UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione

Progettazione e sviluppo di algoritmi su dispositivi edge per l'esecuzione di manovre autonome su veicolo in scala 1:10

Design and development of algorithms on edge devices for executing autonomous maneuvers on a 1:10 scale vehicle

Relatore:
Andrea Bonci

Candidato:
Matteo Colletta

Anno Accademico 2023-2024

UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione

Progettazione e sviluppo di algoritmi su dispositivi edge per l'esecuzione di manovre autonome su veicolo in scala 1:10

Design and development of algorithms on edge devices for executing autonomous maneuvers on a 1:10 scale vehicle

Relatore:
Andrea Bonci

Candidato:
Matteo Colletta

Anno Accademico 2023-2024

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione
Via Brecce Bianche – 60131 Ancona (AN), Italy

Ringraziamenti

Desidero esprimere la mia più sincera gratitudine a tutti coloro che hanno contribuito, sia direttamente che indirettamente, alla realizzazione di questa tesi e alla conclusione di questo percorso universitario. Senza i loro stimoli, questo lavoro non sarebbe stato possibile.

In primis vorrei porgere i miei ringraziamenti al mio relatore, il Prof. Andrea Bonci, che mi ha coinvolto nella Bosch Future Mobility Challenge, porgendo la sua disponibilità e la sua esperienza nel settore durante tutto il percorso di ricerca. Un ringraziamento speciale va anche al Dott. Alessandro Di Biase e alla Dott.ssa Ilaria Pellicani, che hanno contribuito significativamente alla stesura di questo elaborato.

Desidero esprimere la mia più profonda gratitudine alla mia famiglia, che mi ha sempre dato libero arbitrio nelle scelte di vita, supportandomi in ciascuna di esse. Un particolare ringraziamento va a zia Anka e a zio Najni, che sin da giovane hanno instillato in me la curiosità e la necessità di comprendere la natura di ciò che mi circonda, esaminandone i minimi dettagli. Inoltre, nonostante non ci sia un vero e proprio vincolo di parentela tra di noi, vorrei ringraziare il mio gatto Gimbo per essere stato presente nei momenti più duri, porgendomi il suo affetto incondizionato.

Vorrei ringraziare i miei compagni di corso, ormai Dottori, Angjelo Libofsha, Angelo Kollcaku, Valerio Crocetti, Loris Bottegoni, Leonardo Cambiotti e Nicholas Hinterwaldner, per avermi accompagnato da vicino verso questo traguardo, condividendo interminabili giornate di studio in università. Desidero anche ringraziare i Dottori Federico Brunella, Andrea Faccenda e Matteo Losa, con cui ho condiviso l'esperienza della Bosch Future Mobility Challenge 2024.

Un ringraziamento speciale va anche ai miei amici più stretti, che mi hanno accompagnato nella vita quotidiana durante questo percorso. Vorrei infatti ringraziare Pierma, Chiara, Deno, Ripa, Angela, Ale, Cecilia, Rizzi, Marta, Giovanni Edoardo, Sofia, Chiara, Irene, Leonardo, Gionny, Fra e Diegs per avermi accompagnato da vicino in questi anni. Vorrei anche ringraziare Giovanni, Luca, Alessandro, Marko, Demo e Matteo per avermi sostenuto da lontano, indipendentemente dalla distanza che ci separa.

Infine, un pensiero va a tutto coloro che in un modo o nell'altro hanno reso possibile il raggiungimento di questo traguardo, in quanto la legge della causalità detta il funzionamento dell'universo in cui viviamo.

Ancona, Ottobre 2024

Matteo Colletta

Sommario

La seguente tesi ha l'obiettivo di illustrare il processo di sviluppo di algoritmi su microcontrollore per l'esecuzione di manovre di cambio corsia, sorpasso e parcheggio di un veicolo elettrico autonomo in scala 1:10.

Il veicolo è stato fornito dal Bosch Engineering Centre situato a Cluj-Napoca, Romania, come kit base per partecipare alla *Bosch Future Mobility Challenge 2024*. La competizione, su scala internazionale, prevede la progettazione e lo sviluppo di algoritmi adibiti alla guida autonoma del suddetto veicolo in un contesto urbano in scala 1:10. Il veicolo dovrà quindi essere in grado di navigare il percorso in maniera autonoma, rispettando le leggi del codice della strada, tutelando la propria sicurezza e quella di eventuali altri veicoli e pedoni.

Il veicolo fornito è stato modificato in modo da poter svolgere al meglio le task necessarie imposte dal regolamento di gara, mediante l'aggiunta di sensori e modifiche alla disposizione dei componenti del kit fornito. Questi sensori sono stati aggiunti per monitorare al meglio lo stato del veicolo e l'ambiente circostante, aspetti fondamentali nel contesto della guida autonoma. I sensori di cui fa uso il veicolo per navigare in modo autonomo sono una telecamera, un'IMU, un encoder e un LiDAR.

L'obiettivo di questa tesi non consiste però nello spiegare estensivamente gli algoritmi utilizzati per rendere possibile la guida autonoma del veicolo, ma verranno trattate solamente le tecniche e gli algoritmi che hanno reso possibili le manovre di cambio corsia, sorpasso e parcheggio del veicolo.

Indice

Elenco delle figure

Elenco delle tabelle

Parte I.

Introduzione

Capitolo 1.

Introduzione

1.1. Bosch Future Mobility Challenge (BFMC)

La *Bosch Future Mobility Challenge* (BFMC) è una competizione tecnica internazionale avviata dal Bosch Engineering Center Cluj nel 2017. La competizione invita ogni anno team di studenti a sviluppare algoritmi di guida autonoma e connettività su veicoli in scala 1:10, forniti dall'azienda, per navigare in un ambiente designato simulando una città intelligente in miniatura. Gli studenti lavorano per diversi mesi ai loro progetti in collaborazione con esperti Bosch e professori accademici per sviluppare gli algoritmi più performanti.

Attualmente si è giunti alla quinta edizione della challenge in quanto viene riproposta annualmente. Ogni edizione della BFMC aggiunge nuove sfide e task da affrontare, mantenendo alto il livello di competizione anche per i team che partecipano da più edizioni. Ai partecipanti della challenge viene fornito un veicolo in scala 1:10 munito di elettronica essenziale per completare le task richieste dal regolamento, ma viene comunque lasciata la libertà di effettuare modifiche al kit per raggiungere al meglio gli obiettivi e lasciare spazio a soluzioni creative.
Le principali task che il veicolo deve svolgere autonomamente sono:

- Cruise control;
- Lanekeeping;
- Navigazione autonoma di incroci e rotatorie;
- Rilevazione e interpretazione di cartelli stradali;
- Rilevazione ed interpretazione di semafori;
- Mantenimento della distanza di sicurezza da veicoli in moto e pedoni in carreggiata;
- Parcheggio parallelo;

Le numerose task da completare rappresentano una delle principali difficoltà nella gestione simultanea, poiché ciascuna riveste un ruolo cruciale nel funzionamento della

guida autonoma. Un errore nell'esecuzione di una di esse potrebbe compromettere seriamente la sicurezza del veicolo, mettendo a rischio sia i passeggeri che l'ambiente circostante

Tra le novità dell'edizione corrente della challenge c'è un sistema di auto-localizzazione del veicolo, realizzato mediante un sistema di tag mobili ed ancore fisse che permettono al veicolo di ottenere le proprie coordinate in tempo reale, a meno di un margine di errore spesso non trascurabile. Proprio per questo motivo, nonostante questa aggiunta possa smembrare di grande aiuto per il raggiungimento di un livello di autonomia adeguato, tale sistema di localizzazione si è rivelato essere un' ulteriore sfida, in quanto è stato necessario sviluppare algoritmi di sensor fusion al fine di ottenere una stima più accurata della posizione.

Il tracciato di gara è stato mappato con precisione, e su di esso è stato creato un grafo composto da nodi equidistanti collegati da archi orientati per rappresentare la direzione di marcia delle corsie. Grazie alla posizione in tempo reale del veicolo, è possibile determinare le manovre successive che deve compiere.

1.2. Obiettivo della tesi

L'obiettivo di questo elaborato è quello di offrire innanzitutto una panoramica generale sull'hardware impiegato, illustrando le motivazioni che hanno guidato l'integrazione di dei diversi componenti

A seguire verrà illustrato lo schema di connessione dei vari componenti e l'architettura a livello software che ne consegue.

Alla conclusione della panoramica sull'hardware, verranno approfondite le nozioni di fisica alla base degli algoritmi di controllo del veicolo, con particolare attenzione alle manovre di sorpasso e parcheggio.

Oltre a ciò, verranno discusse le implementazioni delle logiche e gli algoritmi necessari per la corretta esecuzione delle suddette manovre, seguite da un'analisi dei dati ottenuti durante i vari esperimenti.

Per concludere, verranno discussi sviluppi futuri degli algoritmi ed eventuali criticità riscontrate.

Parte II.

Hardware utilizzato e architettura del sistema

Capitolo 2.

Hardware utilizzato

In questo capitolo verranno illustrate le componenti presenti nel kit fornito per la competizione, oltre a quelle aggiunte in seguito per implementare le funzionalità necessarie a completare al meglio le task della challenge. In Figura 2.1 è presentato il kit fornito dalla BOSCH, sul quale sono presenti i seguenti componenti:

- STM32 Nucleo F401RE;
- Raspberry Pi 4 8GB;
- Motore Quickrun Fusion SE 1800KV;
- Servo motore;
- IMU (Inertial Measurement Unit) BNO055;
- Batteria;
- Pi Camera V2.1.

Al kit di partenza sono stati aggiunti i seguenti componenti:

- Coral TPU;
- Encoder AMT 103;
- LiDAR.

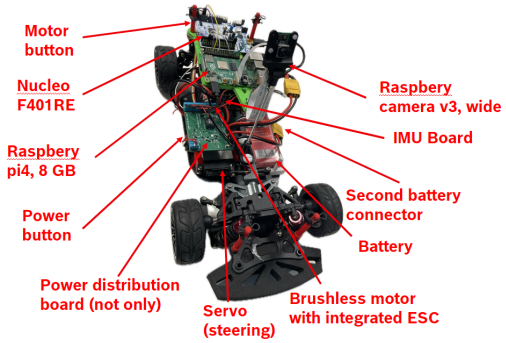


Figura 2.1.: Kit hardware fornito.

2.1. Chassis

Lo chassis del veicolo è il modello Reely TC-04 Onroad-Chassis 1:10, RC model car Electric Road version 4WD ARR (Figura 2.2), uno chassis pensato per la realizzazione di veicoli elettrici in scala 1:10 ad alte prestazioni. Dotato di frame in alluminio rigido e stabile, sospensioni regolabili e trazione integrale con differenziale anteriore e posteriore, esso garantisce un'ottima trazione anche ad alte velocità.

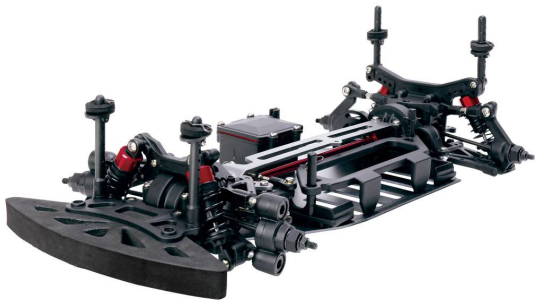


Figura 2.2.: Reely TC-04

Lo chassis è una struttura di suppporto sulla quale sono state montate le componenti necessarie per il corretto funzionamento del veicolo, tra cui batterie, schede di distribuzione, motori, sensori e schede di controllo.

Le informazioni su tale componente sono state prese dal datasheet [2] rilasciato dalla casa madre.

2.2. Motore

Il motore fornito dal kit base è il modello QUICKRUN Fusion SE 1800KV, sviluppato da Hobbywing. Questo dispositivo integra in un'unica unità compatta un motore DC brushless e un regolatore di velocità elettronico (ESC), riducendo l'ingombro e migliorando l'efficienza energetica. Il motore integra la tecnologia FOC (Field Oriented Control) che permette di controllare velocità e coppia senza bisogno di sensori aggiuntivi, garantendo ottime prestazioni anche a basse velocità di manovra.

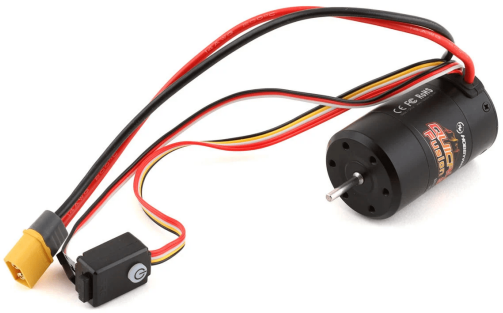


Figura 2.3.: Motore QUICKRUN Fusion SE 1800KV

Il motore utilizzato ha un rating di 1800KV (1800 RPM a vuoto per ogni Volt di tensione fornito) ed essendo alimentato con una tensione di 7.4V é in grado di ruotare a 13.320 RPM, garantendo ottime prestazioni per veicoli RC da corsa. In questo contesto applicativo, tali velocità non sono mai state raggiunte, poiché in ambiente urbano è fondamentale mantenere una velocità moderata per garantire la sicurezza propria e quella dell'ambiente circostante.

Il controllo del motore è stato effettuato fornendo un segnale PWM (Pulse Width Modulation) calcolato mediante un controllore PID in catena chiusa. Per realizzare correttamente il controllo in catena chiusa, poiché non è possibile ricavare i dati di velocità di rotazione del rotore direttamente dal motore, è stato utilizzato un encoder rotativo, in grado di fornire la velocità di rotazione dell'asse del motore in tempo reale.

Anche in questo caso le informazioni sul componente sono state prese dal datasheet [11] rilasciato dalla casa madre.

2.3. Encoder

Per ottenere un feedback sul ciclo di controllo della trazione è stato utilizzato un encoder incrementale a quadratura AMT103, prodotto dalla CUI Devices (Figura 2.4) montato sull'asse del motore mediante un supporto appositamente modellato e stampato in 3D (Figura 2.5).

Figura 2.4.: Encoder AMT103

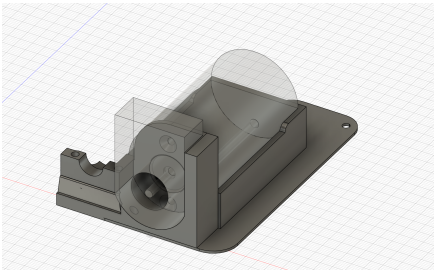


Figura 2.5.: Supporto

Un encoder è un dispositivo elettromeccanico di controllo utilizzato per generare un segnale digitale ad onda quadra, interpretato da un microcontrollore per rilevare la posizione o il suo spostamento, a seconda della tecnologia del sensore utilizzato Un encoder è un dispositivo di controllo elettromeccanico che viene utilizzato per ottenere un segnale digitale ad onda quadra che viene interpretato da un microcontrollore in modo da ottenere un dato di posizione o di scostamento di essa, in funzione della tecnologia adottata dal sensore.

Esistono difatti due tipologie di encoder, che si suddividono in:

1. Encoder assoluti: restituiscono la posizione o l'angolo assoluto dell'attuatore rispetto ad un sistema di riferimento fisso. Un'applicazione tipica di questo tipo di sensore è la misurazione degli angoli dei giunti di manipolatori.

