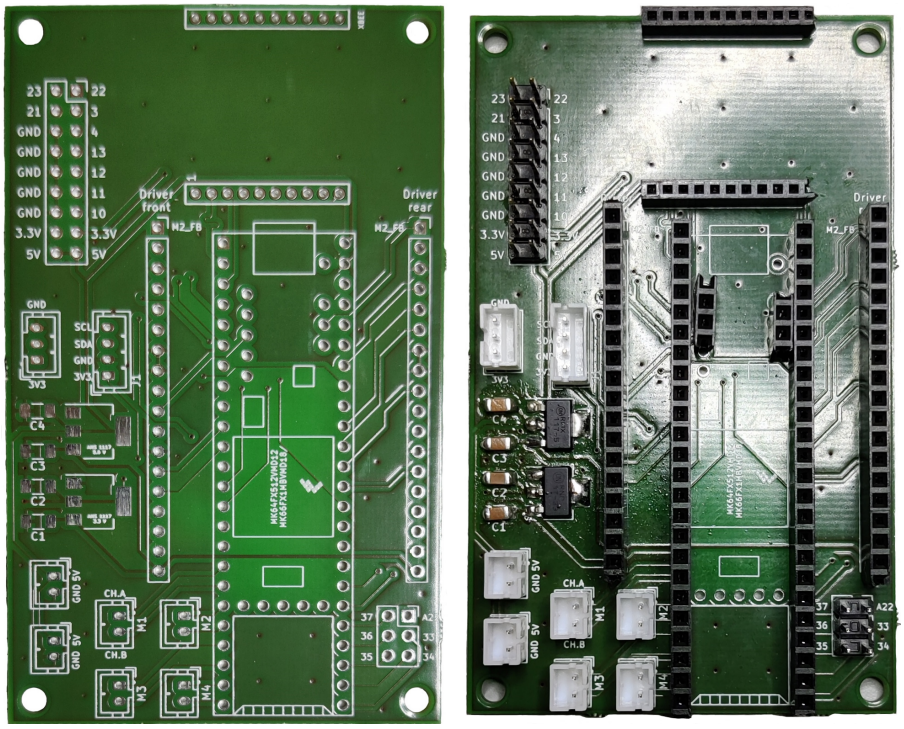


Figura 4.15: PCB

In figura 4.17 sono riportate due immagini dello stato attuale del modulo che alloggia la componentistica elettronica. Come si può ben vedere, i componenti sono molto più accessibili e maggiormente distanziati tra loro.

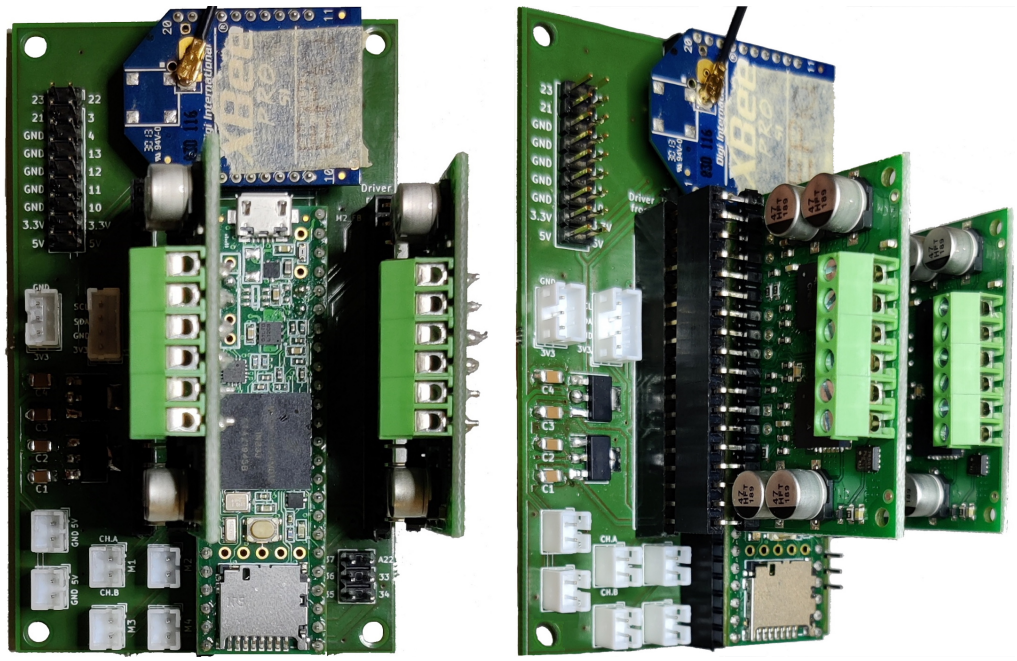


Figura 4.16: Viste del PCB assemblato

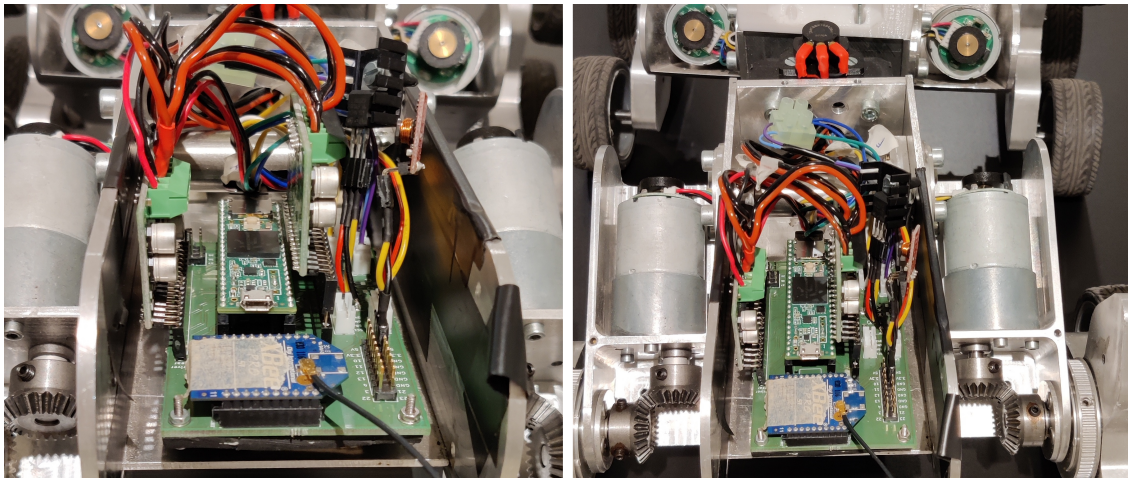


Figura 4.17: Viste posteriori del robot

Capitolo 5

Logica di controllo

In questo capitolo viene trattata la progettazione dei controlli e la logica di funzionamento implementata nel robot.

L'obiettivo principale di questo lavoro è stato costruire una logica di basso livello in grado di ricevere comandi e attuarli efficacemente, e sulla quale, in seguito, si potesse implementare una logica di alto livello in grado di rendere il robot autonomo.

L'implementazione è stata realizzata in C++ mediante l'utilizzo dell'ambiente di sviluppo "Platformio", il quale è un'estensione del software "Visual Studio Code".

È stata scelta tale piattaforma perché consente una maggior semplicità nella scrittura del codice e fornisce strumenti più avanzati rispetto al classico IDE di Arduino[1].

Nonostante la premessa, in questo capitolo non tratteremo aspetti tecnici dell'implementazione, per i quali si rimanda all'appendice A, ma unicamente aspetti logici e funzionali.

5.1 Schema funzionale

L'intera logica ha l'obiettivo di generare un segnale PWM da inviare ai driver, i quali applicheranno una tensione ai motori elettrici proporzionale al duty cycle. Il robot viene controllato mediante l'utilizzo di un joystick programmabile, e tramite una coppia di trasmettitori a radio frequenza (XBee).

Il robot possiede due motori indipendenti per ogni modulo. L'anteriore ha il compito di direzionare il moto imponendo una differenza di velocità tra il lato sinistro e il destro, come visto nel capitolo 2, motivo per il quale tale modulo viene controllato in velocità. Il modulo posteriore ha invece il compito di supportare l'anteriore fornendo potenza quando necessario, per cui si è scelto di implementare un controllo fuzzy. Tale controllo utilizza i PWM dell'anteriore come base di partenza, e dopo opportune elaborazione li invia al posteriore.

Il modulo anteriore è necessario per la corretta movimentazione del robot, infatti i comandi inviati ed elaborati servono inizialmente a generare il suo riferimento in velocità.

[1] Le schede teensy possono essere nativamente programmate mediante l'IDE di Arduino.

Il modulo posteriore, al contrario, può essere disabilitato, riducendo sì le prestazioni, ma consentendo ugualmente la navigazione.

Al fine di garantire una maggiore manovrabilità sono state definite due modalità di moto[2]: la prima permette la normale navigazione del robot (*advancing*), mentre la seconda permette la rotazione sul posto generando un riferimento in velocità tale per cui il lato sinistro e il destro vadano in direzioni opposte (*pivot*).

All'interno del programma c'è una distinzione tra i moduli fisici e i moduli logici, quest'ultimi sono temporanei e variano a seconda delle condizioni e dei comandi inviati. Il modulo di trazione è chiamato "Front" e il modulo di spinta è chiamato "Rear". I due moduli temporanei puntano ai due moduli fisici che invece sono fissi e definiti. Ciò permette di avere la stessa tipologia di controllo sul modulo di trazione e sul modulo di spinta indipendentemente dalle condizioni di moto, banalmente non ci sono differenze tra la marcia avanti e la retromarcia.

Il *front* è sempre controllato in velocità, mentre il *rear* sempre mediante il fuzzy.

In figura 5.1 è mostrato uno schema di quanto detto. Il modulo M1 è di trazione per la direzione di avanzamento scelta e perciò è assegnato al *front*, e di conseguenza abbiamo l'assegnamento anche per l'altro modulo.

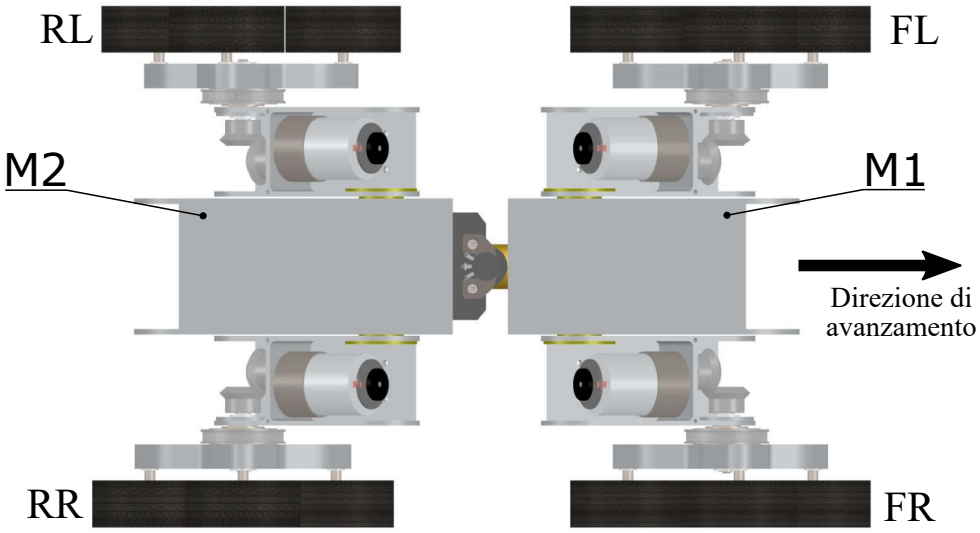


Figura 5.1: Definizione dei moduli

Nel momento in cui si ha un'inversione del verso di avanzamento i due moduli si invertono e avremo che il *front* punterà al modulo M2 e il *rear* al modulo M1. Di default il modulo M1 è il modulo frontale. Come distinzione possiamo dire che il modulo M2 è solidale con le piastre del giunto e con il corpo del potenziometro. Tale sistema è possibile perché il robot è sufficientemente simmetrico, in caso contrario tutto ciò avrebbe creato dei problemi.

[2]Non sono le due modalità intrinseche del tripode.

Ipotizzando quindi che il modulo M1 sia il modulo anteriore, possiamo dare un nome ai quattro motori, come mostrato in figura. Essi stanno rispettivamente per: *front left*, *front right*, *rear left*, *rear right*.

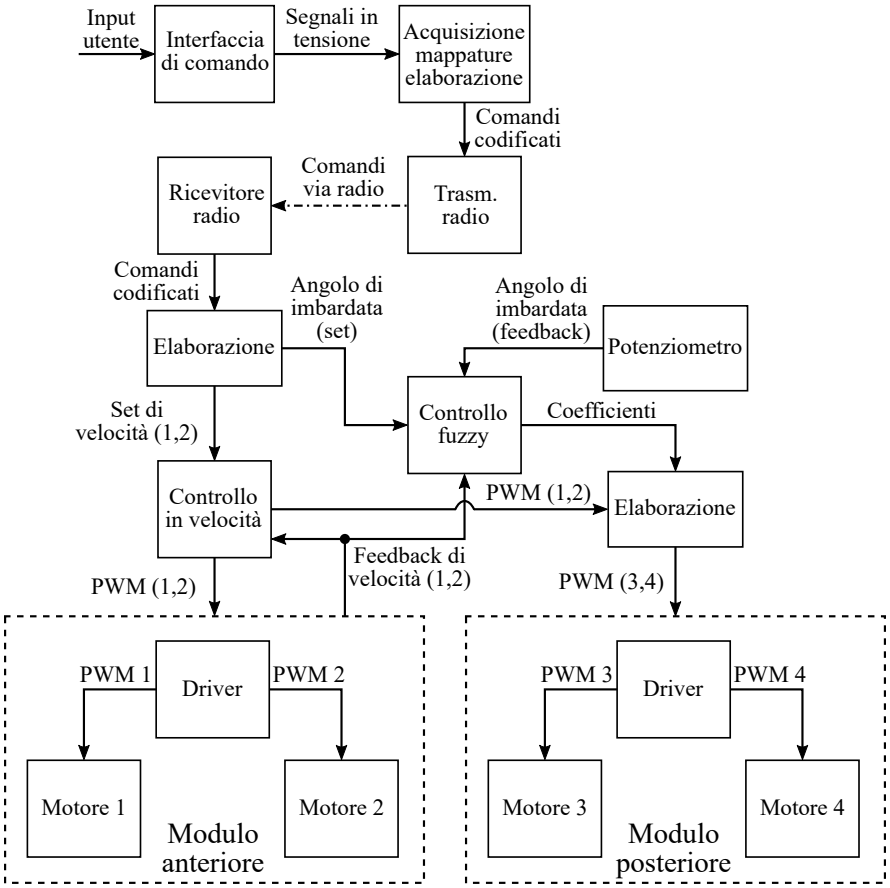


Figura 5.2: Schema del controllo

In figura 5.2 è riportato lo schema del controllo. La prima parte di tale processo non avviene nel robot, ma bensì nel microcontrollore del joystick, il quale sarà il dispositivo di input che prenderemo in considerazione.

A seguito della movimentazione delle leve analogiche vengono generati dei segnali in tensione che, opportunamente elaborati, generano a loro volta le variabili di comando che verranno inviate al robot tramite l'XBee.

Una volta ricevuti i comandi, il microcontrollore di *Epi.q* genera due riferimenti in velocità e un riferimento angolare che sono correlati fra loro. I set di velocità vengono inviati all'omonimo controllo da cui otteniamo i segnali PWM da inviare al driver. I due motori ci restituiscono il feedback di velocità che ci permette di chiudere l'anello. Tale feedback è anche un input del controllo fuzzy, il quale ha lo scopo di generare due coefficienti per il controllo del posteriore.

I segnali PWM del modulo anteriore vengono moltiplicati per due coefficienti e inviati al driver del modulo posteriore. Tali coefficienti vengono generati mediante l'utilizzo di

un controllo fuzzy, il quale prende in input l'errore angolare tra il set e il feedback, e la velocità con cui si sta muovendo il robot.

Si può inoltre scegliere una modalità per cui non viene utilizzato il controllo fuzzy per il calcolo dei coefficienti, ma essi vengono tenuti costanti.

5.2 Codifica dei comandi

Il robot prende in input tre variabili, le prime due definiscono il moto (λ, ν), mentre la terza trasporta informazioni ausiliarie.

Come detto in precedenza, la quantità di dati inviata ad ogni ciclo è pari a 3 byte, il che significa 1 byte per ogni variabile.

Le variabili di moto sono di tipo "int_8t", ovvero rappresentano un intero con segno in 8 bit. Siccome il primo bit serve appunto a definire il segno, tale variabile rappresenta i numeri nell'intervallo [-128, 127]. Per i nostri scopi l'intervallo è stato ridotto a [-125, 125].

La terza variabile denominata *cmd* è di tipo "uint_8t", quindi un intero senza segno. In questo caso si utilizza la variabile per veicolare i singoli bit, i quali attivano o disattivano i singoli controlli e veicolano informazioni sul funzionamento.

In figura 5.3 è riportato il significato di ogni bit. Partendo dal LSB (least significant bit), ovvero il bit più a destra, e numerandoli di conseguenza partendo da zero, abbiamo che i primi due bit rappresentano un numero che va da 0 a 3, il quale altro non è che il numero identificativo della mappatura attualmente utilizzata (la 0 è quella di default).

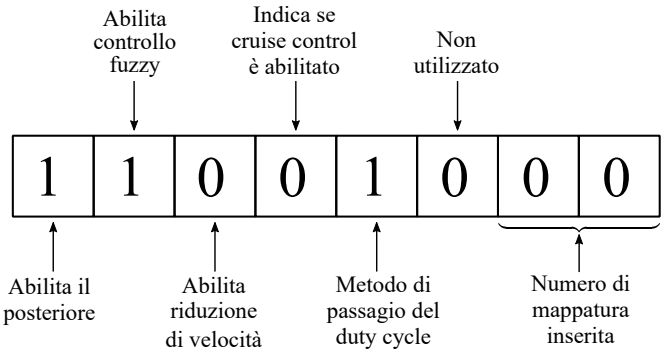


Figura 5.3: Variabile *cmd*

Il bit 2 non viene utilizzato, ed è quindi disponibile.

Il bit 3 ci permette di impostare il metodo di passaggio del duty cycle. Se è pari a 1, abbiamo la media, in caso contrario si ha il passaggio diretto.

Il bit 4 ci da unicamente informazione sul fatto che il cruise control[3] sia in funzione o meno, quindi non ha un effetto attivo sul controllo del robot.

Stesso discorso per il bit 5, il quale comunica se è attiva la modalità di riduzione della velocità massima. Tale modalità ha lo scopo di rendere il robot più governabile andando

[3]È una feature implementata nel joystick, simile al cruise control di una macchina.

a diminuire la velocità massima raggiungibile e di conseguenza si ha un aumento della sensibilità dell'analogico.

Il bit 6 abilita il controllo fuzzy del posteriore, nel caso sia disabilitato c'è un coefficiente costante pari a 0.9.

Infine troviamo il MSB (most significant bit) che consente di abilitare il posteriore. Nel caso in cui si decidesse di utilizzare solo l'anteriore i bit 3 e 6 non vengono presi in considerazione.

La configurazione di default con la quale viene avviato il robot è quella mostrata in figura ed è quella a cui faremo riferimento.

5.2.1 Variabili del moto

In figura 5.4 è riportato lo schema di riferimento per l'angolo di imbardata δ, che ci servirà nella seguente spiegazione. Tale angolo è positivo per curve antiorarie nel caso in cui il modulo frontale sia il modulo M1.

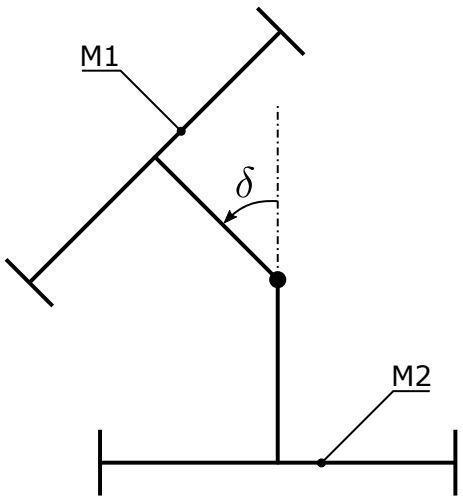


Figura 5.4: Schema di riferimento dell'angolo δ

Il robot riceve i comandi del moto per mezzo di due variabili λ e ν. La prima trasporta informazioni sulla velocità di avanzamento e la seconda sullo sbilanciamento di velocità, ovvero sulla curva da percorrere. Tali variabili danno informazioni anche sulla tipologia di moto (advancing o pivot).

Se $\lambda \neq 0$ siamo in modalità advancing e perciò la variabile ν è in realtà l'angolo di imbardata di riferimento. In caso contrario siamo in modalità pivot, e la variabile ν indica lo sbilanciamento di velocità tra il lato destro e il sinistro in modo tale che abbiano stesso modulo ma verso opposto.

Analizziamo prima questo secondo caso, essendo più semplice. Dato ν andiamo ad eseguire una mappatura lineare in modo tale da ottenere un set di velocità per il motore destro. Il set del motore sinistro è uguale con segno invertito. Ciò consente di generare rotazioni antiorarie nel caso per $\nu > 0$.

In questa modalità il posteriore entra in funzione solo in un certo range di angolo e il duty cycle viene copiato al posteriore con stesso modulo e segno invertito come mostrato in figura 5.5.

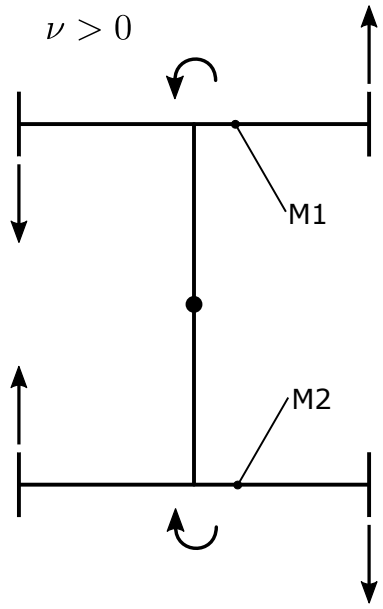


Figura 5.5: Modalità pivot

Nel momento in cui l'angolo δ supera un angolo limite si interrompe l'utilizzo del posteriore e si esegue una rotazione sul posto unicamente con l'anteriore. Attualmente l'angolo limite è pari a 25°, ricordando che l'angolo massimo è pari a 38°.

Questa modalità prevede uno slittamento nel momento in cui il giunto di imbardata va a contatto in una delle due configurazioni estreme. Lo slittamento delle ruote è fisiologico perché il centro di istantanea rotazione si trova a metà dell'interasse delle ruote anteriori, per cui il posteriore verrà trascinato nella rotazione (se non c'è troppo attrito tra le gomme e il terreno).

Inoltre, al fine di evitare manovre brusche, il set di velocità massimo è pari a 3000 rpm.

Comportamento totalmente differente si ha nella modalità advancing, in cui la variabile ν è correlata all'angolo δ. Riprendendo quanto detto nel paragrafo 2.3.2, utilizziamo tali correlazioni al fine di generare un set di velocità che eviti il più possibile lo slittamento delle ruote.

La variabile λ viene mappata nell'intervallo $[-n_{\max}, n_{\max}]$ e la variabile ν nell'intervallo $[-\delta_{\max}, \delta_{\max}]$. Quest'ultima, mediante la relazione 2.16, ci fornisce il coefficiente di curvatura che altro non è che il rapporto tra le velocità dei motori.

Sapendo che la mappatura di λ ci fornisce una velocità di avanzamento media teorica n_{m0}, e ricordando le relazioni 2.17 e 2.20 possiamo scrivere che:

$$n_{\mathrm{l0}} = \frac{2n_{\mathrm{m0}}}{K+1} \tag{5.1}$$

$$n_{\mathrm{r0}} = Kn_{\mathrm{l0}} \tag{5.2}$$

Questi due valori sono dei set di velocità teorici e non è detto che siano compatibili con le reali caratteristiche dei motori, eseguiamo una saturazione e verifichiamo se i valori siano o meno cambiati. Nel caso in cui ci sia una discrepanza, viene ricalcolato il set immutato in funzione del set che effettivamente è stato saturato, sapendo che il loro rapporto è pari a K. Fatto ciò otteniamo i due set di velocità n_{r} e n_{l}.

5.3 Controllo in velocità

In figura 5.6 è riportato lo schema del controllo in velocità implementato. Esso prevede un controllo PID con antiwindup e un feedforward. Quest'ultimo ha lo scopo di migliorare la risposta dinamica del sistema ed evitare che l'integrativa aumenti troppo di valore in condizioni stazionarie, visto che in questo sistema la maggior parte dell'errore viene compensato proprio da questa componente del PID.

Il sistema prende in input un set di velocità che viene saturato ai valori massimi raggiungibili dai motori. Il set viene inviato al feedforward, che mediante una funzione lineare restituisce una prima componente del comando.

La differenza tra set e feedback va in input alle componenti del PID. Le prime due, quella proporzionale e quella derivativa, si sommano al feedforward e passano all'interno di un saturatore dinamico, che in questo caso ha estremi costanti.

La componente integrativa passa invece per un secondo saturatore dinamico i cui estremi dipendono dall'uscita del primo saturatore.

Il valore in uscita viene diviso per il valore massimo di velocità raggiungibile in modo da ottenere un duty cycle il cui valore sarà nell'intervallo $[-1, 1]$. Il segno serve per determinare il verso di rotazione, cioè la direzione di avanzamento, mentre il modulo verrà convertito in un segnale PWM.

5.3.1 Antiwindup

Il fenomeno del windup si presenta quando siamo in dinamica satura, ovvero nella condizione in cui stiamo inviando il comando massimo ma il sistema non riesce comunque ad annullare l'errore. In questa situazione può risultare controproducente accumulare errore nell'integrativa perché il comando non subirà alcuna variazione, ma anzi ci sarà una finestra temporale in cui dovremo attendere che l'integrativa diminuisca il suo valore affinché il sistema torni al suo normale funzionamento.

Per ovviare a tale inconveniente risulta indispensabile evitare di accumulare errore durante i periodi di dinamica satura. Per fare ciò esistono diversi metodi. La nostra scelta è ricaduta su un sistema utilizzato nei controlli analogici, che con una piccola semplificazione è stato riadattato al controllo digitale.

Chiamando u_1 l'uscita del primo saturatore abbiamo che gli estremi del secondo saturatore sono pari a:

$$\mathrm{up} = n_{\max} - u_1$$
$$\mathrm{lo} = -n_{\max} - u_1$$

Nel caso in cui il valore di uscita del secondo saturatore u_2 sia diverso dal valore in ingresso, la quota parte eccedente, che non andrà ad aumentare il valore finale dell'uscita, viene rimandata all'ingresso dell'integrativa mediante un guadagno di disintegrazione k_{dis}.

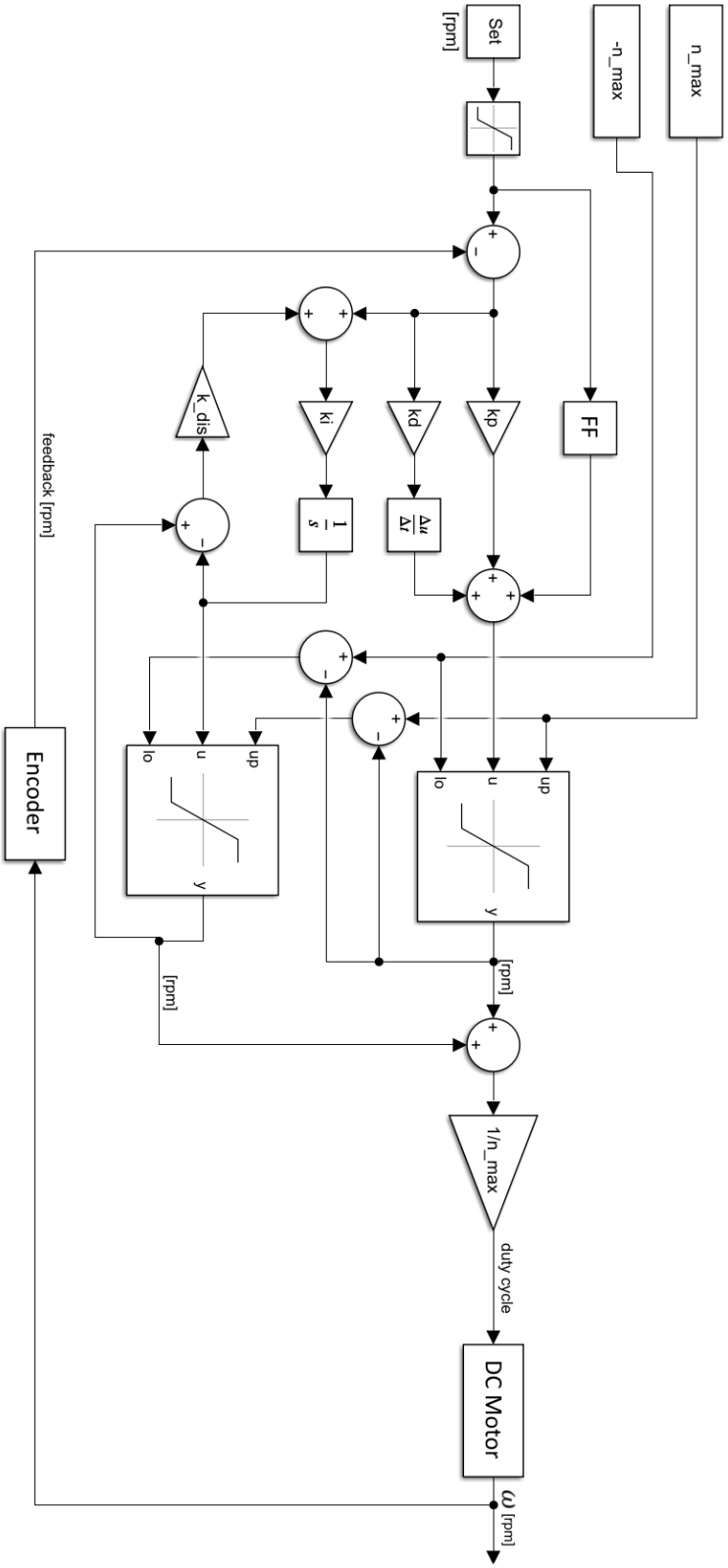


Figura 5.6: Schema del controllo in velocità

In campo analogico si ottengono buoni risultati se $k_{\text{dis}} = k_{\text{i}}$. In campo digitale, come nel nostro caso, non è necessario avere tale guadagno, ma basta semplicemente sostituire il valore dell'integrale, che sarà salvato in una variabile, con il valore u_2.

Questa metodologia che sfrutta i saturatori dinamici ci permette di vincolare il valore dell'integrativa solo nella direzione che ci interessa.

Sapendo che $n_{\max} = 10\,000\,\text{rpm}$ ed immaginando che u_1 sia completamente satura, quindi $u_1 = 10\,000\,\text{rpm}$, avremo che:

$$\text{up} = 10\,000 - 10\,000 = 0\,\text{rpm}$$
$$\text{lo} = -10\,000 - 10\,000 = -20\,000\,\text{rpm}$$

Ciò permette all'integrativa di diminuire il suo valore, ma non di aumentarlo, a differenza di altri metodi che prevedono un semplice blocco. Analizzando i dati si è visto infatti che in certe condizioni c'è una discrepanza tra i metodi, ed il primo ha un tempo di risposta migliore perché consente, anche se l'uscita u_1 è satura, di diminuire il valore finale dell'uscita.

5.3.2 Feedforward

Il feedforward ci consente di inviare un comando in anello aperto che va ad affiancare il PID. Tale funzione ha come unico input il set di velocità che noi imponiamo al motore.

Questo blocco aggiuntivo ci permette, oltre a migliorare i tempi di risposta, di evitare che sia compito dell'integrativa compensare la maggior parte dell'errore anche durante i transitori.

Per costruire la funzione di feedforward sono state eseguite delle prove sperimentali in cui a fronte di un ingresso noto si registrava l'uscita.

L'ingresso veniva fornito sotto forma di numero di giri riferiti all'albero motore, che poi erano divisi per il numero di giri massimo, andando a così ad ottenere un duty cycle.

Invertendo tale relazione ed eseguendo un fitting dei dati generiamo la funzione di feedforward. Il fitting è stato eseguito con un polinomio di primo grado. Vista la presenza del PID, non è necessaria una precisione troppo elevata, lo scopo infatti è avere un'uscita grezza che approssimi l'uscita finale.

In figura 5.7 sono riportati i dati sperimentali e il fitting eseguito. In ascissa troviamo la velocità angolare misurata tramite l'encoder e in ordinata il numero di giri che abbiamo inserito come ingresso, convertito poi in duty cycle.

Entrando in ascissa con il valore di set, otteniamo in ordinata il valore in uscita dal feedforward.

Come si può notare il motore inizia a ruotare con un duty cylce minimo pari a 0.5[4]. Questo valore non è uguale per tutti, ma varia da motore a motore, per tale ragione è stato eseguito un fitting per ognuno di essi.

Nella seguente tabella sono riportati tali coefficienti nella forma $y = mx + q$.

[4]Ricordando che il numero di giri in ingresso diviso per il numero di giri massimo, che è pari a 10 000 rpm, ci da il valore del duty cycle.

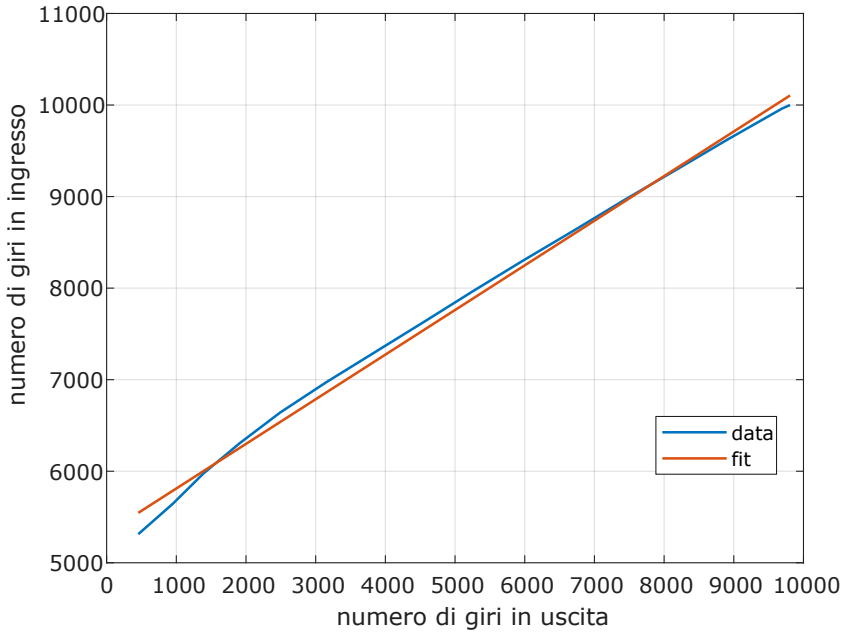


Figura 5.7: Feedforward

Tabella 5.1: Coefficienti del feedforward

Motore	m	q
Anteriore sinistro	0.5900	3909.48
Anteriore destro	0.5816	4221.38
Posteriore sinistro	0.4528	5535.57
Posteriore destro	0.6521	4006.65

5.3.3 Taratura del PID

Viste le evidenti non linearità presenti nei motori, e i diversi attriti presenti nei tripodi la taratura del PID è stata eseguita per via sperimentale, attraverso anche le sensazioni di guida che il robot forniva.

In primo luogo sono stati eseguiti gli step sotto riportati, fino ad arrivare a un buon compromesso, e in seguito sono stati fatti dei piccoli aggiustamenti.

1. Settare k_p basso k_i e k_d pari a zero.

2. Applicare un riferimento ad onda quadra di frequenza pari a circa il 10% della banda passante. L'ampiezza del riferimento deve essere grande ma non si deve andare in saturazione.

3. Aumentare k_p fino ad ottenere un 10% di overshoot. Se il sistema diventa rumoroso, si riduce k_p oppure si cerca di migliorare la qualità della retroazione.

4. Aumentare k_d fino ad eliminare l'overshoot.

5. Aumentare k_i fino ad ottenere un overshoot di circa 5-15%.

6. Eventuale verifica della banda passante con riferimento sinusoidale di frequenza variabile.

La frequenza dell'onda quadra è stata impostata a 0.1 Hz e l'ampiezza pari a 3500 rpm.
In figura 5.8 è riportata la risposta del motore dopo aver eseguito la taratura. Come si può vedere il comportamento dinamico è buono e non vi sono overshoot eccessivi.

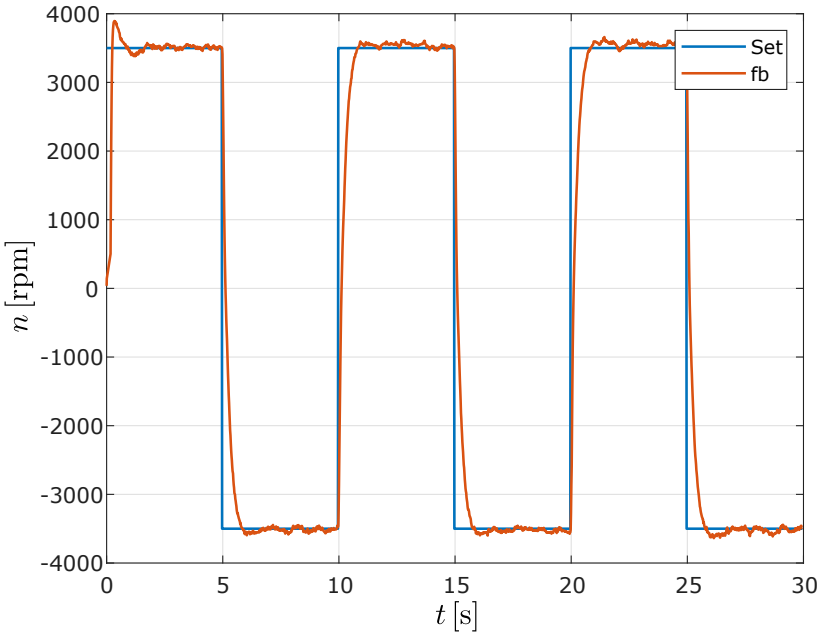


Figura 5.8: Risposta ad onda quadra dopo taratura

Nella tabella 5.2 sono riportati i valori finali dei guadagni del pid:

Tabella 5.2: Guadagni del PID

Motore	k_p (-)	k_i (s^{-1})	k_d (s)
Anteriore sinistro	3.0	15.0	0.03
Anteriore destro	3.1	14.9	0.03
Posteriore sinistro	3.2	15.1	0.03
Posteriore destro	3.2	15.1	0.03

5.4 Controllo del posteriore

Per la movimentazione del modulo posteriore (temporaneo) si è scelto di non usare un controllo in coppia tramite la misura della corrente perché il sensore presente all'interno del driver soffre di disturbi elettromagnetici. Ciò rende le misure estremamente rumorose e poco affidabili, a tal punto che diventerebbe complicato tarare il sistema.

Il controllo viene perciò fatto attraverso il passaggio del duty cycle imposto al modulo anteriore. Questo passaggio può essere fatto o direttamente, quindi il motore sinistro riceve il valore dal motore sinistro e il destro dal destro, o tramite una media, cioè lo stesso valore per entrambi.

Si è visto che il passaggio diretto non porta buoni risultati perché rende decisamente instabile il movimento del robot, il quale tende a posizionarsi in una configurazione di sterzata e difficilmente si riporta ad una di normale avanzamento.

Tramite il passaggio del valore medio il comportamento è migliore, ma si fa ancora fatica a controllare il robot perché tende a derivare a destra o a sinistra e difficilmente mantiene la traiettoria rettilinea.

Il problema è stato parzialmente risolto utilizzando un coefficiente costante di scala che diminuisce il valore imposto, in modo tale che il posteriore fornisca sì un contributo attivo, ma limitato a movimentare il suo peso e non gravare troppo sull'anteriore.

Nel primo caso il posteriore possiede una differenza di spinta tra il lato destro e il sinistro che tende a far ruotare lo stesso. nel secondo caso invece è il modulo anteriore che si occupa di ruotare il posteriore, il quale tenderebbe unicamente ad andare dritto.

Al fine di ottenere un controllo attivo e limitare il problema della deriva si è deciso di implementare un controllo fuzzy che tenga conto dei comandi inviati al robot e della misura angolare acquisita tramite il potenziometro.

Il controllo fuzzy, che vedremo in seguito nel dettaglio, genera due coefficienti moltiplicativi che modulano il duty cycle imposto al retrotreno. Il passaggio del comando non avviene più come mera copia, ma tramite una funzione.

Si è scelto di utilizzare questo tipo di controllo perché è adatto a gestire molteplici input in sistemi non lineari. Nel nostro caso infatti si sfrutterà questa proprietà per definire comportamenti diversi a seconda della velocità di marcia del robot.

5.4.1 Configurazioni di moto

Riprendendo la figura 5.4 che ci mostra la configurazione di base, ovvero quella in cui il modulo M1 è di trazione e l'angolo δ è positivo, possiamo ricavare sei condizioni di moto simmetriche tra loro.

Al fine di semplificare la trattazione ipotizziamo che il comando λ fornito sia positivo, in modo tale che il modulo M1 sia di trazione e sia controllato in velocità.

Dalla variabile ν otteniamo il valore di δ, che per noi sarà un valore di set, e dalla lettura del potenziometro otteniamo il feedback dell'angolo chiamato δ_{fb}, da cui possiamo calcolare l'errore $e = \delta - \delta_{\text{fb}}$.

In figura 5.9 sono riportati i primi due casi. La lunghezza delle frecce sta ad indicare l'intensità del PWM. Il duty cycle imposto all'anteriore deriva dal controllo in velocità, mentre al posteriore viene ipotizzato un valore qualitativo a seconda della condizione.

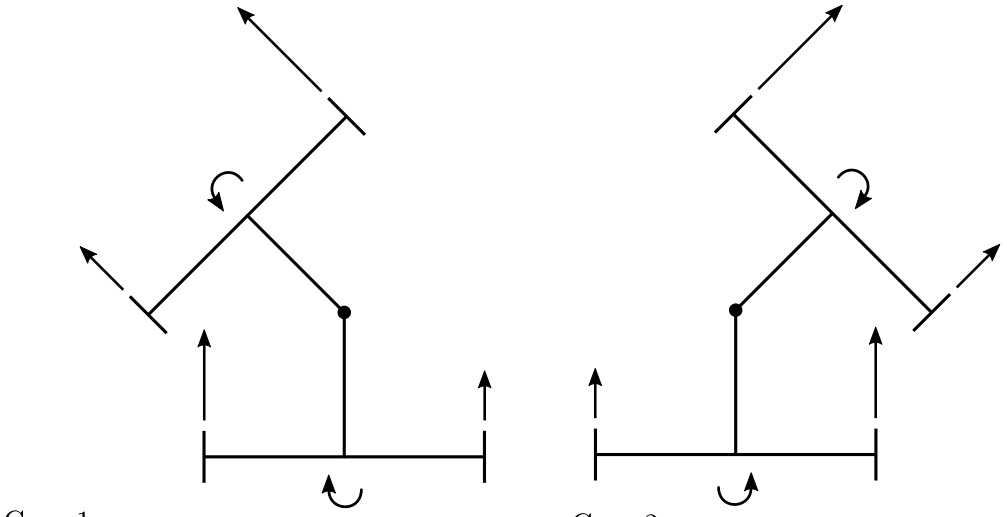


Figura 5.9: Configurazione di moto: casi 1 e 2

Prendendo ad esempio il caso 1, si ha che la curva imposta è antioraria e che $\delta > \delta_{\text{fb}}$, quindi il nostro obiettivo è far sì che il posteriore aiuti il robot a percorrere una curva con un raggio minore. Per far ciò si aumenta il PWM della ruota interna e si diminuisce quello sull'esterna andando a generare così una coppia che ruota il retrotreno, in aggiunta al traino fornito dall'avantreno.

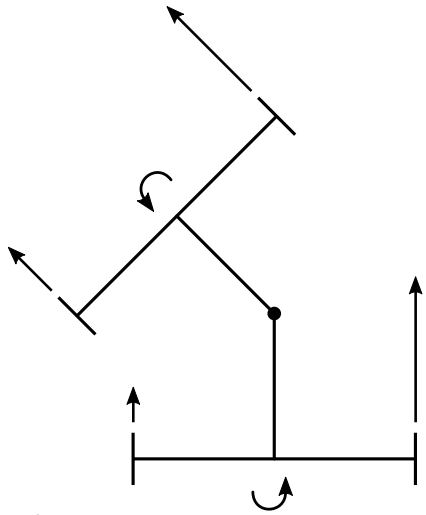
Situazione completamente simmetrica si ha nel secondo caso. Come si può notare i segni sono invertiti, ma le azioni da intraprendere sono esattamente le stesse.

La ruota che viene considerata come interna è funzione del segno di δ_{fb}. Se $\delta_{\text{fb}} > 0$ la posteriore interna sarà la ruota sinistra, in caso contrario sarà la destra.

In figura 5.10 si hanno delle condizioni per cui si sta percorrendo una curva con un raggio troppo piccolo e bisogna aumentarlo. Per fare ciò si invertono le azioni rispetto ai casi precedenti, ovvero si diminuisce il duty cycle della ruota interna e si aumenta quello della ruota esterna, andando a generare una coppia che ruoti il posteriore nello stesso verso dell'anteriore, ma con un'intensità maggiore.

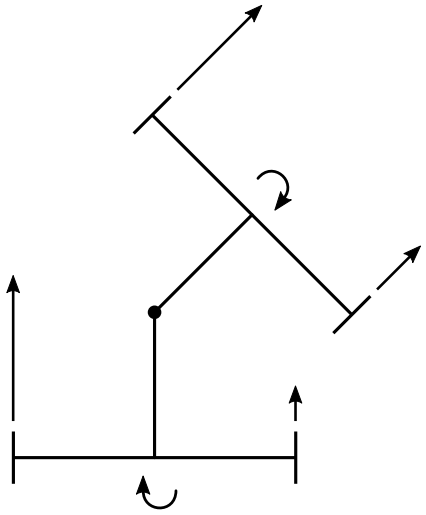
I casi 5 e 6 (figura 5.11) si riferiscono alla condizione di inversione di curva a bassa velocità e alla condizione di partenza da fermo. Al fine di minimizzare il transitorio si applica un duty cycle negativo alla posteriore interna in modo tale che il retrotreno si posizioni nella giusta configurazione. Nel momento in cui l'angolo di feedback cambia segno ci si riporta ai casi 1 e 2 rispettivamente.

Quest'ultima condizione è molto utile perché consente di allineare velocemente i due telai, in uno spazio di circa 20-30 cm. Si è visto sperimentalmente che altrimenti il robot impiegherebbe più di un metro per compiere tale operazione.