2. Encoder incrementali: restituiscono la variazione della posizione o dell'angolo dell'attuatore nel tempo, senza però fornire informazioni sulla posizione o angolo assoluti. Sono molto utili per calcolare la velocità di rotazione degli assi dei motori.

Come già menzionato, l'encoder impiegato è di tipo incrementale, specificatamente un encoder a quadratura a doppio canale. Questo sensore rileva la variazione della posizione angolare nel tempo, indipendentemente dal senso di rotazione, ma non fornisce la posizione assoluta dell'asse del motore, bensì solo il suo spostamento.

Il principio di funzionamento di questo sensore è molto semplice e non varia, a prescindere dalla tecnologia utilizzata per la misurazione. Nel caso più semplice, ovvero quello degli encoder incrementali a fotocellula o fotodiodo, si hanno due componenti:

- Disco forato: questo disco, che presenta fori disposti radialmente a dividere l'angolo giro in sezioni di egual ampiezza, viene fissato all'asse del rotore in modo da ottenere una sincronia rotazionale dei due corpi;

- Sorgente luminosa e foto-transistor: questi due dispositivi, disposti ai lati del disco forato in modo che le sorgenti luminose (LED) illuminino i rispettivi foto-transistor, permettono di ottenere segnali logici alti e bassi in funzione

della posizione del disco rispetto ad essi. Difatti in corrispondenza dei fori, la luce prodotta dalla sorgente viene recepita dal foto-transistor, che attivandosi lascia passare una tensione corrispondente al valore logico alto, mentre il foto-transistor si disattiva nel momento in cui la sorgente viene oscurata dal disco, restituendo un valore di tensione corrispondente al valore logico basso.

Nel caso degli encoder a quadratura, sono presenti due canali (A e B) a ciascuno dei quali è associata una coppia LED/foto-transistor. Questi due canali sono sfasati di 90°elettrici e questo sfasamento permette di ottenere facilmente le informazioni di velocitá e di direzione di rotazione. Difatti la frequenza del treno di impulsi del canale A permette di calcolare la velocità del rotore, mentre lo sfasamento tra il segnale del canale A e quello del canale B permette di ottenere la direzione del moto. Se il segnale fornito dal canale A precede quello fornito dal canale B, il motore sta girando in un verso; Se, invece, il segnale generato dal canale B precede quello generato dal canale A è facilmente intuibile che l'asse del motore sta girando nel verso opposto. Uno schema riassuntivo del funzionamento è riportato in Figura 2.6.

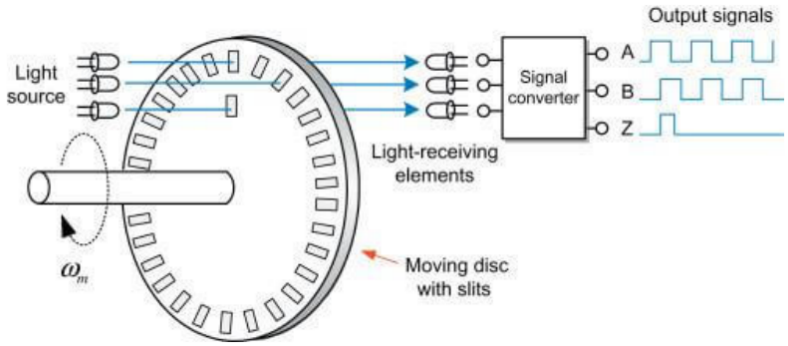


Figura 2.6.: Schema di principio di funzionamento di un encoder incrementale

L'encoder AMT103, utilizzato per la realizzazione del controllo della trazione, presenta il seguente pinout a 5 pin:

- B: pin corrispondente al canale B;
- 5V: pin di alimentazione del sensore;
- A: pin corrispondente al canale A;
- X: pin corrispondente al canale Z, che fornisce la posizione assoluta del rotore, ma inutilizzato nel contesto applicativo;
- G: pin corrispondente alla messa a terra.

L'encoder utilizzato, grazie ad un'alta risoluzione, permette di calcolare con grande accuratezzaa anche piccoli spostamenti longitudinali del veicolo. Questa caratteristica

è fondamentale per l'esecuzione di manovre di precisione quali il parcheggio a basse velocità.

Le informazioni sul componente sono state prese dal datasheet [3] rilasciato dalla casa madre.

2.4. Servo motore

Per operare il meccanismo di sterzo del veicolo è stato impiegato un servo motore Reely RS610WP, utilizzato per spostamenti angolari precisi, essendo controllato in posizione.

Figura 2.7.: Servo motore Reely RS610WP.

Questo motore presenta una coppia di stallo di 7.08Kg/cm se alimentato con una tensione di 4.8V e una velocità di movimento pari a 0.37s per compiere una rotazione pari a 60°alimentato alla suddetta tensione. Presenta invece una coppia di stallo pari a 8.5Kg-cm e una velocità di rotazione pari a 0.3s-60°se alimentato con una tensione di 6.0V. Dato che i miglioramenti in termini di prestazioni sono marginali all'aumentare della tensione di alimentazione a discapito di consumi maggiorati e dunque di un'autonomia ridotta, il servo motore è stato alimentato con una tensione di 4.8V.

Questo motore presenta ottime caratteristiche, perfettamente adatte all'applicazione di interesse, garantendo un'ottima coppia ed un rapido tempo di risposta ai segnali di controllo forniti.

Questo servomotore presenta una tecnologia analogica ed è, quindi, dotato di un potenziometro in retroazione che garantisce un'operazione fluida. Il dispositivo è controllato mediante un segnale PWM fornito in ingresso, calcolato dagli algoritmi di controllo implementati sul microcontrollore.

Le informazioni sul componente sono state prese dal datasheet [15] rilasciato dalla casa madre.

2.5. IMU

L'Inertial Measurement Unit (IMU), è un cluster di sensori che fornisce dati telemetrici riguardanti l'accelerazione, la velocità angolare rispetto agli assi e l'orientamento del sensore nello spazio tridimensionale.

Figura 2.8.: IMU BNO055

Il sensore utilizzato per la realizzazione del veicolo è un sensore BOSCH BNO055, che rientra nella categoria delle IMU NDOF (Nine Degrees Of Freedom) e come suggerito dall'acronimo fornisce dati riguardanti nove gradi di libertà:

- Accelerazione lungo gli assi cardinali, prendendo come sistema di riferimento il sensore: questo dato viene fornito dall'accelerometro integrato;
- Velocità angolare di rotazione rispetto agli assi cardinali, con sistema di riferimento il sensore: questo dato viene fornito dal giroscopio integrato;
- Orientamento assoluto rispetto ai punti cardinali, ottenuto mediante la misurazione dell'intensità del campo magnetico terrestre: questo dato viene fornito dal magnetometro integrato.

Nonostante i dati misurati siano 9 in totale, il giroscopio e il magnetometro forniscono entrambi la misura dell'orientamento del sensore. Questa ridondanza è giustificata dai vantaggi e dagli svantaggi di ciascuna delle due misure, in quanto i dati forniti dal giroscopio sono più precisi per piccoli movimenti ma sono meno stabili a livello numerico in quanto soggetti ad esplosioni di errore nel lungo termine, mentre i dati forniti dal magnetometro sono meno precisi ma più stabili.

Nel caso dell'IMU BNO055, i dati vengono elaborati automaticamente da un microcontrollore presente sulla scheda, che mediante algoritmi di sensor fusion (e.g. Filtro di Kalman) forniscono addizionali dati telemetrici quali angoli di Eulero e versioni più stabili dei dati grezzi misurati dai sensori.

I dati sono successivamente salvati nei registri del sensore e sono accessibili mediante protocolli di comunicazione quali I^2C, UART e SPI. Nel contesto applicativo è stata utilizzata una comunicazione tramite protocollo I^2C.

Tra punti di forza del sensore BNO055 c'è la capacità di calibrarsi automaticamente a seguito dell'accensione e la stabilità numerica dei dati forniti a seguito del pre-processamento di essi effettuato dagli algoritmi on-board, entrambe caratteristiche fondamentali per una corretta esecuzione delle manovre del veicolo.

Questo sensore si è rivelato essere fondamentale per il controllo laterale del veicolo in quanto gli algoritmi di controllo dello sterzo si basano interamente sulla variazione dell'angolo di imbardata (yaw rate) misurata dall'IMU. La trattazione di questo aspetto è riservata ai capitoli successivi di questa tesi.

Le informazioni sul componente sono state prese dal datasheet [4] rilasciato dalla casa madre.

2.6. Batteria e scheda di distribuzione

La batteria fornita con il kit è la *Conrad energy Scale model battery pack (LiPo), 7.4V, 5500mAh, 2 Cells, 20C Softcase, XT90.*

Figura 2.9.: Conrad energy Scale model battery pack

Con una tensione nominale di 7.4V e una capacità di 5500 mAh, è una batteria ai polimeri di litio (LiPo) in grado di erogare una corrente di 5.5A per la durata di un'ora, garantendo una corretta alimentazione al sistema avendo una discreta autonomia. La batteria è realizzata con due celle in serie, al fine di raggiungere la tensione nominale.

Per quanto riguarda la scarica massima, la nomenclatura 20C indica che questo tipo di batteria è in grado di scaricare una corrente pari a 20 volte la specifica nominale (5500mAh), raggiungendo quindi un picco di 110A di corrente continua.

Questo modello usa un connettore XT90, ampiamente utilizzato nel modellismo e capace di sopportare picchi di corrente pari a 90A per 10 secondi. Questa tipologia di connettore è utilizzata per la sicurezza della connessione e la prevenzione di corti circuiti.

La batteria è a sua volta connessa ad una scheda di distribuzione proprietaria appositamente realizzata dalla BOSCH per questa competizione, alla quale è direttamente saldato il motore. Inoltre, grazie a buck e boost converter integrati fornisce ai vari componenti del sistema valori di tensione e corrente adeguati per un corretto funzionamento.

Le informazioni sul componente sono state prese dal datasheet [9] rilasciato dalla casa madre.

2.7. STM32 Nucleo F401RE

La scheda Nucleo F401RE, prodotta da STMicroelectronics, appartiene alla serie F4 della famiglia STM32 ed è basata sul microcontrollore Arm®Cortex®-M4 a 32 Bit, con un clock d 84Mhz. Il microcontrollore dispone di una memoria flash da 512KB e 96KB di RAM: questi due elementi, permettono l'esecuzione di ampie porzioni di codice, grazie ad una grande capacità di archiviazione. La scheda offre inoltre una varietà di interfacce di comunicazione, tra cui numerose porte GPIO (General Purpose Input/Output), I^2C (Inter-Integrated Circuit), SPI (Serial Peripheral Interface) e UART (Universal Asynchronous Reciever/Transmitter) in grado di comunicare con sensori, altri dispositivi e fornire segnali di controllo per attuatori di vario tipo.

In particolare la scheda offre:

- 10 TIMER da 16 e 32 bit con frequenze fino a 84MHz;
- 16 GPIO per ingressi ed uscite analogiche e digitali;
- 3 USART che operano a 10.5 Mbit/s che abilitano la comunicazione seriale

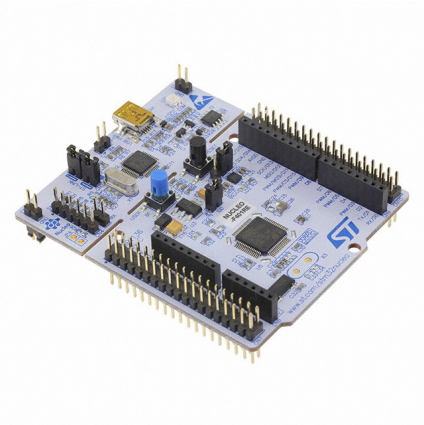


Figura 2.10.: STM32 Nucleo F401RE

In figura 2.10 è mostrata la realizzazione fisica del microcontrollore, che oltre ai pin GPIO è dotata di:

- pulsante nero per il reset della scheda;
- pulsante blu programmabile;
- LED1: indica lo stato della connessione USB;
- LED2: LED programmabile;
- LED3: indica lo stato di alimentazione;

Il microcontrollore può essere alimentato sia mediante porta USB che mediante alimentazione esterna. Per la programmazione ed il debug del microcontrollore sono presenti un debugger/programmatore integrato ST-Link 2.1, che permette un flash rapido del codice mediante la porta micro USB presente sulla scheda. La scheda può facilmente essere programmata mediante l'ambiente di sviluppo STM32CubeIDE, appositamente ideato per il coding sulle schede della famiglia STM32.

Il pinout della scheda è illustrato in Figura 2.11 e comprende 64 pin suddivisi in:

1. Pinout Arduino, con connettori Dupont di tipo femmina:
   - CN6 e CN7 per ingressi ed uscite analogiche;
   - CN5 e CN9 per ingressi ed uscite digitali;
2. Pinout ST Morpho, con connettori Dupont di tipo maschio (CN7 e CN10).

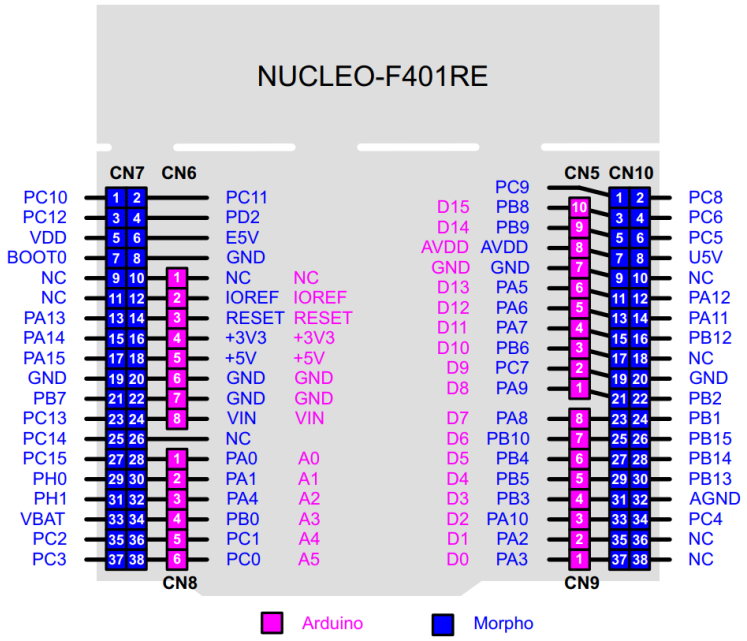


Figura 2.11.: Pinout STM32 Nucleo F401RE

Nel contesto applicativo il microcontrollore è stato utilizzato per i seguenti scopi:

- Ricezione dei setpoint di controllo forniti dalla Raspberry Pi 4 mediante comunicazione seriale;
- Attuazione del motore di trazione;
- Attuazione del servomotore di sterzo e controllo laterale del veicolo;
- Lettura dei dati telemetrici forniti dall'IMU ed invio di essi agli algoritmi di autolocalizzazione presenti su scheda Raspberry Pi 4 mediante comunicazione seriale.

Si può dunque evincere il ruolo fondamentale del microcontrollore nel contesto applicativo e la necessità di disporre di hardware ad alte prestazioni, motivo per cui il microcontrollore scelto è risultato adeguato al carico imposto.

Le informazioni sul componente sono state prese dal datasheet [16] rilasciato dalla casa madre.

2.8. Raspberry Pi 4

La Raspberry Pi 4 (RPi4) è un computer a scheda singola (SBC) sviluppato dalla Raspberry Pi Foundation, progettato per essere un dispositivo economico e versatile, adatto a molteplici applicazioni, dalla didattica all'uso in ambiti professionali. La Raspberry Pi 4 Model B, rilasciata nel 2019, rappresenta un significativo miglioramento rispetto alle versioni precedenti, grazie a specifiche hardware più avanzate che le permettono di gestire carichi di lavoro più complessi.