Caso 3:

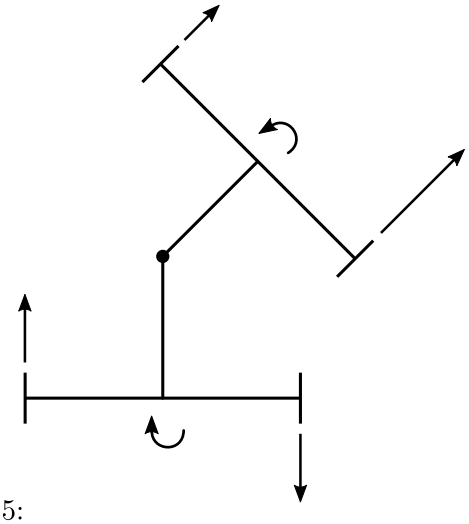
$$\begin{cases} \delta > 0 \\ \delta_{\text{fb}} > 0 \\ e < 0 \end{cases} \Longrightarrow \begin{cases} \text{Allargo la curva} \\ \text{PWM post. interna} \downarrow \\ \text{PWM post. esterna} \uparrow \end{cases}$$



Caso 4:

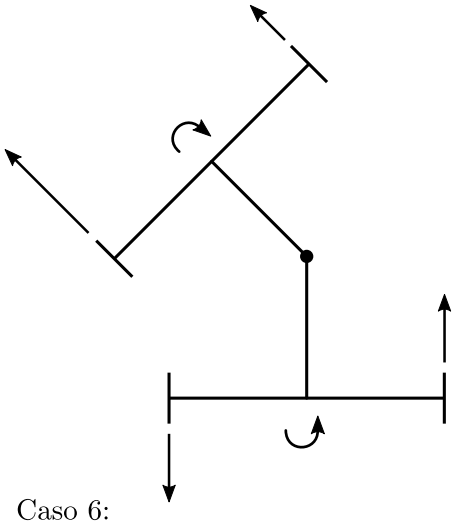
$$\begin{cases} \delta < 0 \\ \delta_{\text{fb}} < 0 \\ e > 0 \end{cases} \Longrightarrow \begin{cases} \text{Allargo la curva} \\ \text{PWM post. interna} \downarrow \\ \text{PWM post. esterna} \uparrow \end{cases}$$

Figura 5.10: Configurazione di moto: casi 3 e 4



Caso 5:

$$\begin{cases} \delta > 0 \\ \delta_{\text{fb}} < 0 \\ e > 0 \end{cases} \Longrightarrow \begin{cases} \text{Stringo la curva} \\ \text{PWM post. interna} \downarrow \\ \text{PWM post. esterna} \uparrow \end{cases}$$



Caso 6:

$$\begin{cases} \delta < 0 \\ \delta_{\text{fb}} > 0 \\ e < 0 \end{cases} \Longrightarrow \begin{cases} \text{Stringo la curva} \\ \text{PWM post. interna} \downarrow \\ \text{PWM post. esterna} \uparrow \end{cases}$$

Figura 5.11: Configurazione di moto: casi 5 e 6

Sfruttando la simmetria è possibile ridurre i casi da sei a tre e per fare ciò è sufficiente scrivere l'errore nel seguente modo:

$$\overline{e} = \text{sign}(\delta_{\text{fb}}) \frac{\delta - \delta_{\text{fb}}}{\delta_{\text{max}}} \tag{5.3}$$

Tale errore è inoltre normalizzato sul valore dell'angolo massimo, quindi il suo intervallo è pari a [-2, 2].

L'altro input che andrà a concorrere nel controllo fuzzy è il modulo della velocità media dell'anteriore, anch'esso normalizzato.

Gli schemi qui presentati serviranno per la costruzione delle regole.

5.4.2 Logica di controllo fuzzy

La logica fuzzy è una teoria logico-matematica che si sviluppa a partire dalla teoria degli insiemi negli anni '60 [13][5].

Si differenzia dalla logica tradizionale poiché propone un certo grado di verità di un enunciato invece di affermare che l'enunciato è semplicemente vero oppure falso. La parola fuzzy significa letteralmente "sfumato", "sfocato".

Essa tende ad eliminare la discontinuità che si ha con la logica tradizionale o binaria introducendo un fuzzy set, cioè una classe di appartenenza con un continuo grado di appartenenza o di membership compreso fra 0 e 1 e una funzione di appartenenza (o membership function) che definisce tale grado in funzione del valore della variabile misurata.

Per fare un esempio, in figura 5.12 sono rappresentati i gradi di verità dell'enunciato "La pressione è alta" in funzione del valore della pressione variabile nel campo $0 < p \leq 10$ bar secondo la logica tradizionale e secondo la logica fuzzy. Date la classe di appartenenza o fuzzy set *alta* la funzione $\mu_{\text{alta}}(p)$ esprime il grado di attivazione di p per la classe *alta*. Nel caso di figura b tale funzione è lineare e definisce i seguenti gradi di appartenenza della pressione alla classe alta nonché tutti i valori per le pressioni intermedie:

$$\mu_{\text{alta}}(4) = 0$$
$$\mu_{\text{alta}}(5) = 0.5$$
$$\mu_{\text{alta}}(6) = 1$$

Le funzioni di appartenenza o membership functions μ definiscono il grado di attivazione di una classe. Le forme di tali funzioni possono essere diverse: per citare alcuni esempi più diffusi si possono avere funzioni a triangolo, a trapezio, lineari a tratti oppure gaussiane.

L'operazione di conversione di valori reali di una variabile, come ad esempio la temperatura, la pressione, la velocità ecc... in valori fuzzy mediante le funzioni di appartenenza è detta fuzzificazione.

Sono state definite alcune operazioni fra gli insiemi fuzzy o fuzzy set come estensione delle operazioni booleane tradizionali. Nella figura 5.13 sono illustrate le operazioni di somma o unione (OR), di prodotto o intersezione (AND) e di negazione o complemento (NOT).

[5] La trattazione teorica e le figure 5.12, 5.13, 5.14, 5.21 e 5.23 sono state prese dalla medesima fonte.

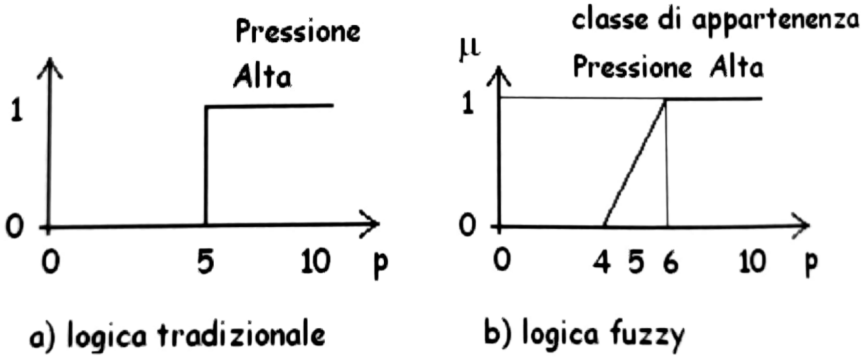


Figura 5.12: Confronto fra logica tradizionale e logica fuzzy

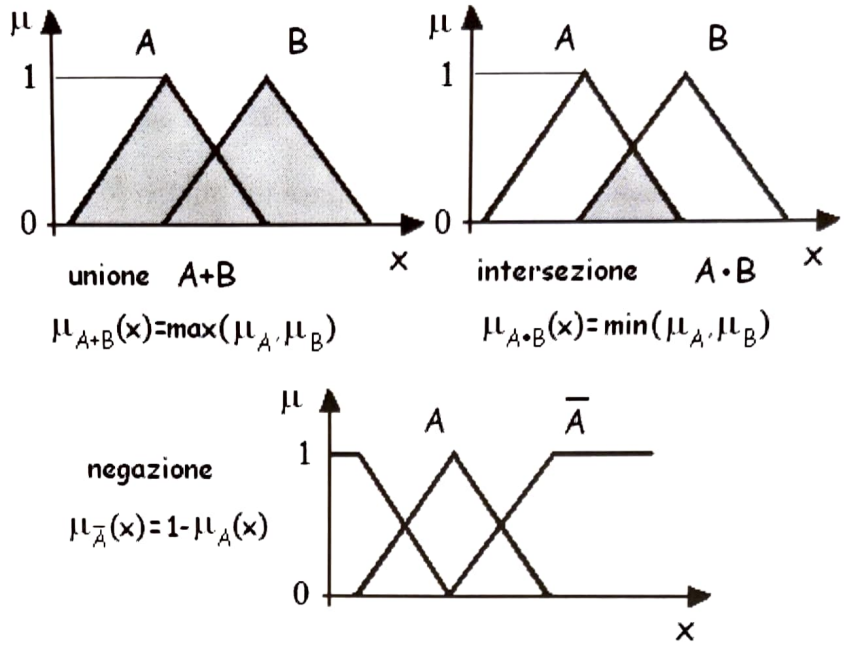


Figura 5.13: Definizione delle operazioni logiche fuzzy OR, AND e NOT

Lo schema tipico di un controllo fuzzy è costituito da tre moduli principali: modulo di fuzzificazione, di applicazione delle regole e di defuzzificazione (figura 5.14). Sia gli ingressi che le uscite sono specificate da valori numerici o *crisp*, mentre il regolatore fuzzy opera su variabili fuzzy, dunque si hanno delle interfacce di fuzzificazione e di defuzzificazione che consentono di associare agli ingressi la loro rappresentazione fuzzy e alla rappresentazione fuzzy dell'uscita l'uscita stessa. La parte centrale dello schema presentato trasforma le rappresentazioni fuzzy degli ingressi X nelle rappresentazioni fuzzy delle uscite Y secondo le regole della logica fuzzy, denominate anche regole di inferenza. L'operazione di defuzzificazione converte i valori fuzzificati Y nei valori numerici o *crisp* dei segnali delle

uscite.

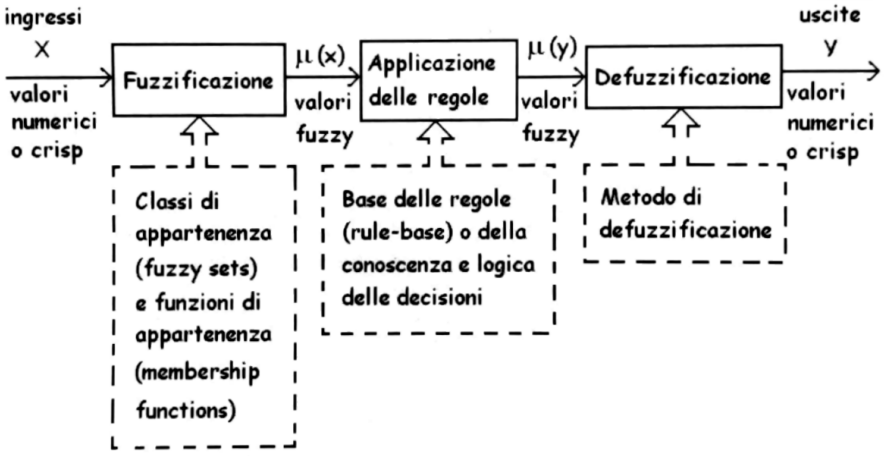


Figura 5.14: Schema tipico di un controllo fuzzy

5.4.3 Definizione dei fuzzy sets

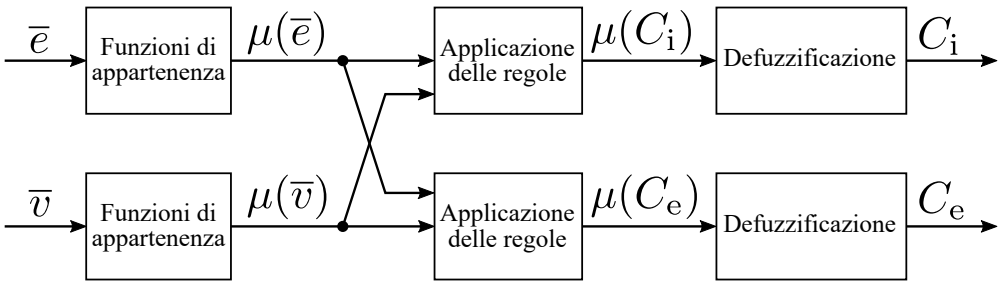


Figura 5.15: Schema funzionale del controllo fuzzy

In figura 5.15 è riportato lo schema del fuzzy implementato. Esso ha due input: l'errore normalizzato $\overline{e}$ e la velocità media normalizzata $\overline{v}$; definiti dalle seguenti equazioni:

$$\overline{e} = \mathrm{sign}(\delta_\mathrm{fb}) \, \frac{\delta - \delta_\mathrm{fb}}{\delta_\mathrm{max}} \tag{5.4}$$

$$\overline{v} = \left| \frac{n_\mathrm{r} + n_\mathrm{l}}{2\, n_\mathrm{max}} \right| \tag{5.5}$$

La velocità media normalizzata viene sempre calcolata tramite il modulo controllato in velocità.

Entrambi gli input concorrono a generare i due output C_i e C_e, che sono rispettivamente il coefficiente moltiplicativo del duty cycle della ruota interna e il coefficiente della ruota esterna, motivo per il quale sono state definite due matrici delle regole, una per ogni uscita.

Tali matrici sono indipendenti tra di loro perché la prima attiva le classi di appartenenza di C_i, mentre l'altra quelle di C_e, nonostante condividano gli input.

Il numero di classi dei singoli fuzzy set è stato definito sulla base dell'ampiezza dell'intervallo, in modo da garantire che ogni singola classe abbia il giusto spazio senza però appesantire troppo le matrici delle regole.

Fuzzy sets di input

Per il primo input $\overline{e}$ sono state definite cinque classi, di cui tre di forma triangolare e due di forma trapezoidale (figura 5.16).

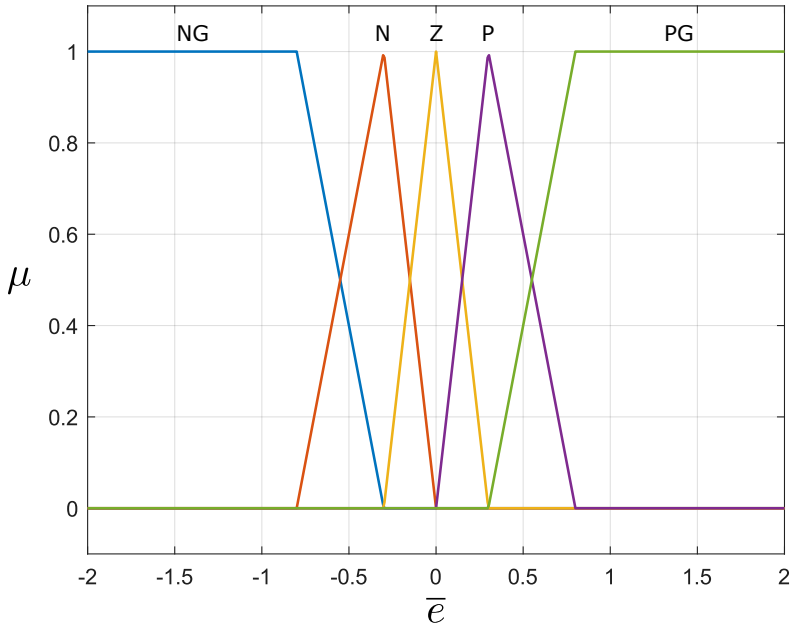


Figura 5.16: Classi di appartenenza dell'errore normalizzato

I nomi delle classi stanno rispettivamente per: negativo grande, negativo, zero, positivo, positivo grande[6]. Esse sono molto schiacciate perché l'intervallo in cui generalmente si muove l'errore è [-1, 1].

Nel caso della velocità (figura 5.17) è stato sufficiente definire tre classi di appartenenza: bassa, media, alta.

Fuzzy sets di output

La scelta delle classi di appartenenza dei due coefficienti è stata fatta sulla base dell'ampiezza del loro intervallo, la quale è in funzione delle configurazioni di moto presentate in precedenza.

[6]Per la scelta della nomenclatura si è fatto riferimento a quella utilizzata in bibliografia [13].

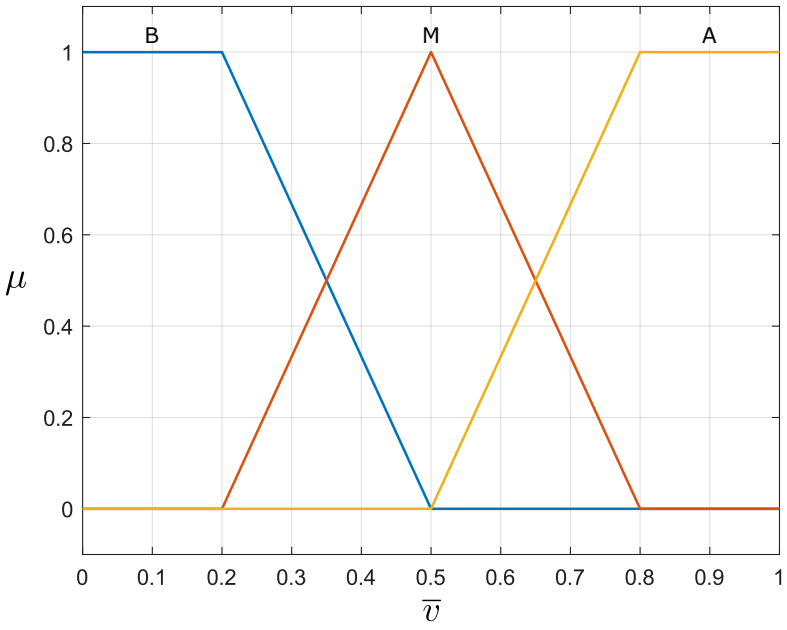


Figura 5.17: Classi di appartenenza della velocità media normalizzata

Il coefficiente di ruota interna può assumere anche valori negativi motivo per il quale ha bisogno di più classi rispetto all'altro.

Nello specifico si ha che C_i varia nell'intervallo [-0.8, 1.4], mentre C_e varia nell'intervallo [0.4, 1.4]. Di conseguenza il primo ha otto classi di appartenenza, mentre il secondo ne ha cinque.

Nelle figure 5.18 e 5.19 sono riportate tali classi. I nomi sono comuni fra le due e stanno per: negativo grande, negativo, zero, riduzione grande, riduzione, positivo, positivo grande, positivo molto grande.

5.4.4 Definizione delle regole

Il controllo fuzzy prevede la scrittura di regole nella forma SE (condizione) ALLORA (azione). Ciò significa che se si attiva un certo input, tramite le regole, si attiva un certo output.

Il metodo più semplice per visualizzare tali regole è mediante una matrice, chiamata *matrice delle regole*, la quale associa ad ogni coppia delle classi di ingresso, una classe di uscita. Avendo due output distinti sono state realizzate due matrici distinte.

Nella tabella 5.3 sono riportate le due matrici. Nel lato sinistro le regole hanno lo scopo di allargare la traiettoria percorsa, mentre nel lato destro accade l'opposto.

Tali regole, soprattutto per la ruota interna, dipendono molto dalla velocità per il fatto che a bassi valori è necessario avere comandi più aggressivi per ottenere una buona compensazione.

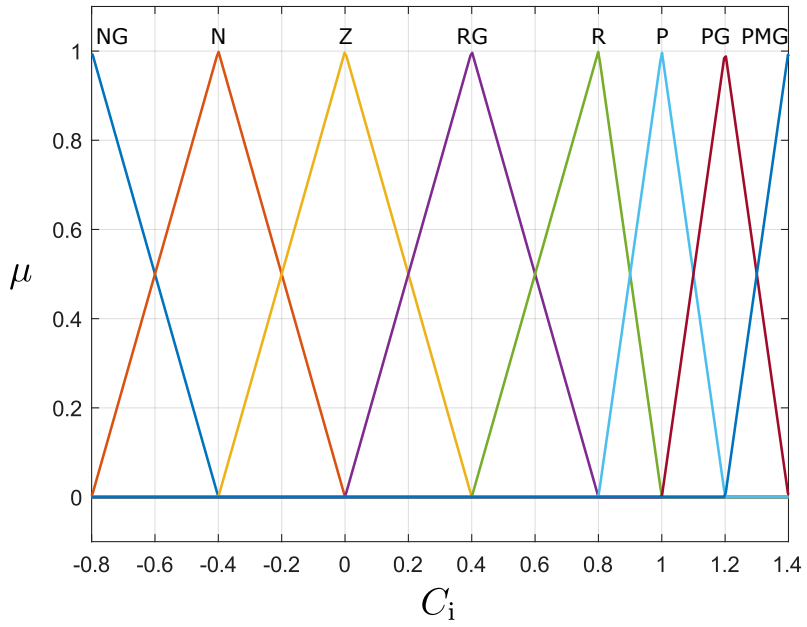


Figura 5.18: Classi di appartenenza del coefficiente della ruota interna

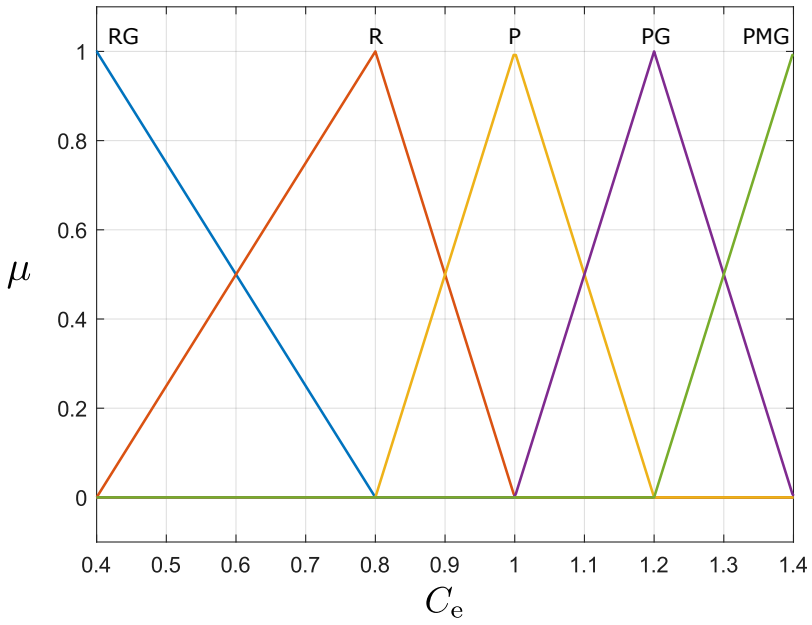


Figura 5.19: Classi di appartenenza del coefficiente della ruota esterna

Tabella 5.3: Matrici delle regole

C_i		$\overline{e}$				
		NG	**N**	**Z**	**P**	**PG**
$\overline{v}$	**B**	NG	N	P	PG	PMG
	M	Z	RG	P	PG	PMG
	A	RG	R	P	PG	PMG

C_e		$\overline{e}$				
		NG	**N**	**Z**	**P**	**PG**
$\overline{v}$	**B**	PMG	PG	P	R	RG
	M	PMG	PG	P	R	R
	A	PMG	PG	P	R	R
		Allargo la curva			Stringo la curva	

5.4.5 Processo di controllo fuzzy

Al fine di semplificare la spiegazione del processo interno del controllo fuzzy ci limitiamo a considerare una sola uscita, il coefficiente C_i.

Dopo aver definito i fuzzy set, la fuzzificazione prevede il calcolo dei gradi di appartenenza delle singole classi a fronte di un ingresso noto.

Prendendo un caso di esempio con i seguenti input:

$$\overline{e} = -0.7$$
$$\overline{v} = 0.6$$

abbiamo che i gradi di attivazione (figura 5.20) sono:

$$\mu_\text{NG}(\overline{e}) = 0.8,\ \mu_\text{N}(\overline{e}) = 0.2,\ \mu_\text{Z}(\overline{e}) = 0,\ \mu_\text{P}(\overline{e}) = 0,\ \mu_\text{PG}(\overline{e}) = 0$$
$$\mu_\text{B}(\overline{v}) = 0,\ \mu_\text{M}(\overline{v}) = 2/3,\ \mu_\text{A}(\overline{v}) = 1/3$$

Una volta calcolati i gradi di attivazione delle classi delle variabili di ingresso occorre calcolare il grado di attivazione delle classi dell'uscita. Per far ciò occorre considerare tutte le regole, ossia tutte le combinazioni possibili degli ingressi definite dalla matrice delle regole e applicare l'operazione di inferenza. Innanzitutto si applica l'operazione di intersezione di insiemi fuzzy (che comporta il calcolo del valore minimo) da cui si ottiene il valore delle premesse μ_p secondo la formula:

$$\mu_\text{p}(\overline{e}_i, \overline{v}_j) = \min(\mu_i(\overline{e}), \mu_j(\overline{v})) \tag{5.6}$$

Dove i e j sono le classi di appartenenza degli ingressi.

Ad esempio:

$$\mu_\text{p}(\overline{e}_\text{N}, \overline{v}_\text{M}) = \min(\mu_\text{N}(\overline{e}), \mu_\text{M}(\overline{v})) = \min(0.2,\ 2/3) = 0.2$$
$$\mu_\text{p}(\overline{e}_\text{Z}, \overline{v}_\text{M}) = \min(\mu_\text{Z}(\overline{e}), \mu_\text{M}(\overline{v})) = \min(0,\ 2/3) = 0$$

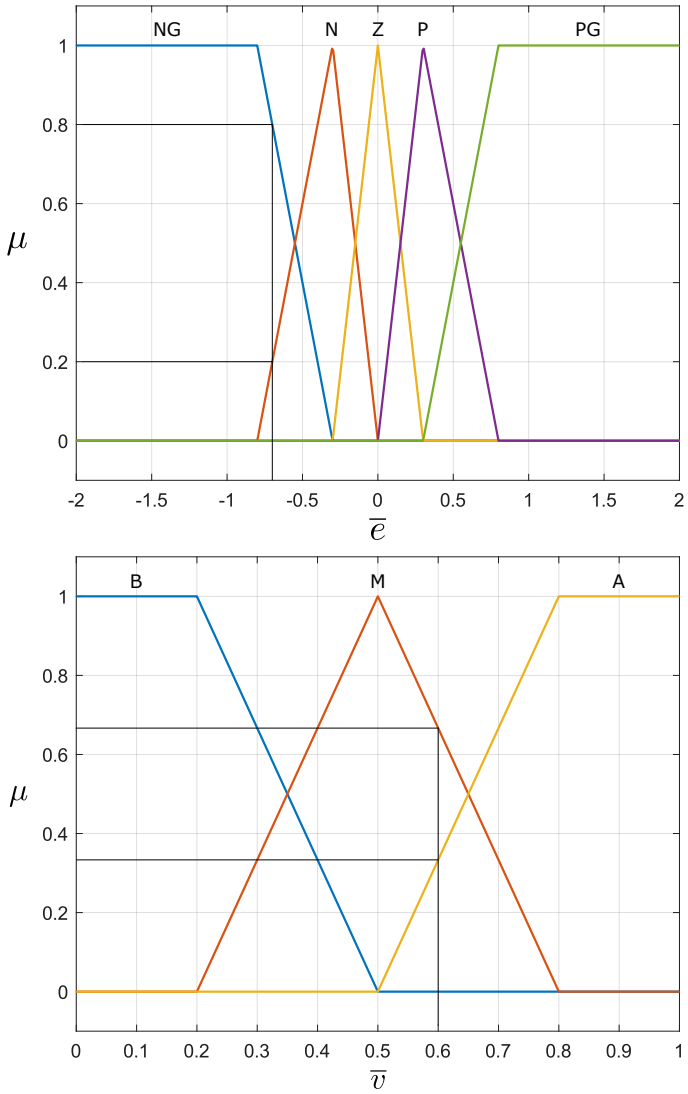


Figura 5.20: Calcolo dei gradi di attivazione

Quindi per il calcolo del grado di attivazione di ciascuna classe dell'uscita si applica l'operazione di unione di insiemi fuzzy (che comporta il calcolo del valore massimo) alle premesse secondo la formula seguente:

$$\mu(u_k) = \max(\mu_{\mathrm{p}}(\overline{e_i}, \overline{v_j})) = \max[\min(\mu_i(\overline{e}), \mu_j(\overline{v}))] \tag{5.7}$$

dove k indica la classe di appartenenza dell'uscita[7] e gli indici i e j individuano le premesse relative alla classe k di uscita secondo la matrice delle regole.

[7] Avendo due uscite separate si è utilizzata una u generica.

Continuando l'esempio numerico, nella tabella 5.4 sono riportati anche i gradi di attivazione degli ingressi $\overline{e}$ e $\overline{v}$ e nelle celle i valori delle premesse μ_{p} cui sono abbinate le classi dell'uscita.

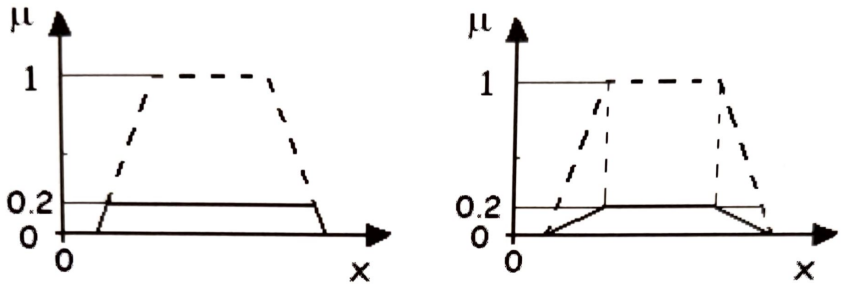
Tabella 5.4: Matrice delle regole con i valori delle premesse

C_{i}		$\overline{e}$				
		0.8 **NG**	0.2 **N**	0 **Z**	0 **P**	0 **PG**
$\overline{v}$	0 **B**	0 NG	0 N	0 P	0 PG	0 PMG
	2/3 **M**	2/3 Z	0.2 RG	0 P	0 PG	0 PMG
	1/3 **A**	1/3 RG	0.2 R	0 P	0 PG	0 PMG

Dalla tabella si evince che le classi dell'uscita che vengono attivate in questo caso sono Z, RG e R, in quanto nelle altre caselle i valori delle premesse sono nulli. Per le classi dell'uscita Z e R il grado di attivazione è univoco, in quanto le relative premesse sono tutte nulle tranne che in un caso. In questo caso non si deve neanche calcolare il valore massimo. Per la classe RG si hanno invece due premesse diverse da zero, dunque occorre applicare l'operazione di unione fuzzy per ottenere il grado di attivazione risultante. Risulta $\mu_{\mathrm{RG}}(C_{\mathrm{i}}) = 1/3$, che è appunto il valore maggiore fra 0.2 e 1/3. Si sono dunque ottenuti, per ogni classe dell'uscita, i relativi gradi di attivazione. Per ottenere la rappresentazione fuzzy dell'uscita C_{i} occorre applicare uno dei seguenti metodi alternativi:

- metodo di *clipping* o *alpha-cut*
- metodo di *scaling*

primo metodo (figura 5.21a) consiste nel limitare superiormente la funzione di appartenenza dell'uscita al grado di attivazione ottenuto per l'uscita. Il secondo metodo (figura 5.21b) consiste in una semplice scalatura della funzione di una costante pari al grado di attivazione dell'uscita.



(a) clipping o alpha-cut (b) scaling

Figura 5.21: Calcolo del grado di attivazione

Il primo metodo è il più diffuso e presenta il vantaggio della semplicità, comporta un'alta velocità di calcolo e semplifica anche l'operazione di defuzzificazione. Lo svantaggio è che si perde una quantità di informazione nell'operazione di taglio dovuta al cambiamento della forma della funzione una volta operato quest'ultimo. Ad esempio, una funzione della forma triangolare si trasforma in una funzione trapezoidale.

Il vantaggio del secondo metodo consiste nella conservazione dell'informazione mantenendo la forma della funzione di appartenenza. Il metodo può essere molto utile nei sistemi esperti fuzzy[8].

A questo punto è possibile effettuare l'operazione di aggregazione dei risultati ottenuti per elaborare l'unico fuzzy set per la variabile di uscita. Le classi dell'uscita che risultano attivate sono Z, RG e R con gradi di attivazione rispettivamente 2/3, 1/3 e 0.2. In figura 5.22 è visibile l'unico fuzzy set ottenuto dall'aggregazione delle tre classi dopo aver applicato il metodo *alpha-cut*.

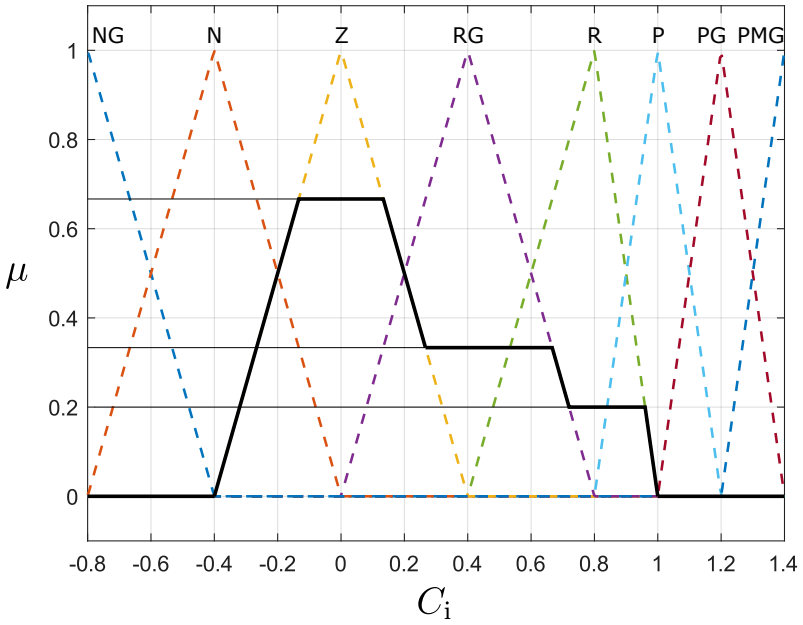


Figura 5.22: Fuzzy set di uscita

La defuzzificazione consiste nella conversione del fuzzy set di uscita in un valore numerico o *crisp value*. Tale conversione può avvenire mediante diversi metodi di defuzzificazione che associano alla rappresentazione fuzzy dell'uscita un valore deterministico. I metodi più usati nella pratica sono:

- metodo del centro di gravità;

[8]Un sistema esperto è un software che cerca di riprodurre le prestazioni di una o più persone esperte in un determinato campo di attività, ed è un'applicazione dell'intelligenza artificiale (Wikipedia).

- metodo dei massimi;

Il secondo è di più semplice applicazione ma può presentare delle discontinuità, oltre a una minore precisione. Esso si divide in tre varianti: FOM (*first of maximum*), LOM (*last of maximum*) e MOM (*middle of maxima*). Per quanto riguarda il metodo FOM il crisp value del segnale di uscita corrisponde all'ascissa del primo massimo della funzione di appartenenza del segnale di uscita, ossia quella più a sinistra. Il metodo LOM invece considera l'ultima ascissa del massimo, quindi quella più a destra. Il metodo MOM considera la media di questi due valori (figura 5.23).

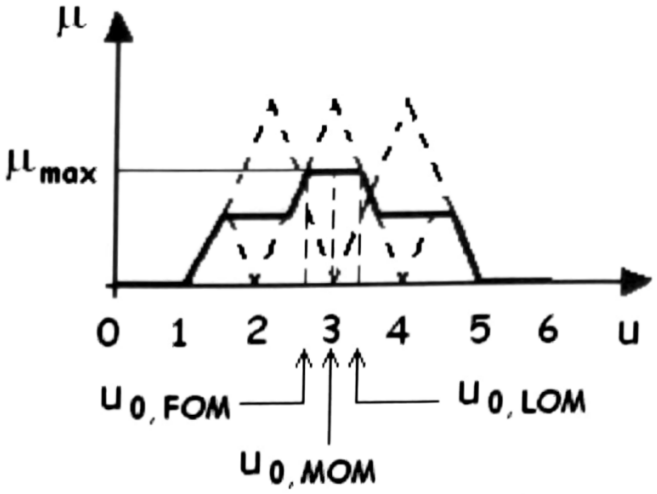


Figura 5.23: Casi di defuzzificazione FOM, MOM, LOM

Il metodo del centro di gravità è uno dei più utilizzati. Esso prevede che l'uscita sia l'ascissa del centro di gravità del fuzzy set. Nel caso continuo si utilizza la formula seguente:

$$u_0 = \frac{\int u\,\mu(u)\,du}{\int \mu(u)\,du} \tag{5.8}$$

mentre nel caso discreto si ha una serie:

$$u_0 = \frac{\sum_i u_i\,\mu(u_i)}{\sum_i \mu(u_i)} \tag{5.9}$$

L'equazione 5.9 consente di semplificare i calcoli, ma di conseguenza ha una precisione minore. Quest'ultima è fortemente legata al passo scelto, più questo è piccolo più la serie convergerà all'integrale.

Nel nostro caso si utilizza il metodo del centro di massa nella versione continua[9], il quale fornisce la massima precisione e la migliore stabilità a fronte di una piccola variazione dell'ingresso.

[9]Si utilizza questa versione perché la libreria che implementa la logica fuzzy ha implementato questo metodo. La libreria è disponibile a questo link: https://github.com/alvesoaj/eFLL.

Il metodo di inferenza qui presentato è quello di Mamdani, ma esistono anche metodi differenti. Uno fra tutti è il metodo di Sugeno, il quale è molto simile al precedente per quanto riguarda la fuzzificazione e l'applicazione delle regole. La differenza si ha nella defuzzificazione. Tale metodologia prevede infatti che le classi di uscita vengano trasformate in dei *singleton*, quindi abbiano un unico valore dell'uscita. Il valore finale viene calcolato come media pesata delle classi.

Applicando al nostro esempio il metodo del baricentro (figura 5.24) otteniamo come valore finale del coefficiente $C_\mathrm{i} = 0.2055$.

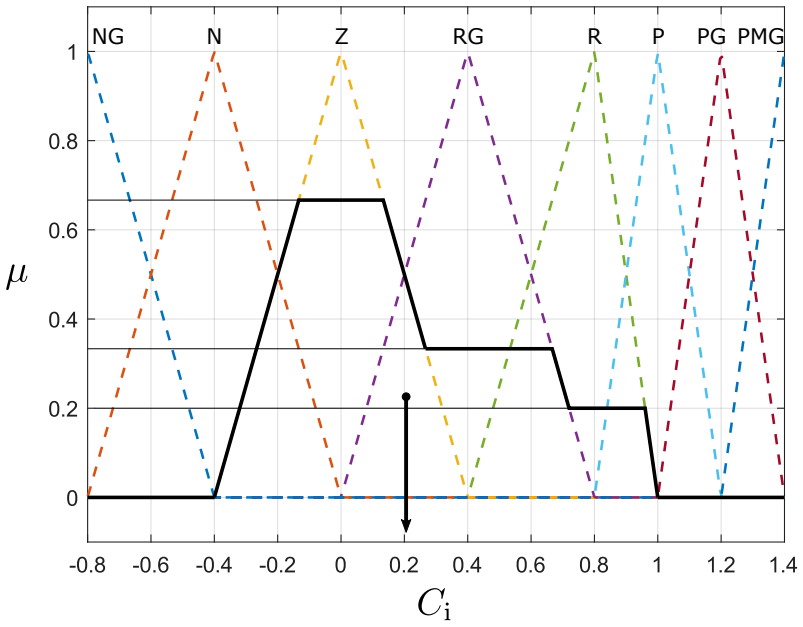


Figura 5.24: Metodo del baricentro applicato

5.4.6 Funzione di approssimazione fuzzy

Il controllo fuzzy non associa le uscite agli ingressi secondo una formula, ma attraverso un algoritmo. La funzione di approssimazione fuzzy evidenzia il legame fra ingressi e uscite.

Riportiamo innanzitutto le regole utilizzate in forma di algebra booleana, perché è così che dovranno essere scritte nel codice sorgente.

Ogni uscita deve avere un'unica espressione, che può essere più o meno articolata. Per il coefficiente di ruota interna avremo:

1. $NG = B \cdot NG$
2. $N = B \cdot N$
3. $Z = M \cdot NG$
4. $RG = M \cdot N + A \cdot NG$

5. $R = A \cdot N$
6. $P = Z$
7. $PG = P$
8. $PMG = PG$

mentre per il coefficiente di ruota esterna avremo:

9. $RG = B \cdot PG$
10. $R = P + PG \cdot (M + A)$
11. $P = Z$
12. $PG = N$
13. $PMG = NG$

Discretizzando gli ingressi, applicandoli e defuzzificando è possibile ottenere la funzione di approssimazione fuzzy, che in questo caso sarà una superficie, visto che abbiamo due input. In figura 5.25 è riportato il grafico del coefficiente di ruota interna, mentre in figura 5.26 quello di ruota esterna.

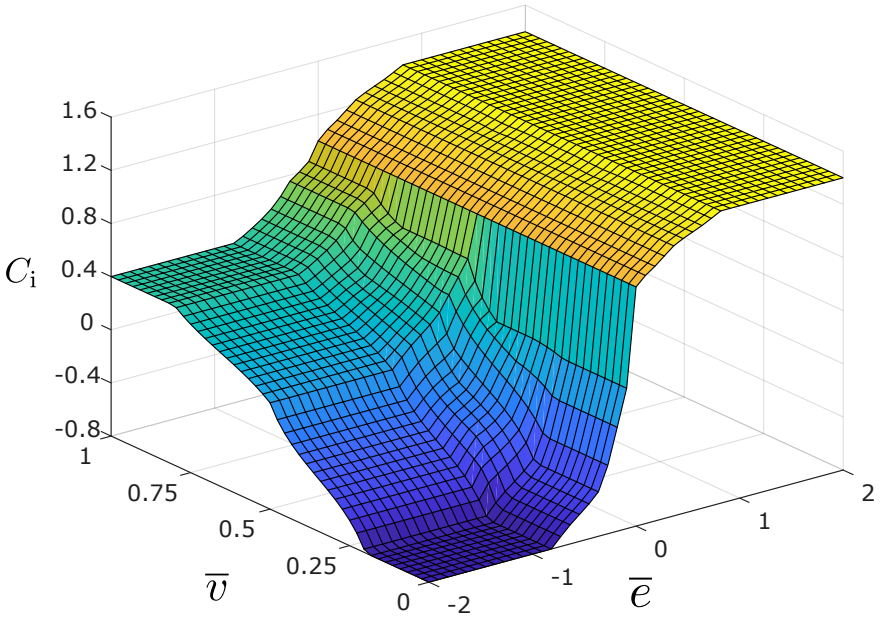


Figura 5.25: Funzione di approssimazione fuzzy: ruota interna

Come si può notare, e come è stato detto in precedenza, a basse velocità il controllo è molto più aggressivo per compensare il duty cycle minore (di norma). Si può inoltre notare che per $\overline{e} > 0$ il coefficiente C_i non dipende dalla velocità, quindi è il coefficiente di ruota esterna che si occupa maggiormente di modulare la spinta in funzione della velocità.

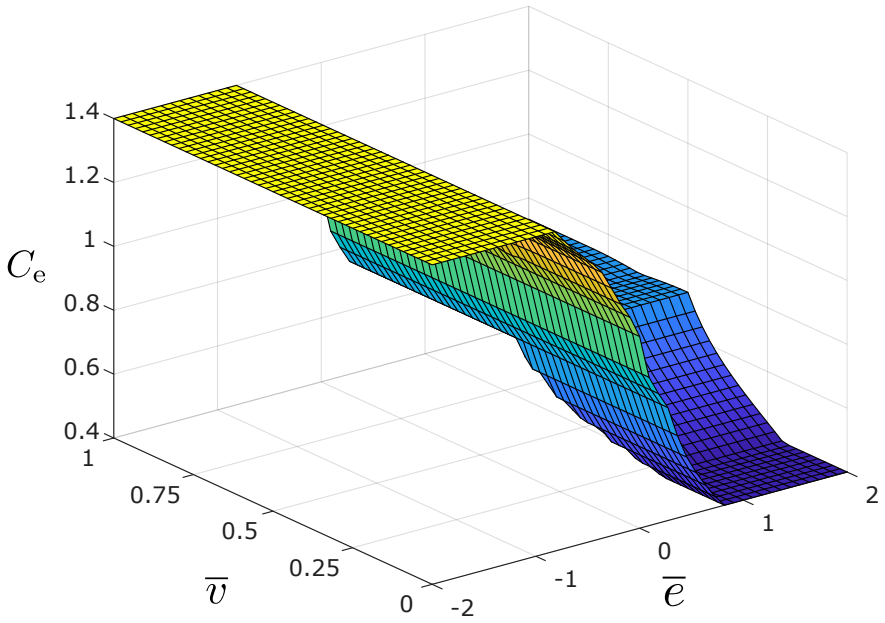


Figura 5.26: Funzione di approssimazione fuzzy: ruota esterna

Compiti invertiti si hanno invece nel caso in cui $\overline{e} < 0$. In definitiva si ha che la ruota che spinge maggiormente è indipendente dalla velocità, almeno per quanto riguarda il coefficiente, perché il duty cycle finale dipende sempre dal duty cycle dell'anteriore. È compito quindi della ruota che "frena" modulare maggiormente il valore del suo coefficiente in funzione della velocità.

Il normale funzionamento del controllo fuzzy prevede che l'errore sia solitamente compreso tra -1 e 1, motivo per il quale le zone esterne risultano indipendenti da tale valore[10], come si può notare dalla presenza di zone in cui è verificata la condizione:

$$\frac{\delta C_i}{\delta \overline{e}} = 0$$

dove δ indica la variazione infinitesima, considerando che i differenziali non sono esatti per via del fatto che la superficie è approssimata ed è generata da un algoritmo e non da una funzione.

5.4.7 Implementazione del controllo fuzzy

In figura 5.27 è riportato lo schema a blocchi del controllo fuzzy. Questa parte non comprende l'intero controllo del posteriore, ma solo il calcolo e l'assegnamento dei coefficienti.

Il primo blocco prevede la ricezione degli input: le velocità di rotazione dei motori anteriori e l'angolo δ imposto dai comandi inviati.

[10]Sono volutamente indipendenti, ricordando che la superficie è stata generata sulla base delle classi e delle regole scelte.

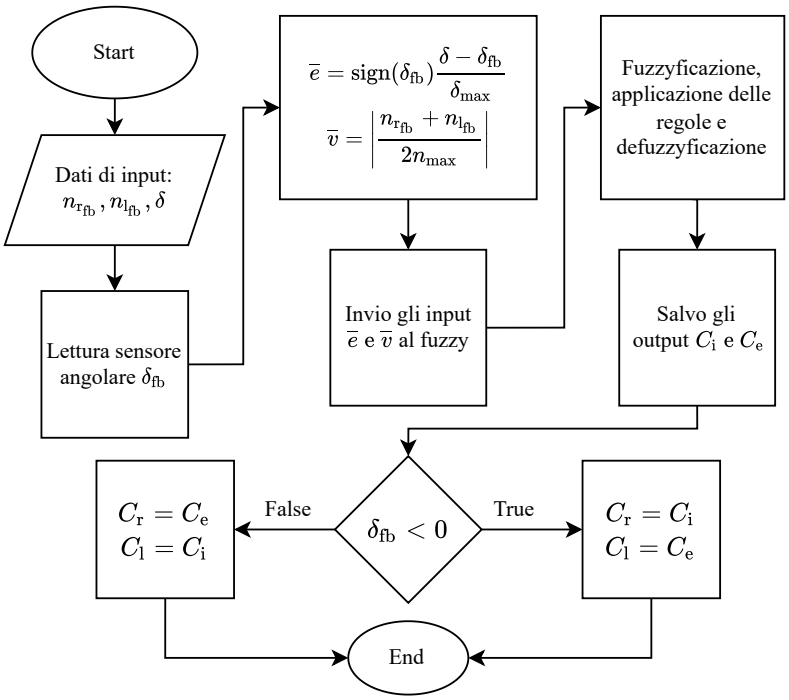


Figura 5.27: Schema a blocchi del controllo fuzzy

Si esegue la lettura del potenziometro e si acquisisce l'angolo δ reale. Al terzo blocco eseguiamo il calcolo delle variabili di input $\overline{e}$ e $\overline{v}$, ed eseguiamo gli step della logica fuzzy.

Otteniamo quindi i due coefficienti che, in funzione del segno di δ_{fb} verranno assegnati rispettivamente alle ruote posteriori destra e sinistra.

Ricordando lo schema in figura 5.4 abbiamo che se $\delta_{fb} > 0$ la ruota interna è la sinistra, in caso contrario è la destra.

5.5 Diagramma di flusso

Analizziamo ora nel dettaglio il diagramma di flusso (figura 5.28) del codice sorgente caricato nel microcontrollore del robot. In questa parte ci siamo concentrati principalmente sull'aspetto funzionale, tralasciando qualche parte di codice, sì importante, ma più marginale, come ad esempio l'elaborazione della variabile di input *cmd* (figura 5.3) e il salvataggio dei dati sulla micro sd, il quale avviene a seguito dell'applicazione dei duty cycle al rear. Si trascura inoltre anche la temporizzazione, che verrà approfondita successivamente.

A seguito della ricezione dei comandi, si esegue un check per stabilire la modalità di moto. Percorrendo prima il ramo della modalità pivot, si ha che il front è sempre il modulo M1, mentre il rear è sempre il modulo M2. Calcoliamo il set di velocità del motore destro mappando la variabile ν, otteniamo l'altro invertendo il segno, e andiamo ad eseguire il controllo in velocità del modulo anteriore (figura 5.6). Calcoliamo i duty cycle da imporre

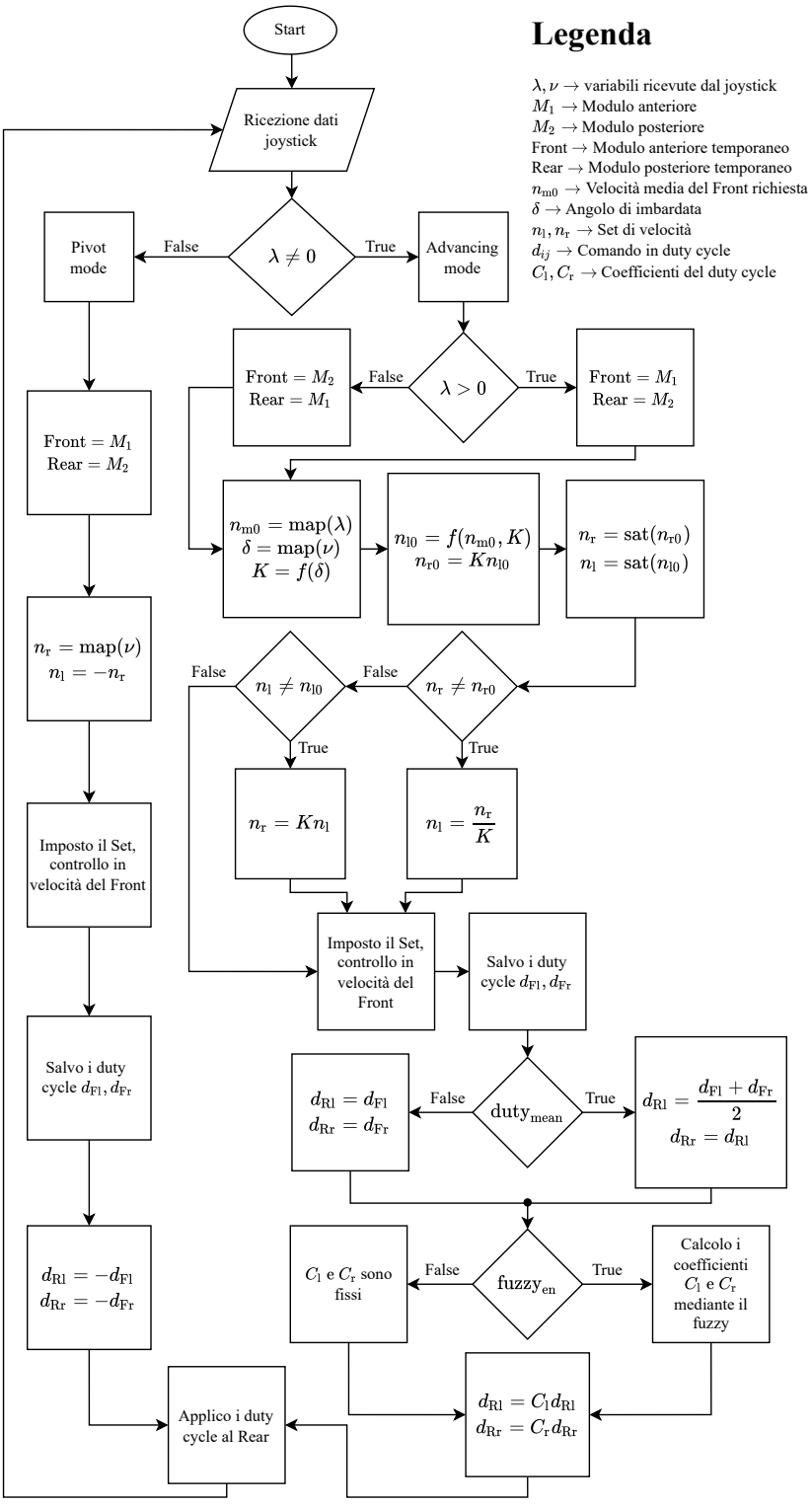


Figura 5.28: Diagramma di flusso del robot

ai motori posteriori invertendo il segno[11] e inviamo il segnale PWM.

Nel caso in cui la variabile λ sia diversa da zero si entra nella modalità di avanzamento. Si esegue un check sul segno al fine di assegnare i due moduli rispettivamente al front e al rear, e in seguito si mappano i comandi ricevuti. La funzione *map* è una funzione di mappatura lineare la cui espressione è la seguente:

$$\text{map}(x, a_1, a_2, b_1, b_2) = \frac{(x - a_1)(b_2 - b_1)}{(a_2 - a_1)} + b_1 \tag{5.10}$$

in cui i parametri a sono gli ingressi e i parametri b le uscite. Tale funzione trasforma a_1 in b_1 e a_2 in b_2 e di conseguenza, essendo lineare, tutti gli altri valori dell'asse reale.

Nel diagramma, per motivi di spazio, non sono stati inseriti gli estremi della mappatura che sono i seguenti:

$$n_{\text{m0}} = \text{map}(\lambda,\ -125,\ 125,\ -n_{\max},\ n_{\max})$$
$$\delta = \text{map}(\nu,\ -125,\ 125,\ -\delta_{\max},\ \delta_{\max})$$

ricordando che $n_{\max} = 10\,000\,\text{rpm}$ e $\delta_{\max} = 38°$. La scelta del numero 125 è stata invece spiegata nella sezione 5.2.

Eseguite le due mappature calcoliamo il coefficiente di curvatura K, definito nel paragrafo 2.3.2, mediante la relazione 2.16.

Dato il coefficiente di curvatura possiamo calcolare le velocità di rotazione teoriche da inviare al controllo in velocità:

$$n_{\text{l0}} = \frac{2\, n_{\text{m0}}}{K + 1} \tag{5.11}$$
$$n_{\text{r0}} = K\, n_{\text{l0}} \tag{5.12}$$

A questo punto eseguiamo la saturazione di entrambi i valori di velocità in modo tale da contenerli nell'intervallo $[-n_{\max},\ n_{\max}]$. Per come sono stati definiti, non possono saturare contemporaneamente, quindi si esegue un controllo su entrambi. Nell'eventualità che uno dei due saturi, l'altro viene ricalcolato mediante il coefficiente K. Avendo ottenuto i due set, possiamo eseguire il controllo in velocità e calcolare i duty cycle.

Eseguiamo il check per il metodo di passaggio del duty cycle (diretto o mediato) e il check del controllo fuzzy. Se quest'ultimo è abilitato si esegue il controllo fuzzy per il calcolo dei coefficienti C_{l} e C_{r}, in caso contrario essi sono costanti e pari a 0.9. Calcoliamo i duty cycle da inviare al posteriore e inviamo il segnale PWM.

A seguito di questa operazione c'è il salvataggio dei dati generati in questo loop nella micro sd.

5.5.1 Temporizzazioni

Il loop nella sua interezza generalmente non viene eseguito in maniera sequenziale, ma è diviso in più sotto cicli, ognuno dei quali si occupa di uno specifico compito. Ogni mini

[11] Qui non è stato esplicitato il fatto che c'è il trasferimento del duty cycle al posteriore solo se l'angolo è minore dell'angolo limite, come detto in precedenza.

loop eseguito genera dei dati che rimangono fissi e persistenti fino a che lo stesso loop non viene rieseguito. Tale persistenza consente agli altri loop di utilizzare tali dati nel momento in cui ne hanno bisogno. Nella tabella 5.5 sono riportati i singoli loop e la loro durata.

Loop	**Cadenza** (ms)	**Descrizione**
Misurazioni motori	10	Include la lettura della corrente e la lettura degli encoder per il calcolo della velocità di rotazione. Ogni motore ha il suo.
Controllo in velocità	10	Esecuzione del controllo in velocità, un loop per ogni motore.
Ricezione dei comandi	20	I comandi vengono ricevuti dal joystick con cadenza fissa, essa può anche essere di un ordine di grandezza superiore alla cadenza del controllo in velocità. Condizione per il corretto funzionamento è che sia pari alla cadenza di invio dei comandi.
Acquisizione angolo	10	Cadenza con cui si effettua la lettura dell'angolo mediante il potenziometro.
Controllo fuzzy	20	Loop per il calcolo dei coefficienti moltiplicativi del duty cycle.
Log dei dati	30	Salvataggio dei dati di navigazione nella micro sd. Viene fatta una fotografia nel momento in cui inizia tale loop.

Tabella 5.5: Durata dei mini loop

La durata del loop delle misurazioni dei motori è figlio delle caratteristiche dell'encoder viste nel paragrafo 4.2.3, e per evitare troppe separazioni si è deciso di accorpare anche l'acquisizione della tensione per il calcolo della corrente.

La cadenza del controllo in velocità dipende anch'essa dall'acquisizione tramite encoder, per cui il suo valore non può essere inferiore al precedente, ma può, senza nessun problema, essere maggiore. In questo caso, per avere la miglior risposta possibile dal sistema, si è scelto di prenderlo uguale. Sotto tale loop viene inglobato anche la semplice applicazione del duty cycle, nel caso in cui i motori non siano controllati in velocità.

La ricezione dei comandi avviene con una frequenza dimezzata rispetto al loop di controllo. Avere la stessa frequenza non comporta alcun vantaggio perché le variazioni dei comandi verrebbero filtrate dalla dinamica dei motori.

L'acquisizione dell'angolo di imbardata avviene con la stessa cadenza delle misurazioni dei motori, il loop è separato perché sono due classi del programma diverse. Non si è scelta la stessa cadenza del controllo fuzzy perché sull'angolo, come sulle velocità e le correnti, viene applicato un filtraggio numerico, per cui è conveniente acquisire più dati.

Il controllo fuzzy viene eseguito ogni 20 ms e non ogni 10 come il controllo in velocità per via del fatto che è molto più dispendioso in termini computazionali, e ciò creerebbe dei rallentamenti nelle esecuzioni dei mini loop che andrebbero a modificare le cadenze

effettive degli stessi. Si è scelto tale valore perché rappresenta un buon compromesso. Ricordando che i duty cycle del posteriore dipendono sia dai coefficienti, ma anche dai duty cycle dell'anteriore, avremo che in un loop su due si avrà una modifica di quest'ultimi mantenendo i coefficienti costanti, mentre nell'altro avremo l'aggiornamento di entrambi. Considerando che i coefficienti dipendono dall'angolo e dalla velocità di avanzamento, e che questi ultimi non hanno una dinamica troppo veloce, possiamo dire che tra due loop consecutivi il loro valore di mantiene pressoché costante.

Digressione sul salvataggio dei dati

Il salvataggio dei dati è sicuramente l'operazione più gravosa che compie il microcontrollore, ogni ciclo vengono infatti scritti circa 350 byte sulla micro sd[12]. Tali dati vengono salvati in un file CSV (comma separated value), un file di testo che contiene i valori numerici separati da una virgola. La cadenza è stata scelta in modo tale da minimizzare l'impatto sugli altri loop, ma in modo tale da fornire una sufficiente quantità di dati per le successive elaborazioni.

I valori salvati sono, in primo luogo, le grandezze che entrano in gioco nel controllo in velocità, comprese le singole componenti del PID, per ogni motore, ed in secondo luogo, le grandezze del controllo in duty cycle.

I file CSV sono comodi per la successiva elaborazione dei dati e per questo motivo sono stati utilizzati. Non sono però il metodo più efficiente per immagazzinare i dati in tempo reale che si può applicare. Al fine di aumentare la frequenza di campionamento può essere conveniente scrivere il file sotto forma di file binario, in modo tale che ogni numero sia salvato con la stessa sequenza di bit contenuta nella ram del microcontrollore, invece di ricorrere alla codifica ascii o utf-8. Ciò permette di risparmiare innanzitutto i byte delle virgole, ma soprattutto i byte utilizzati per rappresentare un float di parecchie cifre.

In un file binario ogni dato occupa una quantità di memoria fissa, cosa non vera in un comune file di testo. Per fare un esempio, se dovessimo rappresentare il numero -1200.55 avremo bisogno di ben 8 byte (uno per ogni carattere), contro i 4 necessari per la sua codifica in forma binaria. Nel caso del float il computo va sempre a vantaggio del file binario, o al massimo in pareggio, visto che ci sono sempre almeno due cifre decimali. Nella codifica degli altri tipi sono pochi i casi in cui il file di testo vince sul file binario.

In conclusione possiamo dire che il file binario può essere un'ottima soluzione per aumentare la frequenza di campionamento dei dati. Tale metodologia non è stata però applicata perché è necessario creare un programma ad hoc per la lettura del file che conosca a priori quando un dato finisce ed inizia il successivo, visto che non è presente il separatore.

5.6 Retromarcia

Verifichiamo ora il funzionamento del robot in retromarcia. Ricapitolando e semplificando quanto detto in precedenza abbiamo che: se il comando λ è positivo, il modulo M1 è di

[12] La micro sd deve essere di buona qualità e di una classe elevata, in modo da avere una velocità di scrittura consona all'utilizzo. Se tali requisiti non fossero soddisfatti si creerebbe un vero e proprio collo di bottiglia che comprometterebbe il corretto funzionamento del robot.

trazione e viene controllato in velocità con un set positivo; al contrario, un λ negativo fa sì che il modulo di trazione sia M2 e che questo riceva un set di velocità negativo. Ricordando che di default il modulo M1 è di trazione possiamo distinguere i motori del lato destro dai motori del lato sinistro.

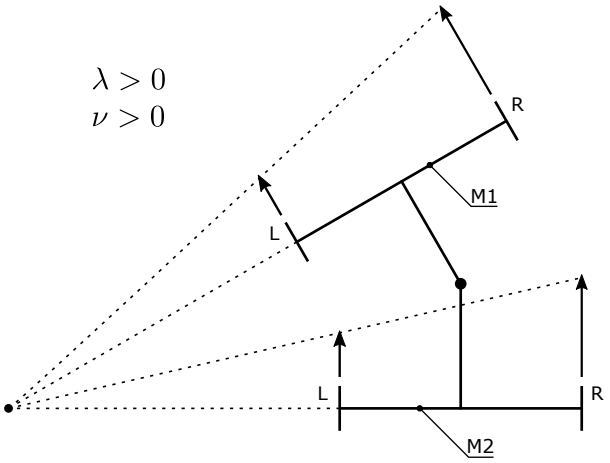


Figura 5.29: Marcia avanti

In figura 5.29 è riportata la condizione di marcia avanti con $\lambda > 0$ e $\nu > 0$. In questa configurazione, con il controllo fuzzy attivo, si ha che la ruota posteriore interna è la ruota sinistra del modulo M2, e l'angolo δ è positivo, in accordo con il segno di ν.

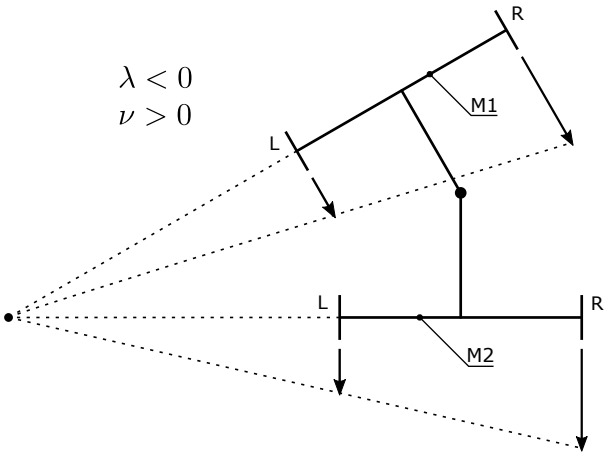


Figura 5.30: Retromarcia

Invertendo il segno di λ (figura 5.30) e lasciando invariato ν avremo che il coefficiente K è rimasto invariato e si è invertito il segno di n_{m0}, di conseguenza, tramite la 5.11 e la 5.12, otteniamo gli stessi set di velocità ma con segni invertiti. Sapendo che il modulo di trazione è diventato M2, questo riceve i set di velocità generati.

Per quanto riguarda il controllo fuzzy, noi sappiamo che se $\delta_{\text{fb}} > 0$ la ruota posteriore interna è la sinistra. In questo caso il modulo posteriore è M1 e, di conseguenza, la ruota posteriore interna diventa la ruota sinistra di tale modulo (come mostrato in figura).

I riferimenti scelti per i lati destro e sinistro consentono il funzionamento del fuzzy anche in retromarcia, senza la necessità di dover introdurre correzioni. Possiamo dire quindi che non ci sono problemi nella gestione della retromarcia.

5.7 Joystick

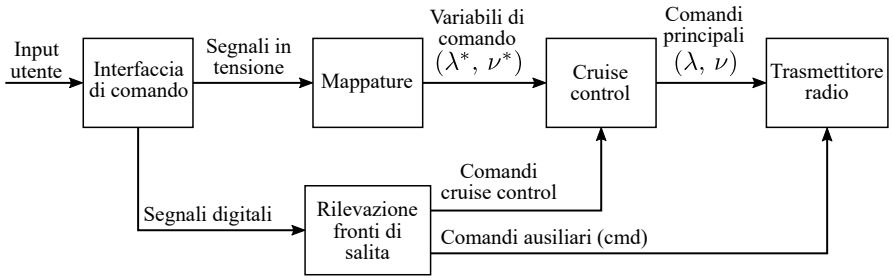


Figura 5.31: Schema funzionale del joystick

In figura 5.31 è riportato lo schema funzionale del joystick. A seguito degli input dell'utente eseguiamo una mappatura degli analogici per convertire i segnali in tensioni in variabili all'interno del microcontrollore, ed eseguiamo una rilevazione dei fronti di salita[13] per verificare se un pulsante è stato premuto[14]. Le variabili generate dalle mappature ed eventuali altri comandi entrano nel cruise control, il quale ha lo scopo di generare un comando fisso qualora fosse necessario e utile. Infine le tre variabili di comando (λ, ν, cmd) arrivano al trasmettitore radio. Il tutto è temporizzato con la stessa cadenza del robot, ovvero i comandi vengono inviati ogni 20 ms.

5.7.1 Mappa dei pulsanti

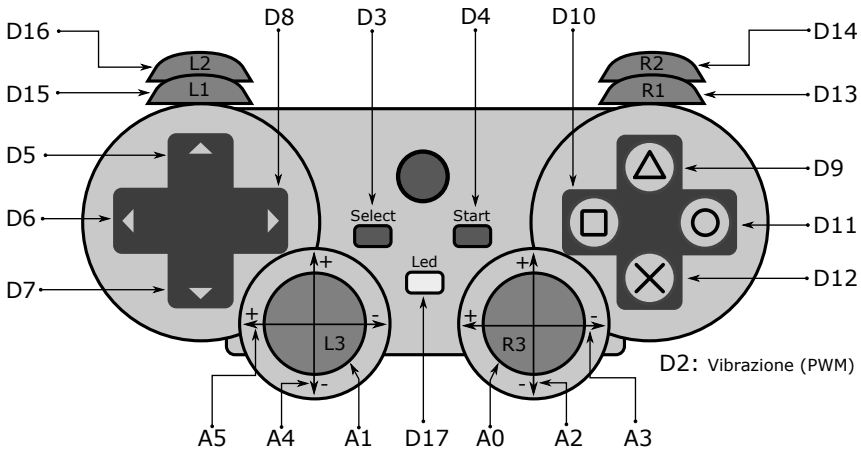
In figura 5.32 è mostrata la mappa dei pulsanti, ognuno di essi corrisponde a un pin digitale o analogico del microcontrollore.

La nomenclatura utilizzata è quella del joystick della Play Station, visto che per la maggior parte delle persone è più familiare, anche se in realtà quella vera è un'altra, come si può notare dalla figura 4.3 mostrata in precedenza.

Con il prefisso **D** si indicano i pin digitali, il cui stato può essere letto mediante la funzione "digitalRead", mentre con il prefisso **A** si indicano i pin analogici, che vengono letti con la funzione "analogRead".

[13] In realtà sono fronti di discesa perché è presente una resistenza di pullup, come verrà spiegato a breve.

[14] Per eseguire questo check viene utilizzata una libreria ottimizzata (JC Button) disponibile al seguente link: https://github.com/JChristensen/JC_Button.



D → pin digitale con pullup (0 se premuto), ex: digitalRead(5)
A → pin analogico (+ → 1023, − → 0), ex: analogRead(5)
A0, **A1** → pulsanti analogici (0 se premuti), ex: analogRead(0)
D17 → pin digitale (digitalWrite(low) → accende il led)

Figura 5.32: Mappa dei pulsanti del joystick

I pin digitali presentano una resistenza di pullup (figura 5.33) che tiene collegati tali pin al livello di tensione HIGH quando non sono premuti. Nel momento in cui il pulsante viene premuto il pin viene collegato al GND, leggendo un livello di tensione LOW.

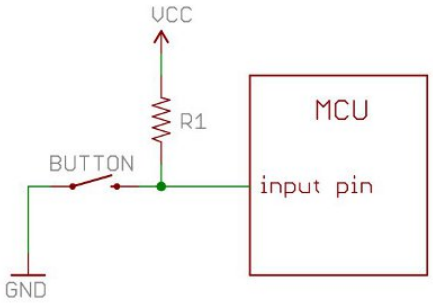


Figura 5.33: Resistenza di pullup

Le levette sono collegate a tre pin analogici ciascuna, i primi due servono per misurare l'inclinazione nelle due direzioni, mentre il terzo è un pulsante a tutti gli effetti, con la stessa resistenza di pullup dei pin digitali.

In tabella 5.6 sono riportati le funzioni associate a ogni pulsante o levetta.

Il controllo del robot avviene per mezzo delle levette analogiche. Dall'acquisizione del loro stato vengono generate le due variabili λ e ν responsabili del moto del robot.

Nello specifico avremo che la leva sinistra mossa verticalmente è associata alla variabile λ, mentre la variabile ν è associata al movimento orizzontale della leva di destra. Se $\lambda = 0$, la variabile ν è associata al movimento orizzontale della leva di sinistra. Tale separazione

Tabella 5.6: Mappa dei pulsanti

Nome	Pin	Funzione
L_v	A4	Variabile λ
L_h	A5	Variabile ν modalità pivot
R_h	A3	Variabile ν modalità advancing
L3	A1	Disabilita cruise control
Up	D5	Cruise control avanzamento (+)
Down	D7	Cruise control avanzamento (−)
Right	D8	Cruise control sterzo (curva oraria)
Left	D6	Cruise control sterzo (curva antioraria)
Triangolo	D9	Mappatura 0 (default)
Cerchio	D11	Mappatura 1
X	D12	Abilita duty cycle medio al posteriore
Quadrato	D10	Disabilita duty cycle medio al posteriore
Select	D3	Abilita riduzione di velocità
Start	D4	Disabilita riduzione di velocità
R1	D13	Abilita controllo fuzzy
L1	D15	Disabilita fuzzy
R2	D14	Abilita posteriore
L2	D16	Disabilita posteriore

è stata fatta per evitare un passaggio improvviso dalla modalità advancing alla modalità pivot.

Le frecce implementano il "cruise control". Simile, per concezione, a quello presente su una normale autovettura, esso consente di inviare un comando fisso senza dover utilizzare gli analogici (tale feature è molto utile nelle prove sperimentali quando si vuole imporre un set a gradino). A seguito della pressione della freccia "up" il valore di lambda viene aumentato di una quantità fissa, caso inverso per la pressione della freccia "down". Una volta inserito il cruise control il joystick invia un segnale costante finché non viene mossa la levetta sinistra, o viene premuto il pulsante L3. Entrambe le condizioni disabilitano il cruise control.

Le frecce left e right fanno la stessa cosa, ma sulla variabile ν, quindi possono essere utilizzate sia per la modalità advancing che per la modalità pivot. I pulsanti triangolo e cerchio consento di eseguire uno switch tra le due mappature caricate (in seguito verrà fatta una trattazione più dettagliata delle mappature). I pulsanti X e quadrato cambiano il modo in cui i duty cycle vengono passati dal front al rear. Di default viene passato il duty cycle medio, quindi è come se fosse stato premuto il pulsante X. Funzioni analoghe si hanno con i pulsanti R1 e L1, che abilitano o disabilitano il controllo fuzzy, e con i pulsanti R2 e L2, che abilitano o disabilitano il modulo posteriore. Infine abbiamo i pulsanti start e select che consento di abilitare la funzione di riduzione della velocità, che consente una migliore manovrabilità del robot quando controllato dal joystick.

Come si può notare, sono state separati i pulsanti che abilitano da quelli che disabilitano una funzione, invece che usare gli stessi come interruttori. Questa scelta è dovuta al fatto

che un pulsante con un comportamento univoco è più facilmente gestibile rispetto ad un pulsante di tipo "toggle". Nella pratica, se premiamo un pulsante sappiamo con certezza cosa sia successo. Nel caso in cui non ci fosse tale separazione sarebbe stato necessario implementare un qualche tipo di feedback per conoscere l'attuale stato del robot[15].

5.7.2 Diagramma di flusso

In figura 5.34 riportiamo un diagramma di flusso che meglio spiega il funzionamento[16]. In seguito alla lettura dello stato dei pulsanti[17], con funzioni specifiche per i successivi check. e della mappatura degli analogici otteniamo tre variabili: λ^*, ν_l^* e ν_r^*. L'apice sta ad indicare che sono i valori ottenuti dalla sola mappatura e non sono passati per il cruise control. Il pedice invece serve per distinguere il valore di ν destinato a diventare un comando pivot o un comando di sterzo in modalità advancing.

Eseguiamo un primo check sui pulsanti up e down, e aumentiamo o diminuiamo la variabile λ, che è reduce dal loop precedente ed è quella che verrà inviata, di una quantità definita pari a λ_0 che nel nostro caso vale circa 1/10 del comando massimo. La persistenza di tale variabile ci permette di mantenere il comando attivo ad ogni loop anche senza alcuna pressione dei tasti o movimento degli analogici. A seguito dell'aumento o della diminuzione si esegue anche una saturazione per evitare che la variabile vada in overflow stravolgendo completamente il suo valore.

In uscita da questo primo blocco eseguiamo una prima verifica su λ che ci consente di capire se il cruise control sia attivo o meno[18] e salviamo il risultato in Λ (variabile booleana). Eseguiamo lo stesso identico processo per la variabile ν e salviamo lo stato del cruise control in N.

Eseguiamo ora una prima verifica per la disattivazione del cruise control. La condizione è la seguente:

$$\lambda^* + \nu_l^* + \overline{\Lambda} = 1 \tag{5.13}$$

la quale ci dice che è sufficiente muovere l'analogico sinistro per disabilitarlo. Per il corretto funzionamento del programma, ovvero per il corretto assegnamento di λ, è stata aggiunta come condizione di attivazione anche la sua negazione: Se $\Lambda = 0$ allora entriamo nel blocco.

I simboli "+" e "||" indicano entrambi l'operazione logica di OR.

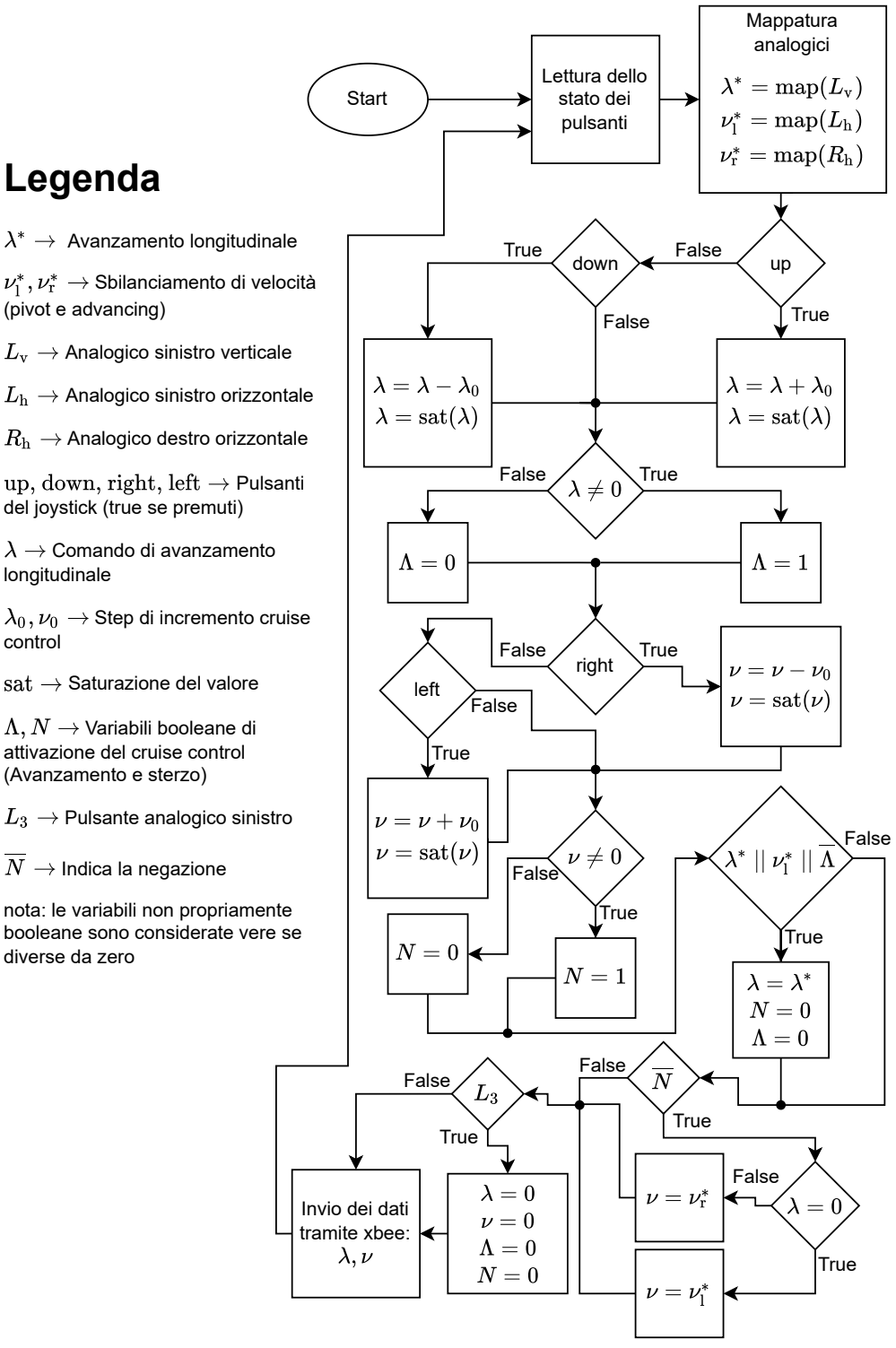
Continuando, troviamo una verifica sulla seconda variabile del cruise control. Nel caso in cui la condizione sia vera e che quindi non stiamo usando il cruise per la variabile ν si

[15] In realtà esiste anche, ma non si è fatta menzione in questa tesi perché è un po' scomodo. In pratica viene utilizzato un terzo XBee collegato a un PC che riceve dati dal robot e stampa a video eventuali cambiamenti di controllo e mappature.

[16] Non tratteremo nello specifico la generazione della variabile cmd, in quanto superflua per i nostri scopi.

[17] Per la lettura dei pulsanti e per evitare il debounce meccanico si utilizza una libreria ottimizzata che si chiama JC_Button ed è disponibile al seguente link: https://github.com/JChristensen/JC_Button.

[18] In realtà questo check viene eseguito all'interno di ogni blocco, dopo la saturazione, per evitare che il comando del loop precedente, magari generato dall'analogico, faccia attivare il cruise control senza che questo lo sia realmente. Nello schema a blocchi è stato posto all'esterno per una questione di ordine e di spazio. Nella pratica, se sia up che down sono falsi si salta direttamente al right, e lo stesso discorso vale per la variabile ν.



Legenda

$\lambda^* \rightarrow$ Avanzamento longitudinale

$\nu_\mathrm{l}^*, \nu_\mathrm{r}^* \rightarrow$ Sbilanciamento di velocità (pivot e advancing)

$L_\mathrm{v} \rightarrow$ Analogico sinistro verticale

$L_\mathrm{h} \rightarrow$ Analogico sinistro orizzontale

$R_\mathrm{h} \rightarrow$ Analogico destro orizzontale

$\mathrm{up}, \mathrm{down}, \mathrm{right}, \mathrm{left} \rightarrow$ Pulsanti del joystick (true se premuti)

$\lambda \rightarrow$ Comando di avanzamento longitudinale

$\lambda_0, \nu_0 \rightarrow$ Step di incremento cruise control

$\mathrm{sat} \rightarrow$ Saturazione del valore

$\Lambda, N \rightarrow$ Variabili booleane di attivazione del cruise control (Avanzamento e sterzo)

$L_3 \rightarrow$ Pulsante analogico sinistro

$\overline{N} \rightarrow$ Indica la negazione

nota: le variabili non propriamente booleane sono considerate vere se diverse da zero

Figura 5.34: Diagramma di flusso del joystick

esegue un check su λ per capire se assegnare a ν il valore ottenuto dall'analogico sinistro (modalità pivot) o dall'analogico destro (modalità advancing).

In ultimo, prima dell'invio dei comandi, si esegue un controllo sullo stato del pulsante L3. Nel caso fosse stato premuto avremmo un azzeramento delle variabili di comando e delle variabili booleane del cruise control.

5.8 Mappature dei comandi

In questa sezione analizzeremo le mappature implementate al fine di correlare graficamente le grandezze di set del controllo in velocità con i comandi inviati al robot. Successivamente, riporteremo tali valori al controllo tramite joystick, esplicitando quali strategie sono state utilizzate per migliorare il comfort di guida.

In questa parte le superfici mostrate si riferiscono unicamente alla modalità advancing visto che la modalità pivot è decisamente più semplice.

5.8.1 Mappature del robot

Analizzando le mappature in funzione delle variabili λ e ν, andremo ad osservarle da un punto di vista più generale, perché non teniamo conto di come sono stati generati tali comandi. Focalizzandoci sul punto di vista del robot le uniche informazioni importanti sono proprio i valori di λ e ν. È compito proprio di quest'ultimo elaborare tali dati e generare i set di velocità tramite l'algoritmo descritto in precedenza.

Discretizzando i due input in 60 punti otteniamo le seguenti superfici.

In figura 5.35 è riportato l'andamento della velocità di rotazione media teorica dei motori, ovvero la media dei due set di velocità che vengono inviati al controllo in velocità. Come si può notare abbiamo un ampio tratto in cui la velocità media dipende solo dalla variabile λ, mentre nelle zone più estreme il comportamento varia a causa delle saturazioni imposte ai singoli set. In queste zone non può essere rispettato il comando λ se $\nu \neq 0$.

La forma delle zone in cui avviene una saturazione è quella riportata nel grafico 2.15, in cui si analizzava la velocità massima in funzione dell'angolo δ.

La zona centrale è un piano perché è stata applicata una mappatura lineare, in caso contrario la forma sarebbe stata differente, come si vedrà in seguito.

Ricordando che $\Delta n = n_\mathrm{r} - n_\mathrm{l}$, in figura 5.36 è riportata la superficie generata.

Ricordando le relazioni 5.11 e 5.12 possiamo scrivere che:

$$\Delta n = 2\, n_\mathrm{m}\, \frac{K-1}{K+1} \tag{5.14}$$

Valutiamo la funzione $(K-1)/(K+1)$ nell'intervallo che ci interessa per capirne la forma (figura 5.37).

Come si può notare, l'andamento è quasi lineare rispetto all'angolo δ e quindi può essere approssimato ad una retta. Possiamo perciò scrivere che $(K-1)/(K+1) \propto \nu$, ricordando che δ è ottenuto mediante una mappatura lineare senza offset di ν.

Al di fuori delle zone di saturazione la relazione di proporzionalità vale anche per la coppia n_m e λ, motivo per il quale, con una piccola approssimazione, possiamo scrivere che:

$$\Delta n \propto \lambda\, \nu \tag{5.15}$$

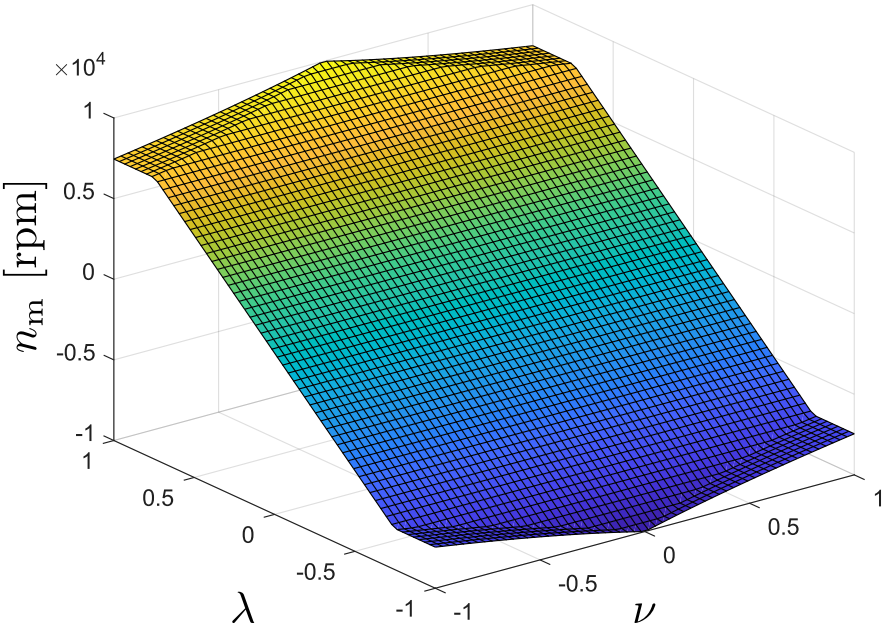


Figura 5.35: Velocità di rotazione media teorica in funzione dei comandi inviati

Grazie a ciò possiamo concludere che la superficie di Δn è un paraboloide iperbolico, il quale ha al centro un punto di sella e le curve di livello sono delle iperboli (figura 5.38).

Le figure 5.39 e 5.40 rappresentano i set di velocità effettivi dei motori. Si possono notare le due zone di saturazione che ogni motore. Più ν è grande (o piccolo) e maggiore è questa zona, per via del fatto che il nostro scopo è rendere indipendente il raggio di curvatura ρ[19] dalla velocità di avanzamento longitudinale.

Come detto in precedenza, $\nu > 0$ genera curve antiorarie, quindi avremo che la ruota destra è esterna alla curva, sia in avanti che in retromarcia, di conseguenza è lei che gioca da fattore limitante nella velocità di percorrenza del robot.

Visto che in tali zone non si riuscirà mai a rispettare il comando λ inviato, si cerca quantomeno di rispettare il comando ν andando a diminuire maggiormente il set della ruota interna.

In figura 5.41 sono riportate anche le curve di livello. È stato preso come esempio solo il set del motore destro per via del fatto che l'altro è semplicemente speculare.

Come si può vedere le curve sono abbastanza rettilinee nella zona centrale in cui λ è vicino allo 0, andando leggermente a divergere per valori crescenti (in modulo) dello stesso. Quello che all'apparenza potrebbe sembrare un piano ha invece subito una leggera "torsione".

[19] Il raggio di curvatura ρ è correlato all'angolo δ in condizioni di regime, ovvero dopo un certo tempo che la curva è iniziata.