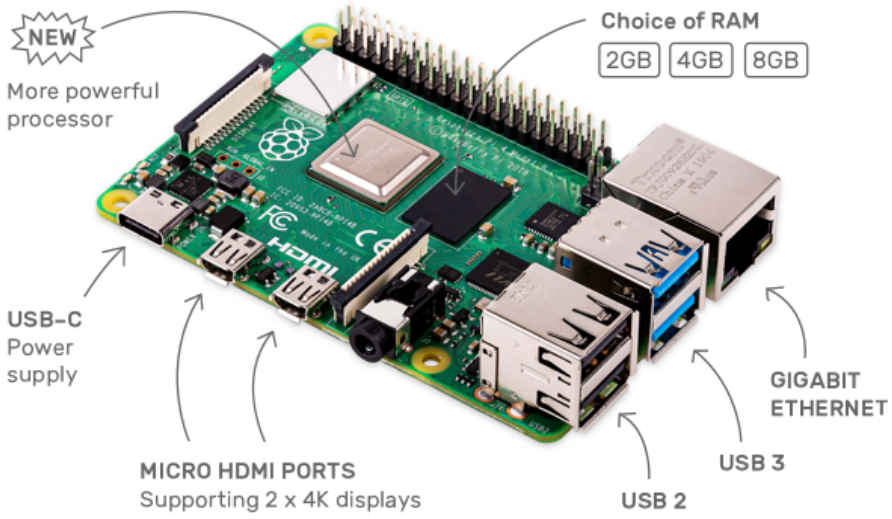


Figura 2.12.: Raspberry Pi 4

Le caratteristiche tecniche principali della RPi4 sono:

- Processore: broadcom BCM2711, un quad-core ARM Cortex-A72 a 64 bit con frequenza di clock a 1.5 GHz, che garantisce prestazioni migliorate rispetto alle versioni precedenti;
- Memoria RAM: disponibile in varianti da 2 GB, 4 GB o 8 GB di RAM LPDDR4, consentendo una maggiore flessibilità a seconda delle esigenze di utilizzo. Nel contesto applicativo è stato utilizzato il modello con 8GB di RAM, per limitare il più possibile colli di bottiglia dovuti a carichi computazionali intensi;
- Connettività: include due porte USB 3.0 e due porte USB 2.0, una porta Gigabit Ethernet, Wi-Fi dual-band 802.11ac, Bluetooth 5.0 e supporto per interfacce GPIO (General Purpose Input/Output) che consentono l'interazione con componenti esterni;
- Supporto video: è dotata di due porte micro-HDMI con supporto per la risoluzione 4K su due monitor contemporaneamente, e la GPU VideoCore VI permette la riproduzione fluida di contenuti multimediali in alta definizione;
- Archiviazione: l'archiviazione avviene tramite una scheda microSD, ma supporta anche l'avvio da dispositivi USB come SSD, migliorando le prestazioni in lettura e scrittura rispetto alle schede microSD. La possibilità di utilizzare schede microSD è molto utile in quanto è stato possibile implementare diverse iterazioni del codice di controllo in maniera parallela, passando da una versione all'altra semplicemente cambiando l'unità di archiviazione;
- Alimentazione: necessita di un alimentatore USB-C con uscita da 5V e 3A per un funzionamento ottimale.

Grazie alla sua compatibilità software, la Raspberry Pi 4 supporta una vasta gamma di sistemi operativi, tra cui Raspberry Pi OS (basato su Debian), Ubuntu, e altre distribuzioni Linux, oltre a sistemi operativi specifici per progetti IoT o server. Nel contesto applicativo è stato scelto l'utilizzo di ROS2 (Robot Operating System 2) in quanto offre ottime prestazioni nel controllo real time ed è dunque adeguato allo sviluppo di progetti nel settore della robotica intelligente.

Inoltre, la scelta di utilizzare la Raspberry Pi 4 è giustificata dal fatto che questo SBC offre un rapporto prestazioni/prezzo molto alto, un fattore fondamentale nell'industria dell'automotive.

Difatti, il ruolo ricoperto dalla RPi4 è quello del "Cervello" della macchina, in quanto ad essa sono affidati i compiti di:

- Lanekeeping: mantenimento del veicolo in corsia, mediante appositi algoritmi di computer vision;
- Auto-localizzazione: calcolo della posizione della macchina in real time mediante la fusione di dati telemetrici forniti da più sensori;

- Path planning: calcolo del percorso ottimale tra due punti della mappa e conseguente navigazione autonoma di incroci e rotatorie;
- Riconoscimento di cartelli stradali e adeguamento alle regole del codice della strada;
- Riconoscimento pedoni e fermata di emergenza;
- Riconoscimento di veicoli in carreggiata e cruise control adattivo;
- Esecuzione di manovre di cambio corsia e sorpasso;
- Auto parking: manovra di parcheggio autonomo;

Il risultato degli algoritmi decisionali sviluppati su RPi vengono tradotti in setpoint di velocità e curvatura, che vengono inviati mediante comunicazione seriale al microcontrollore. Successivamente, quest'ultimo ai occupa anche della loro attuazione mediante appositi algoritmi di controllo.

Le informazioni sul componente sono state prese dal datasheet [12] rilasciato dalla casa madre.

2.9. Coral TPU

La Google Coral TPU (Tensor Processing Unit) è un acceleratore hardware progettato specificamente per l'esecuzione efficiente di modelli di intelligenza artificiale, in particolare di reti neurali profonde, direttamente ai margini della rete (edge computing). Sviluppata da Google, fa parte della linea di prodotti Coral, pensati per portare capacità avanzate di machine learning su dispositivi embedded o a bassa potenza.

Figura 2.13.: Google Coral TPU

La Google Coral TPU presenta le seguenti caratteristiche principali:

- Architettura: la Coral TPU si basa sull'architettura Edge TPU, progettata per ottimizzare l'esecuzione di operazioni su tensori, tipiche dei modelli di machine learning, offrendo un basso consumo energetico e tempi di inferenza estremamente rapidi;

- Prestazioni: la Coral TPU è capace di eseguire fino a 4 trilioni di operazioni al secondo (TOPS), consumando solo 2 watt di potenza, rendendola ideale per dispositivi con limitate risorse energetiche o applicazioni che richiedono inferenza in tempo reale, ad esempio un veicolo autonomo.

- Supporto per TensorFlow Lite: è ottimizzata per eseguire modelli di machine learning compilati in TensorFlow Lite, un framework leggero sviluppato da Google per implementazioni su dispositivi mobili e embedded. Questa ottimizzazione consente di caricare modelli complessi preaddestrati e di eseguire inferenze con latenza minima.

- Efficienza energetica: grazie alla sua architettura specializzata, la Coral TPU offre prestazioni di inferenza comparabili a soluzioni cloud, ma con consumi ridotti, facilitando l'implementazione di applicazioni AI on the edge dove è fondamentale minimizzare l'uso di energia e ridurre la latenza.

La Google Coral TPU è stata aggiunta al kit di base fornito per la competizione in quanto fondamentale per ridurre il carico computazionale della Raspberry Pi 4, sulla quale sono stati implementati diversi algoritmi di visione e reti neurali (e.g. riconoscimento di cartelli stradali). Per il motivo sopracitato la Coral TPU non è illustrata in figura 2.1, ma è stata collegata direttamente alla Raspberry Pi 4 mediante un cavo USB-A.

Le informazioni sul componente sono state prese dal datasheet [5] rilasciato dalla casa madre.

2.10. Telecamera

La telecamera utilizzata per la visione è la Pi Camera V2.1, una fotocamera ufficiale sviluppata dalla Raspberry Pi Foundation, progettata per essere utilizzata con i computer a scheda singola Raspberry Pi. Questa fotocamera rappresenta un aggiornamento rispetto alla prima versione della Pi Camera, offrendo miglioramenti significativi in termini di risoluzione e qualità delle immagini, pur mantenendo una grande versatilità e compatibilità.

Le caratteristiche tecniche principali di questa telecamera sono:

- Sensore Sony IMX219: offre una risoluzione di 8 megapixel (3280 x 2464 pixel), consentendo di scattare immagini ad alta risoluzione e registrare video di qualità;

Figura 2.14.: Pi Camera V2.1

- Capacità video: supporta la registrazione video a diverse risoluzioni, tra cui 1080p a 30 fps, 720p a 60 fps e 480p a 90 fps, rendendola adatta sia per applicazioni di monitoraggio video che per progetti di visione artificiale.
- Dimensioni ridotte: il modulo fotocamera è compatto, con una dimensione di circa 25 mm x 23 mm x 9 mm, il che lo rende ideale per progetti che richiedono un ingombro minimo.
- Connessione: si collega alla Raspberry Pi tramite l'interfaccia CSI (Camera Serial Interface), che garantisce una trasmissione dati ad alta velocità e bassa latenza. L'installazione è semplice, grazie al cavo a nastro fornito con la fotocamera.
- Compatibilità: la Pi Camera V2.1 è compatibile con tutte le versioni della Raspberry Pi, ed è supportata da una vasta gamma di librerie software, come Picamera e OpenCV, per sviluppare applicazioni di visione artificiale e analisi delle immagini. La compatibilità con OpenCV risulta essere fondamentale per lo sviluppo degli algoritmi di visione utilizzati per garantire il livello di autonomia richiesta al veicolo;

La fotocamera è un componente fondamentale per il corretto funzionamento del veicolo, in quanto da essa dipendono gli algoritmi di visione che permettono il mantenimento della corsia (lanekeeping) ed è utilizzata come sorgente per la rete neurale che ha il compito di riconoscere ed identificare i cartelli stradali e gli eventuali pedoni che ostacolano la marcia.

Le informazioni sul componente sono state prese dal datasheet [13] rilasciato dalla casa madre.

2.11. LiDAR

Un LiDAR planare è un sensore che utilizza la tecnologia LiDAR (Light Detection and Ranging) per acquisire dati tridimensionali mediante l'emissione di impulsi laser e la misurazione del tempo impiegato dalla luce a riflettersi sugli oggetti e tornare al sensore. A differenza dei LiDAR tridimensionali, il LiDAR planare raccoglie dati in un solo piano bidimensionale, generando una mappa 2D dell'ambiente circostante. Questo tipo di sensore viene comunemente utilizzato in applicazioni come la navigazione robotica, la mappatura di interni e la rilevazione di ostacoli in tempo reale.

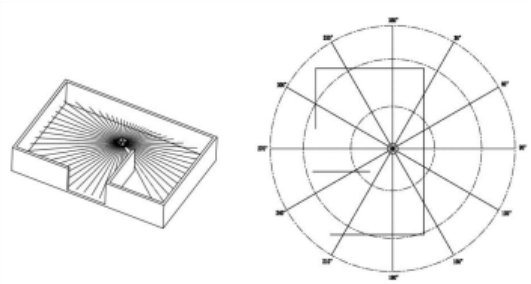


Figura 2.15.: Scenario e output fornito da un LiDAR planare

Al veicolo è stato aggiunto un LiDAR planare LD06 prodotto da LDROBOT, un sensore compatto con un FOV (Field Of View) di 360°. Questo sensore è particolarmente interessante poiché offre ottime prestazioni in un pacchetto compatto e ad un prezzo competitivo, offrendo un raggio di scansione fino a 12 metri, ad una frequenza di 4500Hz. Grazie ad un errore di misura di solo ± 2cm, questo sensore è in grado di mappare un'area vasta con ottima accuratezza e velocità: questa caratteristica lo rende perfetto per permettere al veicolo di reagire all'ambiente circostante con prontezza.

Figura 2.16.: LiDAR LD06

Il dispositivo comunica mediante protocollo UART o USB, permettendo un'ottima connettività con vari dispositivi. Nel contesto applicativo, il sensore è stato collegato alla Raspberry Pi 4 mediante USB e grazie ai driver forniti dal produttore è stato facile integarlo con il sistema operativo ROS2.

Le informazioni sul componente sono state prese dal datasheet [8] rilasciato dalla casa madre.

2.12. Sistema di localizzazione

Con l'edizione 2024 della BFMC è stato aggiunto un sistema di localizzazione in tempo reale (Real Time Location System o RTLS), realizzato mediante una tecnologia a banda ultra larga (Ultra Wide Band o UWB). L'hardware impiegato è un kit di sviluppo QORVO MDEK1001, che fa uso di 12 schede di sviluppo DWM1001-DEV, che possono essere utilizzate in due configurazioni:

- Ancora: scheda montata su un supporto fisso, funge da ricetrasmettitore e collabora con altre ancore per ottenere la posizione di uno o più "tag" mobili mediante algoritmi di triangolazione;
- Tag: scheda montata su un supporto mobile (e.g. veicolo), comunica con le ancore e riceve i propri dati di posizione da esse calcolati.

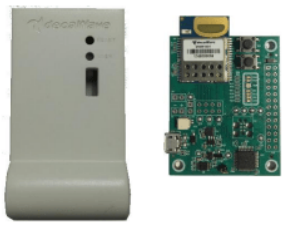


Figura 2.17.: Scheda di sviluppo MDEK1001

In Figura 2.18 è riportata un'illustrazione del funzionamento tipo del kit utilizzato, con una configurazione a 4 ancore e 1 tag.

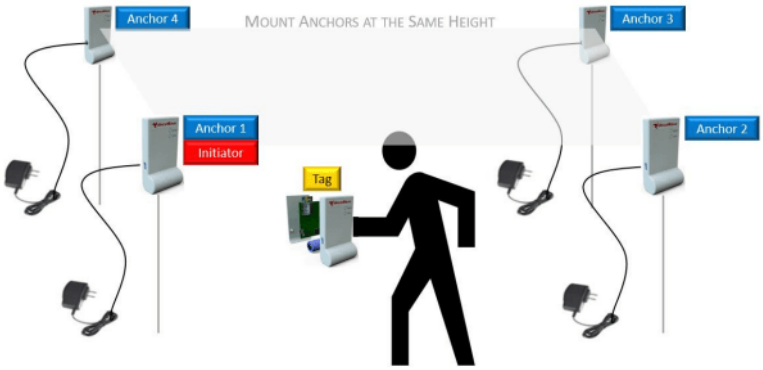


Figura 2.18.: Caso d'uso tipico

Nel contesto applicativo, il sistema di localizzazione comunica direttamente i dati di posizione alla Raspberry Pi ad una frequenza di 50Hz mediante USB. Questi dati vengono poi fusi a quelli forniti dall'IMU per ottenere una migliore stima della posizione del veicolo. Questo aspetto è fondamentale poiché una misura accurata della posizione è necessaria per una corretta navigazione lungo il percorso.

Le informazioni sul componente sono state prese dal datasheet [10] rilasciato dalla casa madre.

2.13. Connessioni elettriche

Di seguito è riportato lo schema delle connessioni elettriche tra i componenti forniti nel kit base. Nonostante le aggiunte, lo schema risulta invariato in quanto i sensori aggiuntivi sono stati collegati direttamente al microcontrollore e alla Raspberry Pi, sfruttando i pin di alimentazione presenti sulle schede. In Figura 2.17 è riportato il diagramma delle connessioni, mentre in Figura 2.18 è mostrata una legenda delle connessioni.