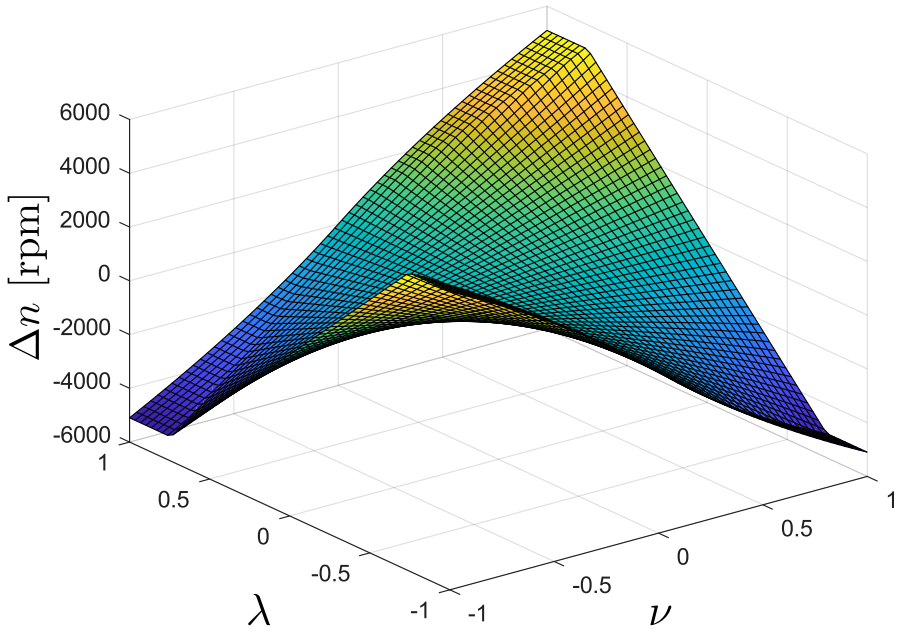


Figura 5.36: Differenza di velocità di rotazione tra il lato destro e il sinistro

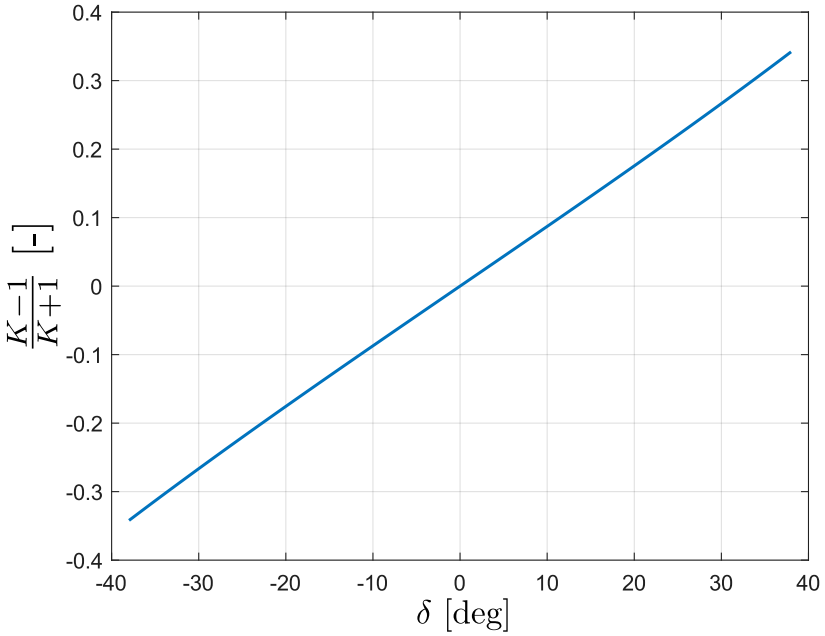


Figura 5.37: Andamento di $(K-1)/(K+1)$

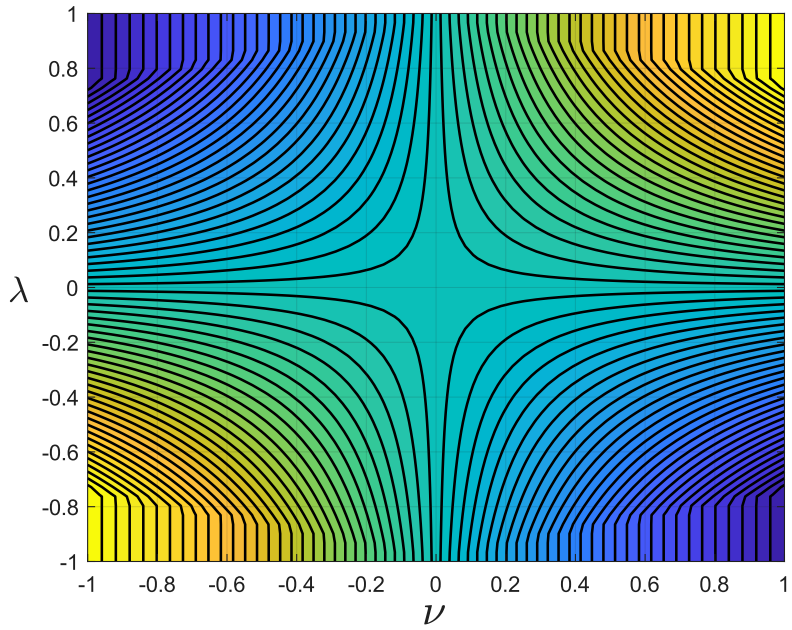


Figura 5.38: Differenza di velocità di rotazione tra il lato destro e il sinistro: curve di livello

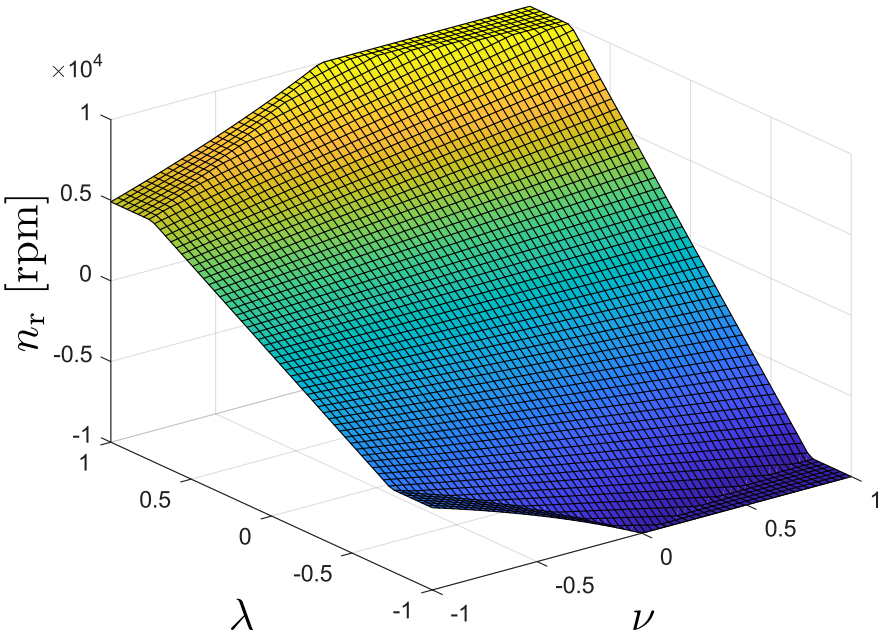


Figura 5.39: Set di velocità motore destro

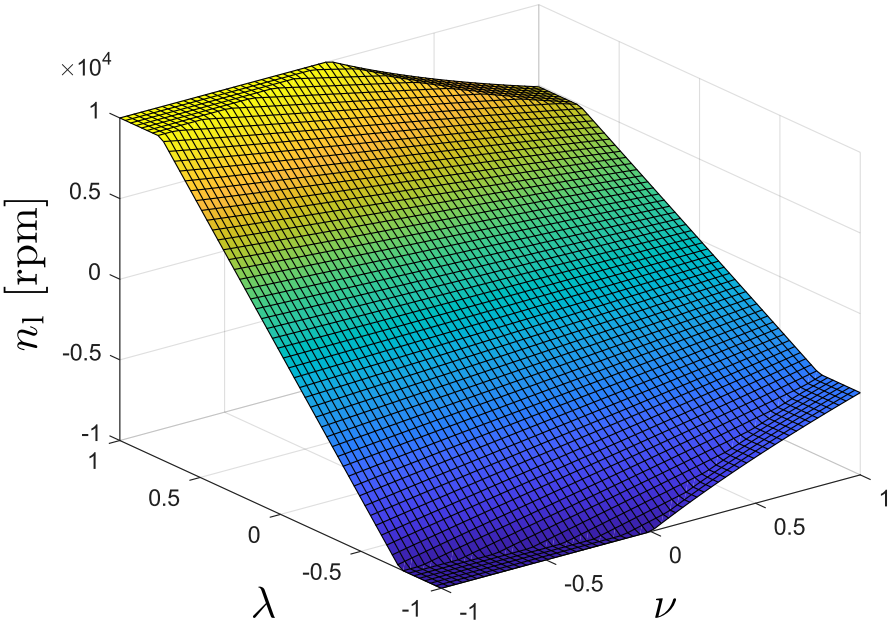


Figura 5.40: Set di velocità motore sinistro

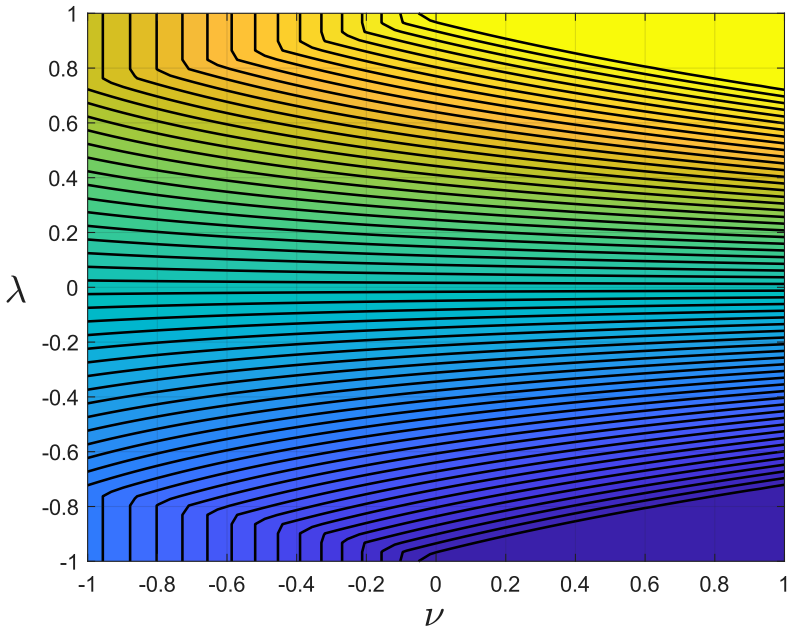


Figura 5.41: Set di velocità motore destro: curve di livello

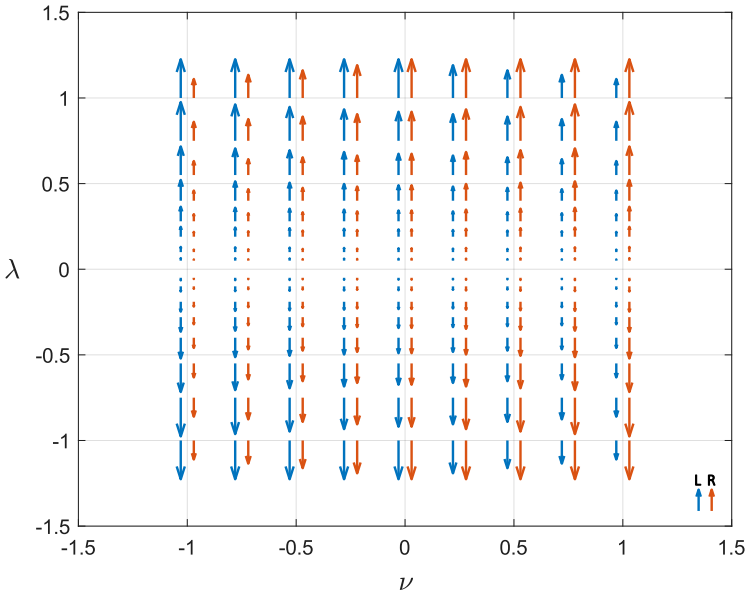


Figura 5.42: Mappa quiver

Riportiamo infine (figura 5.42) una mappa di tipo "quiver" in cui sono mostrati in scala i vettori dei set di velocità in funzione dei comandi inviati. Anche qui si può notare come per bassi valori di λ il comando ν sia poco influente nella generazione del set, mentre per alti valori il contributo diventa decisamente più consistente.

Nel primo quadrante e nel secondo quadrante avremo, rispettivamente, curve antiorarie e orarie. Nel terzo e nel quarto avremo le stesse curve, ma percorse in retromarcia.

5.8.2 Mappature del joystick

Gli analogici presenti sul joystick sono collegati a dei pin analogici i quali hanno al loro interno un ADC (convertitore analogico digitale) la cui risoluzione è impostata di default a 10 bit, il che significa un intervallo che va da 0 a 1023. Riprendendo la figura 5.32 abbiamo che il valore 1023 si raggiunge quando la leva è completamente inclinata verso il +, mentre lo 0 si raggiunge nel caso inverso. In posizione di riposo il valore in uscita è circa pari a 512.

Al fine di generare i comandi λ e ν da inviare al robot è necessario elaborare i segnali acquisiti mediante l'ADC, per fare ciò viene qui presentata una breve trattazione matematica riferita a un analogico generico.

Indicando con ψ l'angolo di inclinazione, possiamo scrivere una relazione che lega tale angolo all'uscita dell'ADC:

$$y_0 = m\psi \tag{5.16}$$

dove m è un coefficiente di proporzionalità. Non ci interessa il suo valore puntuale, o l'intervallo di ψ, in quanto il joystick verrà usato manualmente. Molto più utile ai fine dell'utilizzo è il suo andamento qualitativo, motivo per il quale abbiamo definito tre posizioni

principali della leva: *high*(+), *middle* e *low*(-).

Eseguiamo ora una traslazione verticale della funzione in modo da far coincidere il valore 0 dell'ADC con la posizione *middle* dell'analogico:

$$y_t = y_0 - 512 \tag{5.17}$$

che riportiamo in figura 5.43.

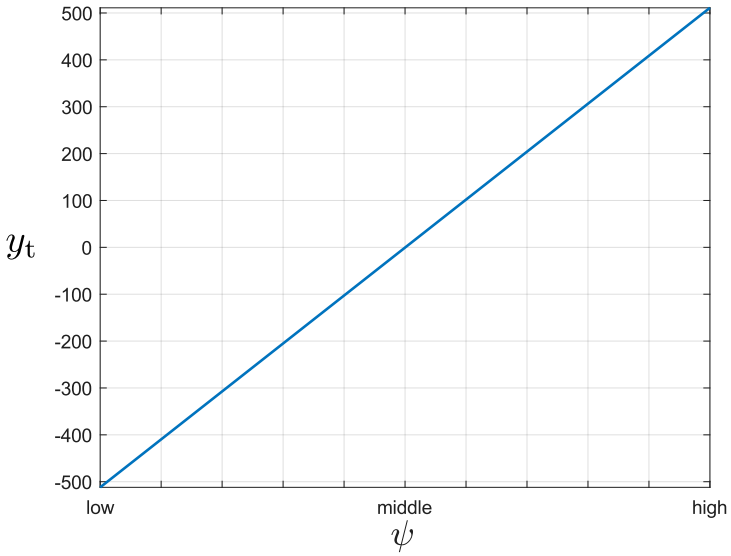


Figura 5.43: Uscita ADC traslata

L'acquisizione della tensione tramite ADC presenta un certo grado di rumore, quindi è necessario inserire una banda morta in cui il segnale viene saturato per evitare che un comando si generi anche senza alcun movimento della leva. Avremo quindi che la nostra uscita saturata è data dalla seguente funzione definita a tratti:

$$y_s = \begin{cases} y_t & \text{se } y_t \leq -h \\ -h & \text{se } -h < y_t < 0 \\ h & \text{se } 0 \leq y_t < h \\ y_t & \text{se } y_t \geq h \end{cases} \tag{5.18}$$

dove h indica la semi ampiezza della banda morta, misurata lungo la scala dell'ADC, ovvero l'ordinata, essendo l'unica misura di cui disponiamo (figura 5.44).

All'interno del codice sorgente h ha un valore pari a 8, quindi molto contenuto, ma per motivi di visibilità, in questa parte il suo valore è stato fissato a 40.

Si è scelto di implementare la banda morta in questa forma particolare perché semplifica le successive trasformazioni. Una volta eseguite diventerà una vera e propria banda morta.

Il passo successivo consiste in una normalizzazione e nell'applicazione del modulo, al fine di avere una funzione il cui codominio è l'intervallo [0, 1]. Avremo quindi che:

$$y_n = \begin{cases} -\text{map}\,(y_s,\, -512,\, -h,\, -1,\, 0) & \text{se } y_s < 0 \\ \text{map}\,(y_s,\, h,\, 511,\, 0,\, 1) & \text{se } y_s > 0 \end{cases} \tag{5.19}$$

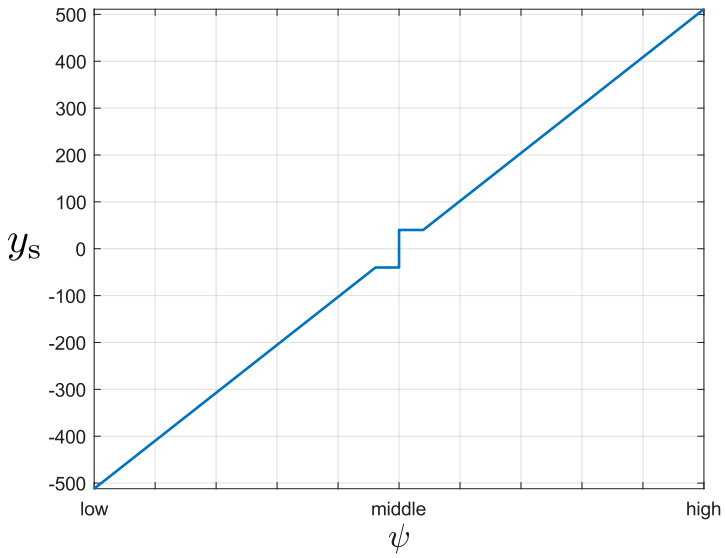


Figura 5.44: Banda morta

dove map è la funzione di mappatura lineare definita in 5.10. Le due mappature non sono perfettamente simmetriche per via del fatto che 1023 non è divisibile per 2, ma l'errore può essere tranquillamente trascurato. La banda morta definita in precedenza ci ha permesso di applicare direttamente la mappatura lineare senza ulteriori manipolazioni, semplificando questo passaggio. in figura 5.45 riportiamo la normalizzazione.

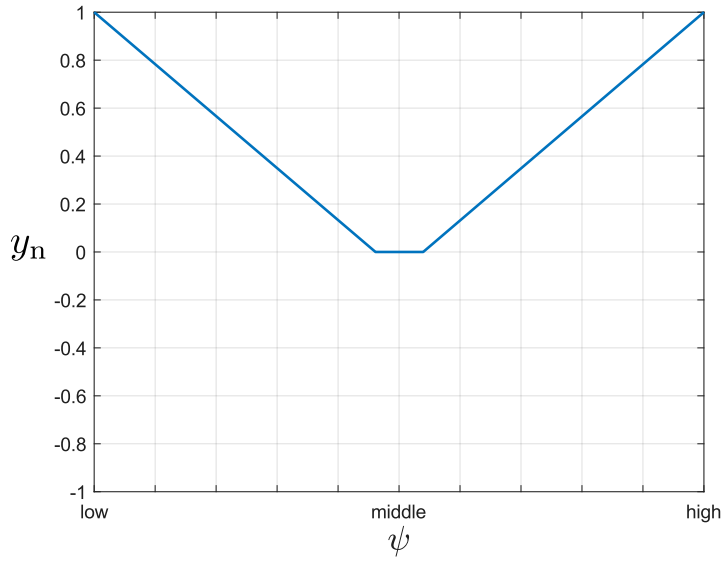


Figura 5.45: Normalizzazione

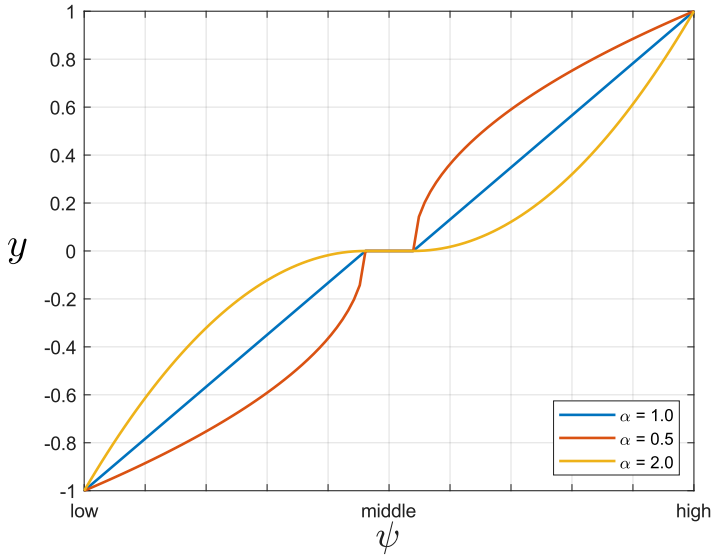


Figura 5.46: Applicazione della mappa

La normalizzazione ci consente ora di applicare una qualsiasi funzione di mappatura da noi definita, in particolare deve valere la seguente condizione:

$$f\colon [0,1] \subseteq \mathbb{R} \to [0,1] \subseteq \mathbb{R}$$

Soddisfatta la quale possono essere implementate le forme che meglio si adattano ai nostri scopi. Applicandola otteniamo il seguente output:

$$y = \begin{cases} -f(y_{\mathrm{n}}) & \text{se } y_{\mathrm{s}} < 0 \\ f(y_{\mathrm{n}}) & \text{se } y_{\mathrm{s}} > 0 \end{cases} \tag{5.20}$$

La funzione da noi scelta è $f(x) = x^{\alpha}$, in modo tale che con un singolo parametro sia possibile rendere più o meno aggressiva la mappatura. Nello specifico, in figura 5.46, riportiamo il confronto tra mappa lineare e mappa di potenza.

Come si può notare, con un esponente maggiore di 1 riusciamo ad ottenere curve via via più morbide nel tratto iniziale, a discapito di una maggiore pendenza nel tratto finale. L'aumento dell'esponente consente perciò un migliore sensibilità nell'intorno della posizione di riposo dell'analogico. Al contrario un esponente minore di 1 rende più aggressivo il comando.

Nella tabella 5.7 sono riportati i coefficienti utilizzati nelle mappature. Ricordiamo che ν_{l} viene generato dall'analogico sinistro ed è riferito alla modalità pivot, mentre ν_{r} è generato dall'analogico destro ed è riferito alla modalità advancing.

Dati questi coefficienti ammorbidiamo sia la modalità advancing che la modalità pivot, mentre rendiamo lo sterzo un po' più reattivo.

Nelle figure 5.47 e 5.48 riportiamo gli andamenti reali di λ e ν_{r}, avendo inserito la giusta semi ampiezza della banda morta. Come si può notare La presenza della banda morta è poco percettibile, se non nel caso dello sterzo.

Tabella 5.7: Coefficienti delle mappature

Variabile	Esponente α
λ	3.0
ν_l	3.0
ν_r	0.7

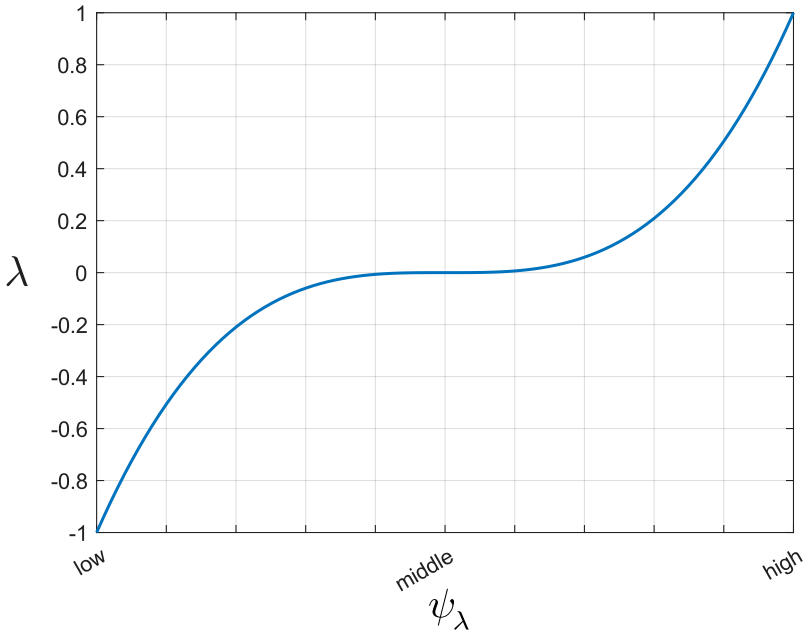


Figura 5.47: Mappatura della variabile λ

Avendo esplicitato e definito come vengono generati i comandi, è possibile ricostruire le superfici presentate in precedenza in funzione degli angoli delle leve del joystick. Anche in questo caso ci concentreremo sulla modalità advancing.

In figura 5.49 troviamo la mappa della velocità di avanzamento media. Rispetto alla sua versione precedente notiamo che si è generata una zona pianeggiante nell'intorno della posizione *middle* di ψ_λ, e sono leggermente diminuite le zone in cui si ha una saturazione. Proprio in queste ultime zone si vede una leggera influenza della mappatura dell'altra variabile. Si ha infatti una discesa leggermente più brusca dal punto di velocità massima verso valori estremi di ψ_ν.

Il discorso analogo può essere fatto per Δn (figura 5.50). In questo caso si nota maggiormente lo scalino generato dal coefficiente $\alpha = 0.7$ della mappatura di ψ_ν.

Infine, nelle figure 5.51 e 5.52 riportiamo le superfici dei due set di velocità che effettivamente verranno usati dal controllo.

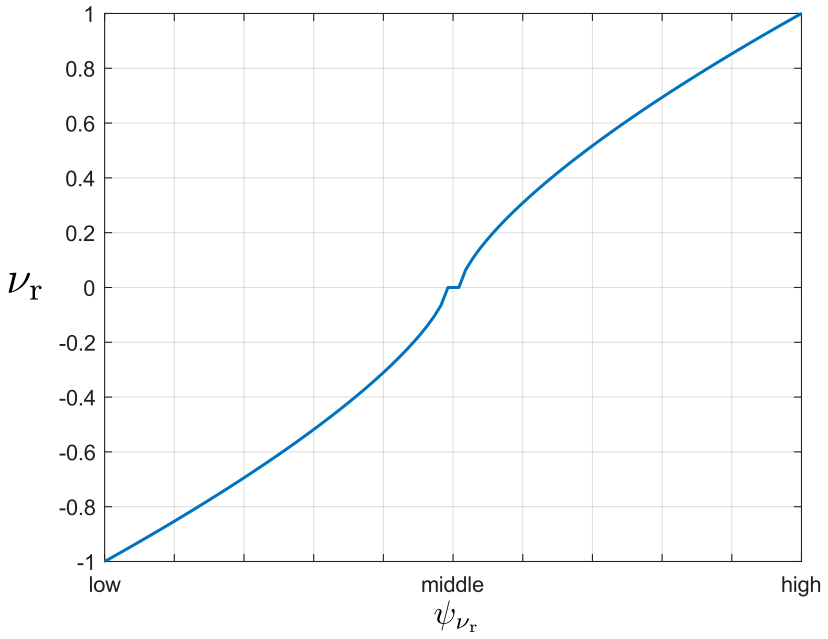


Figura 5.48: Mappatura della variabile ν_{r}

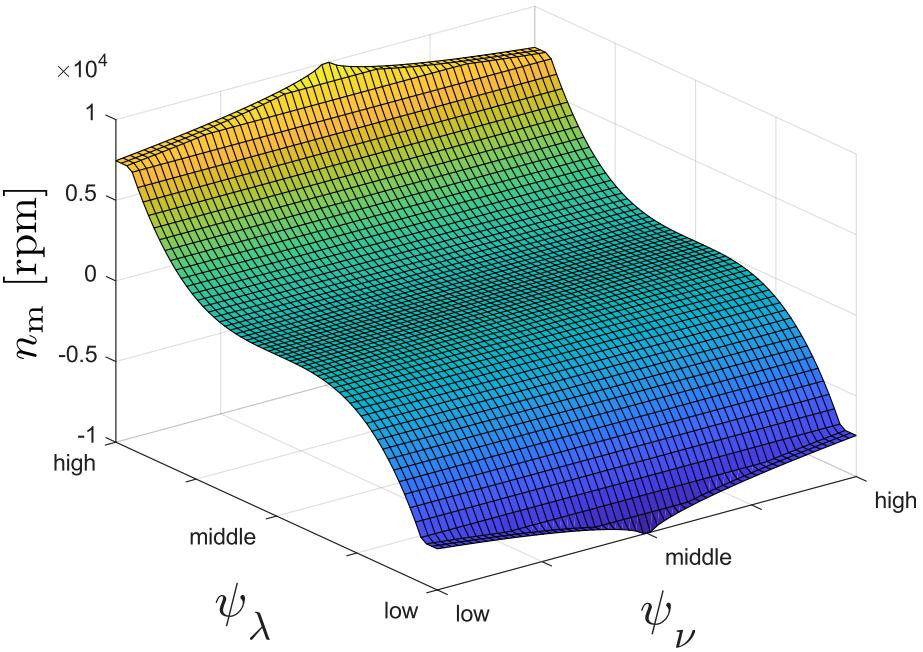


Figura 5.49: Velocità media in funzione del joystick

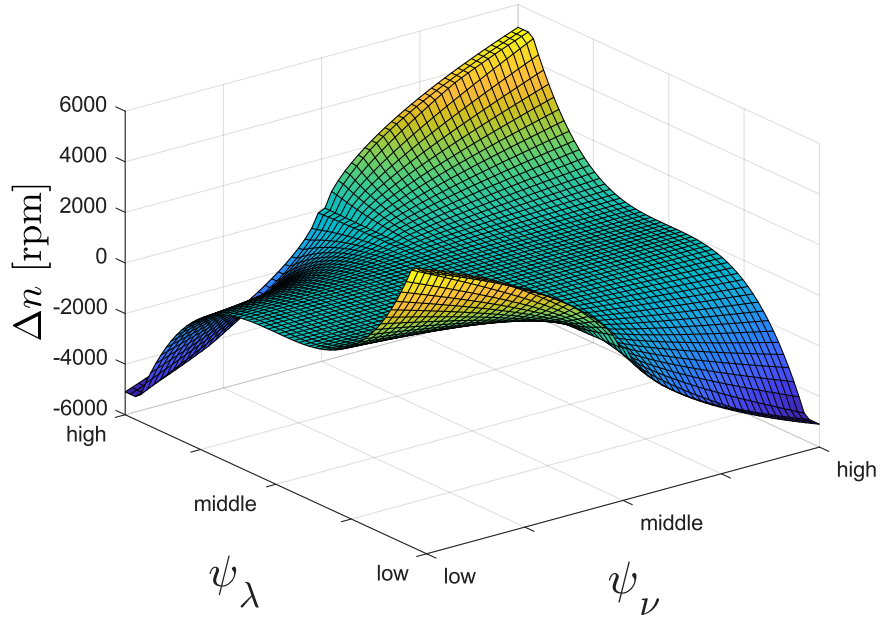


Figura 5.50: Differenza di velocità in funzione del joystick

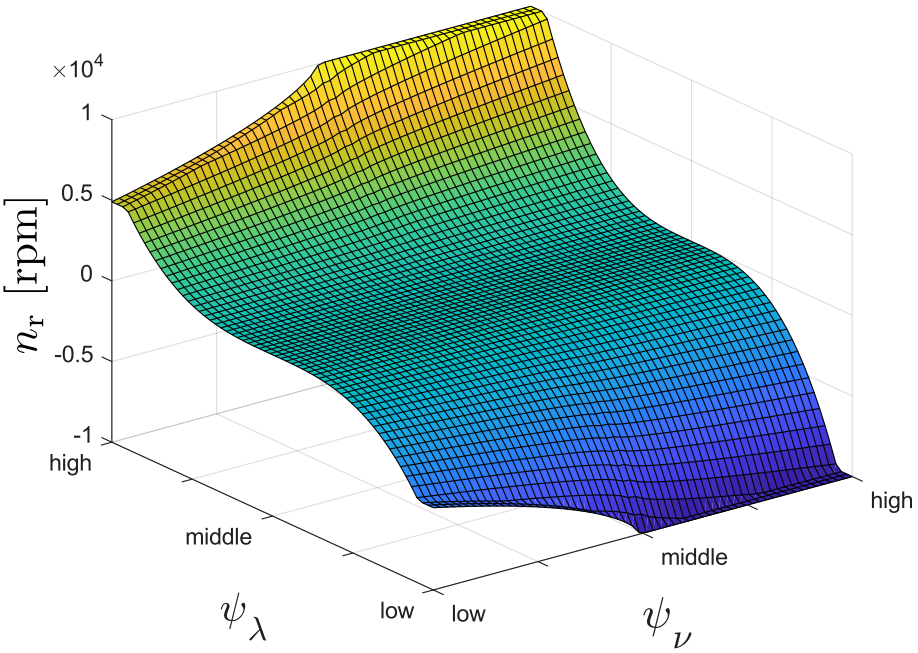


Figura 5.51: Velocità del motore destro in funzione del joystick

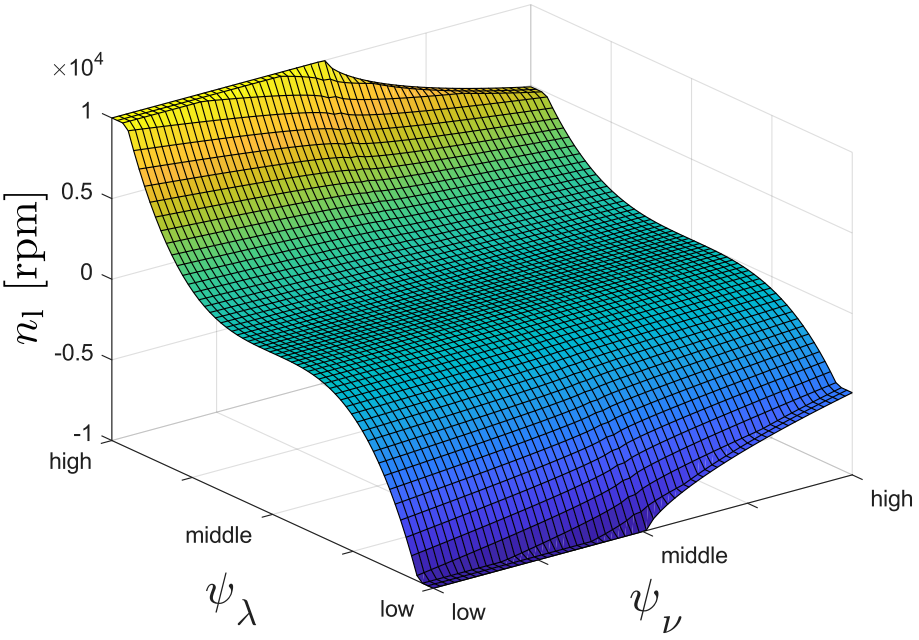


Figura 5.52: Velocità del motore sinistro in funzione del joystick

5.9 Filtraggio numerico

Le acquisizioni del microcontrollore del robot sono filtrate numericamente in tempo reale per diminuire il rumore. Al fine di ridurre al minimo la potenza computazionale richiesta e l'utilizzo di memoria, è stato implementato un filtro **EWMA** (exponentially weighted moving average) [14] [15]. L'EWMA è un filtro di risposta all'impulso infinito del primo ordine che applica pesi che diminuiscono in modo esponenziale senza mai raggiungere lo zero. L'equazione su cui si basa è la seguente:

$$S_t = \alpha Y_t + (1 - \alpha) S_{t-1} \tag{5.21}$$

Dove:

- il coefficiente α, compreso tra 0 e 1, rappresenta il peso del nuovo dato rispetto ai precedenti, quindi il grado di decadimento;
- Y_t rappresenta il nuovo dato;
- S_t rappresenta l'uscita del filtro al tempo t;

Come si può vedere è sufficiente tenere in memoria un solo valore per aver traccia della storia del segnale.

Scrivendo l'equazione per tre istanti temporali otteniamo:

$$\begin{aligned} S_t &= \alpha Y_t + (1-\alpha) S_{t-1} \\ S_{t-1} &= \alpha Y_{t-1} + (1-\alpha) S_{t-2} \\ S_{t-2} &= \alpha Y_{t-2} + (1-\alpha) S_{t-3} \end{aligned}$$

Sostituendo nella prima otteniamo:

$$S_t = \alpha[Y_t + (1-\alpha) Y_{t-1} + (1-\alpha)^2 Y_{t-2}] + (1-\alpha)^3 S_{t-3} \tag{5.22}$$

da cui possiamo scrivere un'espressione più generale:

$$S_t = \alpha \sum_{n=0}^{t} (1-\alpha)^n Y_{t-n} + (1-\alpha)^t S_0 \tag{5.23}$$

Ponendo la condizione iniziale $S_0 = 0$ otteniamo infine:

$$S_t = \alpha \sum_{n=0}^{t} (1-\alpha)^n Y_{t-n} \tag{5.24}$$

I pesi sono una progressione geometrica, la quale è una versione discreta della funzione esponenziale, da cui deriva il nome del filtro. Sapendo che la somma della serie geometrica è pari a:

$$\sum_{n=0}^{+\infty} q^n = \frac{1}{1-q} \tag{5.25}$$

Tale equazione è vera se $q < 1$. Nel nostro caso $q = 1 - \alpha$, perciò possiamo scrivere che:

$$\sum_{n=0}^{+\infty} (1-\alpha)^n = \frac{1}{1-(1-\alpha)} = \frac{1}{\alpha} \tag{5.26}$$

Se il numero di campioni utilizzati è sufficientemente alto possiamo considerare vera la 5.26 e riscrivere la 5.24 nel seguente modo:

$$S_t = \frac{\sum_{n=0}^{t} (1-\alpha)^n Y_{t-n}}{\sum_{n=0}^{t} (1-\alpha)^n} \tag{5.27}$$

In cui a numeratore avremo la somma pesata e a denominatore la somma dei pesi.

Oltre un certo numero di campioni il peso dei precedenti diventa trascurabile, a tal punto che possiamo troncare la sommatoria senza rilevanti variazioni del risultato. Arrestando la sommatoria dopo k termini e facendo il rapporto tra la somma dei pesi omessi e la somma totale dei pesi otteniamo:

$$\begin{aligned} W_{\mathrm{L}} &= \frac{(1-\alpha)^k + (1-\alpha)^{k+1} + (1-\alpha)^{k+2} + \cdots}{1 + (1-\alpha) + (1-\alpha)^2 + \cdots} \\ &= \frac{(1-\alpha)^k [1 + (1-\alpha) + (1-\alpha)^2 + \cdots]}{1 + (1-\alpha) + (1-\alpha)^2 + \cdots} \\ &= (1-\alpha)^k \end{aligned} \tag{5.28}$$

Risolvendo per k otteniamo:

$$k = \frac{\ln(W_{\mathrm{L}})}{\ln(1-\alpha)} \tag{5.29}$$

La media mobile e la EWMA hanno la stessa distribuzione dell'errore se α è vincolato al numero di campioni N della media mobile nel seguente modo:

$$\alpha = \frac{2}{N+1} \tag{5.30}$$

In virtù del fatto che se $N \to +\infty$ allora $\alpha \to 0$, si ha che $\ln(1-\alpha) \to -\alpha$ se N cresce. Quindi possiamo scrivere che:

$$k \approx \frac{\ln(W_{\mathrm{L}})}{-\alpha} = -\ln(W_{\mathrm{L}})\,\frac{N+1}{2} \tag{5.31}$$

Ponendo $k = N$ possiamo riscrivere la relazione per calcolare la percentuale di pesi trascurata.

$$W_{\mathrm{L}} = e^{-\frac{2N}{N+1}} \tag{5.32}$$

Se $N = 10$ avremo che i pesi trascurati sono pari al 16% del totale, quindi sappiamo che la maggior parte del valore di S_t dipende dalle ultime misurazioni.

Calcoliamo ora la costante di tempo τ della EWMA sapendo che è il tempo necessario, a seguito di un ingresso a gradino unitario, per raggiungere un'uscita pari a $1-1/e \approx 63.2\%$. Sapendo che la relazione tra α e τ è:

$$\alpha = 1 - e^{\frac{\Delta t}{\tau}} \tag{5.33}$$

otteniamo che:

$$\tau = -\frac{\Delta t}{\ln(1-\alpha)} \tag{5.34}$$

dove Δt è il tempo di campionamento. Se α è sufficientemente piccolo possiamo approssimare il valore di τ a:

$$\tau \approx \frac{\Delta t}{\alpha} \tag{5.35}$$

Il tempo di campionamento è pari a 10 ms in tutte le applicazioni. In tabella 5.8 sono riportati i coefficienti N scelti per ognuna di esse, con il conseguente calcolo di α e τ.

Tabella 5.8: Pesi del filtro EWMA

Applicazione	N (-)	α (-)	τ (s)
Velocità	15	0.125	0.08
Corrente	20	0.0952	0.105
Angolo	10	0.182	0.055

Capitolo 6

Test sperimentali

I Test sperimentali hanno lo scopo di validare il sistema di controllo implementato facendo un confronto sulle medesime prove tra le seguenti modalità:

- **8WD** controllata;
- **8WD** con posteriore non controllato;
- **4WD** (solo anteriore);

A tal fine sono state scelte tre prove in piano e una prova in salita. Le prove in piano solo le seguenti:

- Accelerazione da fermo con telai allineati;
- Accelerazione da fermo con telai disallineati;
- Curva di raggio costante a velocità costante e rilascio dello sterzo;

Le prove in salita sono state eseguite con inclinazioni variabili sempre con partenza da fermo ma solo nelle modalità **8WD** controllata e **4WD**.

Nelle prove in piano il comando longitudinale λ è stato posto pari a 30, che corrisponde a un set di 2400 rpm, mentre nelle prove in salita il suo valore è stato dimezzato. Per garantire le stesse condizioni in ogni prova, il comando λ viene generato tramite il cruise control, mentre non si è ritenuto necessario per il comando ν, per il quale si è usato l'analogico. Le prove in piano sono state eseguite nelle tre modalità, mentre per quelle in salita è stata tralasciata la modalità **8WD** con posteriore non controllato.

6.1 Prove in piano

6.1.1 Accelerazione da fermo con telai allineati

Posteriore controllato

Con questo primo gruppo di grafici andremo ad analizzare, oltre alle differenze tra le varie modalità, anche il funzionamento del controllo al posteriore per individuare eventuali conflitti con il controllo dell'anteriore.

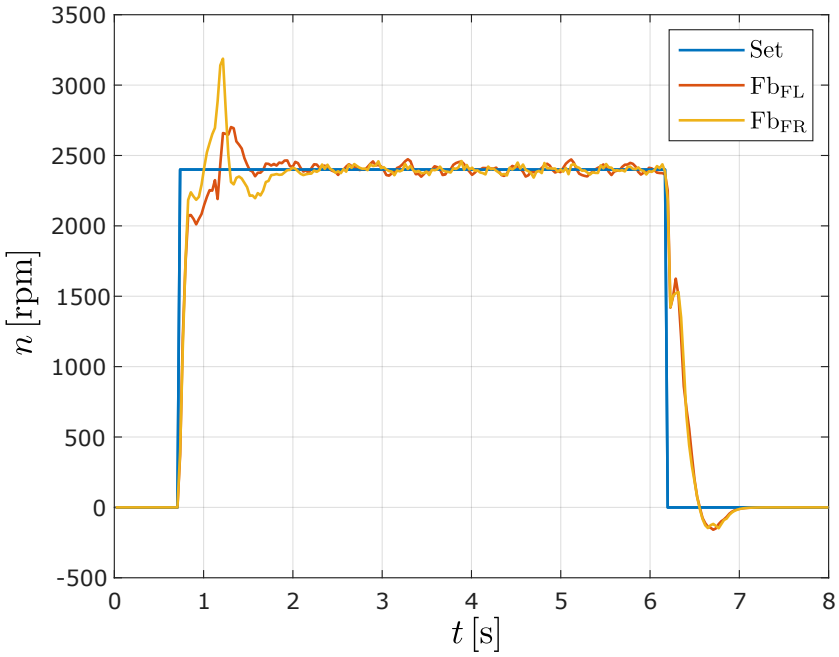


Figura 6.1: Posteriore controllato: velocità anteriore

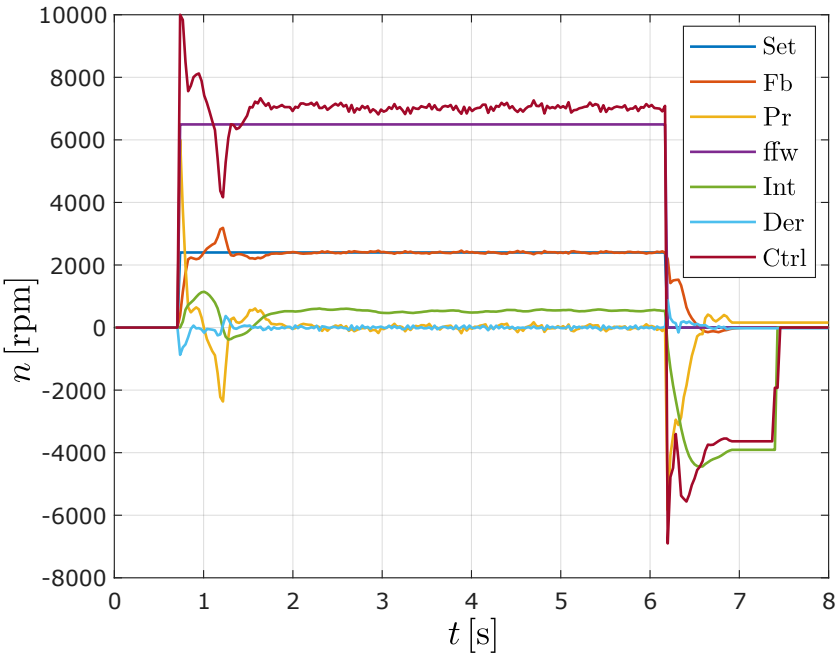


Figura 6.2: Posteriore controllato: PID

In figura 6.1 sono riportate le velocità del modulo 1, che in questo caso funge da anteriore per tutta la prova. Come si può vedere, il transitorio è molto sostenuto e ciò ha delle conseguenze sulla dinamica del robot che visibili dalle oscillazioni di velocità. Tale overshoot dipende dal fatto che il tripode, vista la coppia notevole e il trasferimento di carico al posteriore, tende a entrare in climbing mode senza però effettivamente eseguire il cambio ruote. In condizioni stazionarie le oscillazioni di velocità sono contenute, e in fase di frenata si ha una risposta altrettanto decisa, ma più morbida.

In figura 6.2 sono visibili le varie componenti del PID del controllo in velocità. Prendiamo l'anteriore destro come esempio. La maggior parte dell'uscita (Ctrl) è generata dal feedforward (ffw). La componente proporzionale e la derivativa entrano in gioco principalmente nel transitorio, mentre vediamo che l'integrativa da sempre il suo contributo. L'uscita del PID (Ctrl) è in definitiva il duty cycle che riceverà il motore, visto che tale valore viene diviso per $n_{\max}$. Inoltre, possiamo notare che nel momento in cui il set di velocità va a zero, è compito della proporzionale inizialmente e dell'integrativa poi generare un duty cycle per frenare il robot. Nell'intorno di $t = 7$ siamo fermi, ma il duty cycle è diverso da zero, quindi i motori sono ancora alimentati ma non ricevono una potenza sufficiente per la movimentazione, di conseguenza il controllo non interviene. Per mandare a zero il valore si è implementato un timer che, in caso di set e feedback pari a zero per un certo tempo, impone un reset delle componenti del pid, in particolare viene azzerata la componente integrativa forzatamente.

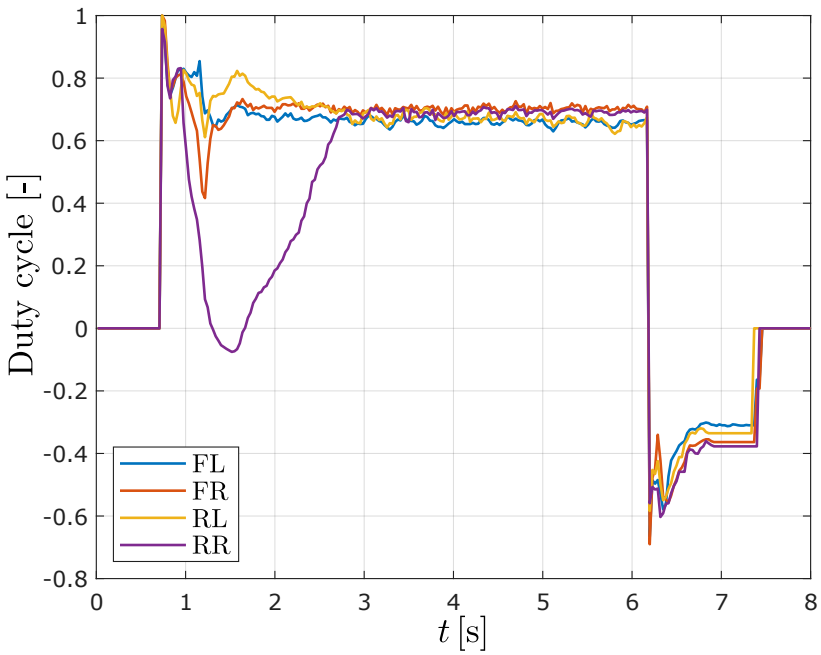


Figura 6.3: Posteriore controllato: duty cycle

Confrontando i quattro duty cycle (figura 6.3) si può notare che a regime i valori sono pressoché simili, soprattutto tra i motori dello stesso lato, sintomo che la spinta è equamente distribuita tra anteriore e posteriore. Nella fase di transitorio il motore posteriore destro

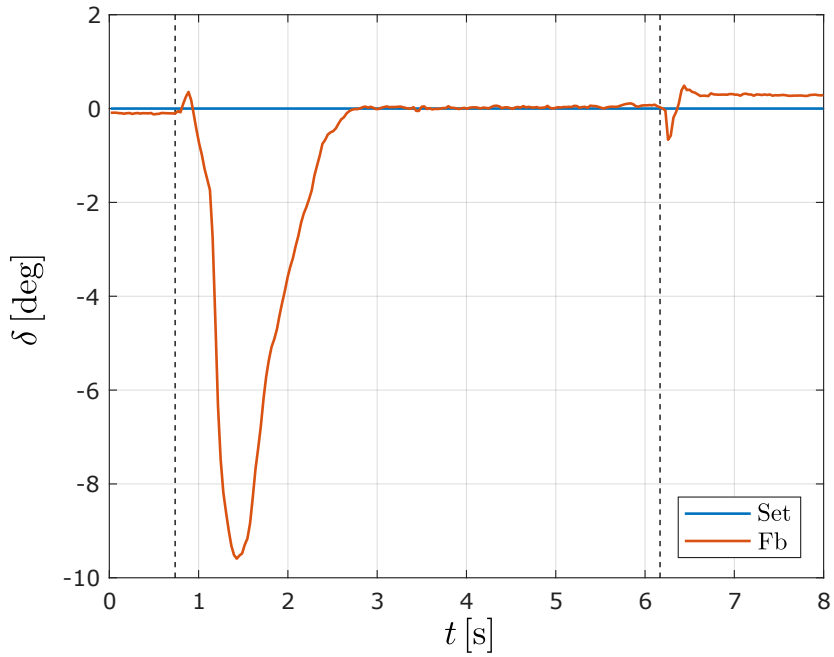


Figura 6.4: Posteriore controllato: angolo di imbardata

ha subito una forte diminuzione della spinta per via del fatto che è entrato in funzione il controllo fuzzy per ripristinare l'allineamento tra i due telai. Anche in frenata i motori lavorano abbastanza sincronizzati.

In figura 6.4 analizziamo l'allineamento dei telai durante la prova. Prima dell'applicazione del set di velocità (prima linea verticale tratteggiata) il potenziometro misura un angolo circa pari a zero. Nel momento in cui il robot inizia a muoversi, a causa di una diversa spinta dei motori, c'è una perdita dell'allineamento che viene poi recuperata tramite il controllo fuzzy, riducendo la spinta sulla ruota posteriore destra. Si ha inoltre una leggera oscillazione in fase di frenata.

Osservando i coefficienti fuzzy (figura 6.5) notiamo come il controllo interviene principalmente tramite la ruota interna (C_{i}), sia quando l'errore è grande, sia quando esso tende a zero. Questo non significa che viene modulata una singola ruota, perché tale coefficiente viene assegnato all'una o all'altra a seconda del segno di δ_{fb}. In particolare, se il set angolare è zero, come in questo caso, avremo uno switch continuo dei coefficienti.

Riportiamo anche il grafico delle correnti (figura 6.6), che è da prendere un po' con le pinze per via dei possibili disturbi elettromagnetici.

Nel primo tratto di salita, molto probabilmente, non possiamo apprezzare il reale andamento a causa dell'elevato coefficiente scelto per il filtro EWMA, ma ciò è stato reso necessario dall'elevata rumorosità del segnale. In condizioni di regime notiamo come le correnti del lato sinistro siano circa le stesse, mentre abbiamo una discrepanza tra il motore anteriore e il motore posteriore del lato destro.

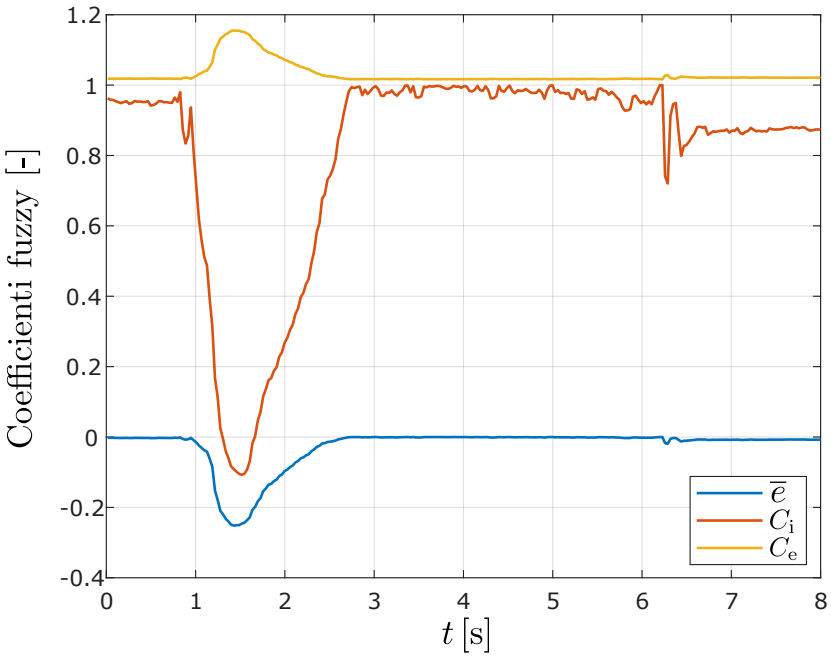


Figura 6.5: Posteriore controllato: coefficienti fuzzy

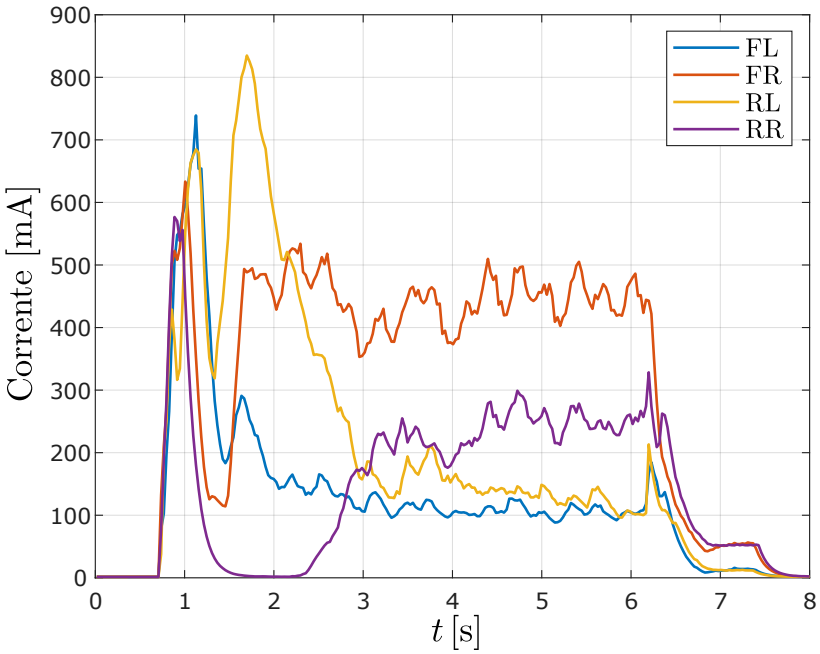


Figura 6.6: Posteriore controllato: correnti

Posteriore non controllato e posteriore disabilitato

Da qui in poi, e ove possibile, accorperemo insieme i grafici per vedere meglio i differenti comportamenti del robot.

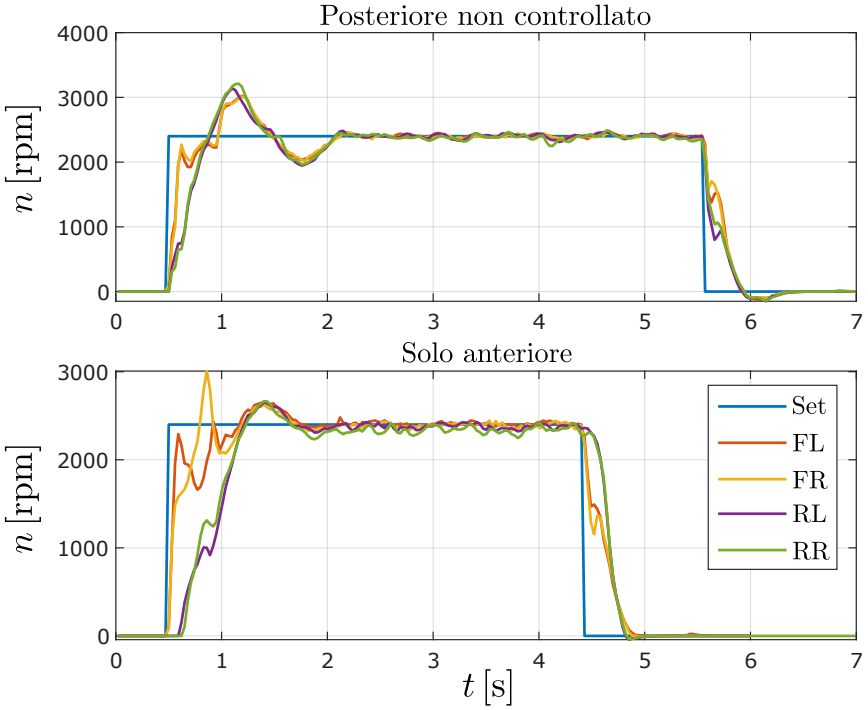


Figura 6.7: Velocità angolare dei motori

In figura 6.7 abbiamo il primo confronto tra posteriore abilitato ma non controllato e il moto tramite il solo anteriore. Come si può vedere la fase di accelerazione è più decisa quando è anche il posteriore a fornire coppia, ma il tempo per arrivare a regime è pressoché lo stesso perché c'è una maggiore oscillazione delle velocità. Nella prova condotta con il solo anteriore si può notare maggiormente l'effetto che la rotazione dei tripodi ha sulla velocità dei motori. Nel momento in cui il tripode inizia la sua rotazione e la ruota più arretrata si solleva c'è una diminuzione di coppia resistente che fa aumentare la velocità di rotazione dei motori che il controllo deve compensare. La fase di frenata è pressoché uguale in entrambi i casi visto che c'è un elevato trasferimento di carico sull'anteriore.

In figura 6.8 possiamo notare come i duty cycle dell'anteriore siano pressoché gli stessi, nel caso di posteriore non controllato, e che c'è un piccolo scarto con il duty del posteriore dovuto al coefficiente di riduzione impostato che è pari a 0.9. Nel caso del solo anteriore si nota invece che in fase di accelerazione il tentativo di rotazione dei tripodi è avvenuto in momenti leggermente diversi e che a regime c'è una differenza consistente tra i duty. Tale fenomeno potrebbe essere causato da differenti attriti dei motori e dei tripodi stessi.

In figura 6.9 riportiamo l'andamento delle correnti. Nel caso di posteriore trascinato si hanno correnti nulle al retrotreno e correnti all'avantreno un po' superiori rispetto all'altro caso, con un trend leggermente decrescente. Con l'applicazione di un coefficiente di

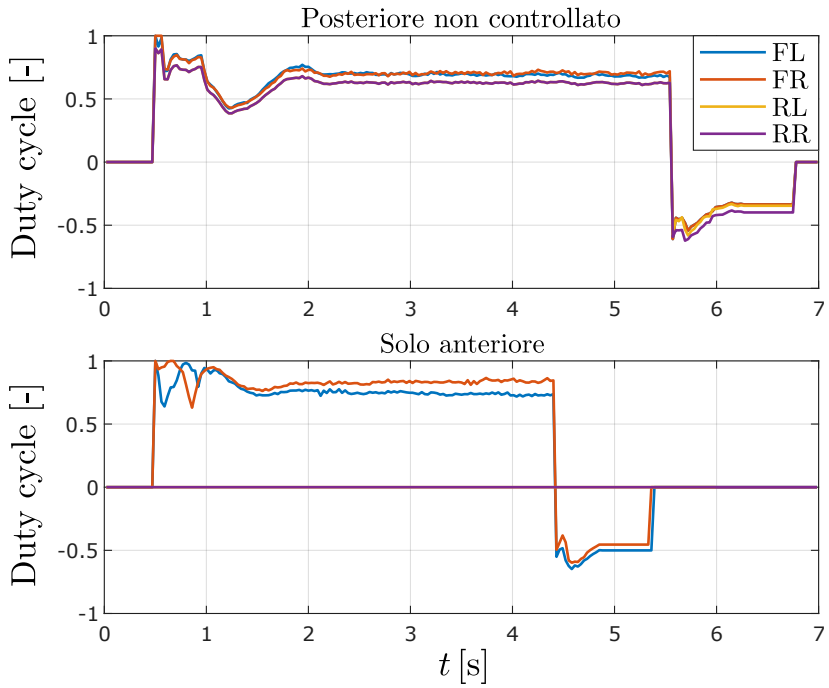


Figura 6.8: Duty cycle

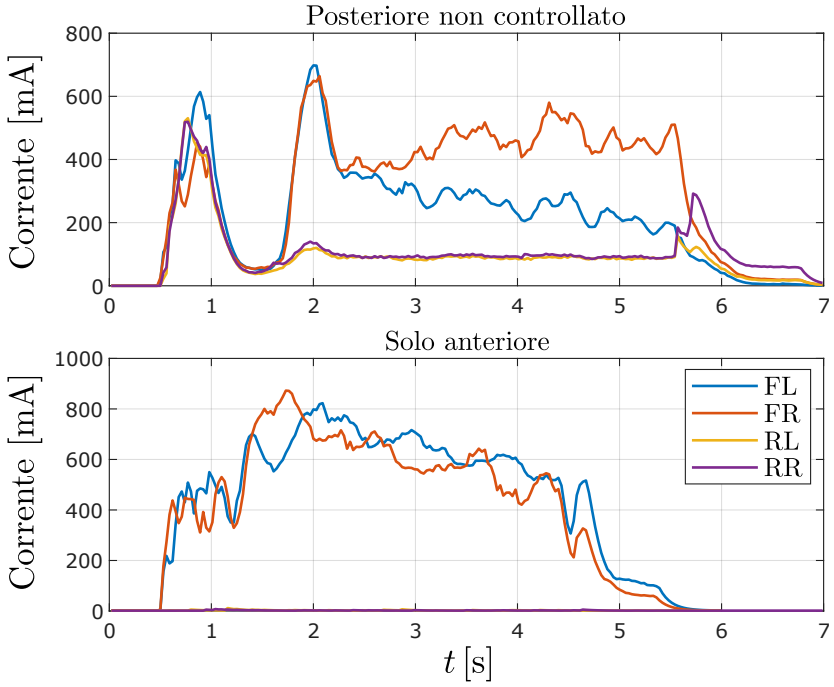


Figura 6.9: Correnti

scala pari a 0.9 si può notare come il posteriore non eserciti un notevole contributo nella movimentazione del robot, soprattutto in condizioni di regime. Nel grafico in alto, si nota anche una progressiva diminuzione della corrente del motore anteriore sinistro. Ciò può dipendere da una differente spinta generata dal posteriore causata da differenti attriti tra il lato destro e il sinistro, e forse anche da errori del sensore stesso.

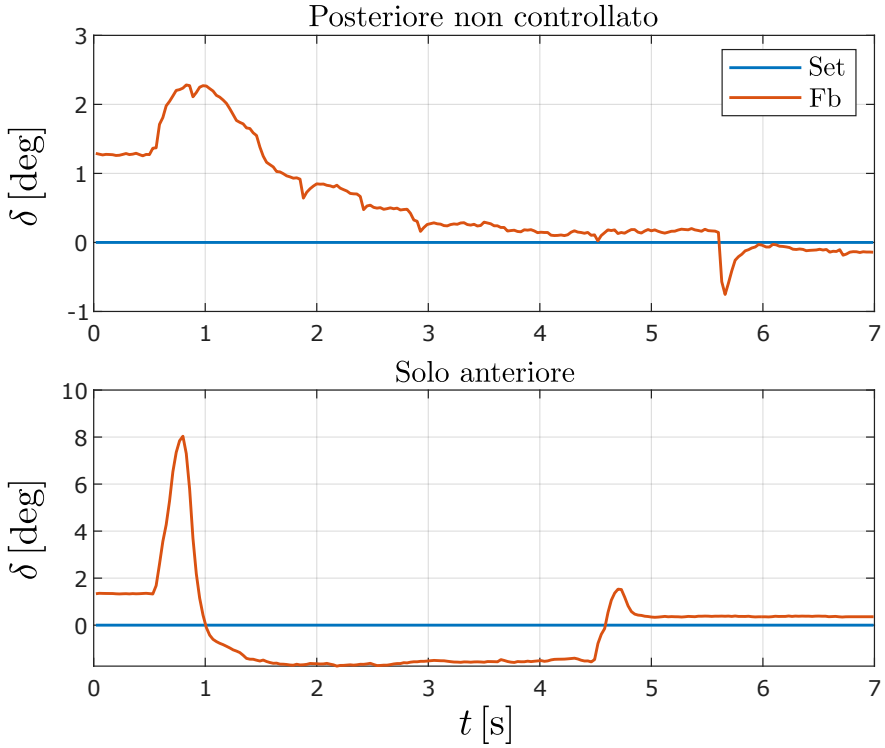


Figura 6.10: Angolo di imbardata

Osservando il grafico dell'angolo δ (figura 6.10) notiamo come in entrambi i casi c'è una leggera rotazione nella fase iniziale che dipende sia dalla spinta non perfettamente simmetrica, sia dalla rotazione del portatreno dei tripodi. Notiamo inoltre che il robot tende a recuperare questo allineamento ma non sempre riesce completamente, come nella prova con il solo anteriore in cui deriva leggermente verso destra.

Osservando il grafico abbiamo che il valore dell'angolo è pari a circa -1.5°, che implica la percorrenza di una curva oraria con un raggio di circa 10 m, quindi abbastanza visibile anche senza troppa strada percorsa. Il solo controllo in velocità non è in grado di compensare efficacemente tale effetto.

Situazione un po' migliore si ha con l'utilizzo del posteriore, indipendentemente se controllato o meno, anche se durante la guida in campo libero si è visto che tale effetto si presenta comunque.

6.1.2 Accelerazione da fermo con telai disallineati

La prova consiste nel posizionare il robot con i telai disallineati e il giunto di imbardata a battuta, quindi con un angolo di 38°, e applicare un comando di avanzamento longitudinale.

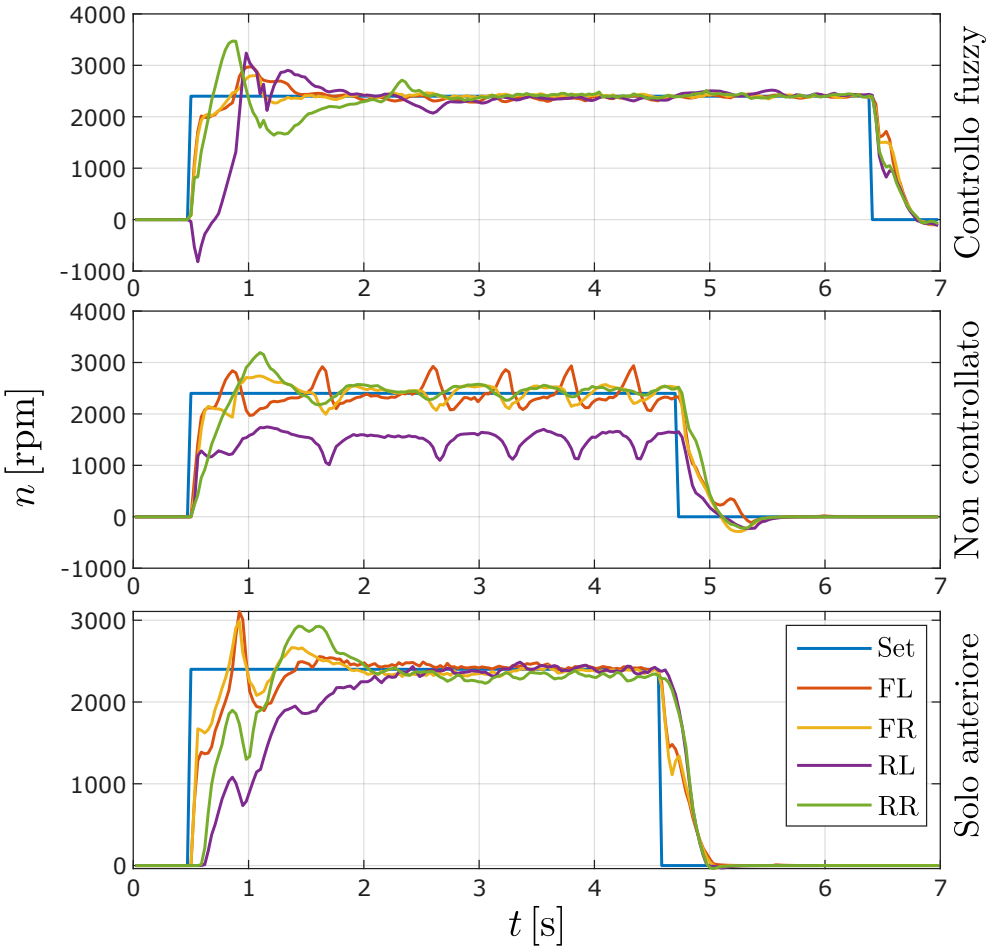


Figura 6.11: Velocità angolare dei motori

In figura 6.11 abbiamo il confronto tra le velocità dei motori nelle tre prove. Notiamo subito che nel caso di posteriore non controllato si hanno delle notevoli oscillazioni per tutta la durata della prova, in aggiunta al fatto che la posteriore sinistra non raggiunge la velocità target. Ciò dipende dal fatto che la spinta del posteriore genera un carico maggiore sul tripode anteriore destro, che è esterno alla curva, ed alleggerisce l'anteriore sinistro, il quale entra in climbing mode e non genera spinta sul terreno (o comunque non paragonabile al destro). Avendo un disequilibrio di spinta, il robot non riesce ad allineare i due telai e continua a percorrere una traiettoria circolare. Possiamo infatti notare che la velocità misurata sul motore posteriore sinistro è inferiore rispetto agli altri e se guardiamo il grafico dell'angolo di imbardata ne abbiamo la conferma. Per evitare che il tripode inizi a ruotare è necessario fornire un set di velocità più graduale, ed evitare strappi improvvisi.

Con l'implementazione del controllo fuzzy questo problema è stato risolto perché si riesce a modulare la spinta generata dal posteriore. Possiamo notare che la velocità della posteriore sinistra è negativa per il primo tratto per consentire il rapido recupero dell'allineamento. Ciò permette al robot di impiegare un minor tempo per arrivare alla velocità richiesta rispetto all'utilizzo del solo anteriore.

Nel grafico della prova condotta con il solo anteriore notiamo inoltre dei picchi di velocità, sintomo di una rotazione dei tripodi causata dall'applicazione della coppia di spunto. Oscillazioni di velocità che vediamo leggermente anche quando è attivo il controllo fuzzy.

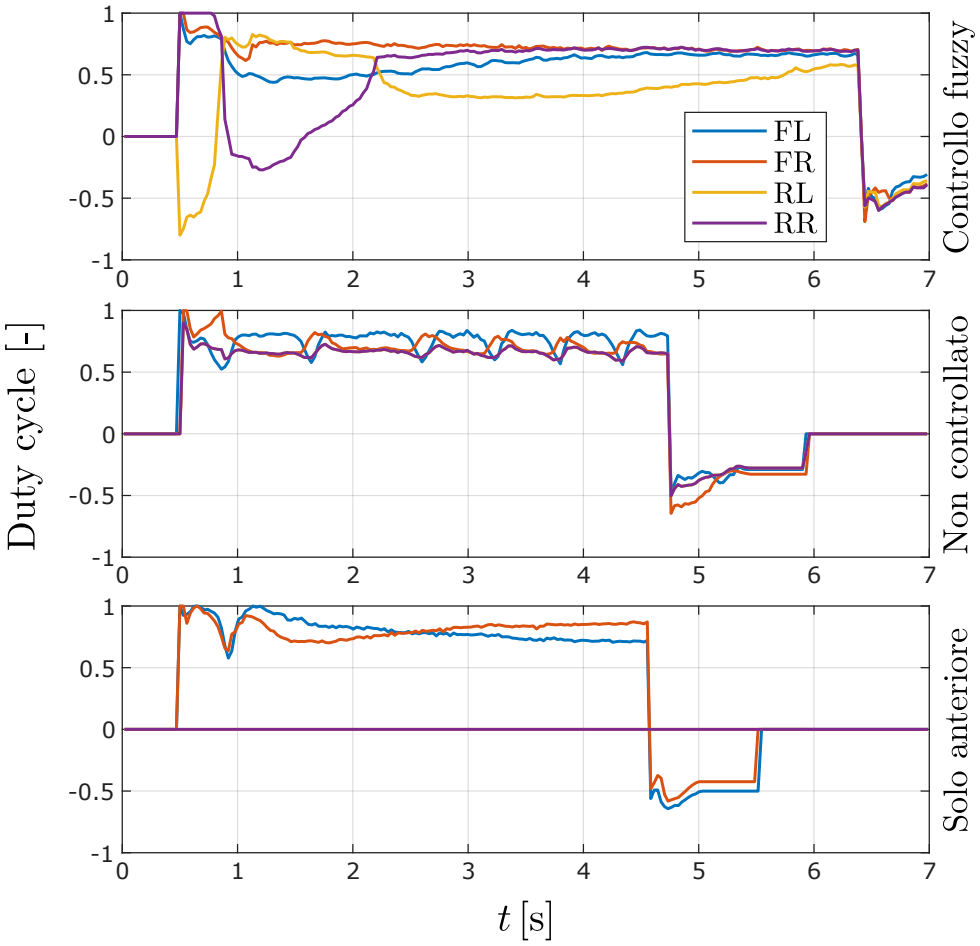


Figura 6.12: Duty cycle

Osservando i duty cycle (figura 6.12) notiamo come il controllo fuzzy lavori attivamente sul posteriore andando a generare grandi variazioni sui comandi del posteriore al fine di compensare quanto prima il disallineamento.

È interessante osservare i diversi comportamenti tramite l'angolo di imbardata (figura 6.13). In primo luogo vediamo come, nel caso di posteriore non controllato, il robot non recuperi l'allineamento. Mentre nelle altre due prove abbiamo che il tempo impiegato per raggiungere la configurazione voluta è praticamente lo stesso, ma le modalità sono differenti.

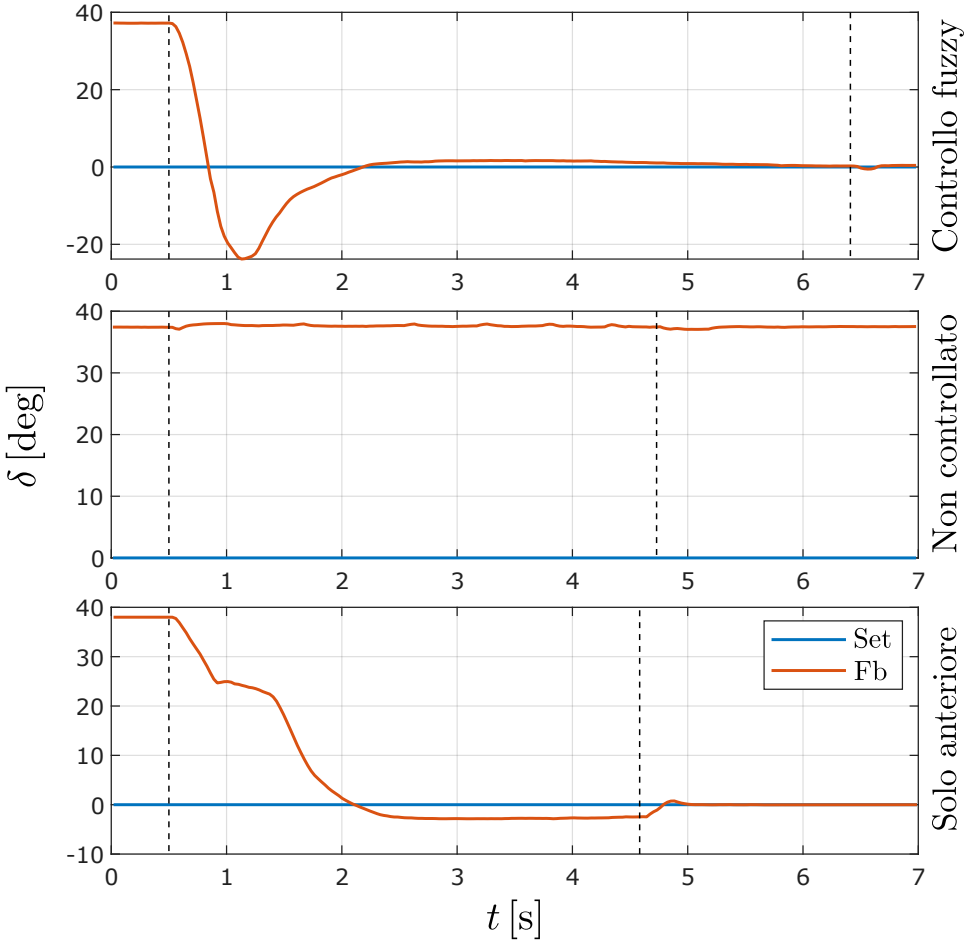


Figura 6.13: Angolo di imbardata

Nel caso del solo anteriore, il robot percorre un tratto di curva prima di raggiungere il suo valore di regime, mentre con il controllo fuzzy ciò non avviene. La forte compensazione imposta dal controllo fa si che ci sia una netta rotazione del posteriore prima di tendere verso $\delta = 0$. In tal modo l'anteriore percorre una traiettoria più rettilinea rispetto all'altro caso. In ultimo possiamo dire che il controllo fuzzy garantisce un minor errore a regime, limitando notevolmente la deriva del robot.

Analizzando le correnti circolanti all'interno dei motori (figura 6.14) possiamo notare un andamento alquanto oscillatorio, nel caso di posteriore non controllato, che rispecchia l'andamento delle velocità.

Nel caso del solo anteriore abbiamo invece un grafico abbastanza simmetrico tra i due lati, il sinistro è però lievemente superiore, infatti il robot tende a derivare verso destra.

Con l'ausilio del controllo fuzzy, le maggiori correnti si hanno nel lato destro del robot, ad eccezion fatta per un paio di picchi durante la fase di accelerazione. Si nota la tendenza delle correnti a convergere verso un valore più o meno comune, se la prova fosse continuata. Riprendendo il grafico dell'angolo di imbardata vediamo come il valore stava pian piano

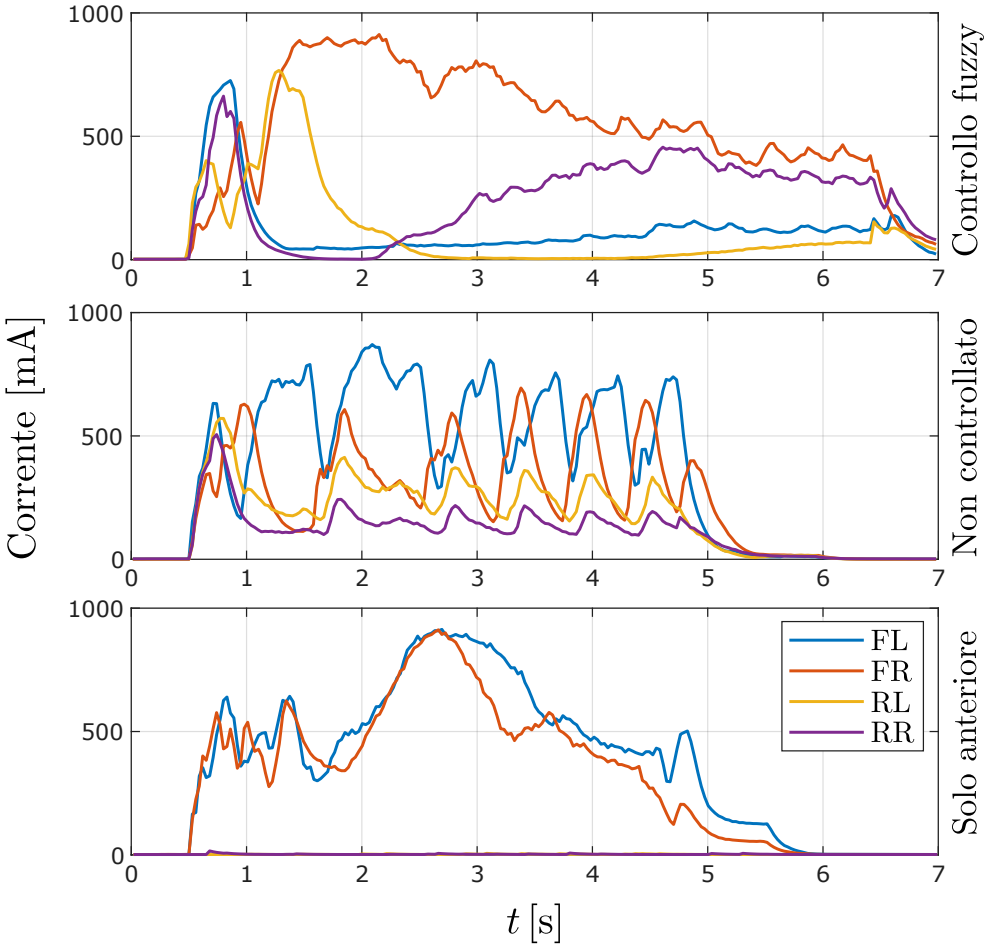


Figura 6.14: Correnti

convergendo al valore di set.

Riportiamo in figura 6.15 l'andamento dei coefficienti fuzzy, sia in termini di ruota interna/esterna, quindi come vengono generati, sia in termini di ruota destra/sinistra, quindi come sono assegnati.

Notiamo subito che il coefficiente che fornisce la maggior compensazione è il C_{i} (grafico in alto), mentre il coefficiente della ruota esterna entra in gioco solo per elevati valori dell'errore, al contrario, tende a uno quando l'errore tende ad annullarsi.

Nel grafico inferiore vediamo come effettivamente i due coefficienti vengono assegnati alla ruota destra o alla sinistra, in funzione di quale sia quella interna.

6.1.3 Rilascio dello sterzo

Questa prova prevede che il robot percorra a velocità costante un tratto rettilineo, una curva e un nuovo tratto rettilineo, al fine di analizzare il comportamento sia in ingresso, ma soprattutto in uscita di curva.

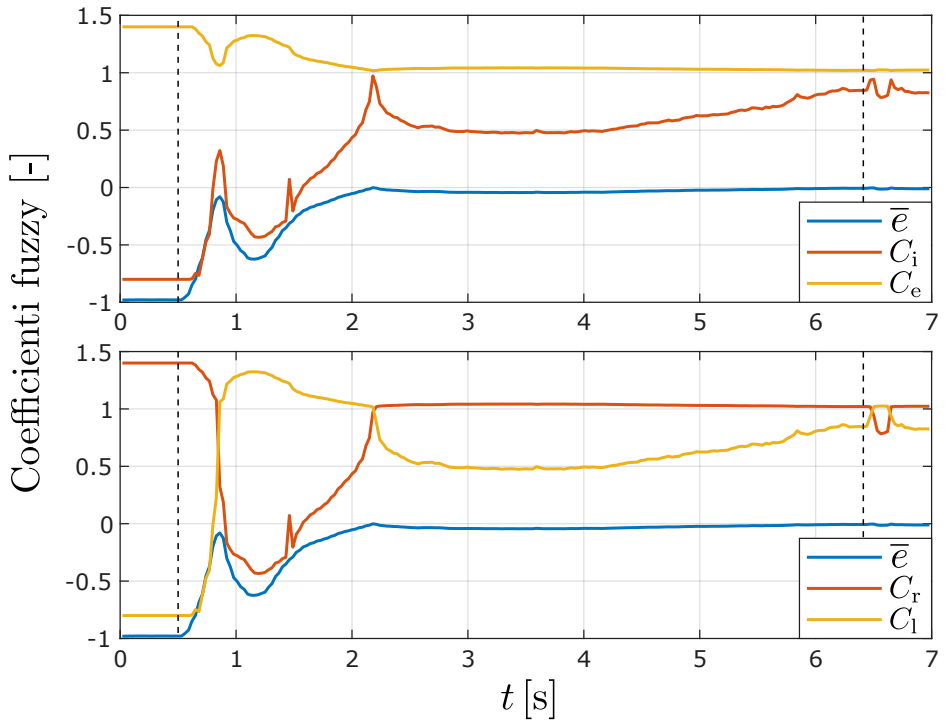


Figura 6.15: Coefficienti fuzzy

Questa volta riportiamo prima il grafico dell'angolo di imbardata (figura 6.16) che è molto più esplicativo degli altri. Come si può notare, l'ingresso in curva è un po' più netto quando si utilizza il controllo fuzzy rispetto alle altre due prove. Il caso con il posteriore non controllato è anche leggermente peggiore rispetto al solo anteriore.

Il controllo fuzzy fa una differenza netta in fase di rilascio dello sterzo. Nel primo caso il tempo che impiega a riallinearsi è circa pari a quello impiegato per l'ingresso curva, mentre lo stesso non si può dire per le altre due prove. Nel caso di posteriore non controllato non riusciamo proprio a percorrere il secondo tratto rettilineo perché, a causa della spinta del posteriore, la ruota anteriore interna ha poco carico verticale e di conseguenza il tripode inizia a ruotare. Senza un rilascio di λ ed una successiva graduale ripresa non si riesce a riportare il robot nella configurazione iniziale.

Osservando la prova condotta con il solo anteriore, è visibile un certo ritardo tra il rilascio del comando e l'effettivo inizio di variazione dell'angolo δ, oltre al fatto che la pendenza della curva di risalita è decisamente inferiore. In aggiunta al ritardo, che significa più strada percorsa in curva, c'è bisogno di un ulteriore tratto per consentire il riallineamento dei due moduli, il tutto si traduce in uno scarso comfort di guida.

Riportando quanto appena detto alla figura 6.17 notiamo come, soprattutto in uscita curva con il controllo fuzzy attivo, le velocità del posteriore si discostano molto da quelle dell'anteriore per consentire il rapido allineamento. Si può osservare inoltre come la posteriore destra (interna) abbiamo una velocità più elevata rispetto all'anteriore destra, sintomo di uno slittamento.

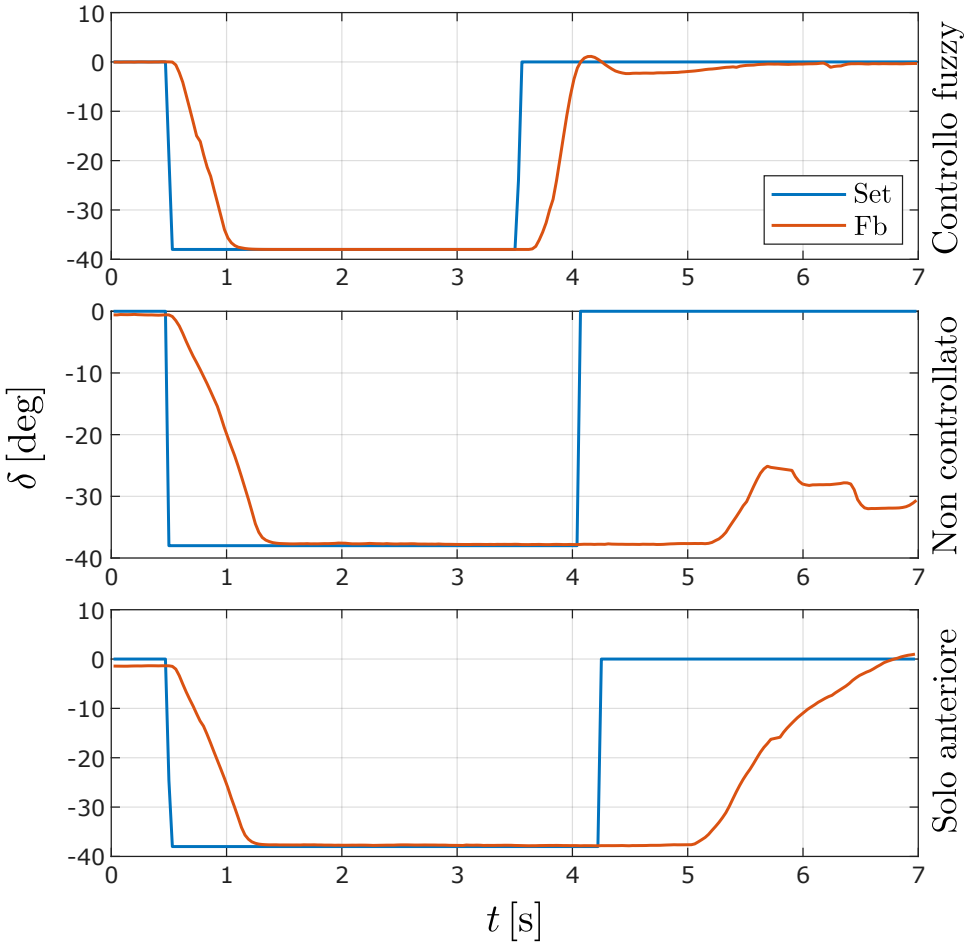


Figura 6.16: Angolo di imbardata

Osservando il grafico del posteriore non controllato si notano le oscillazioni di velocità dovute alla rotazione del tripode e di conseguenza la minore velocità della ruota posteriore.

In generale con il controllo fuzzy si hanno delle velocità che vanno a convergenza molto più velocemente, e al di là dei grafici, il pilotaggio tramite joystick risulta decisamente più semplice.

In figura 6.18 osserviamo come variano i duty cycle nei tre casi presi in esame. Durante la percorrenza della curva, con il controllo fuzzy attivo abbiamo che la ruota anteriore interna riceve un comando basso, mentre è l'esterna che si occupa di fornire gran parte della coppia. Il posteriore invece si mantiene in un livello intermedio tra i due fornendo una spinta di accompagno. Ciò avviene perché il duty è derivato dalla media dell'anteriore moltiplicato per i coefficienti fuzzy, e visto che l'errore angolare è circa zero, perché il giunto è in battuta, essi sono circa pari a uno.

Nelle altre due prove, notiamo come il comando della ruota anteriore interna tenda a diventare negativo per riuscire a convergere con il set di velocità imposto, ricordando che un comando negativo indica che il motore sta frenando.

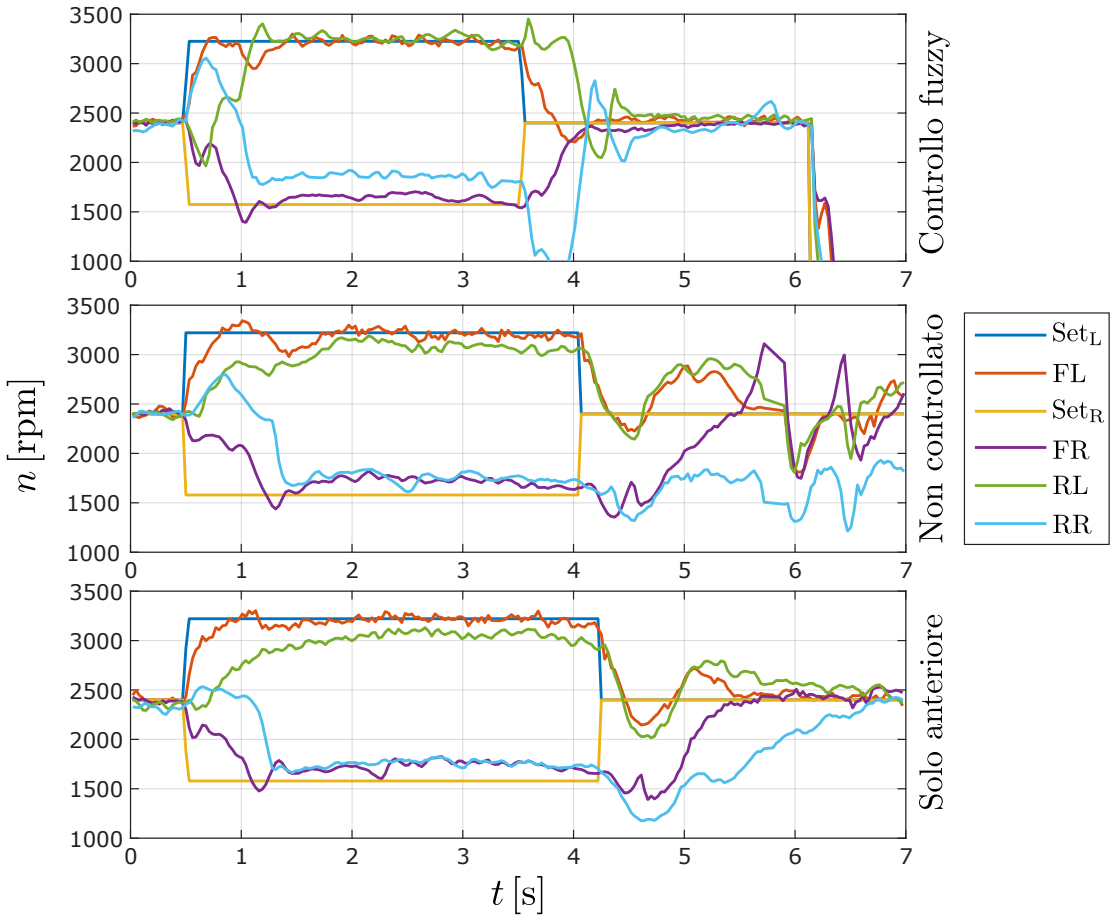


Figura 6.17: Velocità angolare dei motori

Nel momento di rilascio della curva si vede come il fuzzy interviene prontamente per riallineare i due telai, mentre negli altri casi il comando è molto più lento a risalire.

Altro lato positivo del controllo fuzzy è che fa lavorare la ruota anteriore esterna non troppo vicina alla saturazione, come invece accade sotto.

Analizzando le correnti (figura 6.19) notiamo come nel caso di solo anteriore o posteriore non controllato l'unico motore che fornisce coppia è l'anteriore sinistro (ruota esterna), mentre con il controllo fuzzy attivo il carico si ripartisce anche un po' sul posteriore.

6.2 Superamento ostacolo

La prova di superamento ostacolo prevede che il robot parta da fermo già direzionato verso l'ostacolo, si avvicini a velocità costante e lo superi percorrendo un altro piccolo tratto. Invertendo la direzione del moto si ripercorre lo stesso tragitto fino a ritornare nella posizione iniziale. Il terreno prima dell'ostacolo è cemento, mentre oltre l'ostacolo è presente della ghiaia.

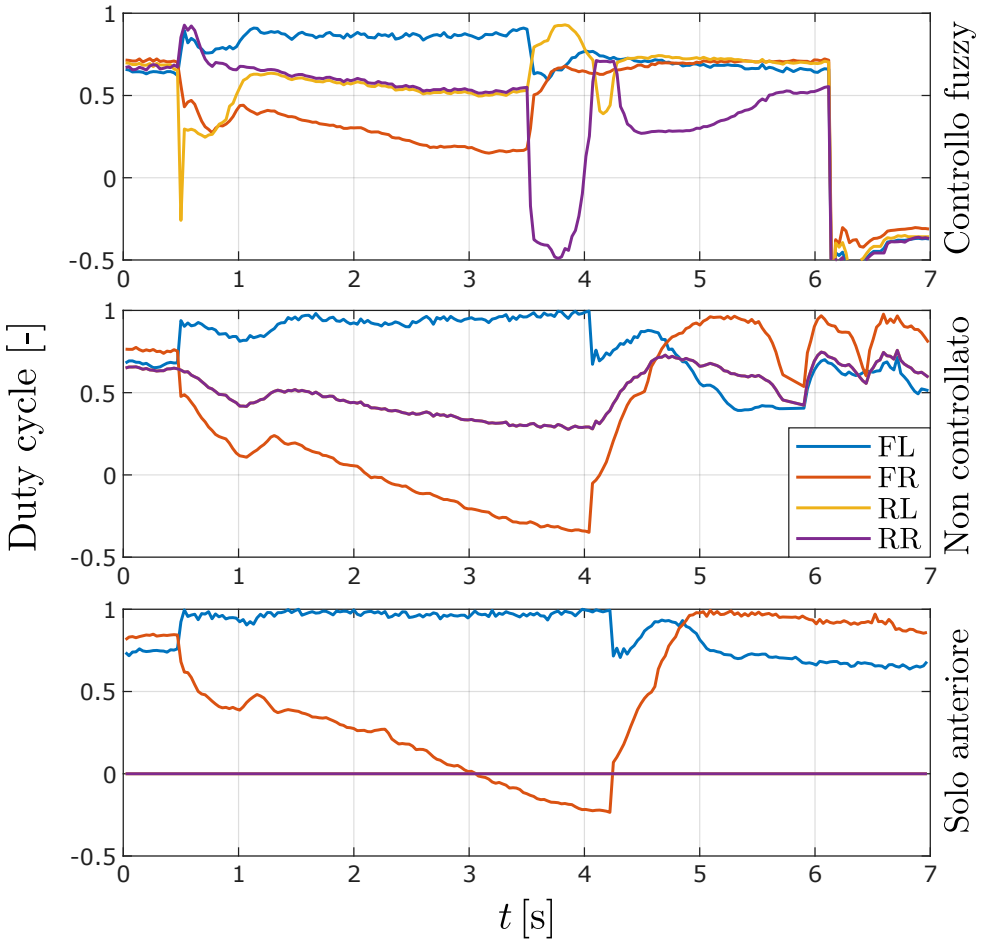


Figura 6.18: Duty cycle

In figura 6.20 è schematizzato l'ostacolo da superare e le sue dimensioni, con una rappresentazione in scala del tripode in modo da avere le giuste proporzioni. Questa tipologia di ostacoli è tra le più difficili da superare perché l'anteriore finisce di scavalcarlo prima che il posteriore inizi.