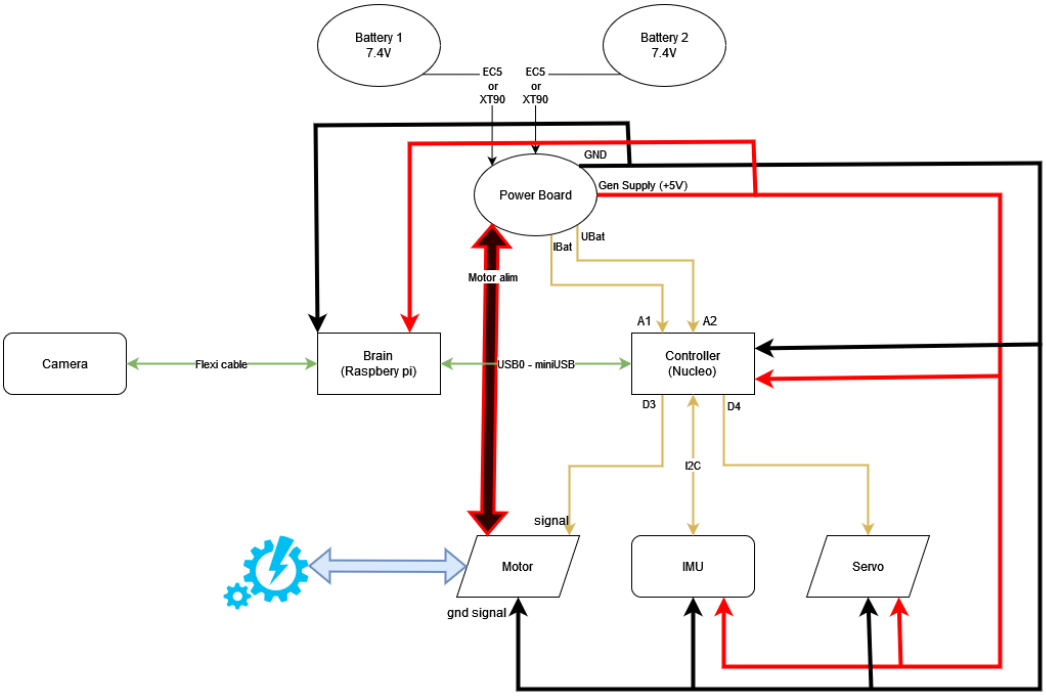


Figura 2.19.: Schema delle connessioni

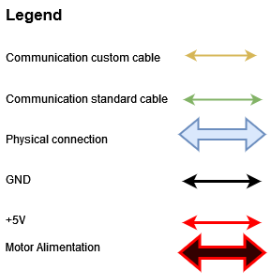


Figura 2.20.: Legenda

In appendice A é riportato uno schema elettrico più dettagliato che è stato realizzato al termine della competizione, evidenziando i pinout dei diversi componenti e le connessioni tra di essi.

Capitolo 3.

Architettura del sistema

3.1. Overview dell'architettura

Il sistema che è stato concepito per affrontare la challenge è un connubio di elementi software ed hardware, che lavorano in sintonia per fornire al veicolo propriocezione, percezione dell'ambiente circostante ed una conseguente capacità di reagire agli stimoli esterni nel modo più adeguato possibile.

L'hardware descritto nel capitolo precedente può essere raggruppato in funzione della descrizione appena fornita. Possiamo quindi distinguere tre gruppi principali di componenti:

1. Sensing: questo gruppo permette al veicolo di ottenere una percezione del proprio stato in relazione all'ambiente circostante. Ne fanno parte i seguenti componenti:
    - QORVO UWB RTLS: sistema di localizzazione, fornisce la posizione del veicolo in relazione alla mappa, consentendo ad esso di pianificare istante per istante le manovre da eseguire;
    - IMU: assieme al sistema di localizzazione, questo sensore permette di ottenere informazioni più dettagliate riguardo allo stato del veicolo, quali accelerazioni, rotazioni e orientamento assoluto rispetto agli assi cardinali. Mediante algoritmi di sensor fusion, questo sensore permette di correggere eventuali errori commessi dal sistema di localizzazione, spesso soggetto ad imprecisioni;
    - Encoder: questo sensore permette di ottenere la velocità esatta del veicolo, ulteriore dato utile all'algoritmo di sensor fusion per correggere la stima della posizione;
    - Telecamera: mediante algoritmi di visione e di lanekeeping, permette di effettuare correzioni della traiettoria del veicolo in tempo reale per mantenerlo in corsia;
    - LiDAR: fornisce una misura dello spazio circostante al veicolo, individuando ostacoli e calcolandone la distanza. Questo sensore svolge un ruolo fondamentale nel corretto funzionamento degli algoritmi di cruise control adattivo e di parcheggio automatico.

2. Decision making: questo gruppo permette al veicolo di effettuare decisioni in maniera autonoma in funzione degli stimoli ricevuti dall'ambiente esterno. I componenti che garantiscono questo livello di autonomia sono:
    - Raspberry Pi 4: rappresenta il "cervello" del veicolo, in quanto su questo SBC sono implementati tutti gli algoritmi decisionali e le reti neurali che permettono al veicolo di navigare in maniera autonoma rispettando le leggi del codice stradale. Grazie all'efficienza del sistema operativo real time ROS2, gli algoritmi sviluppati operano ad alte frequenze garantendo ottimi tempi di risposta a situazioni impreviste;
    - Pi Camera V2.1: componente fondamentale per il corretto funzionamento del veicolo, utilizza algoritmi di visione per il mantenimento della corsia (lanekeeping) e grazie all'implementazione di una rete neurale ad hoc individua ed interpreta cartelli stradali, pedoni e semafori.
3. Actuation: questo gruppo permette al veicolo di navigare l'ambiente circostante, in funzione dei segnali di controllo forniti dal gruppo di decision making. I componenti che permettono al veicolo di spostarsi e navigare l'ambiente sono:
    - STM32 Nucleo F401RE: microcontrollore che ha il fondamentale scopo di interpretare i messaggi comunicati dalla Raspberry Pi 4 trasformandoli, mediante algoritmi di controllo, in segnali di controllo elettrici adibiti all'attuazione dei motori. Un'altra funzione fondamentale che svolge il microcontrollore è quella di raccogliere i dati telemetrici forniti dai vari sensori a bordo ad esso collegati, inviandoli alla RPi per successive elaborazioni;
    - Motore brushless: attuatore adibito allo spostamento longitudinale del veicolo, controllato mediante PID in catena chiusa. Permette al veicolo di navigare alla velocità imposta dagli algoritmi di decision making forniti dalla RPi in maniera fluida e stabile;
    - Servo motore: attuatore adibito allo spostamento laterale del veicolo, anch'esso controllato in catena chiusa mediante controllore PID. Permette al veicolo di sterzare ed eseguire manovre quali navigazione di intersezioni e rotatorie, cambi di corsia, sorpassi e parcheggi.

3.2. Visione top-down dell'architettura e flusso dei dati

Ogni azione del veicolo è il risultato di una ben precisa pipeline, così definita:

1. Sensing dell'ambiente circostante e raccolta dati, mediante l'utilizzo dei sensori a bordo e dei dati forniti dal RTLS;
2. Processing dei dati a disposizione e definizione dello stato corrente del veicolo, mediante appositi algoritmi di sensor fusion e di visione;
3. Decision making e motion planning mediante algoritmi decisionali che permettono, quindi, di definire quindi lo stato futuro del veicolo;
4. Controllo del veicolo, mediante appositi algoritmi di controllo sviluppati su microcontrollore;
5. Attuazione dei motori di trazione e sterzo mediante generazione di appositi segnali di controllo, con conseguente movimento del veicolo;

Questa pipeline viene eseguita ciclicamente ad alte frequenze, in modo da permettere manovre rapide e tempi di risposta brevi, specifiche fondamentali per il controllo in tempo reale del veicolo.

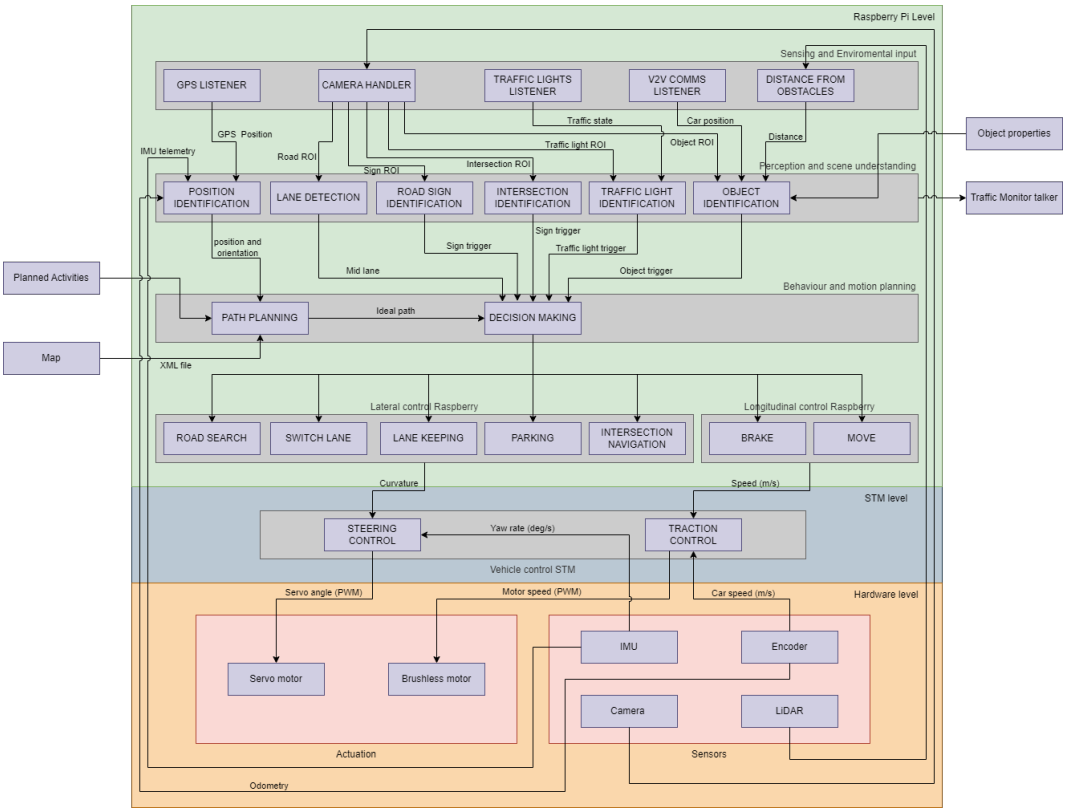


Figura 3.1.: Architettura e flusso dei dati

Dalla Figura 3.1 emerge chiaramente un flusso di dati che, una volta acquisiti dai sensori di bordo, attraversano vari livelli di astrazione. Si parte dagli algoritmi di visione e interpretazione dell'ambiente, insieme al livello decisionale e di pianificazione delle azioni, per arrivare ad un livello inferiore, dove vengono elaborati i valori di velocità del motore e gli angoli di sterzo, fini a giungere ad un livello hardware, rappresentato da segnali elettrici di controllo (e.g. PWM).

3.3. Suddivisione del carico computazionale

A causa della natura dell'architettura descritta nella sezione 3.2, è stato deciso di suddividere in modo netto le responsabilità affidate a ciascuna delle due schede.

La Raspberry Pi, che è decisamente più potente del microcontrollore STM, funge da "cervello" e ad essa sono affidate le funzioni di interpretazione dell'ambiente circostante e pianificazione delle azioni. Al microcontrollore è invece affidato il compito della gestione dei sensori a basso livello, quali IMU ed encoder; invio dei dati telemetrici alla RPi per il processamento; interpretazione dei messaggi di controllo forniti dalla RPi e controllo degli attuatori mediante appositi algoritmi.

La comunicazione tra la Raspberry Pi e il microcontrollore STM in entrambi i versi è fondamentale per il corretto funzionamento dell'architettura, motivo per cui è stata implementata una comunicazione seriale a messaggi a formato fisso ed a frequenza costante, sviluppando varie misure di prevenzione, controllo di errori di interpretazione e correzione di malfunzionamenti.

Questa distinzione tra "alto" e "basso" livello di controllo si è rivelata essere molto efficace per un corretto controllo del veicolo.

In questo elaborato di tesi non verrà però esplorata l'implementazione dell'intera architettura, bensì solo la porzione riguardante il controllo laterale a livello della Raspberry Pi e gli algoritmi di controllo sviluppati su microcontrollore per attuare il motore brushless ed il servo motore in maniera corretta.

Parte III.

Cenni teorici e soluzione analitica

Capitolo 4.

Geometria e cinematica del veicolo

4.1. Introduzione

Per comprendere la dinamica laterale di un veicolo a 4 ruote è necessario comprendere il funzionamento del meccanismo di sterzo. Difatti, la geometria dello sterzo svolge un ruolo fondamentale nella definizione della traiettoria del veicolo e le varie geometrie che può assumere hanno differenti prestazioni in funzione dell'applicazione.

In questo capitolo verranno inizialmente esplorate le diverse misure che definiscono un veicolo, e successivamente si passerà ad analizzare le possibili geometrie di sterzo e il loro funzionamento, evidenziandone punti di forza ed eventuali criticità.

A seguire verrà discussa la relazione tra la geometria di sterzo e la traiettoria descritta in moto dal veicolo utilizzato in competizione.

Successivamente, verrà esplorata la relazione tra la traiettoria del veicolo e lo yaw rate, ossia il parametro che è stato utilizzato per realizzare il controllo laterale del veicolo.

Per concludere verranno riportate le misure caratteristiche del veicolo utilizzato per la competizione, evidenziandone i limiti funzionali.

4.2. Geometria del veicolo

La geometria di un veicolo è definita da un insieme di misure fondamentali, che ne influenzano il design, la maneggevolezza e le prestazioni. Questi parametri sono essenziali per la sua dinamica, oltre che per la sicurezza in moto e il comfort di guida.

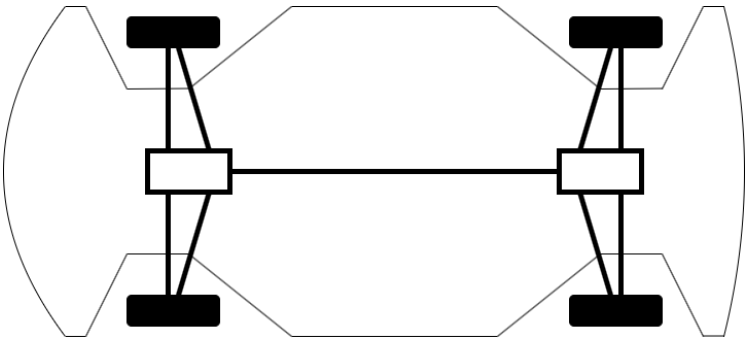


Figura 4.1.: Vista stilizzata dall'alto del veicolo.

Le misure fondamentali sono:

- **Passo**: distanza tra gli assi delle ruote anteriori e posteriori del veicolo. Influisce sulla stabilità di esso, in quanto un passo più lungo offre una guida più stabile, a discapito della manovrabilità. Un passo corto, d'altro canto, aumenta la manovrabilità del veicolo a discapito della stabilità di guida;

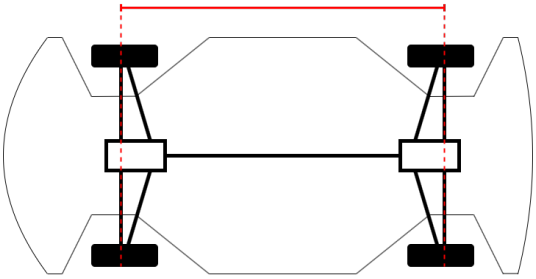


Figura 4.2.: Passo.