Se l'anteriore è l'unico modulo che fornisce coppia, il robot non sarà in grado di superare l'ostacolo perché la forza di tiro che esercita sul posteriore è quasi orizzontale, in aggiunta al fatto che essendo su ghiaia c'è un maggiore slittamento delle ruote. L'unico modo per superare questa tipologia di ostacolo è avere un posteriore attivo.

In figura 6.21 è riportato il grafico delle velocità. Possiamo notare come, nonostante l'ostacolo, le velocità dell'anteriore non subiscono grandi fluttuazioni, al contrario del posteriore.

nella fase di andata, il momento di superamento dell'ostacolo è riconoscibile dalla diminuzione di velocità del posteriore (RL e RR) al tempo $t = 2\,\mathrm{s}$, mentre il tratto di percorrenza sulla ghiaia è riconoscibile dalle elevate fluttuazioni. A produrre tali fluttuazioni concorre anche il controllo fuzzy che applica molte correzioni.

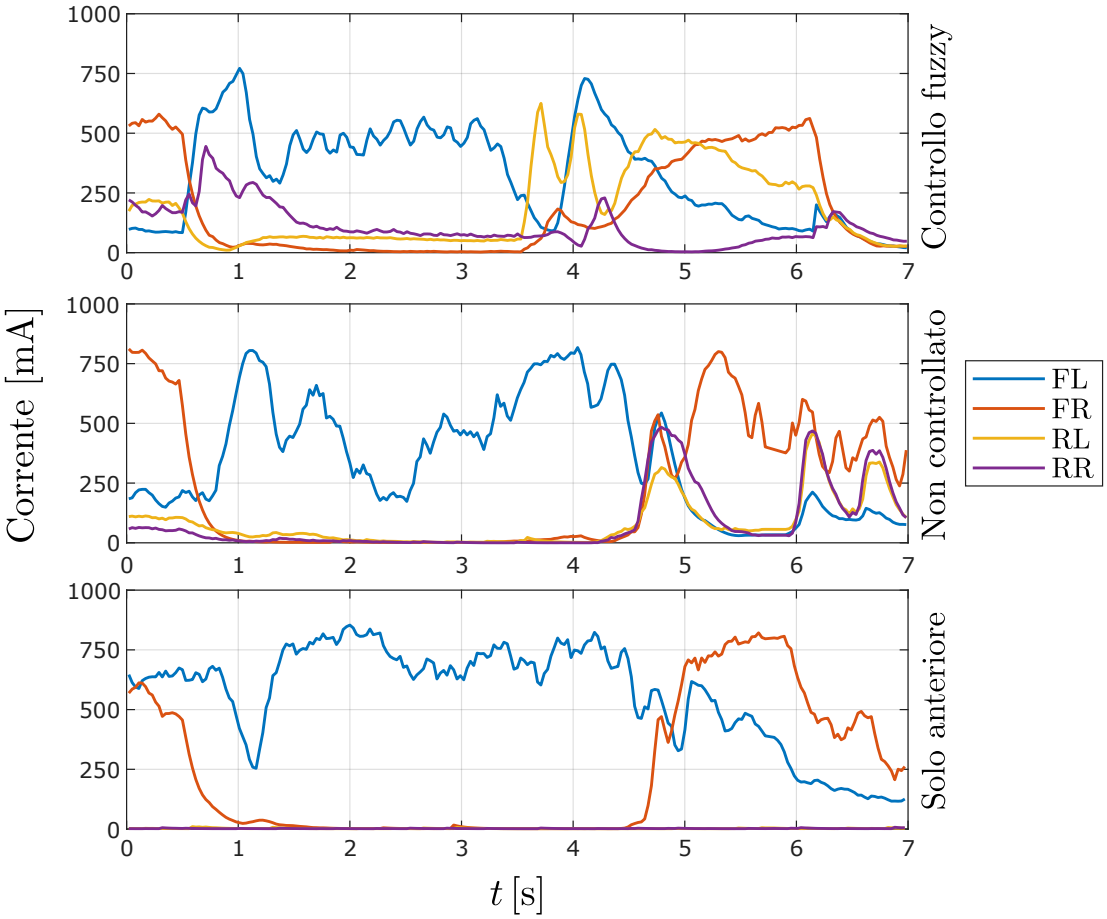


Figura 6.19: Correnti

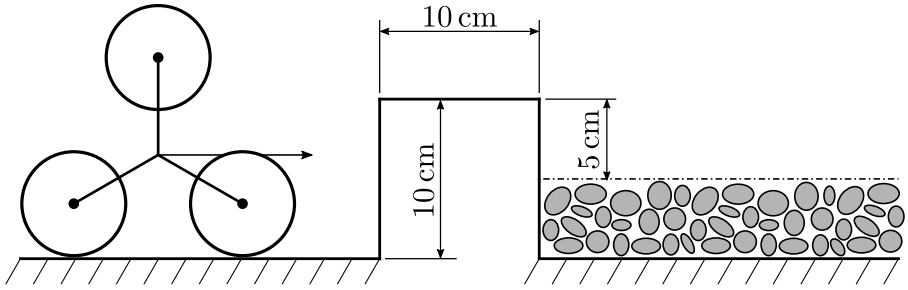


Figura 6.20: Schema ostacolo

Nel momento di inversione di marcia, si nota come il controllo in velocità passa all'altro modulo. Ci ritroviamo così con le stesse dinamiche di prima, ma con i motori invertiti.

Osservando i comandi inviati ai motori (figura 6.22) possiamo notare come ci sia una netta differenza tra il modulo controllato in velocità e il modulo controllato in duty cycle,

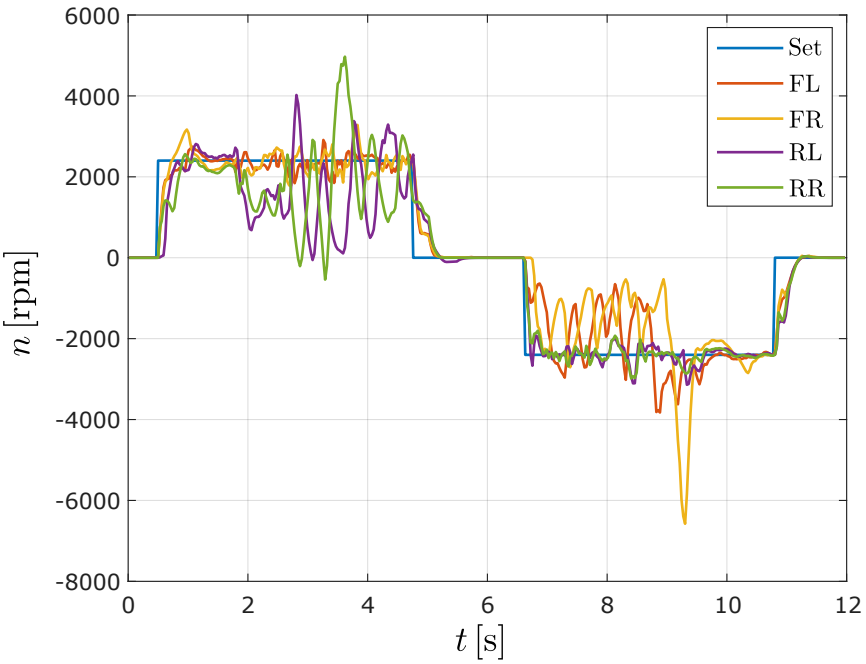


Figura 6.21: Superamento ostacolo: velocità anteriore

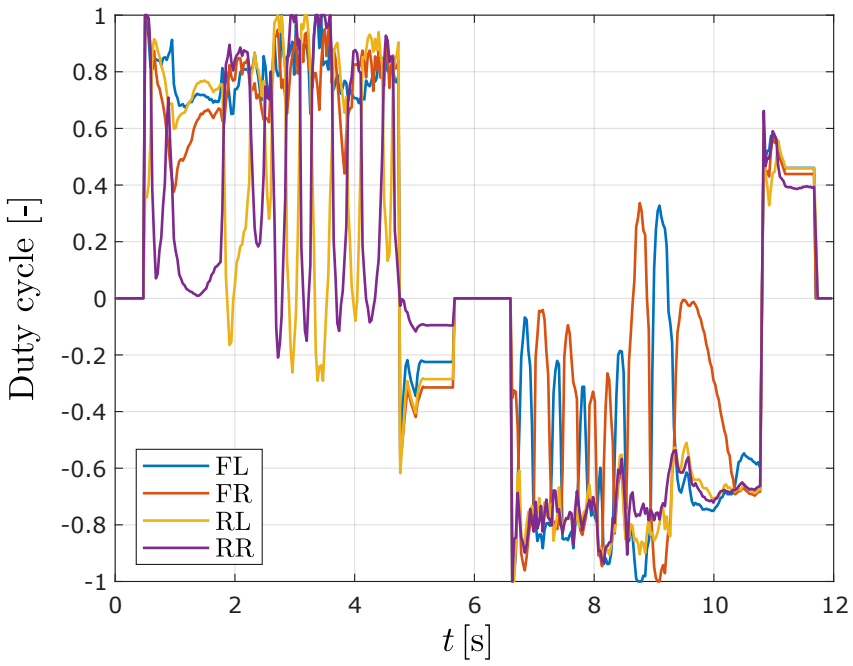


Figura 6.22: Superamento ostacolo: duty cycle

proprio a causa delle elevate correzioni imposte dal controllo fuzzy. In pratica abbiamo un posteriore che spinge a impulsi e non in maniera continua, esso si occupa perlopiù di mantenere l'allineamento. Non riportiamo il grafico delle correnti perché è di difficile lettura ed è facile intuirlo visto l'andamento dei duty cycle.

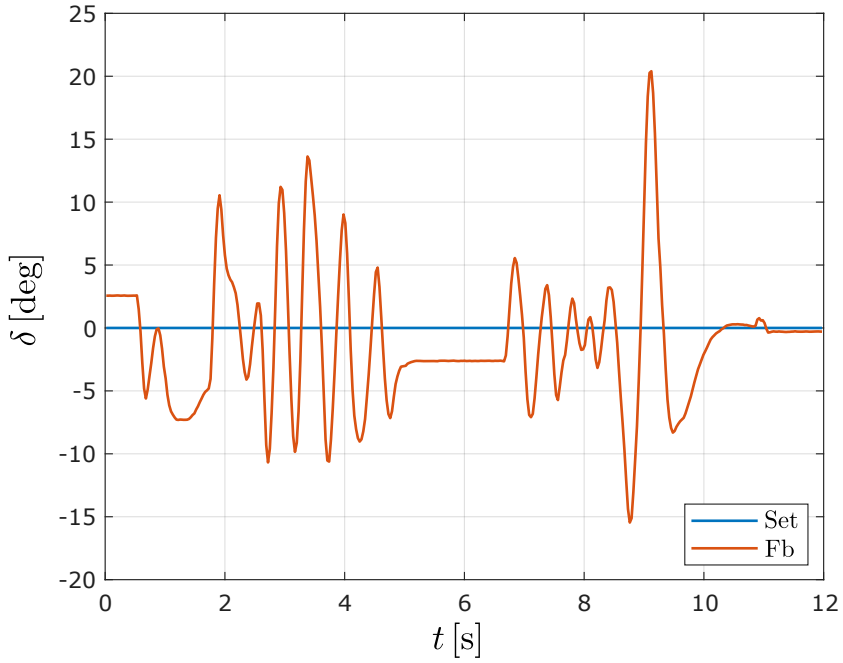


Figura 6.23: Superamento ostacolo: angolo di imbardata

Più interessante è invece l'angolo di imbardata (figura 6.23). A seguito del superamento dell'ostacolo, la ghiaia provoca delle notevoli oscillazioni che il controllo fuzzy cerca di compensare facendo tendere a zero la media in quel tratto. Avremo quindi un robot con il posteriore che si muove un po' a destra e a sinistra, ma che nel complesso riesce comunque ad andare dritto.

6.3 Prove in salita

L'ultimo set di prove effettuato sono le prove in salita. Il robot viene posizionato su una tavola di legno piana la cui inclinazione viene variata. Le prove sono state eseguite imponendo un set di velocità pari a 1200 rpm, con angoli di inclinazione pari a 12°, 18° e 28°.

In questa prima prova mettiamo a confronto la capacità di salita del robot con il controllo fuzzy attivo e con il solo anteriore. L'angolo di inclinazione del piano è pari a 12°. In figura 6.24 riportiamo il grafico della velocità di rotazione. Nel caso di solo anteriore vediamo come i tripodi inizino a ruotare senza riuscire a scaricare coppia al terreno. L'inclinazione è tale da ridurre il carico verticale sui tripodi anteriori, i quali trovano un vincolo alla rotazione ridotto.

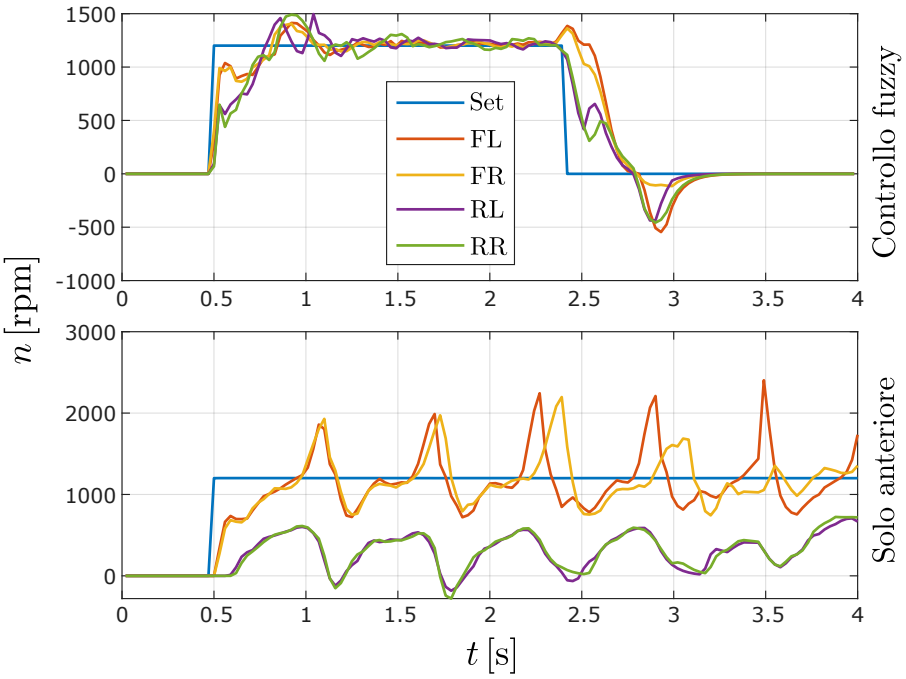


Figura 6.24: Salita $\alpha = 12°$: velocità angolare dei motori

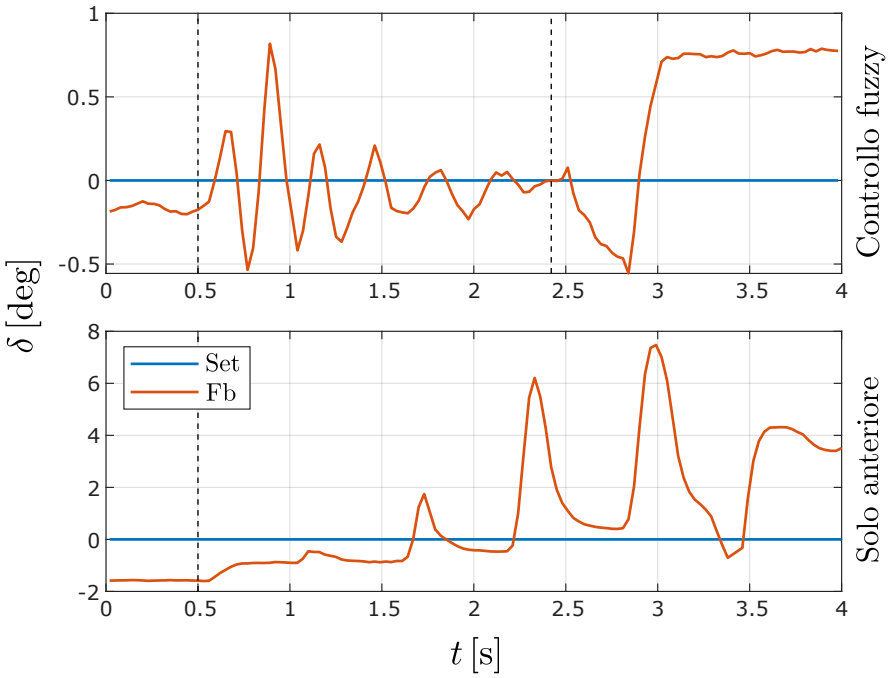


Figura 6.25: Salita $\alpha = 12°$: angolo di imbardata

Eseguendo la prova con il posteriore controllato, il robot riesce a salire facilmente. Notiamo come la velocità è stabile durante la percorrenza. Il picco che si vede al rilascio del comando dipende dal fatto che in quel momento l'anteriore è sceso dalla tavola di legno su cui era posto.

Osservando il grafico dell'angolo di imbardata (figura 6.25) si vede come il controllo fuzzy lavori al fine di mantenere l'allineamento. L'errore rimane sempre contenuto all'interno del grado. Non ha invece senso soffermarsi troppo sull'altro, visto che il robot non è riuscito a concludere la prova.

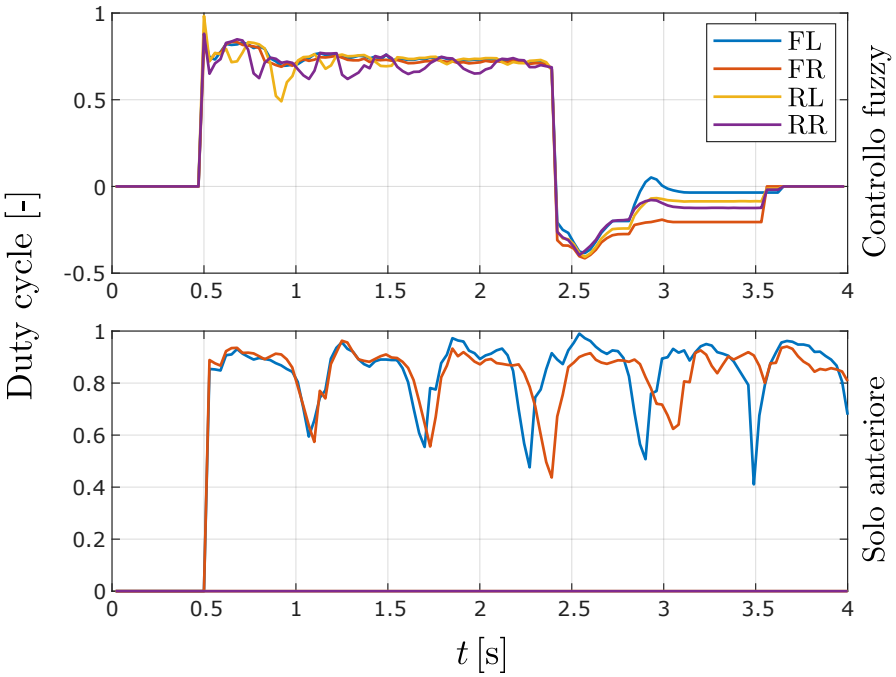


Figura 6.26: Salita $\alpha = 12°$: duty cycle

Dai duty cycle in figura 6.26 vediamo come il controllo fuzzy entri un po' in gioco, ma senza creare eccessive compensazioni. I duty cycle dell'anteriore sono quasi sovrapponibili, ma la conferma sulla ripartizione della spinta l'avremo dal grafico delle correnti.

In figura 6.27 vediamo come le correnti del modulo frontale siano molto simili, segno che la spinta è equamente ripartita. Per quanto riguarda il posteriore, si nota come la posteriore destra dia coppia un po' a singhiozzo, con una forma simile al duty cycle imposto, con lo scopo di mantenere l'allineamento dei telai.

In salita è probabile che il controllo fuzzy applichi correzioni troppo aggressive rispetto ai tempi di risposta del robot, soprattutto per inclinazioni elevate.

Le prove successive sono state svolte solo con il controllo fuzzy attivo per inclinazioni pari a 18° e 28°.

In figura 6.28 notiamo come per $\alpha = 18$ siamo in una condizione limite, in cui i tripodi anteriori accennano a una rotazione senza però completarla, mentre il posteriore ha una spinta non simmetrica a causa delle correzioni applicate dal controllo fuzzy. Aumentando

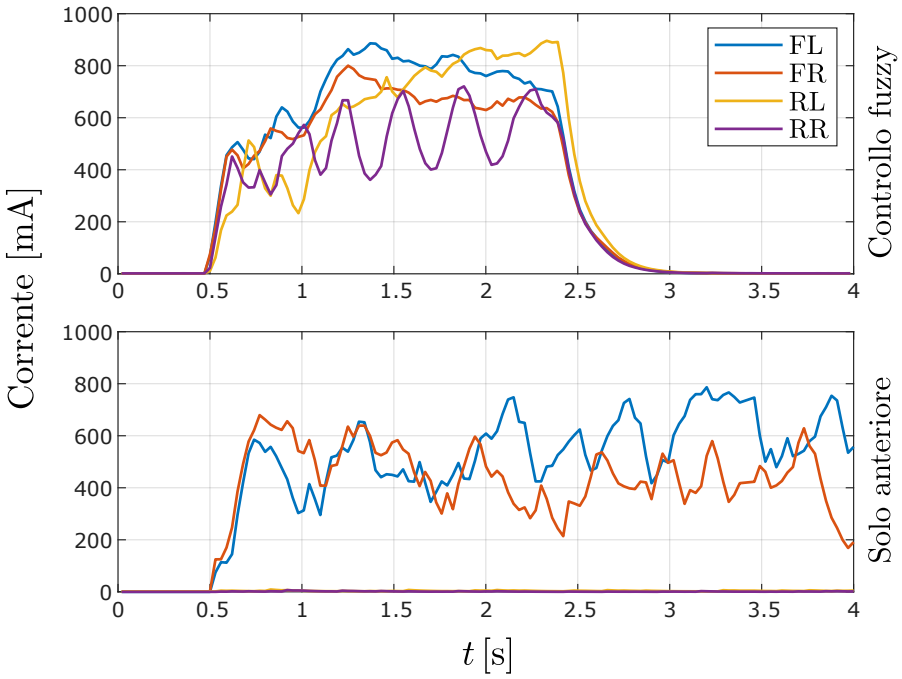


Figura 6.27: Salita $\alpha = 12°$: correnti

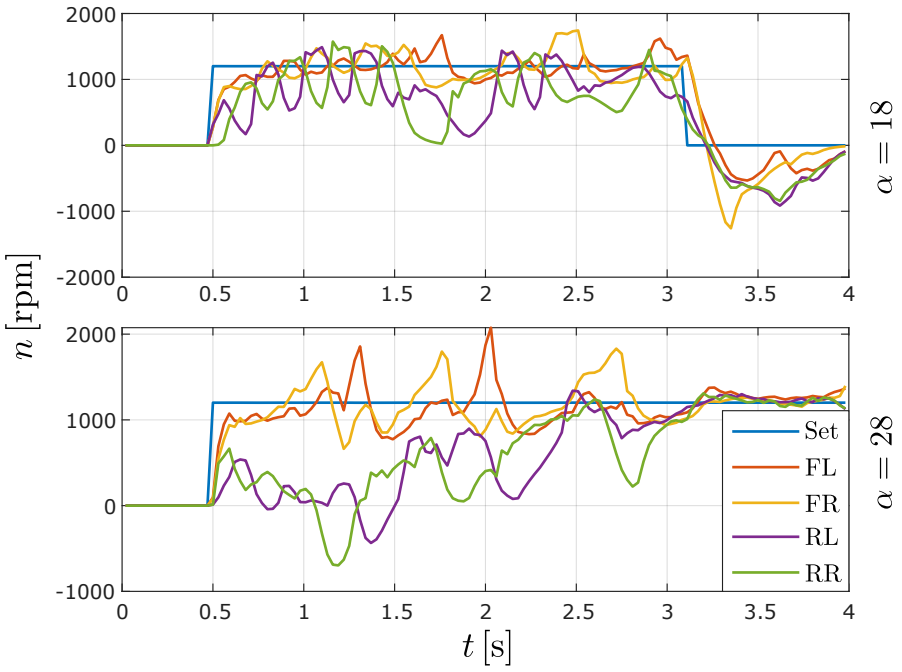


Figura 6.28: Salita con fuzzy abilitato: velocità angolare dei motori

l'inclinazione a 28°, i tripodi anteriori iniziano a ruotare senza che vi sia la possibilità scaricare coppia al terreno. Al tempo $t = 3$ si è applicato un carico verticale[1] per aumentare il vincolo alla rotazione, e come si può notare, sono diminuite le oscillazioni e il robot è riuscito a risalire la rampa.

Il problema quindi non è la mancanza di coppia, ma la struttura del robot stesso che non gli permette di andare oltre certe pendenze. Per risolvere il problema è necessario che la maggior parte della coppia venga fornita dal posteriore, sul quale grava la quasi totalità del peso del robot. L'anteriore deve essere perciò utilizzato solo come supporto, il che rende un po' più difficoltoso il processo di sterzata.

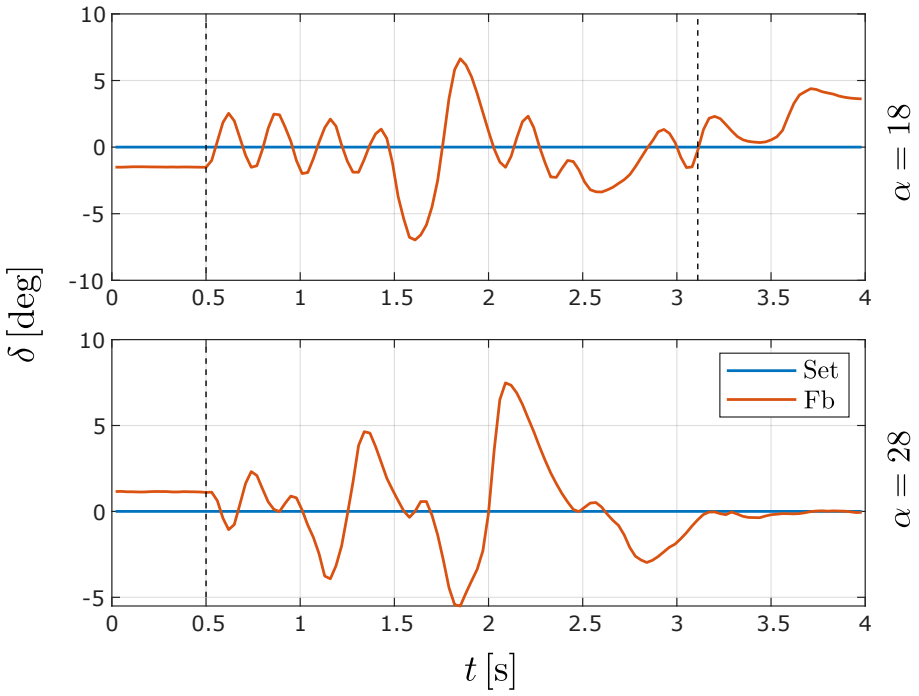


Figura 6.29: Salita con fuzzy abilitato: angolo di imbardata

In figura 6.29 si notano le oscillazioni angolari tra i due moduli, dovuti anche al movimento di rollio causato dalla parziale rotazione dei tripodi. Tali oscillazioni si arrestano nel momento in cui viene applicato il carico verticale nella prova con $\alpha = 28$.

L'andamento dei duty cycle (figura 6.30) è simile ad altri casi visti in precedenza, ovvero quando siamo in presenza di elevate correzioni imposte dal controllo fuzzy. Notiamo come, nel momento di applicazione del carico verticale, rimane ancora margine per la saturazione del comando, e ciò significa che la coppia fornita dai motori è tale da consentire la salita anche con angoli maggiori, se propriamente scaricata al terreno.

In ultimo abbiamo il grafico delle correnti (figura 6.31) che non aggiunge molto a quanto detto sopra.

[1] Ho appoggiato una mano sul modulo.

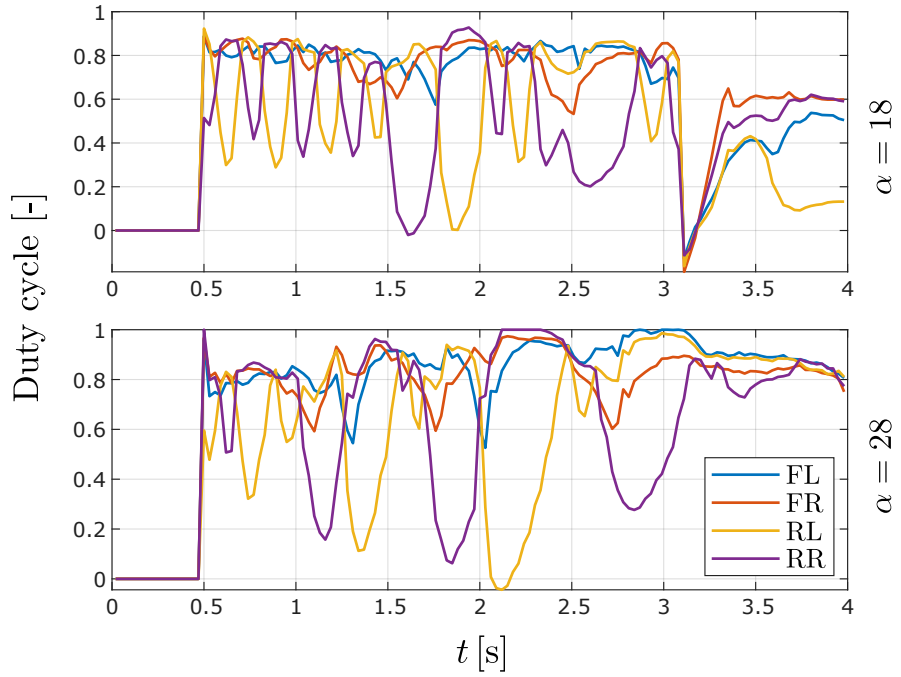


Figura 6.30: Salita con fuzzy abilitato: duty cycle

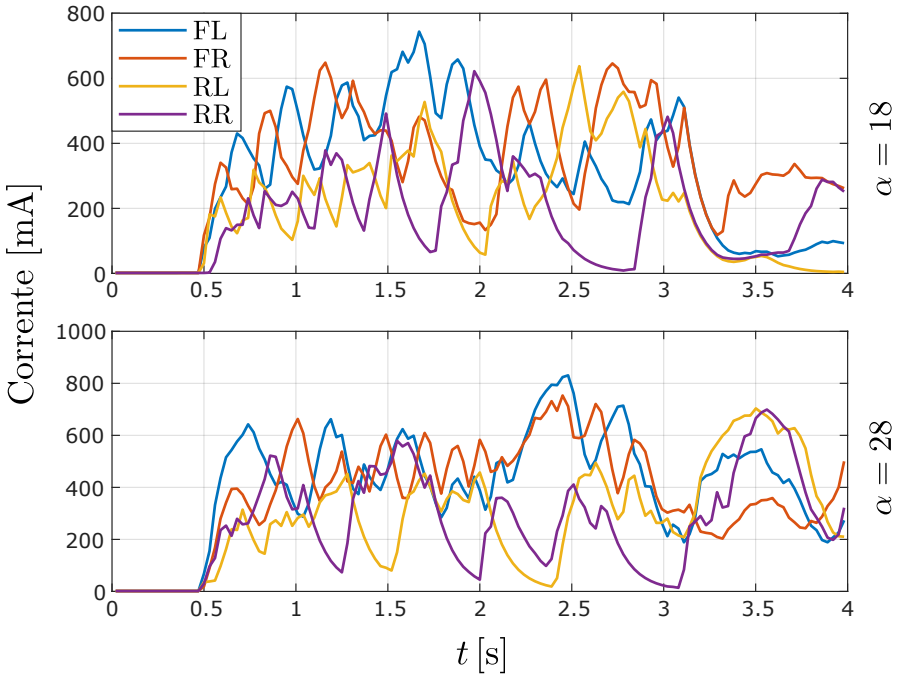


Figura 6.31: Salita con fuzzy abilitato: correnti

Per concludere possiamo dire che il controllo fuzzy ha migliorato decisamente la dinamica del robot in piano rendendolo più reattivo e preciso. Per quanto riguarda salite ripide, è invece necessario studiare un modo per utilizzare maggiormente il posteriore al fine di evitare la rotazione indesiderata dei tripodi.

Capitolo 7

Conclusioni

In questo lavoro di tesi si è analizzato il robot *Epi.q-Mod2* facente parte della famiglia delle unità mobili UGV *Epi.q* all'interno della categoria ibrida Legs-Wheels (**LW**). Dopo un breve studio sulla sua caratteristica più peculiare, l'unità di locomozione a tripode, ci si è concentrati sul miglioramento dell'hardware, in primis con la sensorizzazione del giunto di imbardata e in seguito con l'aggiornamento e la modifica della componentistica elettronica.

Una volta gettate le basi, si è passati allo sviluppo del sistema di controllo. Il robot utilizza un controllo in velocità per il modulo anteriore e un controllo in duty cycle per il posteriore, quest'ultimo utilizza a sua volta un controllo fuzzy. Il controllo in velocità è sufficiente per la movimentazione del robot ma fornisce prestazioni inferiori rispetto al controllo completo.

Al fine di testare il comportamento del robot sono state condotte una serie di prove sia in piano che in salita. Il primo gruppo aveva lo scopo di mettere in luce gli effetti del controllo sulla dinamica del robot, mentre il secondo i limiti meccanici.

Le prove in piano sono state condotte in tre diverse modalità: anteriore controllato in velocità e posteriore controllato in duty cycle con il fuzzy abilitato; anteriore controllato in velocità e posteriore controllato in duty cycle con il fuzzy disabilitato; e infine il solo anteriore controllato in velocità. Dalle prove è emerso che il controllo fuzzy rende il robot più reattivo e preciso alle variazioni dei comandi e più facilmente governabile tramite l'utilizzo di un controller manuale.

Le prove condotte con il posteriore abilitato, ma non controllato attivamente, sono quelle che hanno prodotto i risultati peggiori. Il robot è difficilmente controllabile e si riesce con fatica a far si che vada dritto. Se si percorre una curva abbastanza chiusa e si rilascia lo sterzo è molto probabile che non si raggiunga mai l'allineamento dei telai, a meno di non rilasciare anche il comando di velocità longitudinale.

Il robot, in alternativa, può essere movimentato dal solo anteriore con prestazioni, in termini di manovrabilità, intermedie tra i due casi sopra presentati. La grande differenza la si trova invece nella capacità di superamento ostacoli, caratteristica peculiare del robot, viste le unità tripodi presenti. Il solo anteriore non permette al robot di superare ostacoli particolarmente elevati e soprattutto dalle conformazioni un po' più ostiche. Per ottenere le massime prestazioni in quest'ambito è necessario l'utilizzo del posteriore, meglio se controllato attivamente tramite il fuzzy.

Le prove in salita hanno invece mostrato le limitazioni e gli svantaggi di avere un'unità di locomozione a tripode, rispetto a singole ruote. Nel momento in cui l'inclinazione del piano superava un certo angolo, il trasferimento di carico al posteriore era tale che il carico verticale residuo sull'anteriore non era più sufficiente a bloccare la rotazione del portatreno, impedendo così al flusso di potenza di scaricarsi a terra tramite le ruote e di generare spinta di avanzamento. Il robot rimane perciò in una condizione di stallo in cui i tripodi ruotano sul posto, ma il robot non avanza. L'utilizzo dei due moduli combinati migliora la situazione ma non risolve completamente il problema.

In conclusione possiamo dire che lo sviluppo della logica di controllo ha portato a dei miglioramenti del robot in termini prestazionali e ha migliorato le sensazioni di guida dell'utente.

Gli sviluppi futuri prevedono un miglioramento dell'algoritmo di controllo al fine di avere una maggiore coppia al posteriore nelle prove in salita con lo scopo di minimizzare le possibili rotazioni dei tripodi anteriori quando non richieste. In aggiunta si potrebbe migliorare il controllo fuzzy aggiungendo una componente integrativa con lo scopo di rilassare la componente proporzionale (l'unica presente), al momento abbastanza aggressiva, e di migliorare il comportamento in curve di ampio e medio raggio.

Lo scopo ultimo del robot è la guida autonoma, quindi i successivi sviluppi prevedono anche l'implementazione a bordo di una logica ad alto livello in grado di pianificare il percorso del robot generando i comandi che verranno inviati al basso livello qui presentato.

Appendice A

Codice sorgente

In questa appendice sono riportati i codici sorgente caricati sui microcontrollori di Epi.q e del joystick. Non seguirà una spiegazione dettagliata dell'implementazione, per la quale rimando alle seguente guida [16], ma ci limiteremo a esplicitare la funzione di ogni classe e quali informazioni vengono veicolate.

A.1 Codice sorgente del robot Epi.q

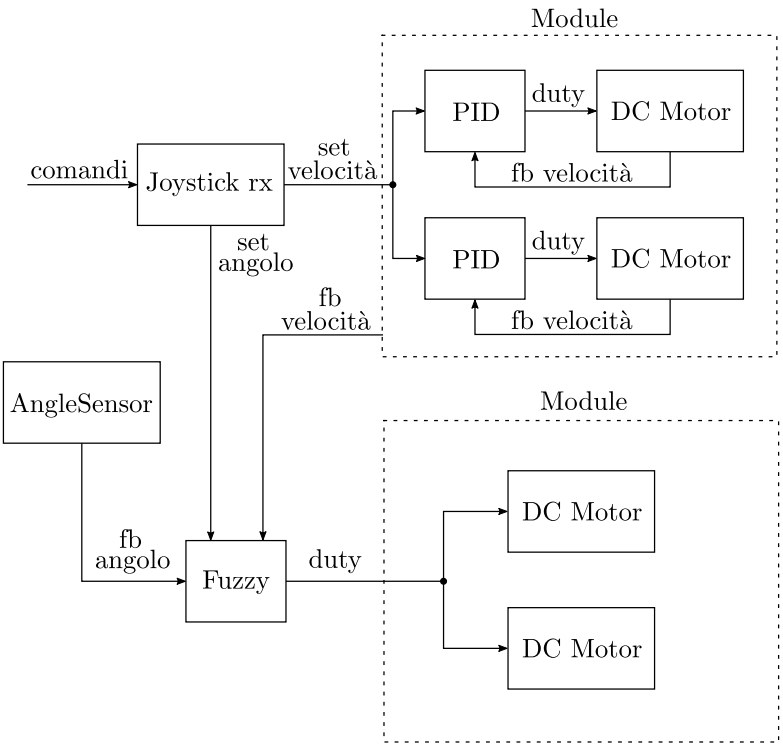


Figura A.1: Schema generale delle classi

In figura A.1 è riportato lo schema generale delle classi utilizzate. La classe "Joystick rx" riceve ed elabora i comandi generando i set di velocità per il modulo anteriore e il set angolare. La classe "Module" contiene al suo interno la classe "DC Motor" che contiene la classe "PID"[1], quest'ultima riceve il set di velocità e genera un duty cycle che è il comando vero e proprio del motore.

La classe "Angle Sensor" si occupa di acquisire i dati del potenziometro. La classe "Fuzzy" contiene tutte le regole e le classi di appartenenza. Tramite gli input forniti genera i duty cycle per il modulo posteriore. Le due classi "Module" ovviamente sono uguali, solo che nella seconda vengono bypassati i PID.

Per ogni classe abbiamo sia il file header che il file sorgente. Nel primo troviamo le dichiarazioni, mentre nel secondo troviamo l'effettiva implementazione.

Tra i vari file qui presentati ce ne sono alcuni che non contengono il codice sorgente delle classi:

- "Functions", qui ci sono le funzioni utilizzate che non sono metodi;
- "Settings", qui sono state dichiarate tutte le costanti, i parametri e le strutture dati utilizzate nel codice;
- "Logging", qui sono state definite le funzioni di log dei dati;
- "Main" è il file principale che contiene il codice vero e proprio.