- **Carreggiata**: distanza tra le linee centrali delle ruote presenti sullo stesso asse, misurata sia anteriormente che posteriormente. Influisce sulla stabilità laterale e sulla capacità di sterzo. Difatti una carreggiata larga offre maggiore stabilità durante le curve;

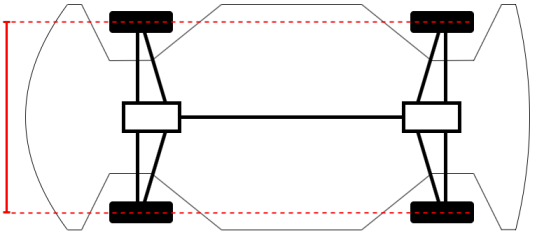


Figura 4.3.: Carreggiata.

- **Lunghezza totale**: lunghezza complessiva del veicolo, dalla parte più anteriore alla parte più posteriore (spesso dal paraurti anteriore a quello posteriore). Questa misura è particolarmente importante nella manovra di parcheggio, in quanto è necessario tenere conto degli ingombri dei paraurti anteriori e posteriori;

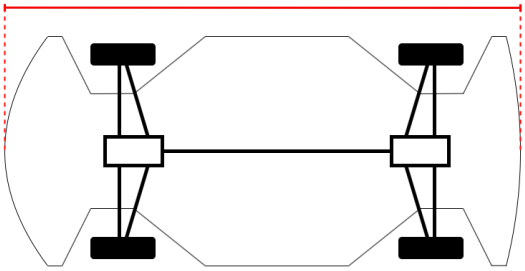


Figura 4.4.: Lunghezza totale.

- **Larghezza totale**: larghezza complessiva del veicolo da lato a lato, inclusi gli specchietti. Questa misura influisce sull'aerodinamica del veicolo;

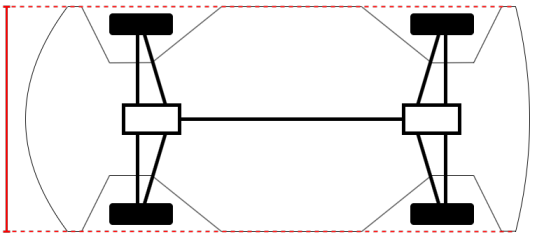


Figura 4.5.: Larghezza totale.

- **Distanza baricentro-asse**: distanza orizzontale tra il baricentro del veicolo e gli assi anteriore e posteriore. Influisce sulla distribuzione del peso, fattore fondamentale per la maneggevolezza, la frenata e la trazione;

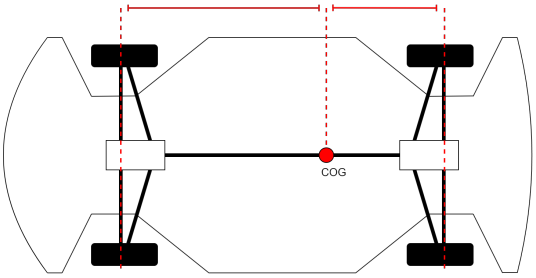


Figura 4.6.: Distanza del baricentro dagli assi.

- **Altezza totale**: distanza dal suolo al punto più alto del veicolo, spesso il tetto. Influisce sull'aerodinamica e sul baricentro;

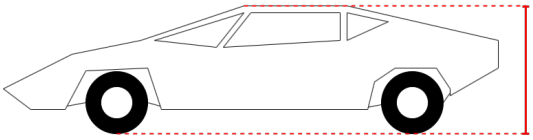


Figura 4.7.: Altezza totale.

- **Altezza da terra**: distanza tra il suolo e il punto più basso della parte inferiore del veicolo. Influisce sulla capacità del veicolo di evitare ostacoli;

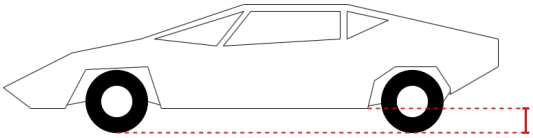


Figura 4.8.: Altezza da terra.

- **Sbocchi (anteriore e posteriore)**: distanza tra gli assi e i relativi paraurti. Influisce sugli angoli di attacco ed uscita, oltre che sull'aerodinamica del veicolo;

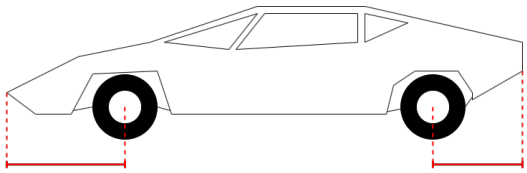


Figura 4.9.: Sbocco anteriore e posteriore.

- **Angoli di attacco ed uscita**: angolo massimo tra il suolo e il paraurti anteriore (angolo di attacco) o il paraurti posteriore (angolo di uscita) quando il veicolo si avvicina ad un tratto di strada pendente.

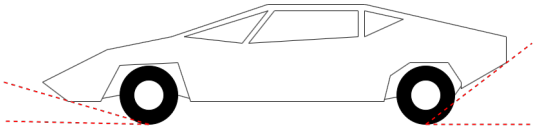


Figura 4.10.: Angolo di attacco (sinistra) e angolo di uscita (destra).

- **Altezza del baricentro**: altezza del baricentro del veicolo rispetto al suolo. Influisce su maneggevolezza, stabilità e probabilità di ribaltamento. In particolare, un baricentro più basso migliora la stabilità in curva;

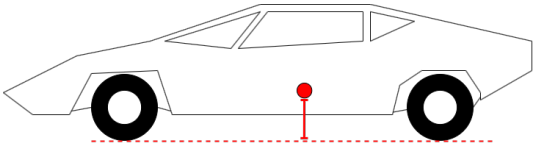


Figura 4.11.: Altezza del baricentro.

- **Geometria delle sospensioni**: la geometria delle sospensioni rappresenta uno dei fattori che incide maggiormente sulla precisione dello sterzo, nonché sulla dinamica della guida. Le misure che definiscono questa geometria sono tre e sono:

  1. **Angolo di Caster**: angolo tra l'asse verticale dello sterzo e l'asse verticale del veicolo, visto di lato;

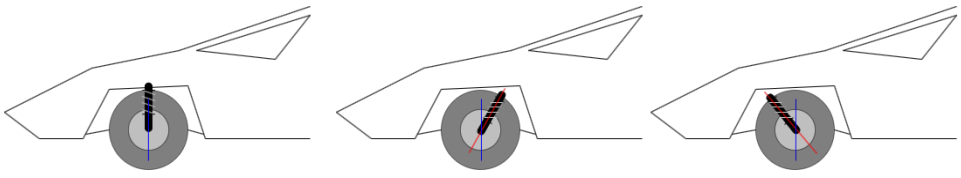


Figura 4.12.: Angolo di Caster neutro (sx), positivo (centro), negativo (dx).

  2. **Angolo di Camber**: inclinazione delle ruote verso l'interno o l'esterno rispetto all'asse verticale, vista frontalmente;

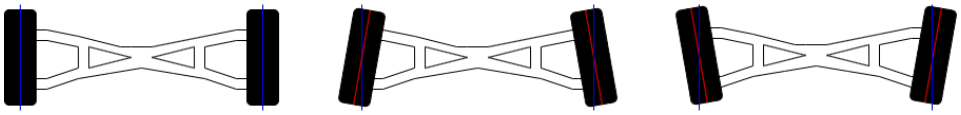


Figura 4.13.: Angolo di Camber neutro (sx), negativo (centro), positivo (dx).

3. **Toe**: angolo a cui le ruote puntano verso l'interno o l'esterno rispetto alla linea centrale del veicolo.

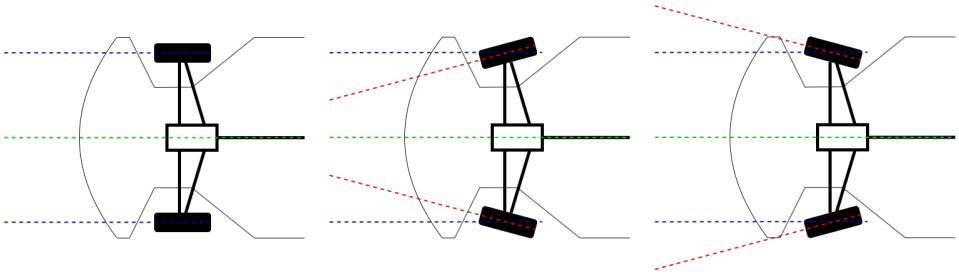


Figura 4.14.: Toe neutro (sx), Toe in (centro), Toe out (dx).

- **Raggio di sterzata**: raggio minimo di una curva che il veicolo può compiere, misurato dal centro della curva alla parte più esterna del veicolo. Questa misura è data dalla combinazione di passo, carreggiata e geometria di sterzo;

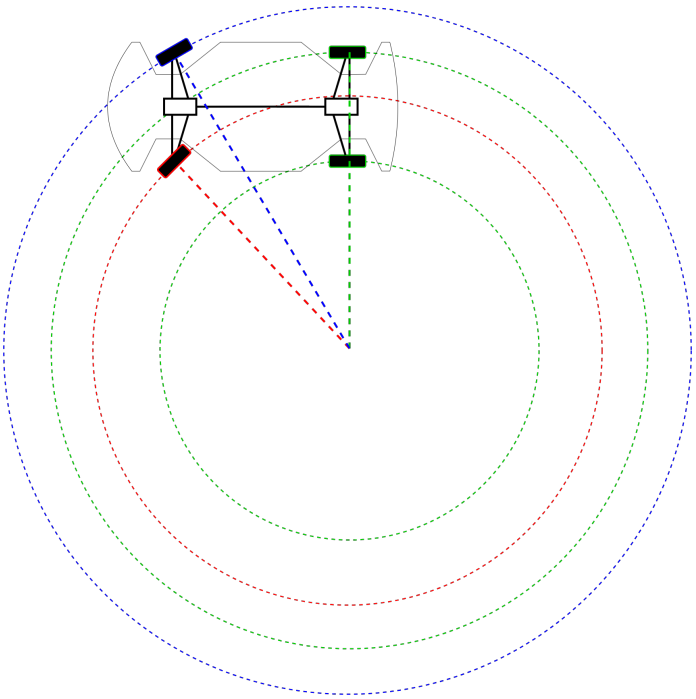


Figura 4.15.: Raggio minimo di sterzata (blu).

4.3. Geometria di sterzo

Varie applicazioni nel contesto automobilistico richiedono differenti implementazioni del meccanismo di sterzo. Nel corso della storia sono stati implementati diversi sistemi per ottimizzare le performance dei veicoli in funzione della loro applicazione.

Tra queste, le principali geometrie di sterzo che verranno esplorate in questa sezione sono:

- Geometria di sterzo parallelo;
- Geometria di sterzo Ackermann;
- Geometria di sterzo a 4 ruote (4WS).

Ciascuna di queste geometrie presenta pro e contro, descritti di seguito.

4.3.1. Geometria di sterzo parallelo

Questo tipo di geometria è quella concettualmente più semplice ed intuitiva, in quanto permette di sterzare entrambe le ruote anteriori con lo stesso angolo di sterzo. Questa condizione è garantita dai vincoli geometrici imposti dal meccanismo di sterzo, realizzato idealmente con 4 assi, due orizzontali e due verticali, giunti agli estremi con possibilità di rotazione sul piano su cui giacciono. Le ruote sono fissate al meccanismo di sterzo in modo tale da muoversi in modo solidale con gli assi verticali.

Una rappresentazione stilizzata della geometria di sterzo parallelo è mostrata in Figura 4.16.

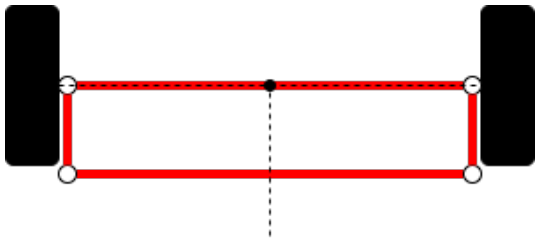


Figura 4.16.: Geometria di sterzo parallelo.

Per realizzare lo sterzo l'asse orizzontale anteriore, congiungente i centri delle ruote, rimane fisso mentre l'asse posteriore viene azionato da un meccanismo in modo da scorrere parallelamente a quello anteriore.

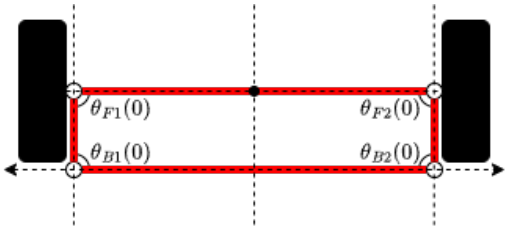


Figura 4.17.: Sterzo parallelo, angoli compresi tra gli assi.

A causa dei vincoli di rigidità imposti dalla struttura, se viene verificata la condizione di parallelismo tra gli assi orizzontali, si ottengono le seguenti relazioni:

$$\theta_{F1}(0) \equiv \theta_{B2}(0) \tag{4.1}$$
$$\theta_{F2}(0) \equiv \theta_{B1}(0) \tag{4.2}$$

Grazie a questa condizione è possibile stabilire una relazione di parallelismo anche tra gli assi verticali, e quindi tra le ruote del veicolo.

Infatti, osservando Figura 4.18, è possibile osservare come il meccanismo di sterzo assuma la forma di un parallelogramma nel caso in cui si voglia sterzare il veicolo.

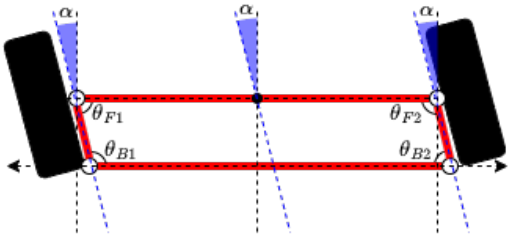


Figura 4.18.: Geometria del meccanismo di sterzo a seguito di uno sterzo di 15° verso sinistra.

Matematicamente si hanno le seguenti relazioni:

$$\theta_{F1} \equiv \theta_{B2} \equiv \theta_{F1}(0) - \alpha \equiv \theta_{B2}(0) - \alpha \tag{4.3}$$
$$\theta_{F2} \equiv \theta_{B1} \equiv \theta_{F2}(0) + \alpha \equiv \theta_{B1}(0) + \alpha \tag{4.4}$$

Dove α rappresenta l'angolo di sterzo applicato al meccanismo, entro i suoi limiti costruttivi.

Considerando l'interezza del veicolo, è possibile stabilire una relazione tra l'angolo di sterzo e la traiettoria da esso descritta in moto. Questa relazione si basa sul concetto del Centro di Rotazione Istantaneo (ICR), ottenuto geometricamente intersecando gli assi di ciascuna ruota. L'ICR rappresenta il centro della circonferenza attorno alla quale ruotano tutti i punti materiali del veicolo e di conseguenza anche il suo baricentro. Nel caso di una geometria di sterzo parallela si ottiene la situazione descritta in Figura 4.19.

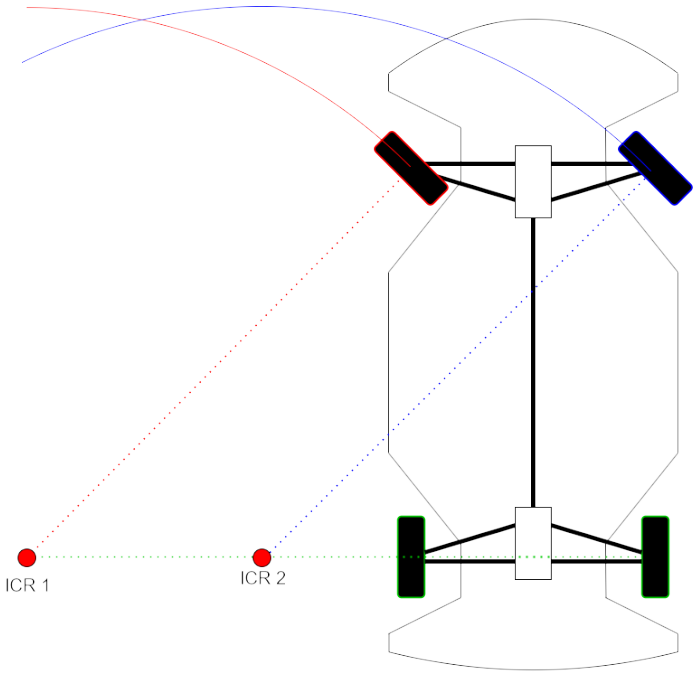


Figura 4.19.: Rappresentazione ICR, sterzo parallelo.

In Figura 4.19 è possibile osservare la presenza di due ICR distinti e la conseguente differenza tra le traiettorie descritte tra le ruote anteriori del veicolo. Questa particolare geometria di sterzo, purché semplice nella realizzazione, presenta questo difetto caratteristico. Difatti, all'aumentare della velocità di marcia del veicolo, la convergenza delle traiettorie descritte dalle ruote fa si che diminuisca la tenuta su strada a causa dell'inevitabile scivolamento degli pneumatici sul suolo. Conseguenze di questo fenomeno sono una diminuzione della tenuta in curva ad alte velocità, una peggiore stabilità laterale ed un'inevitabile maggiore usura degli pneumatici. Un pregio di questa geometria di sterzo è, però, la manovrabilità a basse velocità, garantita dalla posizione dell'ICR generato dalla ruota esterna alla curva (ICR 2 in Figura 4.19).

A causa della natura di questa geometria è possibile semplificarne il modello, trascurando il trasferimento del peso sulle sospensioni e non considerando lo scivolamento degli pneumatici. Infatti, data la simmetria del modello lungo l'asse longitudinale, è possibile sovrapporre le ruote anteriori e posteriori lungo l'asse centrale del veicolo, mantenendone il passo: si ottiene così quello che in letteratura viene comunemente denominato "Modello a bicicletta", che permette di effettuare differenti semplificazioni nel calcolo della dinamica del veicolo.

Una caratteristica fondamentale di questo modello è la rimozione dell'ambiguità dell'ICR, poiché la sua formulazione prevede un unico ICR posizionato nel punto medio del segmento congiungente i due ICR reali.

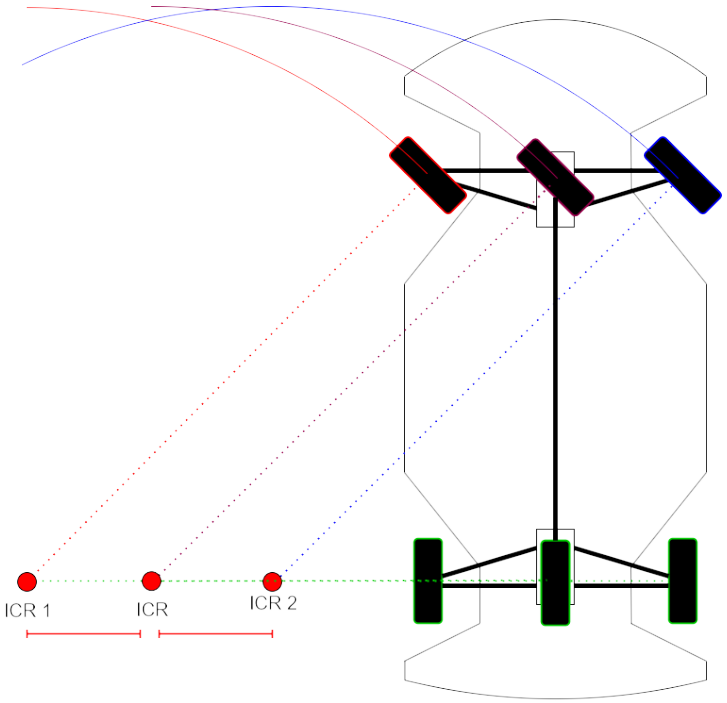


Figura 4.20.: Rappresentazione del modello, con semplificazione "a bicicletta" in sovrapposizione.

4.3.2. Geometria di sterzo Ackermann

La geometria di sterzo Ackermann è realizzata in modo simile a quella parallela in termini meccanici, risultando però radicalmente diversa in termini di costruzione geometrica. Difatti questo tipo di geometria si basa comunque su un sistema di assi rigidi giunti alle estremità ma usa una configurazione trapezoidale, che le conferisce il suo funzionamento caratteristico.

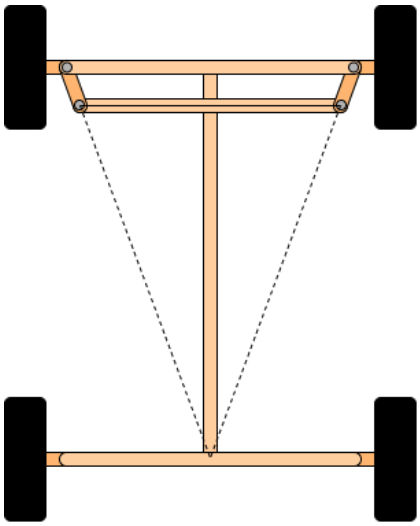


Figura 4.21.: Geometria di Sterzo Ackermann.

La caratteristica che distingue questa geometria di sterzo da quella parallela è il cosiddetto "toe dinamico", o più comunemente, noto come effetto Ackermann. Per spiegare quest'ultimo è necessario però descrivere in primis questa particolare geometria.

Come è possibile osservare in Figura 4.21, i due assi verticali del meccanismo di sterzo sono inclinati di un uguale angolo verso l'interno (o verso l'esterno), rispetto alla geometria parallela introdotta nella sezione precedente. Questa caratteristica conferisce alla geometria di sterzo, nella posizione di sterzo neutrale, la sua caratteristica forma a trapezio isoscele. La configurazione in Figura 4.21 è detta 100% Ackermann, e si ottiene quando i prolungamenti degli assi inclinati si intersecano nel punto medio dell'asse posteriore. Ci sono anche altre configurazioni possibili, che verranno introdotte in seguito.

Considerando una configurazione 100% Ackermann, l'inclinazione degli assi rispetto all'interasse del veicolo, può essere ricavata trigonometricamente. Detti L il passo del veicolo, C la distanza tra i giunti degli elementi sterzanti e α l'angolo compreso tra l'asse posteriore e quello sterzante, si ha che:

$$\alpha = \tan^{-1}\left(\frac{C}{2 \cdot L}\right) \tag{4.5}$$

Poiché l'angolo α definisce le caratteristiche di questa geometria, è detto "angolo di Ackermann".

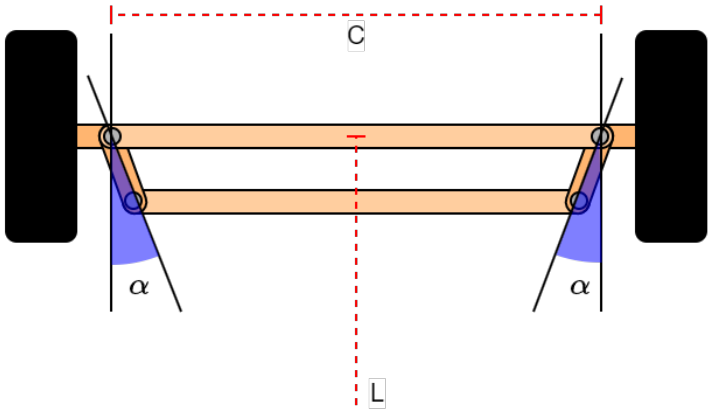


Figura 4.22.: Angolo di Ackermann

Come introdotto in precedenza, una conseguenza di questa geometria è una differenza, più o meno pronunciata, tra gli angoli di sterzo della ruota interna rispetto a quella esterna durante la sterzata. Nel caso di una configurazione 100% Ackermann, l'angolo di sterzo della ruota interna sarà maggiore rispetto a quello della ruota esterna, in quanto la traiettoria da essa percorsa sarà una circonferenza di raggio minore.

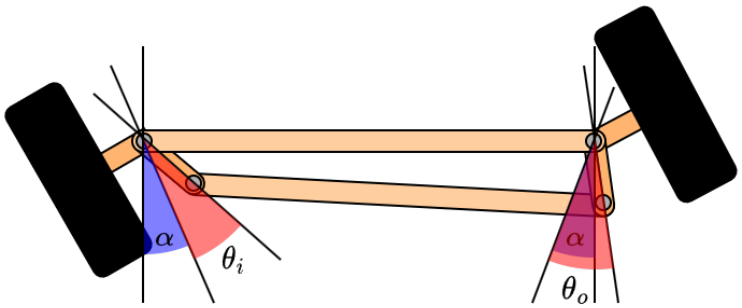


Figura 4.23.: Sterzata verso sinistra

In Figura 4.23 è riportato un possibile funzionamento di questa geometria di sterzo, evidenziando l'angolo di sterzo della ruota che si trova nella zona più interna della traiettoria (θ_i) e quello della ruota più esterna (θ_o).

Dato un angolo di sterzo, imposto meccanicamente dalla rotazione del volante di guida, si hanno le seguenti relazioni:

$$\theta_{ack} = \frac{\theta_{in}}{\gamma} \tag{4.6}$$

$$\theta_o = \tan^{-1}\left(\frac{L \cdot \tan(\theta_{ack})}{W + 0.5 \cdot \tan(\theta_{ack})}\right) \tag{4.7}$$

$$\theta_i = \tan^{-1}\left(\frac{L \cdot \tan(\theta_{ack})}{W - 0.5 \cdot \tan(\theta_{ack})}\right) \tag{4.8}$$

$$\frac{C}{L} = \cot(\theta_o) - \cot(\theta_i) \tag{4.9}$$

Dove:

- θ_{in} rappresenta l'angolo di sterzo applicato dal volante;
- γ rappresenta un rapporto di riduzione, dato dal meccanismo che trasforma la rotazione del volante in rotazione del meccanismo di sterzo;
- θ_{ack} rappresenta l'angolo di sterzo effettivamente applicato al veicolo, che ne definisce la traiettoria;
- W rappresenta la misura della carreggiata del veicolo;
- L rappresenta la misura del passo del veicolo,

Inoltre, esiste una relazione fondamentale tra gli angoli di sterzo della ruota interna, di quella esterna e di quella "ideale" che rappresenta la traiettoria del veicolo. Questa relazione si basa sulla percentuale di Ackermann (p_{ack}) ed è la seguente:

$$\theta_o = \theta_i - p_{ack} \cdot (\theta_i - \theta_{ack}) \tag{4.10}$$

Nel caso specifico di una geometria 100% Ackermann come quella presa in esame, la Formula (4.10) si semplifica in:

$$\theta_o = \theta_{ack} \tag{4.11}$$

Per il calcolo dell'angolo di sterzo della ruota interna alla curva vale invece la relazione (4.8).

Una volta ricavati gli angoli di sterzo delle due ruote frontali, è possibile calcolare i raggi di curvatura delle traiettorie circolari da esse percorse. Poiché a causa della realizzazione meccanica dello sterzo, le ruote non sterzano attorno ai loro baricentri ma attorno a dei giunti che si trovano in una posizione più interna rispetto alla carreggiata, durante i calcoli bisogna tenere conto di questa peculiarità del meccanismo.

Sia C la distanza tra i due giunti sterzanti, W la carreggiata del veicolo, L il suo passo, θ_i e θ_o gli angoli di sterzo rispettivamente della ruota interna ed esterna alla curva, si hanno le seguenti relazioni:

$$R_i = \frac{L}{\sin(\theta_i)} - \frac{W - C}{2} \tag{4.12}$$

$$R_o = \frac{L}{\sin(\theta_o)} + \frac{W - C}{2} \tag{4.13}$$

$$R_m = \frac{R_o - R_i}{2} \tag{4.14}$$

Le relazioni (4.12) e (4.13) forniscono le misure dei raggi di curvatura delle traiettorie descritte rispettivamente dalla ruota interna ed esterna, mentre la relazione (4.14) fornisce il raggio medio.

Tracciando le due circonferenze di raggio R_i e R_o, tangenti alla direzione di ciascuna delle due ruote anteriori nei rispettivi punti di contatto con il terreno, si ottiene, dalla loro intersezione con il prolungamento dell'asse posteriore, un unico centro di rotazione istantaneo (ICR) del veicolo.

La presenza di un unico ICR, rigoroso a livello geometrico, rappresenta il principale vantaggio di questa geometria di sterzo. Questa caratteristica, in condizioni ideali di non slittamento delle ruote, presenta diversi benefici:

- Minimizzazione del consumo degli pneumatici: non essendo presente slittamento, gli pneumatici si consumano in modo uniforme e in maniera ridotta;
- Massimizzazione dell'agilità del veicolo: a basse velocità, il veicolo segue meglio la traiettoria desiderata ed è in grado di compiere manovre precise con raggi di curvatura molto stretti;
- Migliore stabilità in curva: a causa del corretto allineamento delle ruote si evitano fenomeni di sovrasterzo e sottosterzo.

Nonostante i benefici, durante la sterzata ad alte velocità, sono presenti anche dei difetti. Un sistema di sterzo può però essere ottimizzato per il controllo ad alte velocità modificando la percentuale di Ackermann.

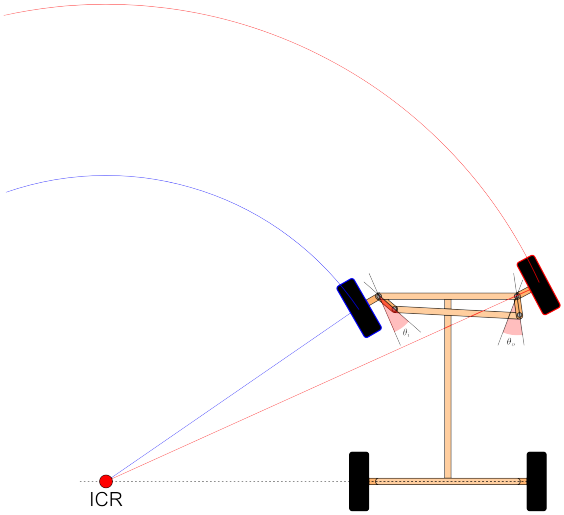


Figura 4.24.: Geometria Ackermann, traiettorie e raggi di curvatura.