Codice A.1: File header "PID.h"

```
#pragma once
#include <Arduino.h>
#include <Settings.h>
#include <Functions.h>

class PID
{
private:
  float kp; // proportional gain
  float ki; // integral gain
  float kd; // derivative gain
  OL_gain_struct OL_fw_gain = {0.0f, 0.0f};
  OL_gain_struct OL_bw_gain = {0.0f, 0.0f};
  float error = 0.0f; // RPM   error between setrence and measured value
  float ffw = 0.0f;   // feedforward (in realta' e' piu' un controllo in
    anello aperto)
  float ctrl = 0.0f;  // control output [rpm] - [N*mm]
  float INT = 0.0f;   //      Integral part
  float DER = 0.0f;   //      derivative part
  float last_measure = 0.0f;
  float set = 0.0f; // e' una ridondanza rispetto alla classe del motore,
     ma e' piu' comodo per creare la struttura
  float fb = 0.0f;
```

[1] Stile matrioska.

```
22   uint32_t t = 0; // salva il tempo per la componente derivativa e
       integrativa
23   bool kd_en;     // siccome il derivativo molto probabilmente non verra'
       usato inserisco un flag per evitare operazioni inutili
24
25 public:
26   PID(float _kp, float _ki, float _kd);
27
28   void setGain(pid_param _pidGain);
29
30   void setOlGain(OL_gain_struct _fw, OL_gain_struct _bw);
31
32   float Calc_ffw(float _set); // calcola il feedforward e lo salva nella
       variabile specifica
33
34   float PID_ctrl(float _set, float _fb, float _lim);
35
36   pid_data get_data(); // ritorna una struttura con tutti i valori del
       pid
37
38   pid_param get_gain(); // ritorna i guadagni del pid
39
40   pid_settings get_settings(); // ritorna i guadagni del pid e del
       feedforward
41
42   void Reset(); // fa un reset della parte integrativa, derivativa e
       della variabile che misura il tempo
43                 // nel dubbio un reset non fa mai male
44
45   void Res_int(); // manda a zero la componente integrativa
46 };
```

Codice A.2: File sorgente “PID.cpp”

```
1  #include "PID.h"
2
3  PID::PID(float _kp, float _ki, float _kd) // costruttore
4      : kp(_kp),
5        ki(_ki),
6        kd(_kd)
7  {
8    kd_en = abs(_kd) < epsilon ? false : true; // variabile booleana per
       evitare di eseguire il calcolo della componente derivativa se il
       guadagno viene posto a zero
9  }
10
11 void PID::setGain(pid_param _pidGain)
12 {
13   kp = _pidGain.kp;
14   ki = _pidGain.ki;
15   kd = _pidGain.kd;
16   kd_en = abs(kd) < epsilon ? false : true; // se kd = 0, non viene
       eseguita la parte inerente
17   error = 0;
18   INT = 0;
```

```
19    DER = 0;
20    ctrl = 0;
21  }
22
23  void PID::setOlGain(OL_gain_struct _fw, OL_gain_struct _bw)
24  {
25    OL_fw_gain = _fw;
26    OL_bw_gain = _bw;
27  }
28
29  float PID::Calc_ffw(float _set)
30  {
31    if (abs(_set) < ffw_th)
32    {
33      return 0.0f;
34    }
35    else
36    {
37      OL_gain_struct OL_gain = set < 0.0f ? OL_bw_gain : OL_fw_gain;
38      float _ffw = OL_gain.n_1 * set + OL_gain.n_0;
39      return _ffw;
40    }
41  }
42
43  float PID::PID_ctrl(float _set, float _fb, float _lim)
44  {
45    uint32_t _t = micros();
46    float dt = _t - t;
47    t = _t;
48    set = _set;
49    fb = _fb;
50    error = set - fb;
51
52    ffw = Calc_ffw(set);
53
54    if (kd_en)
55    {
56      DER = (-kd * (fb - last_measure) * 1e6f) / dt; // Note: the
       derivative term use just the measure and not the value to avoid
       derivative kick.
57      last_measure = fb;
58    }
59    ctrl = constrain(kp * error + ffw + DER, -_lim, _lim); // calcolo un
       parte del valore di uscita e lo saturo
60
61    float _int = (ki * error * dt) * 1e-6f;
62    INT = constrain(INT + _int, -_lim - ctrl, _lim - ctrl); // saturo l'
       integrativa con dei limiti dinamici
63    ctrl += INT;
64    return ctrl;
65  }
66
67  pid_data PID::get_data()
68  {
69    pid_data _data{set, fb, error, ffw, INT, DER, ctrl};
```

```
  return _data;
}

pid_param PID::get_gain()
{
  pid_param _gain{kp, ki, kd};
  return _gain;
}

pid_settings PID::get_settings()
{
  pid_settings _settings{{kp, ki, kd}, OL_fw_gain, OL_bw_gain};
  return _settings;
}

void PID::Reset()
{
  t = micros();
  DER = 0.0f;
  last_measure = 0.0f;
  ctrl = 0.0f;
  error = 0.0f;
  INT = 0.0f;
  ffw = 0.0f;
}

void PID::Res_int()
{
  INT = 0.0f;
}
```

Codice A.3: File header “DC_Motor.h”

```
#pragma once
#include <Arduino.h>
#include <Settings.h>
#include <Functions.h>
#include <Encoder.h>
#include <PID.h>

/*
nomenclatura variabili:
set -> riferimento esterno, viene inviato al motore, ma in realta' e' l'
    ingresso del pid
fb -> feedback del motore, puo' essere la velocita' o la corrente,
    dipende dal tipo di controllo impostato
ctrl -> variabile in uscita dal pid. e' il vero input del motore, che
    verra' poi convertito in pwm

queste 3 variabili si trovano sia nella classe DC_Motor che nella classe
    PID.
*/

// implementazione del DC_Motor
// al suo interno contiene la classe pid e la classe encoder
```

```
19 class DC_Motor
20 {
21 private:
22   byte PIN_IN1; // i pin hanno gli stessi nomi del driver
23   byte PIN_IN2;
24   byte PIN_PWM;
25   byte PIN_FB;
26   byte PIN_EN;
27   byte V_IN1 = 0; // valore che consente al motore la rotazione per l'
       avanzamento (i valori finali sono assegnati dal costruttore)
28   byte V_IN2 = 0; // valore che consente al motore la rotazione per l'
       avanzamento
29   byte FW_dir;    // Verso di rotazione che permette avanzamento
30
31   Encoder enc;           // encoder
32   uint32_t t = 0;        //[us] tempo per calcolare la velocita'
33   float set = 0.0f;      // riferimento in ingresso al pid
34   float ctrl = 0.0f;     // uscita dal pid
35   float speed = 0.0f;    //[rpm] velocita' di rotazione del motore (albero
       encoder)
36   float current = 0.0f; //[mA] corrente circolante nel motore
37   uint16_t pwm;          // pwm attualmente impostato
38   byte dir;              // direzione di rotazione (vedi definizioni
       FORWARD ecc.)
39   bool en = false;       // se true -> motore abilitato (enable)
40
41   uint32_t t_cleanup = 0; //[ms] WatchDog timer -> manda a zero il pwm
       dopo un tot di tempo in cui il set e' 0
42   uint32_t t_acq = 0;     //[us] tempo per acquisizioni (velocita',
       corrente)
43   uint32_t t_cmd = 0;     //[us] tempo per comandi (chiama la funzione
       che richiama il pid)
44
45   uint8_t type_ctrl = NONE; // definisce se controllo in velocita' o duty
       cycle
46
47 public:
48   PID pid;
49
50   DC_Motor(byte _IN1, byte _IN2, byte _PWM, byte _FB, byte _EN, byte
       _ENCA, byte _ENCB, byte _FW_dir);
51
52   void init(pid_settings _pidGain); // inizializza i vari pin e salva i
       parametri del pid
53
54   void set_type_ctrl(uint8_t _type_ctrl); // imposta il controllo in
       coppia o in velocita' (TC - SC)
55
56   void enable(bool _en); // si possono passare indifferentemente i valori
       (true, false), (HIGH, LOW), (1, 0)
57
58   byte readDirection(float _val); // in funzione del segno del valore
       passato ritorna "forward" o "backward"
59
60   byte getDirection(); // ritorna la direzione di rotazione
```

```
61
62    void setDirection(byte _dir); // Imposta il verso di rotazione del
        motore
63
64    float getSpeed(); // velocita' angolare encoder [RPM]
65
66    uint16_t getPWM(); // ritorna il valore di pwm attualmente impostato
67
68    void setPWM(uint16_t _pwm); // set manuale del pwm
69
70    float getDuty(); // ritorna il valore di duty cycle
71
72    void setDuty(float _duty); // imposta il duty cycle (simile a setPWM)
73
74    float getCurrent(); // current feedback [mA]
75
76    float getTorque(); //[N*mm] ritorna il valore di coppia erogata dal
        motore
77
78    void setRef(float _set); // imposta il riferimento del motore (se non
        viene chiamato _ctrl_loop non ha alcun effetto)
79
80    float getRef(); // ritorna il valore di set impostato
81
82    void Reset(); // reset dei feedback e pid, chiamarlo se non viene
        eseguito il mainloop per un po' di tempo
83
84    void Reset_light(); // non azzera i timer, resetta
85
86    void _readSpeed(); // viene chiamata dal mainloop
87
88    void _readCurrent(); // viene chiamata dal mainloop
89
90    void _ctrl_loop(); // viene chiamata dal mainloop
91
92    void mainloop(); // deve essere chiamato il piu' spesso possibile
93
94    void mainloop2(); // fa solo la parte delle misurazioni
95 };
```

Codice A.4: File sorgente “DC_Motor.cpp”

```
1  #include "DC_Motor.h"
2
3  DC_Motor::DC_Motor(byte _IN1, byte _IN2, byte _PWM, byte _FB, byte _EN,
       byte _ENCA, byte _ENCB, byte _FW_dir)
4      : PIN_IN1(_IN1),
5        PIN_IN2(_IN2),
6        PIN_PWM(_PWM),
7        PIN_FB(_FB),
8        PIN_EN(_EN),
9        FW_dir(_FW_dir),
10       enc(_ENCA, _ENCB),
11       dir(FREE),
12       pid(1.0f, 0.0f, 0.0f)
```

```
13 {
14   if (FW_dir == CCW)
15   {
16     V_IN1 = 0;
17     V_IN2 = 1;
18   }
19   else if (FW_dir == CW)
20   {
21     V_IN1 = 1;
22     V_IN2 = 0;
23   }
24 }
25
26 void DC_Motor::init(pid_settings _Gain)
27 {
28   pinMode(PIN_IN1, OUTPUT);
29   pinMode(PIN_IN2, OUTPUT);
30   pinMode(PIN_PWM, OUTPUT);
31   analogWriteFrequency(PIN_PWM, PWM_FREQUENCY); // imposto frequenza del
       pwm
32   pinMode(PIN_EN, OUTPUT);
33   pid.setGain(_Gain.SpeedCtrl);             // imposto i guadagni del pid
34   pid.setOlGain(_Gain.OL_fw, _Gain.OL_bw); // imposto i guadagni in
       anello aperto di velocita'
35   Reset();                                  // effettuo un reset per "
       azzerare i timer"
36 }
37
38 void DC_Motor::set_type_ctrl(uint8_t _type_ctrl)
39 {
40   if (type_ctrl != _type_ctrl)
41   {
42     Reset_light();
43     type_ctrl = _type_ctrl;
44   }
45 }
46
47 void DC_Motor::enable(bool _en)
48 {
49   if (en != _en)
50   {
51     Reset_light();
52     en = _en;
53     digitalWrite(PIN_EN, en);
54   }
55 }
56
57 byte DC_Motor::readDirection(float _val)
58 {
59   return (_val < 0.0f ? BACKWARD : FORWARD);
60 }
61
62 byte DC_Motor::getDirection()
63 {
64   return dir;
```

```
65 }
66
67 void DC_Motor::setDirection(byte _dir) // modificare funzione o inserirne
       un'altra che richiama questa
68 {
69   if (dir != _dir)
70   {
71     dir = _dir;
72     switch (dir)
73     {
74     case BRAKE:
75       analogWrite(PIN_PWM, 0);
76       digitalWrite(PIN_IN1, HIGH);
77       digitalWrite(PIN_IN2, HIGH);
78       break;
79
80     case FREE:
81       analogWrite(PIN_PWM, 0);
82       digitalWrite(PIN_IN1, LOW);
83       digitalWrite(PIN_IN2, LOW);
84       break;
85
86     case FORWARD:
87       digitalWrite(PIN_IN1, V_IN1);
88       digitalWrite(PIN_IN2, V_IN2);
89       break;
90
91     case BACKWARD:
92       digitalWrite(PIN_IN1, !V_IN1);
93       digitalWrite(PIN_IN2, !V_IN2);
94       break;
95     }
96   }
97 }
98
99 float DC_Motor::getSpeed()
100 {
101   return speed;
102 }
103
104 uint16_t DC_Motor::getPWM()
105 {
106   return pwm;
107 }
108
109 void DC_Motor::setPWM(uint16_t _pwm)
110 {
111   _pwm = constrain(_pwm, 0u, MAX_PWM);
112   pwm = _pwm;
113   analogWrite(PIN_PWM, pwm);
114 }
115
116 float DC_Motor::getDuty()
117 {
118   float _c = dir == BACKWARD ? -1.0f : 1.0f;
```

```
119   return _c * (static_cast<float>(pwm) / static_cast<float>(MAX_PWM));
120 }
121
122 void DC_Motor::setDuty(float _duty)
123 {
124   _duty = constrain(_duty, -1.0f, 1.0f);
125   setDirection(readDirection(_duty));
126   setPWM(static_cast<uint16_t>(roundf(abs(_duty) * static_cast<float>(
        MAX_PWM))));
127 }
128
129 float DC_Motor::getCurrent()
130 {
131   return current;
132 }
133
134 float DC_Motor::getTorque() // ottiene il valore di coppia attualmente
        erogata dal motore con una banda morta
135 {
136   float _current = current >= C_deadband ? current : 0.0f; // banda morta
137   return dir == FORWARD ? k_torque * _current : -k_torque * _current;
138 }
139
140 void DC_Motor::setRef(float _set) // la saturazione non viene fatta qui,
        quindi il set del DC_Motor e il set del PID potrebbero essere diversi
141 {
142   set = _set;
143 }
144
145 float DC_Motor::getRef()
146 {
147   return set;
148 }
149
150 void DC_Motor::Reset()
151 {
152   speed = 0.0f;
153   current = 0.0f;
154   set = 0.0f;
155   ctrl = 0.0f;
156   setPWM(0);
157   t = micros();
158   t_acq = t;
159   t_cmd = t;
160   enc.write(0);
161   pid.Reset();
162 }
163
164 void DC_Motor::Reset_light()
165 {
166   pid.Reset();
167   set = 0.0f;
168   ctrl = 0.0f;
169   setPWM(0);
170 }
```

```
171
172 void DC_Motor::_readSpeed() // funzione richiamata dal mainloop
173 {
174   int C = enc.readAndReset();
175   uint32_t tn = micros();
176   uint32_t dt = tn - t;
177   t = tn;
178   float _speed = (static_cast<float>(C) * 1e6f * 60.0f) / (static_cast<
      float>(CPR * dt)); // shaft speed [RPM]
179   _speed = FW_dir == CW ? _speed : -_speed;
180   speed = EWMA_filter(alpha_s, speed, _speed);
181 }
182
183 void DC_Motor::_readCurrent() // funzione richiamata dal mainloop
184 {
185   float _current = (static_cast<float>(analogRead(PIN_FB) * 3300 * 1000))
        / (4095.0f * CURRENT_FB); // current feedback [mA]
186   current = EWMA_filter(alpha_c, current, abs(_current));
                                  // applico il filtro passa basso ("media
      mobile")
187 }
188
189 void DC_Motor::_ctrl_loop() // metodo che utilizza la funzione del pid,
      richiamato dal mainloop
190 {
191   if (en)
192   {
193     if (type_ctrl == SC) // se non viene impostato alcun controllo la
      funzione non viene eseguita
194     {
195       float _set = constrain(set, -n_max, n_max); // saturo il
      riferimento, ma non lo salvo
196       bool flag = false;
197       if (abs(_set) > epsilon || abs(speed) > 50) // se uno dei due non e
      ' zero resetto il timer
198       {
199         t_cleanup = millis();
200         flag = true;
201       }
202       else if (millis() - t_cleanup > dt_cleanup) // se entrambi sono
      zero ed e' passato abbastanza tempo resetto integrativa
203       {
204         t_cleanup = millis();
205         pid.Res_int();
206         setPWM(0);
207       }
208       if (flag)
209       {
210         ctrl = pid.PID_ctrl(_set, speed, n_max); // valore in uscita dal
      pid
211         setDuty(ctrl / n_max);                   // applicazione del
      comando
212       }
213     }
214     else if (type_ctrl == DC)
```

```
215     {
216       setDuty(set);
217     }
218   }
219 }
220
221 void DC_Motor::mainloop()
222 {
223   if (micros() - t_acq >= dt_meas) // loop per la lettura dei feedback
224   {
225     t_acq = micros();
226     _readSpeed();
227     _readCurrent();
228   }
229
230   if (micros() - t_cmd >= dt_cmd)
231   {
232     t_cmd = micros();
233     _ctrl_loop();
234   }
235 }
236
237 void DC_Motor::mainloop2()
238 {
239   if (micros() - t_acq >= dt_meas) // loop per la lettura dei feedback
240   {
241     t_acq = micros();
242     _readSpeed();
243     _readCurrent();
244   }
245 }
```

Codice A.5: File header “Module.h”

```
1  #pragma once
2  #include <Arduino.h>
3  #include <Streaming.h>
4  #include <DC_Motor.h>
5
6  class Module
7  {
8  private:
9    uint8_t type_ctrl; // definisce il tipo di controllo
10
11 public:
12   DC_Motor *Left;
13
14   DC_Motor *Right;
15
16   Module(DC_Motor *_Left, DC_Motor *_Right);
17
18   void set_type_ctrl(uint8_t _type_ctrl);
19
20   void setRef(Ref _set);
21
```

```
22   Ref getSpeed(); //[rpm] Ritorna la velocita' dei due motori
23
24   Ref getDuty(); //[-] ritorna il duty cycle dei due motori
25
26   void enable(bool _en);
27
28   void Reset();
29
30   void mainloop();
31 };
```

Codice A.6: File sorgente “Module.cpp”

```
1  #include "Module.h"
2
3  Module::Module(DC_Motor *_Left, DC_Motor *_Right)
4      : Left(_Left),
5        Right(_Right) {}
6
7  void Module::set_type_ctrl(uint8_t _type_ctrl)
8  {
9    if (type_ctrl != _type_ctrl)
10   {
11     type_ctrl = _type_ctrl;
12     Left->set_type_ctrl(_type_ctrl);
13     Right->set_type_ctrl(_type_ctrl);
14   }
15 }
16
17 void Module::setRef(Ref _set)
18 {
19   Left->setRef(_set.L);
20   Right->setRef(_set.R);
21 }
22
23 Ref Module::getSpeed()
24 {
25   Ref ref{Left->getSpeed(), Right->getSpeed()};
26   return ref;
27 }
28
29 Ref Module::getDuty()
30 {
31   Ref ref{Left->getDuty(), Right->getDuty()};
32   return ref;
33 }
34
35 void Module::enable(bool _en)
36 {
37   Left->enable(_en);  // il pin e' condiviso anche con l'altro motore
38   Right->enable(_en); // ripeto il comando per settare variabile interna
       anche sull'altro motore
39 }
40
41 void Module::Reset()
```

```
{
  Left->Reset();
  Right->Reset();
}

void Module::mainloop()
{
  Left->mainloop();
  Right->mainloop();
}
```

Codice A.7: File header “AngleSensor.h”

```
#pragma once
#include <Arduino.h>
#include <Settings.h>
#include <Functions.h>

// Potenziometro per la misura dell'angolo delta
class AngleSensor
{
private:
  byte pin;                // pin analogico a cui e' collegato in
    potenziometro
  float angle = 0.0f;      //[deg] angolo misurato (e' applicato un
    filtro di media mobile)
  float angle_last = 0.0f; //[deg] angolo misurato al loop precedente
  float omega = 0.0f;      //[deg/s] velocita' angolare di imbardata
  uint32_t t = 0;          // variabile per tenere il tempo (angolo)
  uint32_t t_w = 0;        // variabile per tenere il tempo (velocita'
    angolare)
  uint32_t counter = 0;    // contatore per il calcolo della velocita'
    angolare

  void _readAngle();
  void _calcOmega(); // calcola la velocita' angolare

public:
  AngleSensor(byte _pin);

  AngleSensor(const AngleSensor &_AS) = delete;

  int read(); // lettura analogica del pin

  void mainloop();

  float getAngle(); //[deg] ritorna angolo di imbardata

  float getOmega(); //[rad/s] ritorna velocita' angolare di imbardata
};
```

Codice A.8: File sorgente “AngleSensor.cpp”

```
#include "AngleSensor.h"

```

```
AngleSensor::AngleSensor(byte _pin)
    : pin(_pin) {}

int AngleSensor::read()
{
  return analogRead(pin);
}

void AngleSensor::_readAngle()
{
#ifndef Inverted
  float _angle = map_float(read(), MinAngleRef, MaxAngleRef, -MaxAngle,
    MaxAngle);
#else
  float _angle = -map_float(read(), MinAngleRef, MaxAngleRef, -MaxAngle,
    MaxAngle);
#endif
  _angle = constrain(_angle, -MaxAngle, MaxAngle); // saturo per evitare
    eventuali oscillazioni di tensione
  angle = EWMA_filter(alpha_a, angle, _angle);
}

void AngleSensor::_calcOmega()
{
  float dt = t - t_w;
  t_w = t;
  omega = (angle - angle_last) * 1e6 / dt;
  angle_last = angle;
}

void AngleSensor::mainloop()
{
  if (micros() - t >= dt_angle)
  {
    t = micros();
    _readAngle();
    counter++;
    if (counter % omegaCoeff == 0)
    {
      _calcOmega();
    }
  }
}

float AngleSensor::getAngle()
{
  return angle;
}

float AngleSensor::getOmega()
{
  return omega;
}
```

Codice A.9: File header “Joystick_rx.h”

```
#pragma once
#include <Arduino.h>
#include <Settings.h>
#include <Functions.h>
#include <EasyTransfer.h>

class Joystick_rx
{
private:
  float ni = 0.0f;              //[RPM]    Forward motion reference
  float dn = 0.0f;              //[RPM]    Pivot motion reference
  float delta = 0.0f;           //[deg]     Angle reference
  float K = 0.0f;               // rapporto numero di giri lato destro e sinistro K = nr / nl
  float nr = 0.0f;              //[rpm] motore destro modulo 1
  float nl = 0.0f;              //[rpm] motore sinistro modulo 1
  uint32_t WD_timer = 0;        //[ms] timer watch dog per rilevare prolungata assenza di segnale
  uint32_t t = 0;               //[us] tiene il tempo per il mainloop
  Stream *theStream = nullptr; // inizializzato con un puntatore nullo
  M_data_extra _data_extra{0, false};

public:
  TXRX_DATA_STRUCT joydata = joydata_init; // inizializzazione

  M_aux_cmd aux_cmd{true, adv_mod, false, true};

  EasyTransfer ETJoy;

  Joystick_rx();

  Joystick_rx(const Joystick_rx &_JS) = delete;

  void begin(Stream *theStream); // inizializza la comunicazione

  Ref getRef(); // ritorna il riferimento in velocita'

  float getAngleRef(); // ritorna il riferimento in angolo

  float calc_K(float _delta, bool reverse); // calcola il fattore K = nr / nl, rapporto delle velocita' dei motori, che e' funzione dell' angolo delta

  void _update(); // qui e' implementata la mappa lineare

  void Reset(); // svuota il buffer, azzera i timer ed azzera i riferimenti

  M_aux_cmd getAuxCmd(); // ritorna gli altri comandi inviati dal joystick

  M_data_extra getDataExtra(); // ritorna i dati aggiuntivi che non influiscono direttamente nel controllo

  void mainloop(); // loop principale
```

```
49
50   void log_cmd_change(); // stampa a video nel caso in cui ci sia stato
       un cambiamento dei comandi ausiliari
51 };
```

Codice A.10: File sorgente “Joystick_rx.cpp”

```
1  #include "Joystick_rx.h"
2
3  Joystick_rx::Joystick_rx() {} // costruttore, nessun argomento passato
4
5  void Joystick_rx::begin(Stream *_theStream)
6  {
7    theStream = _theStream;
8    ETJoy.begin(details(joydata), theStream);
9    WD_timer = millis();
10   t = micros();
11 }
12
13 float Joystick_rx::calc_K(float _delta, bool reverse)
14 {
15 #ifndef Inverted
16   float _a = reverse ? b : a; // imposta i parametri in base al fatto che
       si vada in avanti o in retromarcia
17   float _b = reverse ? a : b;
18 #else
19   float _a = reverse ? a : b; // imposta i parametri in base al fatto che
       si vada in avanti o in retromarcia
20   float _b = reverse ? b : a;
21 #endif
22   _delta *= DEG_TO_RAD; // converto in radianti
23   float d = (I * sin(_delta)) / (2.0f * (_a * cos(_delta) + _b));
24   return (1 + d) / (1 - d);
25 }
26
27 void Joystick_rx::_update()
28 {
29   if (ETJoy.receiveData())
30   {
31     WD_timer = millis(); // comand recived -> reset timer
32
33     if (joydata.lambda == 0 && joydata.nu != 0) // modalita' pivot
34     {
35       nr = pivotCoeff * map_float(joydata.nu, -JS_out_max, JS_out_max, -
       n_max, n_max); // eseguo la mappatura del motore di destra la
       moltiplico per il coefficiente
36       nl = -nr;
37       aux_cmd.reverse = false; // siccome siamo in modalita' pivot
       considero che siamo sempre in marcia avanti (non fa differenza)
38       aux_cmd.mod = pivot_mod;
39     }
40
41     else // modalita' advancing
42     {
```

```
43        aux_cmd.mod = adv_mod;
                // imposto la modalita' di moto, serve per il controllo
    fuzzy
44        aux_cmd.reverse = joydata.lambda < 0 ? true : false;
                // controllo se siamo in avanzamento o in retromarcia
45        ni = map_float(joydata.lambda, -JS_out_max, JS_out_max, -n_max,
    n_max);       // mappatura della velocita' di avanzamento
46        delta = map_float(joydata.nu, -JS_out_max, JS_out_max, -MaxAngle,
    MaxAngle); // mappatura dell'angolo delta inviato
47        K = calc_K(delta, aux_cmd.reverse);
48        float nl0 = (2 * ni) / (K + 1);     // numero di giri teorico del
    lato sinistro
49        float nr0 = nl0 * K;                // numero di giri teorico del
    lato destro  //(2 * K * ni) / (K + 1)
50        nr = constrain(nr0, -n_max, n_max); // saturo entrambi
51        nl = constrain(nl0, -n_max, n_max);
52        if (nr0 != nr) // se uno dei due satura ricalcolo l'altro
53        {
54          nl = nr / K;
55        }
56        else if (nl0 != nl)
57        {
58          nr = nl * K;
59        }
60        // dn = nr - nl;
61      }
62
63      aux_cmd.rear_en = joydata.cmd & Rear_en_bit; // comandi per attivare
    o disattivare modulo
64      aux_cmd.fuzzy_en = joydata.cmd & fuzzy_bit;  // attiva /disattiva il
    fuzzy
65      aux_cmd.duty_mean = joydata.cmd & Duty_bit;  // sceglie il tipo di
    trasferimento di duty cycle al posteriore
66
67      _data_extra = {static_cast<uint8_t>(joydata.cmd & Map_bit),
    static_cast<bool>(joydata.cmd & SpeedRed_bit), static_cast<bool>(
    joydata.cmd & Cruise_control_bit)};
68    }
69
70    if (millis() - WD_timer > Connection_WD) // se non si ricevono comandi
     per un certo tempo, il set torna a 0
71    {
72      nr = 0.0f;
73      nl = 0.0f;
74      WD_timer = millis();
75    }
76  }
77
78  Ref Joystick_rx::getRef()
79  {
80    Ref SpeedRef{nl, nr};
81    return SpeedRef;
82  }
83
84  float Joystick_rx::getAngleRef()
```

```
85 {
86   return delta;
87 }
88
89 void Joystick_rx::Reset()
90 {
91   Serial1.clear();
92   ni = 0;
93   dn = 0;
94   delta = 0.0f;
95   WD_timer = millis();
96   t = micros();
97   joydata = joydata_init;
98   nr = 0.0f;
99   nl = 0.0f;
100 }
101
102 M_aux_cmd Joystick_rx::getAuxCmd()
103 {
104   return aux_cmd;
105 }
106
107 M_data_extra Joystick_rx::getDataExtra()
108 {
109
110   return _data_extra;
111 }
112
113 void Joystick_rx::mainloop()
114 {
115   if (micros() - t >= dt_joy)
116   {
117     t = micros();
118     _update();
119   }
120 }
121
122 void Joystick_rx::log_cmd_change()
123 {
124   const static char *fmt PROGMEM = "%s: %s\n";
125   const static char *fmt_2 PROGMEM = "Mappatura: %d\n";
126   const static char *enable PROGMEM = "abilitato";
127   const static char *disable PROGMEM = "disabilitato";
128   const static char *rear PROGMEM = "Posteriore";
129   const static char *fuzzy PROGMEM = "fuzzy";
130   const static char *sp_red_str PROGMEM = "Riduzione di velocita'";
131   const static char *cruise_str PROGMEM = "Cruise control";
132   const static char *duty_str PROGMEM = "Trasferimento duty cycle medio";
133   static bool rear_en = true;
134   static bool fuzzy_en = true;
135   static bool sp_red = false;
136   static bool cruise_control = false;
137   static bool duty_mean = false;
138   static uint8_t Map = 0;
139   if (aux_cmd.rear_en != rear_en)
```

```
140   {
141     rear_en = aux_cmd.rear_en;
142     theStream->printf(fmt, rear, (rear_en ? enable : disable));
143   }
144   else if (aux_cmd.fuzzy_en != fuzzy_en)
145   {
146     fuzzy_en = aux_cmd.fuzzy_en;
147     theStream->printf(fmt, fuzzy, (fuzzy_en ? enable : disable));
148   }
149   else if (aux_cmd.duty_mean != duty_mean)
150   {
151     duty_mean = aux_cmd.duty_mean;
152     theStream->printf(fmt, duty_str, (duty_mean ? enable : disable));
153   }
154   else if (_data_extra.speedReduction != sp_red)
155   {
156     sp_red = _data_extra.speedReduction;
157     theStream->printf(fmt, sp_red_str, (sp_red ? enable : disable));
158   }
159   else if (_data_extra.cruise_control != cruise_control)
160   {
161     cruise_control = _data_extra.cruise_control;
162     theStream->printf(fmt, cruise_str, (cruise_control ? enable : disable
      ));
163   }
164   else if (_data_extra.Map != Map)
165   {
166     Map = _data_extra.Map;
167     theStream->printf(fmt_2, Map);
168   }
169 }
```

Codice A.11: File header “FuzzySettings.h”

```
1 #pragma once
2 #include <Arduino.h>
3 #include <Settings.h>
4 #include <Functions.h>
5 #include <Fuzzy.h>
6
7 Fuzzy* FuzzySetup();
```

Codice A.12: File sorgente “FuzzySettings.cpp”

```
1 #include "FuzzySettings.h"
2
3 Fuzzy *FuzzySetup()
4 {
5   ////Fuzzy//////
6   static Fuzzy *fuzzy = new Fuzzy();
7
8   // variabili per modificare piu' agevolmente le classi di input e
      output
9   // ogni varole indica il picco della classe
10
```

```
11  // input: errore normalizzato
12  float I1_NG = -0.8f;
13  float I1_N = -0.3f;
14  float I1_Z = 0.0f;
15  float I1_P = 0.3f;
16  float I1_PG = 0.8f;
17
18  // input: velocita' media
19  float I2_B = 0.2f;
20  float I2_M = 0.5f;
21  float I2_A = 0.8f;
22
23  // output: Coefficiente ruota interna
24  const float O1_NG = -0.8f;
25  const float O1_N = -0.4f;
26  const float O1_Z = 0.0f;
27  const float O1_RG = 0.4f;
28  const float O1_R = 0.8f;
29  const float O1_P = 1.0f;
30  const float O1_PG = 1.2f;
31  const float O1_PMG = 1.4f;
32  float O1_inf = 2 * O1_NG - O1_N;   // Calcolo l'estremo inferiore
      affinche' la classe NG sia simmetrica
33  float O1_sup = 2 * O1_PMG - O1_PG; // stessa cosa per il superiore
34
35  // output: Coefficiente ruota esterna
36  const float O2_RG = 0.4f;
37  const float O2_R = 0.8f;
38  const float O2_P = 1.0f;
39  const float O2_PG = 1.2f;
40  const float O2_PMG = 1.4f;
41  float O2_inf = 2 * O2_RG - O2_R;
42  float O2_sup = 2 * O2_PMG - O2_PG;
43
44  // FuzzyInput (AngleRef - Angle, normalizzato)
45  static FuzzySet *NG_err = new FuzzySet(-2.0f, -2.0f, I1_NG, I1_N);
46  static FuzzySet *N_err = new FuzzySet(I1_NG, I1_N, I1_N, I1_Z);
47  static FuzzySet *Z_err = new FuzzySet(I1_N, I1_Z, I1_Z, I1_P);
48  static FuzzySet *P_err = new FuzzySet(I1_Z, I1_P, I1_P, I1_PG);
49  static FuzzySet *PG_err = new FuzzySet(I1_P, I1_PG, 2.0f, 2.0f);
50
51  // FuzzyInput (|velocita' media anteriore|)
52  static FuzzySet *B_speed = new FuzzySet(0.0f, 0.0f, I2_B, I2_M);
53  static FuzzySet *M_speed = new FuzzySet(I2_B, I2_M, I2_M, I2_A);
54  static FuzzySet *A_speed = new FuzzySet(I2_M, I2_A, 1.0f, 1.0f);
55
56  // FuzzyOutput (ruota interna)
57  static FuzzySet *NG_ri = new FuzzySet(O1_inf, O1_NG, O1_NG, O1_N);
58  static FuzzySet *N_ri = new FuzzySet(O1_NG, O1_N, O1_N, O1_Z);
59  static FuzzySet *Z_ri = new FuzzySet(O1_N, O1_Z, O1_Z, O1_RG);
60  static FuzzySet *RG_ri = new FuzzySet(O1_Z, O1_RG, O1_RG, O1_R);
61  static FuzzySet *R_ri = new FuzzySet(O1_RG, O1_R, O1_R, O1_P);
62  static FuzzySet *P_ri = new FuzzySet(O1_R, O1_P, O1_P, O1_PG);
63  static FuzzySet *PG_ri = new FuzzySet(O1_P, O1_PG, O1_PG, O1_PMG);
64  static FuzzySet *PMG_ri = new FuzzySet(O1_PG, O1_PMG, O1_PMG, O1_sup);
```

```
65
66  // FuzzyOutput (ruota esterna)
67  static FuzzySet *RG_re = new FuzzySet(O2_inf, O2_RG, O2_RG, O2_R);
68  static FuzzySet *R_re = new FuzzySet(O2_RG, O2_R, O2_R, O2_P);
69  static FuzzySet *P_re = new FuzzySet(O2_R, O2_P, O2_P, O2_PG);
70  static FuzzySet *PG_re = new FuzzySet(O2_P, O2_PG, O2_PG, O2_PMG);
71  static FuzzySet *PMG_re = new FuzzySet(O2_PG, O2_PMG, O2_PMG, O2_sup);
72
73  // Building input and output
74  static FuzzyInput *err = new FuzzyInput(1);
75  err->addFuzzySet(NG_err);
76  err->addFuzzySet(N_err);
77  err->addFuzzySet(Z_err);
78  err->addFuzzySet(P_err);
79  err->addFuzzySet(PG_err);
80  fuzzy->addFuzzyInput(err);
81
82  static FuzzyInput *speed = new FuzzyInput(2);
83  speed->addFuzzySet(B_speed);
84  speed->addFuzzySet(M_speed);
85  speed->addFuzzySet(A_speed);
86  fuzzy->addFuzzyInput(speed);
87
88  static FuzzyOutput *Ci = new FuzzyOutput(1); // coefficiente ruota
      interna
89  Ci->addFuzzySet(NG_ri);
90  Ci->addFuzzySet(N_ri);
91  Ci->addFuzzySet(Z_ri);
92  Ci->addFuzzySet(RG_ri);
93  Ci->addFuzzySet(R_ri);
94  Ci->addFuzzySet(P_ri);
95  Ci->addFuzzySet(PG_ri);
96  Ci->addFuzzySet(PMG_ri);
97  fuzzy->addFuzzyOutput(Ci);
98
99  static FuzzyOutput *Ce = new FuzzyOutput(2); // coeffciente ruota
      esterna
100 Ce->addFuzzySet(RG_re);
101 Ce->addFuzzySet(R_re);
102 Ce->addFuzzySet(P_re);
103 Ce->addFuzzySet(PG_re);
104 Ce->addFuzzySet(PMG_re);
105 fuzzy->addFuzzyOutput(Ce);
106
107 // Building FuzzyRule
108
109 // Ruota interna
110
111 // rule 1: NG = B * NG
112 static FuzzyRuleAntecedent *ifBAndNG = new FuzzyRuleAntecedent();
113 ifBAndNG->joinWithAND(B_speed, NG_err);
114
115 static FuzzyRuleConsequent *thenNG_ri = new FuzzyRuleConsequent();
116 thenNG_ri->addOutput(NG_ri);
117
```

```
118   static FuzzyRule *fuzzyRule1 = new FuzzyRule(1, ifBAndNG, thenNG_ri);
119   fuzzy->addFuzzyRule(fuzzyRule1);
120
121   // rule 2: N = B * N
122   static FuzzyRuleAntecedent *ifBAndN = new FuzzyRuleAntecedent();
123   ifBAndN->joinWithAND(B_speed, N_err);
124
125   static FuzzyRuleConsequent *thenN_ri = new FuzzyRuleConsequent();
126   thenN_ri->addOutput(N_ri);
127
128   static FuzzyRule *fuzzyRule2 = new FuzzyRule(2, ifBAndN, thenN_ri);
129   fuzzy->addFuzzyRule(fuzzyRule2);
130
131   // rule 3: Z = M * NG
132   static FuzzyRuleAntecedent *ifMAndNG = new FuzzyRuleAntecedent();
133   ifMAndNG->joinWithAND(M_speed, NG_err);
134
135   static FuzzyRuleConsequent *thenZ_ri = new FuzzyRuleConsequent();
136   thenZ_ri->addOutput(Z_ri);
137
138   static FuzzyRule *fuzzyRule3 = new FuzzyRule(3, ifMAndNG, thenZ_ri);
139   fuzzy->addFuzzyRule(fuzzyRule3);
140
141   // rule 4: RG = M * N + A * NG
142   static FuzzyRuleAntecedent *ifMAndN = new FuzzyRuleAntecedent();
143   ifMAndN->joinWithAND(M_speed, N_err);
144
145   static FuzzyRuleAntecedent *ifAAndNG = new FuzzyRuleAntecedent();
146   ifAAndNG->joinWithAND(A_speed, NG_err);
147
148   static FuzzyRuleAntecedent *ifMAndNOrAAndNG = new FuzzyRuleAntecedent()
        ;
149   ifMAndNOrAAndNG->joinWithOR(ifMAndN, ifAAndNG);
150
151   static FuzzyRuleConsequent *thenRG_ri = new FuzzyRuleConsequent();
152   thenRG_ri->addOutput(RG_ri);
153
154   static FuzzyRule *fuzzyRule4 = new FuzzyRule(4, ifMAndNOrAAndNG,
        thenRG_ri);
155   fuzzy->addFuzzyRule(fuzzyRule4);
156
157   // rule 5: R = A * N
158   static FuzzyRuleAntecedent *ifAAndN = new FuzzyRuleAntecedent();
159   ifAAndN->joinWithAND(A_speed, N_err);
160
161   static FuzzyRuleConsequent *thenR_ri = new FuzzyRuleConsequent();
162   thenR_ri->addOutput(R_ri);
163
164   static FuzzyRule *fuzzyRule5 = new FuzzyRule(5, ifAAndN, thenR_ri);
165   fuzzy->addFuzzyRule(fuzzyRule5);
166
167   // rule 6: P = Z
168   static FuzzyRuleAntecedent *ifZ = new FuzzyRuleAntecedent();
169   ifZ->joinSingle(Z_err);
170
```

```
171  static FuzzyRuleConsequent *thenP_ri = new FuzzyRuleConsequent();
172  thenP_ri->addOutput(P_ri);
173
174  static FuzzyRule *fuzzyRule6 = new FuzzyRule(6, ifZ, thenP_ri);
175  fuzzy->addFuzzyRule(fuzzyRule6);
176
177  // rule 7: PG = P
178  static FuzzyRuleAntecedent *ifP = new FuzzyRuleAntecedent();
179  ifP->joinSingle(P_err);
180
181  static FuzzyRuleConsequent *thenPG_ri = new FuzzyRuleConsequent();
182  thenPG_ri->addOutput(PG_ri);
183
184  static FuzzyRule *fuzzyRule7 = new FuzzyRule(7, ifP, thenPG_ri);
185  fuzzy->addFuzzyRule(fuzzyRule7);
186
187  // rule 8: PMG = PG
188  static FuzzyRuleAntecedent *ifPG = new FuzzyRuleAntecedent();
189  ifPG->joinSingle(PG_err);
190
191  static FuzzyRuleConsequent *thenPMG_ri = new FuzzyRuleConsequent();
192  thenPMG_ri->addOutput(PMG_ri);
193
194  static FuzzyRule *fuzzyRule8 = new FuzzyRule(8, ifPG, thenPMG_ri);
195  fuzzy->addFuzzyRule(fuzzyRule8);
196
197  // Ruota esterna
198  // rule 9: RG = B * PG
199  static FuzzyRuleAntecedent *ifBAndPG = new FuzzyRuleAntecedent();
200  ifBAndPG->joinWithAND(B_speed, PG_err);
201
202  static FuzzyRuleConsequent *thenRG_re = new FuzzyRuleConsequent();
203  thenRG_re->addOutput(RG_re);
204
205  static FuzzyRule *fuzzyRule9 = new FuzzyRule(9, ifBAndPG, thenRG_re);
206  fuzzy->addFuzzyRule(fuzzyRule9);
207
208  // rule 10: R = P + PG * (M + A)
209  static FuzzyRuleAntecedent *ifMOrA = new FuzzyRuleAntecedent();
210  ifMOrA->joinWithOR(M_speed, A_speed);
211
212  static FuzzyRuleAntecedent *ifPGAnd_MOrA_ = new FuzzyRuleAntecedent();
213  ifPGAnd_MOrA_->joinWithAND(PG_err, ifMOrA);
214
215  static FuzzyRuleAntecedent *ifPOrPGAnd_MOrA_ = new FuzzyRuleAntecedent
       ();
216  ifPOrPGAnd_MOrA_->joinWithOR(P_err, ifPGAnd_MOrA_);
217
218  static FuzzyRuleConsequent *thenR_re = new FuzzyRuleConsequent();
219  thenR_re->addOutput(R_re);
220
221  static FuzzyRule *fuzzyRule10 = new FuzzyRule(10, ifPOrPGAnd_MOrA_,
       thenR_re);
222  fuzzy->addFuzzyRule(fuzzyRule10);
223
```

```
224   // rule 11: P = Z
225   static FuzzyRuleAntecedent *ifZ_2 = new FuzzyRuleAntecedent();
226   ifZ_2->joinSingle(Z_err);
227
228   static FuzzyRuleConsequent *thenP_re = new FuzzyRuleConsequent();
229   thenP_re->addOutput(P_re);
230
231   static FuzzyRule *fuzzyRule11 = new FuzzyRule(11, ifZ_2, thenP_re);
232   fuzzy->addFuzzyRule(fuzzyRule11);
233
234   // rule 12: PG = N
235   static FuzzyRuleAntecedent *ifN = new FuzzyRuleAntecedent();
236   ifN->joinSingle(N_err);
237
238   static FuzzyRuleConsequent *thenPG_re = new FuzzyRuleConsequent();
239   thenPG_re->addOutput(PG_re);
240
241   static FuzzyRule *fuzzyRule12 = new FuzzyRule(12, ifN, thenPG_re);
242   fuzzy->addFuzzyRule(fuzzyRule12);
243
244   // rule 13: PMG = NG
245   static FuzzyRuleAntecedent *ifNG = new FuzzyRuleAntecedent();
246   ifNG->joinSingle(NG_err);
247
248   static FuzzyRuleConsequent *thenPMG_re = new FuzzyRuleConsequent();
249   thenPMG_re->addOutput(PMG_re);
250
251   static FuzzyRule *fuzzyRule13 = new FuzzyRule(13, ifNG, thenPMG_re);
252   fuzzy->addFuzzyRule(fuzzyRule13);
253
254   return fuzzy; // ritorno il puntatore
255 }
```

Codice A.13: File header “Functions.h”

```
1  #pragma once
2  #include <Arduino.h>
3  #include <Settings.h>
4
5  #ifdef memctrl
6  uint32_t FreeMem();
7  #endif
8
9  // funzione per la mappatura (float)
10 float map_float(float x, float in_min, float in_max, float out_min, float
       out_max);
11
12 // filtro di "media mobile"
13 // exponentially weighted moving average (EWMA)
14 float EWMA_filter(float alfa, float measure_filt, float measure_now);
15
16 // funzione segno, ritorna 1 o -1
17 float sign(float val);
18
19 // float smoothgrad(float, float, float, float, float);
```

```
20
21 loop_time loop_duration(bool _return); // calcola la durata massima del
       loop in un intervallo di tempo a scelta
22
23 #ifdef blocked_ctrl
24 void loop_state_ctrl(Stream &_stream); // stampa ogni tot cicli per
       leggere eventuali blocchi
25 #endif
26
27 #ifdef debug_block
28 void debug_T_print(Stream &_stream, const char *_msg, uint32_t dt); //
       stampa sullo stream voluto in messaggio inviato ogni tot tempo [ms]
29 #endif
```

Codice A.14: File sorgente “Functions.cpp”

```
1  #include <Arduino.h>
2  #include "Functions.h"
3
4  #ifdef memctrl
5  uint32_t FreeMem()
6  {
7      uint32_t stackTop;
8      uint32_t heapTop;
9
10     // current position of the stack.
11     stackTop = (uint32_t)&stackTop;
12
13     // current position of heap.
14     void *hTop = malloc(1);
15     heapTop = (uint32_t)hTop;
16     free(hTop);
17
18     // The difference is (approximately) the free, available ram.
19     return stackTop - heapTop;
20 }
21 #endif
22
23 ////////////////////////////////////////
24 //// Map() function for float types ////
25 float map_float(float x, float in_min, float in_max, float out_min, float
       out_max)
26 {
27     return (x - in_min) * (out_max - out_min) / (in_max - in_min) +
       out_min;
28 }
29
30 ////////////////////////////////////////
31 //// EWMA Filter (digital low pass) ////
32 float EWMA_filter(float alfa, float measure_filt, float measure_now)
33 {
34     return measure_filt = (1.0f - alfa) * measure_filt + alfa *
       measure_now;
35 }
36
```

```
37 float sign(float val)
38 {
39     return (val < 0.0f ? -1.0f : 1.0f);
40 }
41
42 loop_time loop_duration(bool _return)
43 {
44     static uint32_t n_loop = 0; // salva il numero di loop
45     static uint32_t start_time = micros(); // tempo iniziale
46     static uint32_t loop_t = 0; // variabile che salva la durata massima
47     static uint32_t t = micros(); // variabile che tiene il tempo
48     if (_return) // se true resetta la variabile e ritorna il valore di
       durata massima
49     {
50         uint32_t t_max = loop_t;
51         loop_t = 0;
52         uint32_t t_mean = static_cast<float>(micros() - start_time) /
       static_cast<float>(n_loop) + 0.5f;
53         n_loop = 0;
54         start_time = micros();
55
56         return {t_mean, t_max};
57     }
58     else // altrimenti calcola la durata del loop e verifica se sia
       maggiore del massimo trovato finora
59     {
60         uint32_t _t = micros();
61         uint32_t _dt = _t - t;
62         t = _t;
63         if (_dt > loop_t)
64         {
65             loop_t = _dt;
66         }
67         n_loop++;
68         return {0, 0};
69     }
70 }
71
72 #ifdef blocked_ctrl
73
74 void loop_state_ctrl(Stream &_stream)
75 {
76     static uint32_t n_loop = 0;
77     const static uint32_t t0 = millis(); // riferimento temporale
78     static uint32_t t = 0;
79     uint32_t _t = millis();
80     n_loop++;
81     if (_t - t >= 3000)
82     {
83         t = _t;
84 #ifdef memctrl
85         _stream.printf(F("t: %d, n_loop: %d, free memory: %d\n"), _t - t0
       , n_loop, FreeMem());
86 #else
87         _stream.printf(F("t: %d, n_loop: %d\n"), _t - t0, n_loop);
```

```
88 #endif
89     }
90 }
91
92 #endif
93
94 #ifdef debug_block
95 void debug_T_print(Stream &_stream, const char *_msg, uint32_t dt)
96 {
97     static uint32_t t = 0;
98     if (millis() - t >= dt)
99     {
100         t = millis();
101         _stream.println(_msg);
102     }
103 }
104 #endif
```