Come introdotto in precedenza, la geometria di sterzo Ackermann può essere progettata in modo da meglio adattarsi all'applicazione di interesse. Difatti, modificando l'angolo di Ackermann e di conseguenza la percentuale di Ackermann, si possono ottenere diverse configurazioni, divise in tre categorie:

- **Pro-Ackermann**: configurazioni con angoli di Ackermann diversi da quelli definiti da una configurazione 100%, che permettono comunque di ottenere un effetto Ackermann più o meno pronunciato. La percentuale di Ackermann è un valore maggiore di 1% e può anche superare il 100%, accentuando ulteriormente l'effetto di toe-in dinamico;
- **Sterzo parallelo**: configurazione con un angolo di Ackermann pari a 0. La geometria di Ackermann è difatti una versione più generale di quella esplorata nella sezione precedente;
- **Anti-Ackermann**: configurazioni con angoli di Ackermann negativi, risultanti in valori di percentuale negativi. Geometrie appartenenti a questa categoria presentano un funzionamento opposto a quello delle pro-Ackermann. Infatti la ruota esterna sterza di più rispetto a quella interna, ottenendo quindi un effetto di toe-out dinamico. Questa geometria è vastamente utilizzata in gare ad alta velocità con curve larghe, quali, ad esempio, quelle di Formula 1.

Una buona rappresentazione di questa categorizzazione è mostrata in Figura 4.25, nella quale sono presenti 4 aree distinte:

- **Area 1**: geometria pro-Ackermann, con valore di percentuale superiore al 100%;
- **Area 2**: geometria pro-Ackermann, con valore di percentuale compreso tra 1% e 99%;
- **Area 3**: geometria di sterzo parallelo, con valore di percentuale nullo;
- **Area 4**: geometria anti-Ackermann, con valori di percentuale negativi;

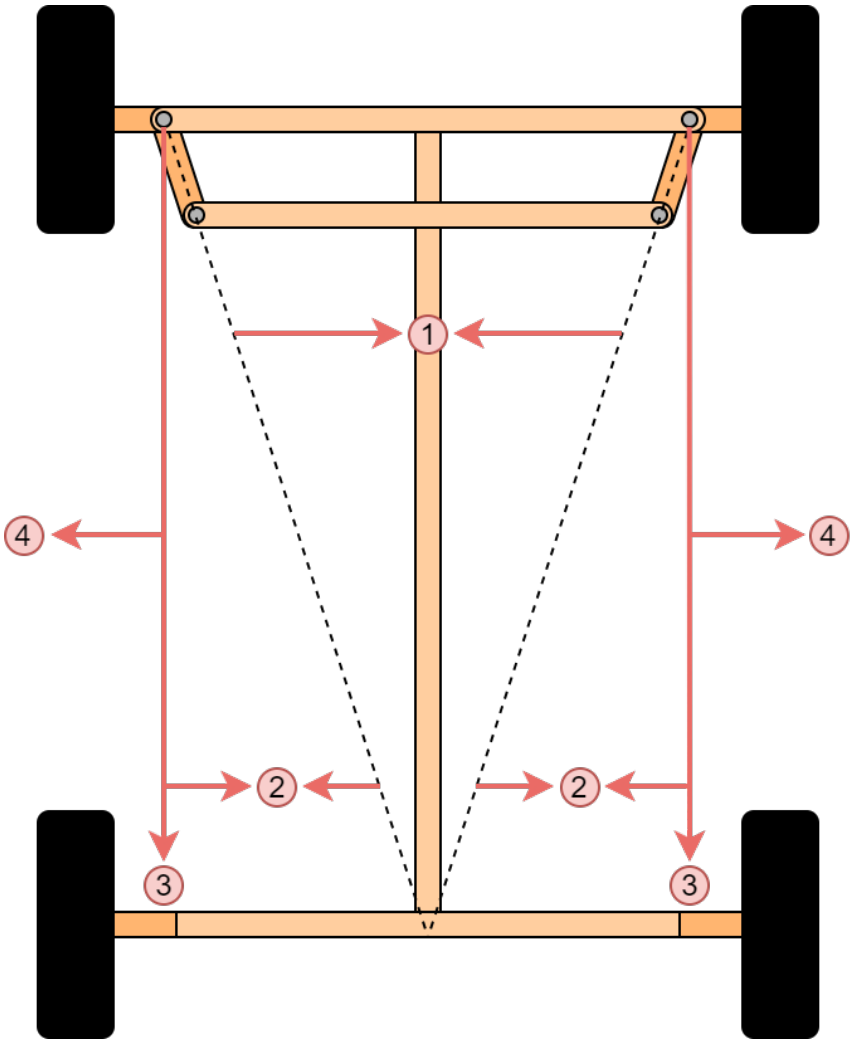


Figura 4.25.: Configurazioni della geometria Ackermann

4.3.3. Geometria di sterzo a 4 ruote (4WS)

Introduzione

La geometria di sterzo a 4 ruote, denominata anche "All-Wheel Steering" (AWS), rappresenta un sistema di sterzo radicalmente diverso da quelli analizzati in precedenza. La caratteristica che differenzia questa geometria dalle altre è la possibilità di sterzare tutte e 4 le ruote del veicolo, come implicato dal nome.

Questa particolare geometria non è ampiamente utilizzata nel contesto automobilistico, in quanto è presente solo su alcuni veicoli di lusso o rover.

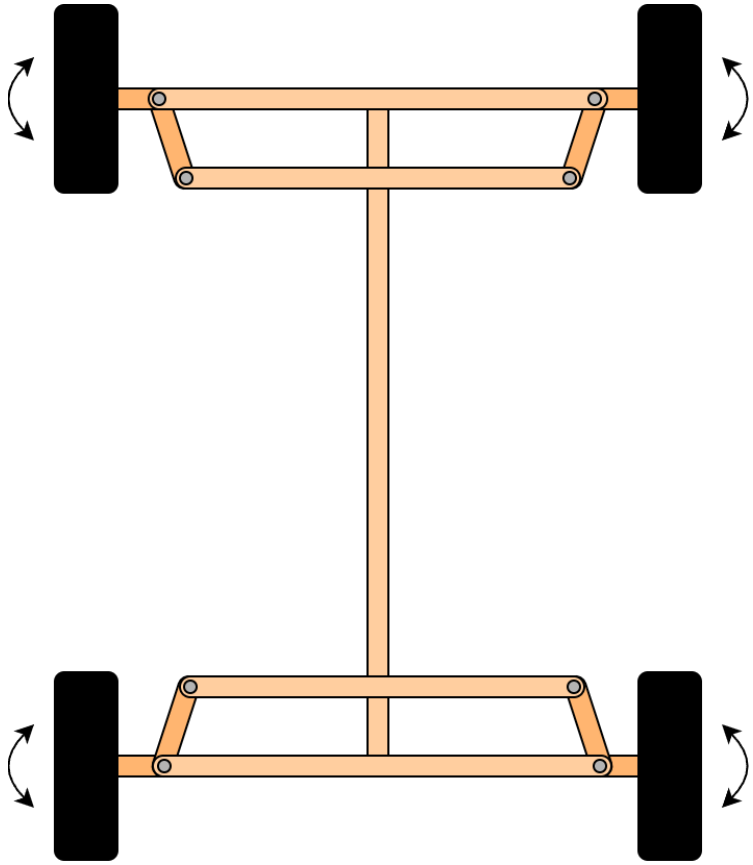


Figura 4.26.: Geometria di sterzo 4WS.

Caratteristiche principali

Come già introdotto, questa geometria consente di sterzare tutte e 4 le ruote del veicolo. In base al tipo di manovra richiesto, le ruote posteriori possono sterzare sia nella direzione opposta rispetto a quelle anteriori, fornendo un raggio di sterzata minore a basse velocità, sia nella stessa direzione, fornendo una sterzata più stabile ad alte velocità.

Fasi della geometria 4WS

La possibilità di articolare in modo indipendente il meccanismo di sterzo anteriore e quello posteriore permette di definire tre configurazioni, anche chiamate fasi. Esse sono:

- Fase neutra
- Fase positiva
- Fase negativa

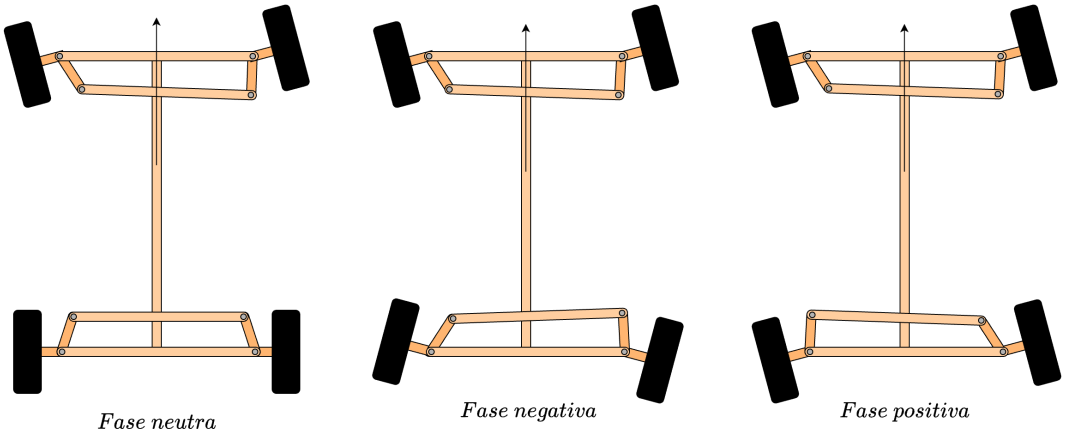


Figura 4.27.: 4WS, fasi.

Come descritto in Figura 4.27, la fase neutra equivale ad un meccanismo di sterzo anteriore, del tutto similare a quello descritto nelle sezioni precedenti. Durante le manovre a bassa velocità, dove è richiesta massima maneggevolezza, lo sterzo opera con una fase negativa, ovvero sterzando le ruote posteriori di un angolo di segno opposto rispetto a quelle anteriori.

Ad alte velocità, ad esempio durante un sorpasso in autostrada, lo sterzo viene attuato invece con una fase positiva, ovvero sterzando le ruote posteriori di un angolo di segno concorde a quello delle ruote anteriori; Risultato di questo metodo di sterzo, è una traslazione laterale del veicolo, piuttosto che una rotazione di esso.

Tipologie di sterzo a 4 ruote

La geometria di sterzo a 4 ruote si differenzia in due categorie principali:

- 4WS passivo;
- 4WS attivo;

Il sistema 4WS passivo prevede una connessione meccanica tra il meccanismo di sterzo anteriore e quello posteriore, che si articola in maniera opposta al primo a basse velocità e in maniera conforme ad alte velocità.

Il sistema 4WS attivo prevede, invece, un totale disaccoppiamento dei due meccanismi di sterzo, che vengono azionati in funzione dei dati acquisiti dai sensori a bordo del veicolo. Il funzionamento è simile a quello del sistema passivo ma è possibile ottenere un comfort e una maneggevolezza migliori utilizzando ulteriori dati acquisiti durante la guida, quali accelerazioni e rotazioni del veicolo.

Geometria di sterzo

Il sistema 4WS impiega spesso una geometria di sterzo di tipo Ackermann, anch'essa basata sul concetto di ICR. In questo caso, l'ICR non si trova però sul prolungamento dell'asse posteriore, a meno che il veicolo non stia sterzando utilizzando una fase neutra.

Nel caso in cui il veicolo sterzi con una fase negativa, l'ICR si trova in una posizione compresa tra l'asse anteriore e quello posteriore, garantendo quindi un raggio di curvatura nettamente inferiore rispetto a quello che si avrebbe le caso di una geometria di sterzo puramente di tipo Ackermann.

Se, invece, il veicolo sterza con una fase positiva, l'ICR si troverà dietro all'asse posteriore del veicolo, permettendo di seguire traiettorie con raggi di curvatura maggiori e dunque garantendo una guida più stabile.

Un esempio illustrativo è riportato in Figura 4.28 .

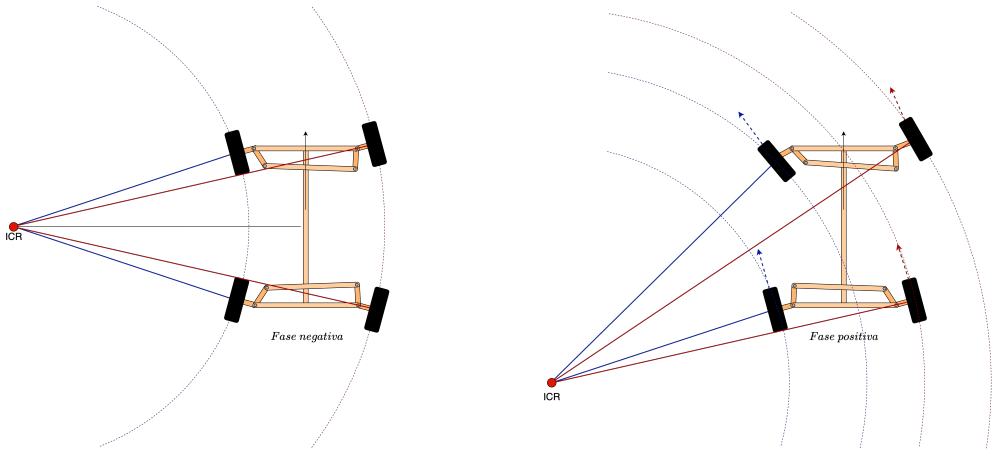


Figura 4.28.: ICR, fase positiva e negativa.

Pregi e difetti della geometria

Tra i punti di forza di questa geometria di sterzo si hanno sicuramente la sua versatilità, data dalla capacità di adattarsi al meglio alle condizioni di guida del veicolo.

Un difetto notabile è sicuramente rappresentato dalla complessità di questa geometria e dalla necessità di un addizionale meccanismo di sterzo. Nello sterzo 4WS

attivo è inoltre richiesta la presenza di sensori a bordo per ottimizzare ulteriormente gli angoli di sterzo. Questa complessità si traduce automaticamente in un maggiore costo di produzione e di manutenzione dei vari meccanismi: questo fa si che siano solitamente impiegati in veicoli di lusso o comunque di fascia più alta.

4.4. Cinematica del veicolo

4.4.1. Introduzione

Nelle sezioni precedenti è stata introdotta la geometria del veicolo e in particolare quella dello sterzo. Infatti, comprendere i principi su cui si basa il controllo laterale di un veicolo è fondamentale per una corretta progettazione e realizzazione di tali sistemi di controllo.

L'angolo di sterzo delle ruote del veicolo, ottenuto ruotando il volante, mediante la geometria di sterzo utilizzata, rappresenta una delle più importanti misure che determinano l'evoluzione del moto del veicolo nel tempo. In precedenza è stato introdotto il concetto di ICR e sono state descritte le traiettorie idealmente percorse dalle ruote del veicolo e dai punti materiali appartenenti ad esso.

Esiste, come verrà discusso nelle prossime sezioni, uno stretto legame tra gli aspetti geometrici trattati finora e il moto effettivo del veicolo nel tempo.

Lo scopo di questa sezione è quello di formulare in modo rigoroso un modello cinematico del veicolo utile alla realizzazione del suo controllo, evidenziando le varie misure che verranno impiegate per la sua realizzazione.