Codice A.15: File header “Settings.h”

```
1 #pragma once
2 #include <Arduino.h>
3
4 // definizioni per il debug
5 //  #define memctrl     //serve per stampare la memoria disponibile a
    runtime
6 //  #define blocked_ctrl //stampa ogni tanto a video per verificare che
    non ci sia un blocco del micro
7 //  #define debug_block     //serve per il debug nel caso il micro si
    blocchi, per circoscrivere la funzione che ha causato tale blocco
8 //  #define Inverted //se definito imposta di default il modulo
    posteriore come modulo anteriore
9
10 // durata dei loop
11 const uint32_t dt_meas = 10 * 1000;  //[us] tempo in microsecondi,
    velocita' e corrente;
12 const uint32_t dt_cmd = 10 * 1000;   //[us] intervallo di tempo
    variazione del comando al motore
13 const uint32_t dt_joy = 20 * 1000;   //[us]tempo campionamento comandi
14 const uint32_t dt_angle = 10 * 1000; //[us] tempo di campionamento angolo
15 const uint32_t omegaCoeff = 3;       // coefficiente che indica ogni
    quanti angoli misurati viene calcolata la velocita' angolare di
    imbardata
16 const uint32_t dt_cleanup = 500;     //[ms] delta di tempo per cui se il
    set e' 0 viene pulito il valore dell'integrativa per mandare il pwm a
     zero
17 const uint32_t Connection_WD = 800;  //[ms]timer di sicurezza nel caso il
     robot non riceva piu' comandi
18 const uint32_t dt_fuzzy = 20 * 1000; //[us] intervallo per il controllo
    fuzzy
19 const uint32_t dt_log = 30 * 1000;   //[us] intervallo salvataggio dei
    dati
20
21 // filtri di media mobile
```

```
22 // alpha = 2/(N+1) dove N e' il numero di campioni della "media mobile
       approssimata"
23 //  _s -> feedback di velocita'
24 //  _c -> feedback di corrente
25 //  _a -> feedback angolo di imbardata
26 const float alpha_s = 2.0f / (15.0f + 1.0f); // peso filtro passa basso
27 const float alpha_c = 2.0f / (20.0f + 1.0f); // il precedente e' per la
       velocita', questo e' per la corrente
28 const float alpha_a = 2.0f / (10.0f + 1.0f); // filtro passa basso
29
30 // Strutture dati utilizzate
31 struct pid_data // struttura per leggere i dati del pid
32 {
33   float set;
34   float fb;
35   float error;
36   float ffw;
37   float INT;
38   float DER;
39   float ctrl;
40 };
41
42 struct pid_param // struttura per settare i parametri del pid
43 {
44   float kp;
45   float ki;
46   float kd;
47 };
48
49 struct OL_gain_struct // coefficienti del polinomio, il numero indica il
       grado
50 {
51   float n_1;
52   float n_0;
53 };
54
55 struct pid_settings // struttura per il passaggio dei parametri del pid
56 {
57   pid_param SpeedCtrl;
58   OL_gain_struct OL_fw;
59   OL_gain_struct OL_bw;
60 };
61
62 struct Ref // struttura per il trasferimento dei riferimenti di velocita'
63 {
64   float L; // left side
65   float R; // right side
66 };
67
68 struct M_aux_cmd // struttura per inviare comandi ausiliari al motore
69 {
70   bool rear_en;   // abilita / disabilita posteriore
71   uint8_t mod;    // indica la modalita' attiva (Advancing o Pivot)
72   bool reverse;   // indica se si sta procedendo o meno in retro marcia
73   bool fuzzy_en;  // abilita / disabilita il controllo fuzzy
```

```
74   bool duty_mean; // definisce il tipo di trasferimento di duty cycle al
       posteriore
75 };
76
77 struct M_data_extra // struttura per il passaggio di informazioni
       aggiuntive che non influiscono nel controllo
78 {
79   uint8_t Map;
80   bool speedReduction;
81   bool cruise_control;
82 };
83
84 struct TXRX_DATA_STRUCT // struttura per il trasferimento dei dati da
       Joystick
85 {
86   int8_t nu;     // Pivot command
87   int8_t lambda; // Advancing command
88   uint8_t cmd;   // contano i singoli bit, non il numero nel suo
       complesso
89 };
90
91 ////////////////////////////////////////
92 /// MOTOR Pololu 37Dx54L 50:1 DATA ///
93 ////////////////////////////////////////
94
95 // Costanti per definire il moto
96 #define BRAKE 0
97 #define FREE 1
98 #define FORWARD 2
99 #define BACKWARD 3
100
101 // verso di rotazione
102 #define CW 0
103 #define CCW 1
104
105 // definizione della tipologia di controllo
106 #define NONE 0 // nessun controllo impostato
107 #define SC 1   // speed control
108 #define DC 2   // duty control
109
110 // impostazioni di adc e pwm
111 #define PWM_RESOLUTION 10     // risoluzione del PWM (bit)
112 #define A_RESOLUTION 12       // risoluzione adc microcontrollore (bit)
113 #define PWM_FREQUENCY 4882.8f // Hz frequenza del pwm (non e' un
       impostazione globale, ma puo' essere settata per il singolo pin)
114 // frequenza di defaul 488.28 Hz
115
116 // pinout
117 // example: {IN1, IN2, PWM, FB, EN, ENC_A, ENC_B, FW_dir}
118 //  FW_dir: verso di rotazione del motore che permette l'avanzamento
       guardando la ruota dentata
119 #ifndef Inverted // in condizioni normali il pinout giusto e' questo
120 #define FL_pinout {9, 24, 8, A25, 7, 25, 26, CCW} // front left
121 #define FR_pinout {5, 6, 2, A26, 7, 27, 28, CW} // front right
122 #define RL_pinout {39, 38, 14, A11, 15, 29, 30, CCW} // rear left
```

```
123 #define RR_pinout {17, 16, 20, A10, 15, 31, 32, CW} // rear right
124
125 #else // siccome voglio usare il posteriore come se fosse l'anteriore
        devo invertire il verso di rotazione che consente l'avanzamento
126 #define FL_pinout {9, 24, 8, A25, 7, 25, 26, CW} // front left
127 #define FR_pinout {5, 6, 2, A26, 7, 27, 28, CCW} // front right
128 #define RL_pinout {39, 38, 14, A11, 15, 29, 30, CW} // rear left
129 #define RR_pinout {17, 16, 20, A10, 15, 31, 32, CCW} // rear right
130 #endif
131
132 // parametri PID
133 // parametri del pid, controllo in velocita'
134 // prime parentesi graffe {kp, ki, kd}
135 // nelle altre invece abbiamo i coefficienti di una retta y = mx + q
136 // avremo quindi {m, q}, quelli con segno meno sono per la retromarcia
137 #ifndef Inverted
138 #define Motor_1_pid {{3.0f, 15.0f, 0.03f}, {0.5900f, 3909.48f}, {0.5048f,
        -4585.98f}}
139 #define Motor_2_pid {{3.0f, 15.0f, 0.03f}, {0.4796f, 5077.31f}, {0.5816f,
        -4221.38f}}
140 #define Motor_3_pid {{3.0f, 15.0f, 0.03f}, {0.4528f, 5535.57f}, {0.4875f,
        -5323.77f}}
141 #define Motor_4_pid {{3.0f, 15.0f, 0.03f}, {0.6521f, 4006.65f}, {0.6558f,
        -3820.87f}}
142
143 #else
144 #define Motor_1_pid {{3.0f, 15.0f, 0.03f}, {0.5048f, 4585.98f}, {0.5900f,
        -3909.48f}}
145 #define Motor_2_pid {{3.0f, 15.0f, 0.03f}, {0.5816f, 4221.38f}, {0.4796f,
        -5077.31f}}
146 #define Motor_3_pid {{3.0f, 15.0f, 0.03f}, {0.4875f, 5323.77f}, {0.4528f,
        -5535.57f}}
147 #define Motor_4_pid {{3.0f, 15.0f, 0.03f}, {0.6558f, 3820.87f}, {0.6521f,
        -4006.65f}}
148 #endif
149
150 #ifndef Inverted
151
152 #define MFL_pid Motor_1_pid
153 #define MFR_pid Motor_2_pid
154 #define MRL_pid Motor_3_pid
155 #define MRR_pid Motor_4_pid
156
157 #else
158
159 #define MFL_pid Motor_4_pid
160 #define MFR_pid Motor_3_pid
161 #define MRL_pid Motor_2_pid
162 #define MRR_pid Motor_1_pid
163
164 #endif
165
166 // controllo in coppia
167 const float k_torque = 0.24f;     //[N*mm/mA] Torque constant
```

```
168 const float torque_max = 1000.0f; //[N*mm] coppia massima permessa al
    motore
169 const float C_deadband = 50.0f;   // banda morta per il controllo in
    coppia applicata alla corrente
170
171 // controllo in velocita'
172 const float n_max = 10000.0f; // rpm  velocita' a vuoto del motore
173 const float ffw_th = 50.0f;   //[rpm] valore di soglia oltre il quale
    entra in funzione il feedforward
174
175 // sensibilita' dei feedback
176 const float CURRENT_FB = 525.0f; //[mV/A] guadagno del feedback
177 const uint8_t CPR = 64;          // impulsi per giro dell'encoder
178
179 const float epsilon = 1e-6f; // costante piccola arbitraria per confronto
180
181 // parametri pwm
182 const uint16_t MAX_PWM = static_cast<uint16_t>(pow(2, PWM_RESOLUTION) -
    0.5f); // imposto il pwm massimo in base alla risoluzione impostata (
    il -0.5 deriva da -1 + 0.5, quest'ultimo serve per l'arrotondamento)
183
184 const float dutyCoeff = 0.9f; // coefficiente di riduzione del duty cycle
     al posteriore quando non e' abilitato il controllo fuzzy
185
186 ////////////////////
187 //////Joystick//////
188
189 // definizione delle posizioni dei bit nella variabile
190 // sono gli stessi anche nella classe Joystick_tx
191 #define Rear_en_bit 0b10000000
192 #define fuzzy_bit 0b01000000
193 #define SpeedRed_bit 0b00100000
194 #define Cruise_control_bit 0b00010000
195 #define Duty_bit 0b00001000
196 #define Map_bit 0b00000011
197
198 #define joydata_init {0, 0, Rear_en_bit | fuzzy_bit | Duty_bit} // inizio
     con posteriore e fuzzy abilitati, e trasferimento diretto del duty
199
200 #define adv_mod 1   // modalita' avanzamento
201 #define pivot_mod 2 // modalita' pivot
202
203 const uint32_t XBEE_BAUD = 38400ul; // Serial1 (0(RX) e 1(TX))
204 const float JS_out_max = 125.0f;    // valore massimo di lambda e nu
205 const float pivotCoeff = 0.3f;      // coefficiente di riduzione del
    comando in modalita' pivot
206 const float pivotAngleLim = 25.0f;  // angolo limite per l'uso del
    posteriore nella modalita' pivot
207
208 ////////////////////////////
209 ///////AngleSensor////////
210 #define NoRef 100.0f // numero arbitrariamente scelto da passare quando
    la velocita' di avanzamento del robot e' bassa (non si esegue il
    calcolo del riferimento)
211
```

```
212 const float MaxAngleRef = 2836.0f; // lettura analogica con angolo pari a
        +38 [deg] (curva antioraria)
213 const float MinAngleRef = 1507.0f; // lettura analogica con angolo pari a
        -38 [deg] (Curva oraria)
214 const float MaxAngle = 38.0f;      //[deg] Angolo massimo del robot (
    simmetrico rispetto allo zero)
215
216 // misure fisiche del robot per il calcolo dell'angolo (va bene qualsiasi
        unita' di misura, basta che ci sia coerenza)
217 const float I = 270.0f; //[mm] interasse ruote
218 const float a = 132.0f; //[mm] modulo anteriore: distanza tra asse
    centrale ruote ed asse del giunto
219 const float b = 139.0f; //[mm] modulo posteriore: distanza tra asse
    centrale ruote ed asse del giunto
220
221 ////////////////////////////
222 /////////Logging//////////
223 /*Quantita' di dati salvati nel log
224 1 -> completo
225 2 -> normale
226 3 -> minimale
227
228 1: timestamp, lambda, nu, (set, speed, err, ffw, int, der, ctrl, duty,
    current)x4, angleSet, angleRef, angle, omega, err_norm(angle), Ci, Ce
    , Cr, Cl, motion_mode, map, speedReduction
229 2: timestamp, (set, speed, int, ctrl, duty, current)x4, angleRef, angle,
    omega, err_norm(angle), Cr, Cl, motion_mode, map
230 3: timestamp, (set, speed, duty, current)x4, angleRef, angle, omega,
    motion_mode, map
231
232 */
233 #define completo 1
234 #define normale 2
235 #define minimale 3
236
237 #define Logger completo // definire qui che tipo di log si desidera (
    impostare un tempo di campionamento adeguato in funzione della scelta
     fatta)
238
239 struct loop_time
240 {
241   uint32_t mean;
242   uint32_t max;
243 };
244
245 struct Other_log_data // la struttura serve per passare i valori alla
    funzione di Log
246 {
247   float *AngleRef;
248   float *err_norm;
249   float *Ci;
250   float *Ce;
251   float *Cr;
252   float *Cl;
253   M_data_extra *_extra;
```

```
254   loop_time t_loop;
255 };
```

Codice A.16: File header “Logging.h”

```
1  #pragma once
2  #include <Arduino.h>
3  #include <Epi_q.h>
4  #include <SD.h>
5
6  #define NUMEL(x) (sizeof(x) / sizeof((x)[0]))
7
8  void Log_dyn(File &_file, DC_Motor **_M_arr, uint8_t _n_motor,
       AngleSensor &_AS, Other_log_data &_OD);
9  /*
10 Funzione di log dinamica,
11 salva i dati in funzione del parametro "Logger" impostato nel file "
       Settings.h"
12 */
13
14 void plot_init_config(Stream &_stream, DC_Motor **_M_arr, uint8_t
       _n_motor);
```

Codice A.17: File sorgente “Logging.cpp”

```
1  #include "Logging.h"
2
3  void Log_dyn(File &_file, DC_Motor **_M_arr, uint8_t _n_motor,
       AngleSensor &_AS, Other_log_data &_OD)
4  {
5    // "Set_%d[rpm], speed_%d[rpm], err_%d[rpm], ffw_%d[rpm], INT_%d[rpm],
       ctrl_%d[rpm], Duty_%d[-], Current_%d[mA], "
6    // "AngleSet[deg], AngleRef[deg], Angle[deg], Omega[deg/s], err_norm
       [-], Ci[-], Ce[-], Cr[-], Cl[-], Map[-], Speed_reduction[-],
       loop_duration[us]"
7    static DC_Motor *M;
8    static pid_data _data;
9  #if Logger == completo
10 #define n_times(x) x, x, x, x, x, x, x, x, x // il numero di x e' pari al
       numero di parametri in "fmt_header"
11   const static char *fmt_header PROGMEM = "Set_%d, speed_%d, err_%d, ffw_
       %d, INT_%d, der_%d, ctrl_%d, Duty_%d, Current_%d, "; // stringa di
       formattazione dell'header per dati motore
12   const static char *fmt_motor = "%.2f, %.2f, %.4f, %.2f, %.2f, %.2f, %.2
       f, %.4f, %.1f, "; // stringa di formattazione dei parametri del
       motore
13 #define Motor_param M->getRef(), M->getSpeed(), _data.error, _data.ffw,
       _data.INT, _data.DER, _data.ctrl, M->getDuty(), M->getCurrent()
14   const static char *header_2 PROGMEM = "AngleRef, Angle, Omega, err_norm
       , Ci, Ce, Cr, Cl, Map, Speed_reduction, t_loop_mean, t_loop_max"; //
       stringa per header degli altri parametri
15   const static char *fmt_other = "%.3f, %.3f, %.3f, %.4f, %.4f, %.4f, %.4
       f, %.4f, %d, %d, %d, %d\n"; // stringa di formattazione degli altri
       parametri
```

```
16 #define Other_param *(_OD.AngleRef), _AS.getAngle(), _AS.getOmega(), *(
       _OD.err_norm), *(_OD.Ci), *(_OD.Ce), *(_OD.Cr), *(_OD.Cl), _OD._extra
       ->Map, _OD._extra->speedReduction, _OD.t_loop.mean, _OD.t_loop.max
17
18 #elif Logger == normale
19 #define n_times(x) x, x, x, x, x
20   const static char *fmt_header PROGMEM = "Set_%d, speed_%d, INT_%d,
       Duty_%d, Current_%d, ";
21   const static char *fmt_motor = "%.2f, %.2f, %.2f, %.4f, %.1f, ";
22 #define Motor_param M->getRef(), M->getSpeed(), _data.INT, M->getDuty(),
       M->getCurrent()
23   const static char *header_2 PROGMEM = "AngleRef, Angle, Omega, Cr, Cl,
       Map";
24   const static char *fmt_other = "%.3f, %.3f, %.3f, %.4f, %.4f, %d";
25 #define Other_param *(_OD.AngleRef), _AS.getAngle(), _AS.getOmega(), *(
       _OD.Cr), *(_OD.Cl), _OD._extra->Map
26
27 #elif Logger == minimale
28 #define n_times(x) x, x, x
29   const static char *fmt_header PROGMEM = "Set_%d[rpm], speed_%d[rpm],
       Duty_%d[-], ";
30   const static char *fmt_motor = "%.2f, %.2f, %.4f, ";
31 #define Motor_param M->getRef(), M->getSpeed(), M->getDuty()
32   const static char *header_2 PROGMEM = "Angle, Map";
33   const static char *fmt_other = "%.3f, %d";
34 #define Other_param _AS.getAngle(), _OD._extra->Map
35 #endif
36   static bool flag = true; // Serve per stampare l'header sul file
37
38   if (flag)
39   {
40     flag = false;
41     _file << "TimeStamp, ";
42     for (uint8_t i = 1; i <= _n_motor; i++)
43     {
44       _file.printf(fmt_header, n_times(i));
45     }
46     _file.println(header_2);
47   }
48   const static uint32_t t0 = millis();   // salvo il tempo al primo loop
       cosi' da avere il timestamp che parte da zero, o almeno da un numero
       basso
49   _file << millis() - t0 << ", ";        // scrivo il timestamp
50   for (uint8_t i = 0; i < _n_motor; i++) // ciclo sui motori
51   {
52     M = _M_arr[i];
53     _data = M->pid.get_data();
54     _file.printf(fmt_motor, Motor_param); // stampo i dati dei motori
55   }
56   _file.printf(fmt_other, Other_param); // stampo gli altri dati
57   _file.flush();
58 }
59
60 void plot_init_config(Stream &_stream, DC_Motor **_M_arr, uint8_t
       _n_motor)
```

```
{
  const static char *fmt_config_time PROGMEM = "\n--loops time--\n"
                                               "dt_meas: %d [ms]\n"
                                               "dt_cmd: %d [ms]\n"
                                               "dt_joy: %d [ms]\n"
                                               "dt_angle: %d [ms]\n"
                                               "omegaCoeff: %d [-]\n"
                                               "dt_cleanup: %d [ms]\n"
                                               "Connection_WD: %d [ms]\n"
                                               "dt_fuzzy: %d [ms]\n"
                                               "dt_log: %d [ms]\n\n"
                                               "--motor config--\n\n";

  _stream.printf(fmt_config_time, dt_meas / 1000, dt_cmd / 1000, dt_joy / 1000,
                 dt_angle / 1000, omegaCoeff, dt_cleanup, Connection_WD,
                 dt_fuzzy / 1000, dt_log / 1000);

  const static char *fmt_config_motor PROGMEM = "%s\n"
                                                "kp: %.3f, ki: %.3f, kd: %.3f\n"
                                                "Ol_fw: %.4f, %.4f\n"
                                                "Ol_bw: %.4f, %.4f\n\n";

  const static char *motor_name[4] PROGMEM = {"Front left",
                                              "Front right",
                                              "Rear left",
                                              "Rear right"};

  for (uint8_t i = 0; i < _n_motor; i++)
  {
    pid_settings settings = _M_arr[i]->pid.get_settings();
    _stream.printf(fmt_config_motor, motor_name[i],
                   settings.SpeedCtrl.kp, settings.SpeedCtrl.ki,
                   settings.SpeedCtrl.kd, settings.OL_fw.n_1,
                   settings.OL_fw.n_0, settings.OL_bw.n_1,
                   settings.OL_bw.n_0);
  }
}
```

Codice A.18: File sorgente “main.cpp”

```
#include <Arduino.h>
#include <Epi_q.h>
#include <Logging.h>
#include <Test.h>
#include <EEPROM.h>

// nel caso si usi il posteriore come se fosse l'anteriore scambio i pinout dei vari motori
// in modo che il motore 1 sia sempre il front left dalla nuova prospettiva del robot
#ifndef Inverted
DC_Motor MFL(FL_pinout);
DC_Motor MFR(FR_pinout);
```

```
12 DC_Motor MRL(RL_pinout);
13 DC_Motor MRR(RR_pinout);
14 #else
15 DC_Motor MFL(RR_pinout);
16 DC_Motor MFR(RL_pinout);
17 DC_Motor MRL(FR_pinout);
18 DC_Motor MRR(FL_pinout);
19 #endif
20
21 DC_Motor *M_arr[4] = {&MFL, &MFR, &MRL, &MRR};
22 const uint8_t n_motor = NUMEL(M_arr);
23
24 Module M1(&MFL, &MFR); // Modulo anteriore
25 Module M2(&MRL, &MRR); // Modulo posteriore
26
27 Module *Front; // modulo controllato in velocita'
28 Module *Rear;  // modulo controllato in duty cycle e controllo fuzzy
29
30 Joystick_rx JS{}; // uso le parentesi graffe per chiamare il costruttore,
31                   //  se usassi le tonde verrebbe letto come
    dichiarazione di una funzione
32
33 AngleSensor YawAngleSensor(A21);
34
35 uint32_t t_log = 0;
36 uint32_t t_fuzzy = 0;
37
38 File file;
39
40 M_aux_cmd aux_cmd;
41 M_data_extra data_extra;
42
43 bool reverse_mode = false; // variabile usata per trovare i cambi di
    direzione
44
45 float err_norm = 0.0f; // errore angolare normalizzato
46 float ni_norm = 0.0f;  // modulo velocita' media front normalizzata
47 float AngleRef = 0.0f; // angolo di riferimento
48
49 float Cr = 0.0f; // coefficiente ruota destra
50 float Cl = 0.0f; // coefficiente ruota sinistra
51 float Ci = 0.0f; // coefficiente ruota interna
52 float Ce = 0.0f; // coefficiente ruota esterna
53
54 Other_log_data OD{&AngleRef, &err_norm, &Ci, &Ce, &Cr, &Cl, &data_extra,
    {0, 0}};
55
56 Fuzzy *fuzzy; // Puntatore alla classe fuzzy, viene ritornato dalla
    funzione FuzzySetup
57
58 bool log_en = true;
59
60 void setup()
61 {
62   Serial1.begin(XBEE_BAUD);
```

```
63    Serial1 << "Init setup..." << endl;
64  #ifdef memctrl
65    Serial1 << "Free memory: " << FreeMem() << endl;
66  #endif
67
68    fuzzy = FuzzySetup(); // costruisco il fuzzy e ritorno il puntatore
        alla classe
69
70    JS.begin(&Serial1);
71    if (!SD.begin(BUILTIN_SDCARD))
72    {
73      Serial1 << "SD fail..." << endl;
74      log_en = false;
75      delay(4000);
76    }
77    analogWriteResolution(PWM_RESOLUTION);
78    analogReadResolution(A_RESOLUTION);
79
80    MFL.init(MFL_pid);
81    MFR.init(MFR_pid);
82    MRL.init(MRL_pid);
83    MRR.init(MRR_pid);
84
85    plot_init_config(Serial1, M_arr, n_motor);
86
87    Front = &M1;
88    Rear = &M2;
89
90    Front->set_type_ctrl(SC);
91    Rear->set_type_ctrl(DC);
92    Front->enable(true);
93    Rear->enable(true);
94
95    if (log_en)
96    {
97      uint8_t counter = EEPROM.read(0);
98      char name[13];
99      sprintf(name, "Log_%03d.txt", ++counter);
100     EEPROM.update(0, counter);
101     file = SD.open(name, FILE_WRITE);
102   }
103
104   Serial1 << "End setup..." << endl;
105 #ifdef memctrl
106   Serial1 << "Free memory: " << FreeMem() << endl;
107 #endif
108
109   Front->Reset();
110   Rear->Reset();
111   JS.Reset();
112   t_log = micros();
113   t_fuzzy = micros();
114 }
115
116 void loop()
```

```
117 {
118   JS.mainloop();                          // chiamo il loop del joystick
119   aux_cmd = JS.getAuxCmd();               // leggo i comandi ausiliari
120   if (aux_cmd.reverse != reverse_mode) // eseguo questa parte solo se c'e
        ' un cambio di valore
121   {
122     reverse_mode = aux_cmd.reverse;
123     if (!aux_cmd.reverse) // imposto il modulo frontale in funzione della
       direzione di marcia
124     {
125       Front = &M1;
126       Rear = &M2;
127     }
128     else
129     {
130       Front = &M2;
131       Rear = &M1;
132     }
133     Front->set_type_ctrl(SC);   // assegno al front il controllo in
      velocita' nel caso sia stato scambiato
134     Front->Left->Reset_light(); // eseguo un reset del pid e di alcune
      variabili
135     Front->Right->Reset_light();
136     Rear->set_type_ctrl(DC); // stessa cosa con il rear, ma con il
      controllo in duty
137     Rear->Left->Reset_light();
138     Rear->Right->Reset_light();
139   }
140   Rear->enable(aux_cmd.rear_en); // abilito / disabilito modulo
141
142   Front->setRef(JS.getRef()); // assegno il riferimento in velocita' al
       front
143   Front->mainloop();          // chiamo il loop del front
144
145   YawAngleSensor.mainloop(); // chiamo il loop del sensore angolare
146
147   Ref dutyRef = Front->getDuty(); // mi salvo il duty cycle impostato al
       front
148
149   float angleMeasured = YawAngleSensor.getAngle(); // angolo misurato dal
       sensore
150
151   if (aux_cmd.mod == pivot_mod) // controllo se sono in modalita' pivot
152   {
153     if (abs(angleMeasured) < pivotAngleLim)
154     {
155       Cr = -1.0f; // uso i coefficienti per invertire il duty cycle del
      posteriore
156       Cl = -1.0f;
157     }
158     else
159     {
160       Cr = 0.0f; // in caso contrario li mando a zero entrambi e uso solo
       l'anteriore
161       Cl = 0.0f;
```

```
162     }
163   }
164   else if (aux_cmd.mod == adv_mod)
165   {
166     if (aux_cmd.duty_mean) // se abilitato passo al posteriore il duty
      medio (che dovra' comunque essere moltiplicato per il coefficiente)
167     {
168       dutyRef.L = (dutyRef.L + dutyRef.R) / 2.0f;
169       dutyRef.R = dutyRef.L;
170     }
171
172     AngleRef = JS.getAngleRef();
173
174     if (!aux_cmd.fuzzy_en) // senza il controllo fuzzy il coefficiente e'
       costante
175     {
176       Cr = dutyCoeff;
177       Cl = dutyCoeff;
178     }
179     else if (micros() - t_fuzzy >= dt_fuzzy)
180     {
181
182       t_fuzzy = micros();
183
184       Ref frontSpeed = Front->getSpeed();
185
186       if (AngleRef != NoRef) // se ho un riferimento angolare esegue il
      fuzzy
187       {
188         err_norm = sign(angleMeasured) * (AngleRef - angleMeasured) /
      MaxAngle; // calcolo l'errore normalizzato da passare al fuzzy
189         ni_norm = abs(frontSpeed.L + frontSpeed.R) / (2.0f * n_max); //
      modulo della velocita' media normalizzata
190         ni_norm = constrain(ni_norm, 0.0f, 1.0f); // saturo a 1 perche'
      fluttuazioni di velocita' potrebbero far salire il valore
191
192         fuzzy->setInput(1, err_norm); // inserisco il primo input
193         fuzzy->setInput(2, ni_norm);  // inserisco il secondo input
194
195         fuzzy->fuzzify();
196
197         Ci = fuzzy->defuzzify(1); // output 1 -> coefficiente ruota
      interna
198         Ce = fuzzy->defuzzify(2); // output 2 -> coefficiente ruota
      esterna
199
200         if (angleMeasured < 0.0f) // imposto i coefficienti in funzione
      di quale sia la ruota interna alla curva
201         {
202           Cr = Ci;
203           Cl = Ce;
204         }
205         else
206         {
207           Cr = Ce;
```

```
208           Cl = Ci;
209         }
210       }
211       else // se non ho un riferimento angolare metto i coefficienti a
      zero
212       {
213         Cl = 0.0f;
214         Cr = 0.0f;
215       }
216     }
217   }
218
219   dutyRef = {Cl * dutyRef.L, Cr * dutyRef.R}; // eseguo il calcolo del
      duty da mandare al rear
220   Rear->setRef(dutyRef); // assegno i riferimenti
221
222   Rear->mainloop();
223   if (log_en)
224   {
225     if (micros() - t_log >= dt_log)
226     {
227       t_log = micros();
228       OD.t_loop = loop_duration(true);
229       Log_dyn(file, M_arr, n_motor, YawAngleSensor, OD);
230     }
231     // JS.log_cmd_change();
232     loop_duration(false); // misura la durata massima di un singolo loop
      tra una chiama "loop_duration(true)" e l'altra
233   }
234 #ifdef blocked_ctrl
235   loop_state_ctrl(Serial1); // stampa a video ogni tanto per vedere se il
       micro si pianta o meno
236 #endif
237 }
```

A.2 Codice sorgente del joystick di comando

L'approccio utilizzato per la scrittura del codice sorgente è lo stesso utilizzato sopra, ma visto che la mole di righe di codice è decisamente inferiore si è deciso di inserire le costanti, i parametri, le strutture dati e le funzioni ausiliarie direttamente del file header della classe.

Il file "main.cpp" non contiene quasi niente se non il richiamo del metodo di inizializzazione e il mainloop. Tutto il codice è scritto all'interno della classe.

Codice A.19: File header "Joystick_tx.h"

```
1 #pragma once
2 #include <Arduino.h>
3 #include <SoftPWM.h>
4 #include <SoftPWM_timer.h>
5 #include <EasyTransfer.h>
6 #include <Streaming.h>
7 #include <JC_Button.h>
8
```

```
9  #define Rear_en_bit 0b10000000
10 #define fuzzy_bit 0b01000000
11 #define SpeedRed_bit 0b00100000
12 #define Cruise_control_bit 0b00010000
13 #define Duty_bit 0b00001000
14 #define Map_bit 0b00000011
15
16 #define joydata_init {0, 0, Rear_en_bit | fuzzy_bit | Duty_bit} // inizio
       con posteriore e fuzzy abilitati
17
18 float map_float(float x, float in_min, float in_max, float out_min, float
       out_max);
19
20 float lin(float _x, float _coeff); // funzione per mappatura lineare, non
       fa niente il pratica
21
22 float pol(float _x, float _coeff); // funzione polinomiale di mappatura
23
24 #undef powf
25
26 float powf(float _x, float _exp); // funzione esponenziale di mappatura
27
28 const unsigned long XBEE_BAUD = 38400ul; // Serial1 Xbee
29 // const uint32_t B_delay = 1000;           //[ms] durata long press
30 const uint32_t dt_loop = 20 * 1000; //[us] durata minima del loop
31 const int AmpSemiBand = 8;          // Ampiezza della banda morta
32 const float out_max = 125.0f;       // valore massimo per le variabili nu
       e lambda
33 const float SpeedRedCoeff = 0.7f;   // coefficiente di riduzione di
     velocita' quando si attiva apposita modalita'
34 const int8_t lambda_cc_step = 15;   // incremento di lambda del cruise
     control
35 const int8_t nu_cc_step = 30;       // incremento di nu  del cruise
     control
36
37 // Script to run on the DF Robot wireless joystick
38 // The board is a Leonardo board, so set it correctly in the IDE
39 // pinout joystick
40 #define leftPadV 4       // analog up/dw left pad (lambda)
41 #define leftPadH 5       // analog L/R left pad
42 #define rightPadV 2      // analog up/dw right pad
43 #define rightPadH 3      // analog L/R right pad (nu)
44 #define leftPadButton 1  // analog 0 quando premuto, 1023 quando
     rilasciato
45 #define rightPadButton 0 // analog 0 quando premuto, 1023 quando
     rilasciato
46 #define vibrationPin 2   // digital
47 #define L1_pin 15        // Left Z1 digital
48 #define L2_pin 16        // Left Z2 digital
49 #define R1_pin 13        // Right Z1 digital
50 #define R2_pin 14        // Right Z2 digital
51 #define up_pin 5         // digital permette di attivare l'anteriore
52 #define down_pin 7       // digital permette di disattivare l'anteriore
53 #define right_pin 8      // digital permette di attivare il posteriore
54 #define left_pin 6       // digital permette di disattivare il posteriore
```

```
55 #define LED_pin 17        // Pin del led
56 #define triangle_pin 9    // digital pulsante "1"
57 #define circle_pin 11     // digital pulsante "2"
58 #define x_pin 12          // digital pulsante "3"
59 #define square_pin 10     // digital pulsante "4"
60 #define start_pin 4       // digital pulsante start
61 #define select_pin 3      // digital pulsante select
62
63 // mappature abilitate sul joystick
64 // mappatura 0 -> pulsante 1
65 // mappatura 1 -> pulsante 2
66 // mappatura 2 -> pulsante 3
67 // mappatura 3 -> pulsante 4
68
69 #define L_map_0 {powf, 3.0f} // mappatura per il lambda
70 #define N_map_0 {powf, 3.0f} // mappatura per il nu in modalita' pivot
71
72 #define L_map_1 {powf, 3.5f}
73 #define N_map_1 {powf, 3.0f}
74
75 #define L_map_2 {pol, 0.0f}
76 #define N_map_2 {powf, 2.5f}
77
78 #define L_map_3 {lin, 1.0f} // potrei usare la mappatura esponenziale con
       coefficiente 1, ma cosi' risparmio un po' di conti
79 #define N_map_3 {lin, 1.0f} // nella mappatura lineare posso inserire il
     coefficiente che voglio
80
81 #define lambda_map {L_map_0, L_map_1} // metto insieme le mappature e le
     passo all'array
82 #define nu_map {N_map_0, N_map_1}
83
84 #define N_MAP_TOT 2 // numero di mappature salvate
85
86 struct TXRX_DATA_STRUCT
87 {
88   int8_t nu;     // Pivot command
89   int8_t lambda; // Advancing command
90   uint8_t cmd;   // contano i singoli bit, non il numero nel suo
       complesso
91 };
92
93 struct Map
94 {
95   float (*func)(float, float); // puntatore a funzione per scegliere la
       funzione di mappatura
96   float _coeff; // coefficiente da passare alla funzione
97 };
98
99 class Joystick_tx
100 {
101 private:
102   TXRX_DATA_STRUCT joydata = joydata_init;
103   uint32_t t = 0;
104   uint8_t nMap = 0; // mappatura scelta [0, 3]
```

```
105   const Map L_map_array[N_MAP_TOT] = lambda_map;
106   const Map N_map_array[N_MAP_TOT] = nu_map;
107   float speedCoeff = 1.0f;
108   int8_t lambda = 0;
109   int8_t nu_l = 0;         // mappa l'analogico sinistro
110   int8_t nu_r = 0;         // mappa l'analogico destro
111   bool nu_cc = false;      // cruise control
112   bool lambda_cc = false; // cruise control
113
114 public:
115   Button up;
116   Button down;
117   Button right;
118   Button left;
119   Button tri; //"1"
120   Button cir; //"2"
121   Button x;   //"3"
122   Button sqr; //"4"
123   Button start;
124   Button select;
125   Button R1;
126   Button R2;
127   Button L1;
128   Button L2;
129   AnalogButton L3;
130   EasyTransfer ETJoy;
131   Joystick_tx();
132   Joystick_tx(const Joystick_tx &_JS) = delete;
133   void begin();
134   int8_t PadMapping(int _read, const int _semiAmp, const float _exp,
       float (*_func)(float, float)); // mappatura della leva analogica
135   void getState();
136   void update();
137   void sendCMD();
138   void vibrating();
139   void mainloop();
140 };
```

Codice A.20: File sorgente "Joystick_tx.cpp"

```
1  #include "Joystick_tx.h"
2
3  float map_float(float x, float in_min, float in_max, float out_min, float
       out_max)
4  {
5    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min
       ;
6  };
7
8  float lin(float _x, float _coeff) // funzione di mappatura lineare
9  {
10   return _x;
11 }
12
13 float pol(float _x, float _coeff) // funzione polinomiale di mappatura
```

```
{
  return 2 * (_coeff - 1) * pow(_x, 3) - 3 * (_coeff - 1) * pow(_x, 2) +
    _coeff * _x;
}

float powf(float _x, float _exp) // funzione di mappatura di potenza
{
  return static_cast<float>(pow(_x, _exp));
}

Joystick_tx::Joystick_tx()
    : up(up_pin),
      down(down_pin),
      right(right_pin),
      left(left_pin),
      tri(triangle_pin),
      cir(circle_pin),
      x(x_pin),
      sqr(square_pin),
      start(start_pin),
      select(select_pin),
      R1(R1_pin),
      R2(R2_pin),
      L1(L1_pin),
      L2(L2_pin),
      L3(leftPadButton)
{
}

void Joystick_tx::begin()
{
  Serial1.begin(XBEE_BAUD);
  ETJoy.begin(details(joydata), &Serial1);
  up.begin();
  down.begin();
  right.begin();
  left.begin();
  tri.begin();
  cir.begin();
  x.begin();
  sqr.begin();
  start.begin();
  select.begin();
  R1.begin();
  R2.begin();
  L1.begin();
  L2.begin();
  L3.begin();
  pinMode(LED_pin, OUTPUT);
  pinMode(vibrationPin, OUTPUT);
  SoftPWMBegin();
  SoftPWMSet(vibrationPin, 0);
  SoftPWMSetFadeTime(vibrationPin, 1800, 1200);
  digitalWrite(LED_pin, HIGH);
  t = micros();
```

```
68 }
69
70 int8_t Joystick_tx::PadMapping(int _read, const int _semiAmp, const float
       _coeff, float (*_func)(float, float))
71 {
72   _read -= 512; // sposto la posizione neutra del joystick al valore 0
73   float out = 0;
74   if (_read >= 0)
75   {
76     _read = constrain(_read, _semiAmp, 511);                              //
       i limiti della parte positiva e di quella negativa sono asimmetrici,
       pero' amen, non fa differenza
77     out = map_float(_read, _semiAmp, 511.0f, 0.0f, out_max) / out_max; //
       eseguo una prima mappatura e normalizzo
78     out = (*_func)(out, _coeff) * out_max * speedCoeff + 0.5f;          //
       eseguo una seconda mappatura e lo rimoltiplico per il fondo scala e
      per il coefficiente di riduzione di velocita' (la somma di 0.5 serve
      per l'arrotondamento in fase di casting)
79   }
80   else
81   {
82     _read = constrain(_read, -512.0f, -_semiAmp);
83     out = map_float(_read, -512.0f, -_semiAmp, -out_max, 0.0f) / (-
      out_max); // normalizzo con segno positivo
84     out = (*_func)(out, _coeff) * (-out_max) * speedCoeff - 0.5f;
          // mappatura, cambio segno e sottrazione per arrotondamento
85   }
86   return static_cast<int8_t>(out); // casting nel tipo voluto
87 }
88
89 void Joystick_tx::getState()
90 {
91   up.read();
92   down.read();
93   right.read();
94   left.read();
95   tri.read();
96   cir.read();
97   x.read();
98   sqr.read();
99   start.read();
100  select.read();
101  R1.read();
102  R2.read();
103  L1.read();
104  L2.read();
105  L3.read();
106  lambda = PadMapping(analogRead(leftPadV), AmpSemiBand, L_map_array[nMap
       ]._coeff, L_map_array[nMap].func); // uso la funzione polinomiale per
       mappare la velocita' di avanzamento
107  nu_l = PadMapping(analogRead(leftPadH), AmpSemiBand, N_map_array[nMap].
       _coeff, N_map_array[nMap].func);   // Mappatura per la modalita'
       pivot
108  nu_r = PadMapping(analogRead(rightPadH), AmpSemiBand, 0.7, powf); //
       Mappatura dello sterzo per la modalita' advancing
```

```
109 }
110
111 void Joystick_tx::update()
112 {
113   // implementazione del cruise control
114   if (up.wasPressed()) // aumenta lambda (avanzamento)
115   {
116     joydata.lambda += lambda_cc_step;
117     joydata.lambda = constrain(joydata.lambda, -out_max, out_max);
118     lambda_cc = (joydata.lambda != 0 ? true : false);
119   }
120   else if (down.wasPressed())
121   {
122     joydata.lambda -= lambda_cc_step;
123     joydata.lambda = constrain(joydata.lambda, -out_max, out_max);
124     lambda_cc = (joydata.lambda != 0 ? true : false);
125   }
126
127   if (right.wasPressed())
128   {
129     joydata.nu -= nu_cc_step;
130     joydata.nu = constrain(joydata.nu, -out_max, out_max);
131     nu_cc = (joydata.nu != 0 ? true : false);
132   }
133   else if (left.wasPressed()) // aumenta nu (sterzo)
134   {
135     joydata.nu += nu_cc_step;
136     joydata.nu = constrain(joydata.nu, -out_max, out_max);
137     nu_cc = (joydata.nu != 0 ? true : false);
138   }
139
140   if (lambda != 0 || nu_l != 0 || !lambda_cc) // se una delle tre
      condizioni si verifica disabilito il cruise control
141   {
142     joydata.lambda = lambda;
143     lambda_cc = false;
144     nu_cc = false;
145   }
146
147   if (!nu_cc)
148   {
149     joydata.nu = (joydata.lambda == 0 ? nu_l : nu_r);
150   }
151
152   if (L3.wasPressed()) // resetta il cruise control
153   {
154     joydata.lambda = 0;
155     joydata.nu = 0;
156     lambda_cc = false;
157     nu_cc = false;
158   }
159
160   if (lambda_cc || nu_cc)
161   {
162     joydata.cmd |= Cruise_control_bit;
```

```
163   }
164   else
165   {
166     joydata.cmd &= ~Cruise_control_bit;
167   }
168   // fine implementazione del cruise control
169
170   // scelta mappatura
171   if (tri.wasPressed())
172   {
173     nMap = 0;                 // mappatura 0 (default)
174     joydata.cmd &= ~Map_bit; // pulisco i bit della mappatura
175     joydata.cmd |= nMap;     // scrivo i bit della mappatura
176   }
177   else if (cir.wasPressed())
178   {
179     nMap = 1; // mappatura 1
180     joydata.cmd &= ~Map_bit;
181     joydata.cmd |= nMap;
182   }
183   // fine mappature
184
185   // passaggio di duty cycle
186   if (x.wasPressed())
187   {
188     joydata.cmd |= Duty_bit; // imposto a 1 il bit corrispondente
189   }
190   else if (sqr.wasPressed())
191   {
192     joydata.cmd &= ~Duty_bit;
193   }
194
195   // riduzione di velocita'
196   if (select.wasPressed())
197   {
198     speedCoeff = SpeedRedCoeff;  // riduco la velocita' massima (cambia
      leggermente la mappatura selezionata)
199     joydata.cmd |= SpeedRed_bit; // imposto a 1 il bit corrispondente
200   }
201   else if (start.wasPressed())
202   {
203     speedCoeff = 1.0f;            // ripristino la velocita' massima
204     joydata.cmd &= ~SpeedRed_bit; // imposto a 0 il bit corrispondente
205   }
206
207   // controllo fuzzy
208   if (R1.wasPressed())
209   {
210     joydata.cmd |= fuzzy_bit; // attiva il controllo fuzzy
211   }
212   else if (L1.wasPressed())
213   {
214     joydata.cmd &= ~fuzzy_bit; // disattiva il controllo fuzzy
215   }
216
```

```
217   // posteriore
218   if (R2.wasPressed()) // attiva posteriore
219   {
220     joydata.cmd |= Rear_en_bit;
221   }
222   else if (L2.wasPressed()) // disattiva posteriore
223   {
224     joydata.cmd &= ~Rear_en_bit;
225   }
226 }
227
228 void Joystick_tx::sendCMD()
229 {
230   ETJoy.sendData();
231 }
232
233 void Joystick_tx::vibrating()
234 {
235   uint8_t _speed = (abs(joydata.lambda) + 0.2 * abs(joydata.nu));
236   _speed = constrain(_speed, 0, 100);
237   SoftPWMSetPercent(vibrationPin, _speed);
238 }
239
240 void Joystick_tx::mainloop()
241 {
242   if (micros() - t >= dt_loop)
243   {
244     t = micros();
245     getState();
246     update();
247     sendCMD();
248     vibrating();
249   }
250 }
```

Codice A.21: File sorgente “main.cpp”

```
1  #include <Arduino.h>
2  #include <Joystick_tx.h>
3
4  Joystick_tx Joystick{};
5
6  void setup()
7  {
8    Joystick.begin();
9  }
10
11 void loop()
12 {
13   Joystick.mainloop();
14 }
```

Lista dei simboli

ω	$\mathrm{rad} \cdot \mathrm{s}^{-1}$	Velocità angolare
τ_0	-	Rapporto di trasmissione con portatreno bloccato
τ_p	-	Rapporto di trasmissione con ruote bloccate
τ_m	-	Rapporto di trasmissione del motoriduttore
z	-	Numero di denti
m	mm	Modulo del dente
l_A	mm	Lunghezza del braccetto
l_R	mm	Distanza tra giunto e attacco del braccetto del modulo posteriore
l_F	mm	Distanza tra giunto e attacco del braccetto del modulo anteriore
l_L	mm	Lunghezza dell'arto, interasse tra ruota solare e satelliti esterni
r_W	mm	Raggio degli pneumatici
i	mm	Interasse tra il gli appoggi degli pneumatici dello stesso modulo
β_R	mm	Angolo di inclinazione del braccetto posteriore
β_F	mm	Angolo di inclinazione del braccetto anteriore
a	mm	Distanza della ruota solare del centro del giunto (modulo anteriore)
b	mm	Distanza della ruota solare del centro del giunto (modulo posteriore)
h_tot	mm	Altezza totale del robot
l_tot	mm	Lunghezza totale del robot
w_tot	mm	Larghezza totale del robot
ρ	mm	Raggio di curvatura della traiettoria percorsa
v_r	$\mathrm{m} \cdot \mathrm{s}^{-1}$	Velocità lineare tripode destro
v_l	$\mathrm{m} \cdot \mathrm{s}^{-1}$	Velocità lineare tripode sinistro
v_m	$\mathrm{m} \cdot \mathrm{s}^{-1}$	Velocità lineare del modulo
Δv	$\mathrm{m} \cdot \mathrm{s}^{-1}$	Differenza di velocità tra il lato destro e il sinistro
n_r	rpm	Velocità angolare motore destro
n_l	rpm	Velocità angolare motore sinistro
n_m	rpm	Media delle velocità angolari dei motori dello stesso modulo
Δn	rpm	Differenza di velocità angolare tra il motore destro e il sinistro
δ	deg	Angolo di imbardata relativo
K	-	Coefficiente di curvatura, rapporto tra le velocità di uno stesso modulo
λ	-	Comando di velocità longitudinale
ν	-	Comando di sbilanciamento delle velocità
k_p	-	Guadagno proporzionale del PID
k_i	-	Guadagno integrativo del PID
k_d	-	Guadagno derivativo del PID

$\overline{e}$	-	Errore angolare normalizzato
$\overline{v}$	-	Velocità di avanzamento normalizzata
C_{i}	-	Coefficiente della ruota interna
C_{e}	-	Coefficiente della ruota esterna
C_{r}	-	Coefficiente della ruota destra
C_{l}	-	Coefficiente della ruota sinistra
d	-	Duty cycle
ψ	rad	Angolo di inclinazione della leva analogica del joystick
y	-	Lettura delle leve analogiche tramite ADC del microcontrollore
h	-	Semi-ampiezza della banda morta nella lettura degli analogici

Bibliografia

[1] "Robot." *Wikipedia, L'enciclopedia libera.* 25 set 2021, 07:48 UTC. 7 gen 2022, 15:20, <https://it.wikipedia.org/w/index.php?title=Robot&oldid=123157064>.

[2] L. Bruzzone, G. Quaglia, "Locomotion systems for ground mobile robots in unstructured environments", *Mechanical sciences*, 2012, 3.2: 49-62.

[3] L. Bruzzone, P. Fanghella, "Functional redesign of Mantis 2.0, a hybrid leg-wheel robot for surveillance and inspection", *Journal of Intelligent & Robotic Systems*, 2016, 81.2: 215-230.

[4] J. Kim et al., "Wheel & Track hybrid robot platform for optimal navigation in an urban environment", In *Proceedings of SICE Annual Conference 2010*, IEEE, 2010, p. 881-884.

[5] F. Michaud et al., "Multi-modal locomotion robotic platform using leg-track-wheel articulations", *Autonomous Robots*, 2005, 18.2: 137-156.

[6] G. Quaglia et al., "Epi.q-1.2: a new hybrid mobile mini robot", *Proceedings of RAAD*, 2008, 15-17.

[7] G. Quaglia et al., "Epi.Q Mobile robots family", In *ASME International Mechanical Engineering Congress and Exposition*, 2011, p. 1165-1172.

[8] G. Bozzini et al., "Design of the small mobile robot Epi.q-2", *Proceedings of AIMETA*, 2009.

[9] G. Quaglia et al., "A modular approach for a family of ground mobile robots", *International Journal of Advanced Robotic Systems*, 2013, 10.7: 296.

[10] G. Quaglia et al., "UGV epi.q-mod", In *Advances on Theory and Practice of Robots and Manipulators*, Springer, Cham, 2014, p. 331-339.

[11] M. Callegari, P. Fanghella, F. Pellicano, "Trasmissioni di Potenza", *Meccanica applicata alle macchine*, seconda edizione, Città Studi, 2017, pp. 394 - 397.

[12] A.Botta et al., "An Estimator for the Kinematic Behaviour of a Mobile Robot Subject to Large Lateral Slip", *Applied Sciences*, 2021, 11.4: 1594.

[13] V. Viktorov, F. Colombo, "Controllo Fuzzy", *Automazione dei sistemi automatici*, quinta edizione, CLUT, 2010, pp.165-194.

[14] "Moving average." *Wikipedia, L'enciclopedia libera.* 31 gen 2022, 17:25 UTC. 1 mar 2022, 10:30, <https://en.wikipedia.org/wiki/Moving_average>.

[15] "Exponential smoothing." *Wikipedia, L'enciclopedia libera.* 26 set 2021, 23:45 UTC. 1 mar 2022, 12:30, <https://en.wikipedia.org/wiki/Exponential_smoothing>.

[16] "Guida Librerie Epi.Q", *YouTube*, caricato da Simone Pantanetti, 3 gen. 2022, <https://www.youtube.com/watch?v=VG_z7QVV3H8>. Ultimo accesso 3 gen. 2022.

[17] L. Bruzzone et al., "Experimental assessment and evolution perspectives of the epi.q mobile robot architecture", *International Journal Of Robotics Research*, 2010, 29: 81-91.

[18] A. Botta et al., "Modelling and experimental validation of articulated mobile robots with hybrid locomotion system", In *The International Conference of IFToMM ITALY*, Springer, Cham, 2020, p. 758-767.

[19] F. Minozzo, *Sperimentazione su robot mobile*, 2016, Politecnico di Torino.