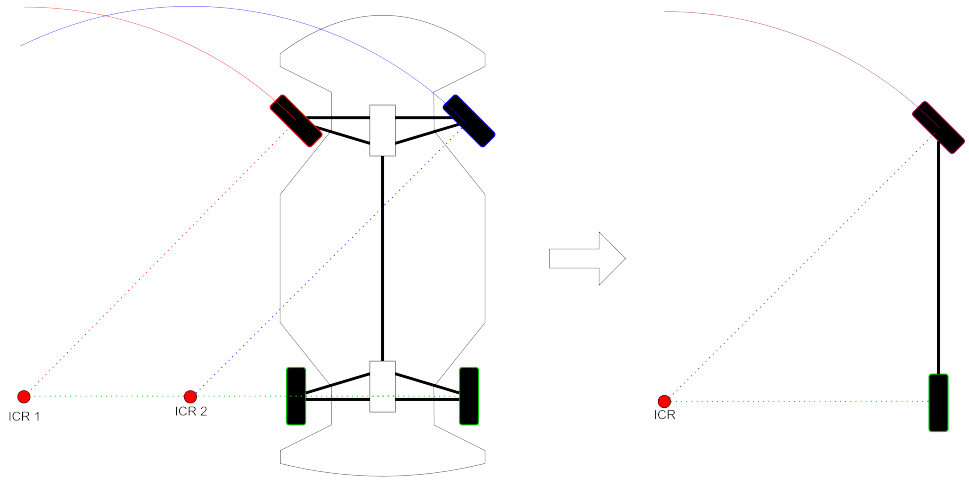


Figura 4.29.: Semplificazione del modello cinematico

4.4.2. Sistemi di riferimento

La cinematica permette di descrivere geometricamente il moto nello spazio, in relazione a differenti sistemi di riferimento. Per descrivere in modo completo e formale il moto di un corpo rigido, in questo caso quello di un veicolo, è dunque necessario fornire una formulazione rigorosa dei vari sistemi di riferimento utili a tale scopo.

Sistema di riferimento inerziale

Nel contesto della dinamica e del controllo dei veicoli, il sistema di riferimento inerziale è solidale con quello della terra.

Per descrivere questo sistema di riferimento viene utilizzato il sistema di coordinate Cartesiane destrorso, indicando gli assi con X, Y e Z.

L'asse Z è verticale ed è orientato in direzione opposta rispetto alla forza di gravità e gli assi X-Y rappresentano il piano orizzontale, perpendicolare alla forza di gravità.

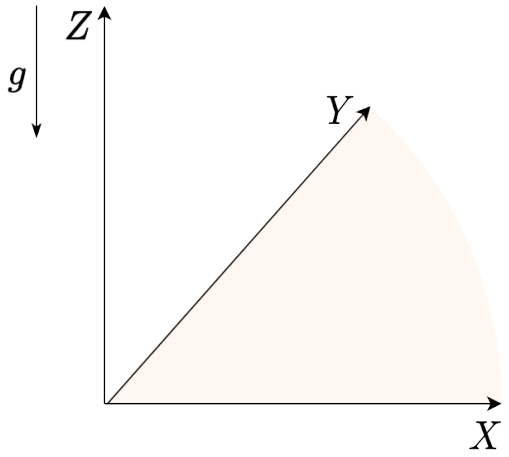


Figura 4.30.: Sistema di riferimento inerziale

Sistema di riferimento del veicolo

Questo sistema di riferimento è solidale con il veicolo, rispetto ad un suo punto, solitamente rappresentato dal centro di gravità o dal punto centrale dell'asse posteriore. Gli assi di questo sistema sono indicati con X_v, Y_v e Z_v e sono orientati in modo che X_v punti verso il fronte del veicolo, Y_v verso il lato sinistro e Z_v verso l'alto, secondo lo standard internazionale ISO 8855.

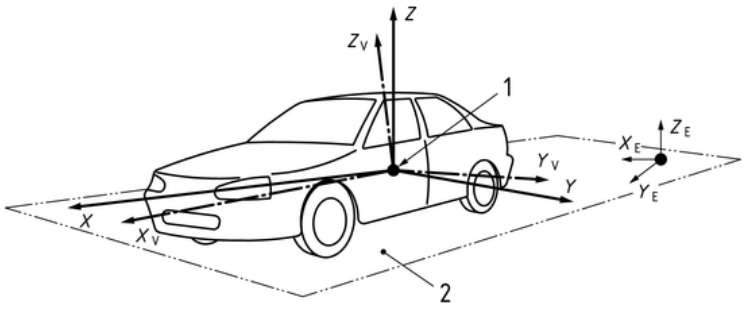


Figura 4.31.: Sistema di riferimento del veicolo, ISO 8855.

Sistema di riferimento orizzontale

L'origine di questo sistema di riferimento è rappresentato dal punto di riferimento del veicolo, ma gli assi X, Y e Z di questo sistema non sono orientati allo stesso modo. Difatti gli assi X ed Y sono rispettivamente le proiezioni ortogonali degli assi X_v e Y_v sul piano orizzontale del sistema di riferimento inerziale, mentre l'asse Z è ortogonale ad esso.

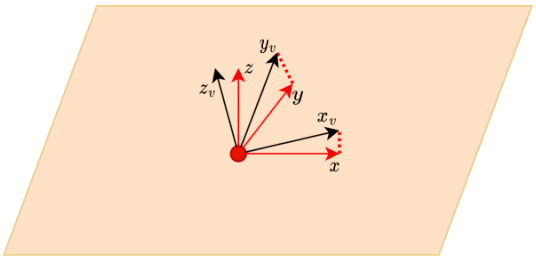


Figura 4.32.: Sistema di riferimento orizzontale.

Sistema di riferimento della traiettoria

Un ulteriore sistema di riferimento fondamentale per la formalizzazione del moto di un veicolo è quello della sua traiettoria.

Sia $P(s)$ una curva parametrizzata in $s \in \mathbb{R}$, tale che $P(s)$ sia differenziabile almeno una volta $\forall s \in \mathbb{R}$.

L'origine del sistema di riferimento della traiettoria segue la curva $P(s)$, denominata traiettoria, e le sue coordinate sono descritte come segue:

- asse d: direzione tangenziale alla traiettoria;
- asse e: direzione perpendicolare all'asse d, orientata verso la sua sinistra.
- asse n: asse ortogonale sia all'asse d che all'asse e, a formare un sistema di coordinate Cartesiano destrorso.

Gli assi d ed e danno origine al piano stradale, il quale è tangente alla superficie terrestre nel punto di origine, assunto esso differenziabile.

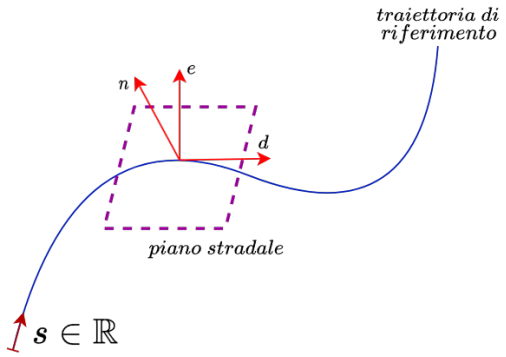


Figura 4.33.: Sistema di riferimento della traiettoria

4.4.3. Modello cinematico del veicolo

Per modellare la cinematica del veicolo, è stato utilizzato il modello cinematico "a bicicletta", descritto formalmente di seguito:

> "Il modello cinematico a bicicletta è una semplificazione del modello cinematico a quattro ruote, dove le due ruote posteriori e le due ruote posteriori sono sovrapposte a formare rispettivamente una ruota posteriore ed una anteriore (immaginarie)"

Questo modello è stato introdotto nella sezione 4.3.1, senza però approfondirne gli aspetti formali che verranno discussi di seguito.

In Figura 4.29 è mostrata un'illustrazione formale del modello cinematico a bicicletta, caratterizzato da:

- ICR: centro istantaneo di rotazione del veicolo, e di conseguenza di tutti i punti materiali che lo compongono;
- C: punto materiale del veicolo, descritto dalle sue coordinate nel sistema di riferimento inerziale;
- $\vec{v}$: vettore della velocità istantanea del veicolo, di modulo costante v;
- l_r, l_f: rispettive distanze del punto C dalle due ruote del veicolo. Vale la condizione $l_r + l_f = L$, dove L è il passo del veicolo;
- δ: angolo di sterzo della ruota anteriore rispetto alla direzione di marcia del veicolo, ottenuta prolungando l'asse del veicolo;
- β: angolo di slittamento del veicolo, considerato nel punto materiale C;
- ψ: angolo di imbardata del veicolo, comunemente conosciuto come angolo di yaw. Rappresenta lo scostamento angolare tra l'asse X_v del sistema di riferimento del veicolo e l'asse X del sistema di riferimento inerziale ;

- ω: velocità angolare del veicolo attorno all'ICR, ossia la velocità angolare assunta da qualsiasi suo punto materiale.

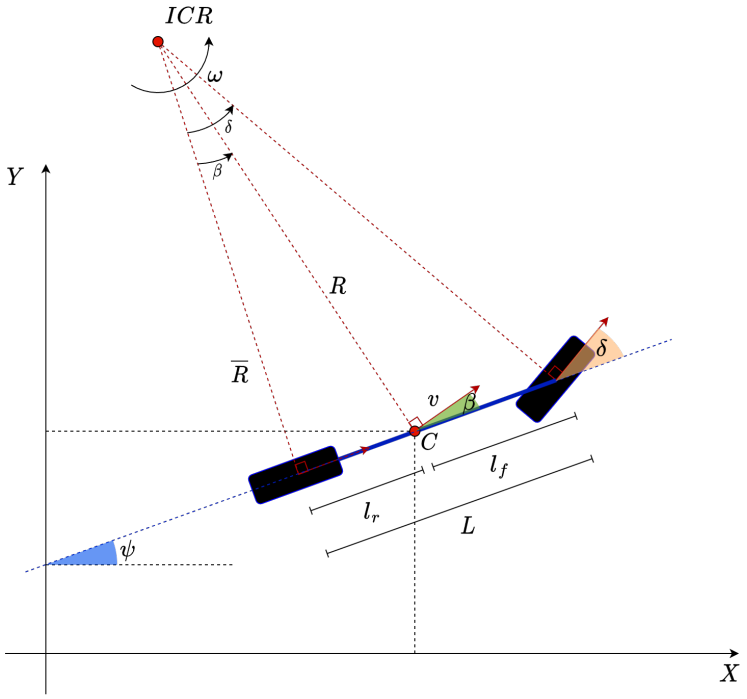


Figura 4.34.: Modello cinematico a bicicletta

Assumendo che il veicolo si muova ad una velocità costante v e le ruote non scivolino sul terreno durante lo sterzo, dal modello si possono estrapolare le seguenti relazioni che descrivono rispettivamente la velocità di un qualsiasi punto materiale rispetto agli assi e la variazione dell'angolo di imbardata, meglio conosciuta come yaw rate.

$$\dot{x}(t) = v\cos(\psi + \beta) \tag{4.15}$$

$$\dot{y}(t) = v\sin(\psi + \beta) \tag{4.16}$$

$$\dot{\psi}(t) = \omega = \frac{v}{R} \tag{4.17}$$

Nella Relazione (4.17), va notato che il valore di R rappresenta il raggio della circonferenza che congiunge l'ICR con il punto C. Utilizzando semplici relazioni geometriche e trigonometriche, è possibile ottenere ulteriori relazioni che arricchiscono il modello.

Difatti, è possibile innanzitutto ricavare la relazione tra l'angolo β e δ, applicando semplici formule trigonometriche sui triangoli descritti dai diversi raggi di curvatura e dal passo del veicolo. Si ottiene quanto segue:

$$\tan\delta = \frac{l_f + l_r}{\overline{R}} \implies \overline{R} = \frac{l_f + l_r}{\tan\delta} \tag{4.18}$$

dove $\overline{R}$ rappresenta il raggio della circonferenza congiungente l'ICR alla ruota posteriore. Dunque, applicando nuovamente relazioni trigonometriche, si ottiene:

$$\tan\beta = \frac{l_r}{\overline{R}} = \frac{l_r}{l_f + l_r}\tan\delta \implies \beta = \arctan\left(\frac{l_r}{l_f + l_r}\cdot\tan\delta\right) \tag{4.19}$$

Rimane ora da ricavare una relazione tra lo yaw rate, l'angolo di sterzo δ e l'angolo di slittamento β: ciò può essere fatto tramite relazioni trigonometriche mediante la seguente relazione:

$$\cos\beta = \frac{\overline{R}}{R} \implies \frac{1}{R} = \frac{1}{\overline{R}}\cdot\cos\beta = \frac{1}{l_f + l_r}\cdot\tan\delta\cdot\cos\beta \tag{4.20}$$

Sostituendo nella Relazione (4.17) si ottiene infine:

$$\dot{\psi}(t) = \frac{v}{l_f + l_r}\cdot\cos\beta\cdot\tan\delta \tag{4.21}$$

Dalla Relazione (4.21) è evidente come il valore dello yaw rate in un dato istante di tempo è strettamente legato al punto in cui viene misurato, in quanto è presente una stretta dipendenza tra questo e l'angolo di slittamento β, che dipende appunto dalla posizione del punto C rispetto all'asse del veicolo . Per disaccoppiare il valore dello yaw rate dal valore dell'angolo di slittamento è dunque necessario posizionare il punto C sull'asse posteriore del veicolo, ottenendo dunque la relazione:

$$\dot{\psi}(t) = \frac{v}{L}\cdot\tan\delta \tag{4.22}$$

Poiché vale la relazione:

$$\frac{1}{\overline{R}} = \frac{\tan\delta}{L} \tag{4.23}$$

si ha che:

$$\dot{\psi}(t) = \frac{v}{\overline{R}} \tag{4.24}$$

Questa relazione, che lega il valore dello yaw rate al raggio di curvatura descritto dal veicolo rispetto all'asse posteriore, risulta essere fondamentale per il controllo di quest'ultimo in quanto permette di sterzare il veicolo nella maniera corretta indipendentemente dalla geometria di sterzo.

4.5. Dimensioni e geometria di sterzo del veicolo di sviluppo

In questa sezione verranno descritte le dimensioni del veicolo utilizzato nel contesto applicativo, la sua geometria di sterzo e i suoi limiti funzionali.

Il veicolo presenta le seguenti dimensioni:

- **Passo**: 260mm
- **Carreggiata**: 175mm
- **Distanza tra i giunti di rotazione dello sterzo**: 130mm

Per quanto riguarda la geometria di sterzo, essa è di tipo puramente Ackermann, ovvero con un fattore di Ackermann pari al 100%. I limiti funzionali della geometria sono rappresentati dall'angolo massimo di sterzo della ruota interiore rispetto alla curva, che può ruotare di un angolo massimo di 23°.

Con i parametri ricavati sperimentalmente, è possibile ricavare il raggio minimo della traiettoria circolare che il veicolo può descrivere, ovvero pari a 55cm. Come verrà trattato nei capitoli successivi, questa proprietà del veicolo si appresta bene ad eseguire correttamente le manovre richieste dalla competizione.

Capitolo 5.

Scenari di guida

5.1. Introduzione

In questo capitolo verranno descritti i vari scenari della competizione che il veicolo deve essere in grado di navigare in completa autonomia, riportando le misure accurate della pista di gara fornite dalla documentazione ufficiale della competizione. Verrà illustrato in dettaglio il tracciato e le sezioni di particolare interesse per questo elaborato, evidenziandone la topologia.

Comprendere al meglio il percorso di gara permette, infatti, di adattare nel miglior modo possibile le soluzioni sviluppate, cercando comunque di mantenere un buon grado di generalità nella loro formulazione.

5.2. Tracciato di gara

Il tracciato di gara simula un contesto urbano in scala 1:10, con sezioni ad alta densità di incroci e sezioni più lineari, quali, ad esempio, il tratto di autostrada. In Figura 5.1 è riportato il tracciato nella sua interezza.

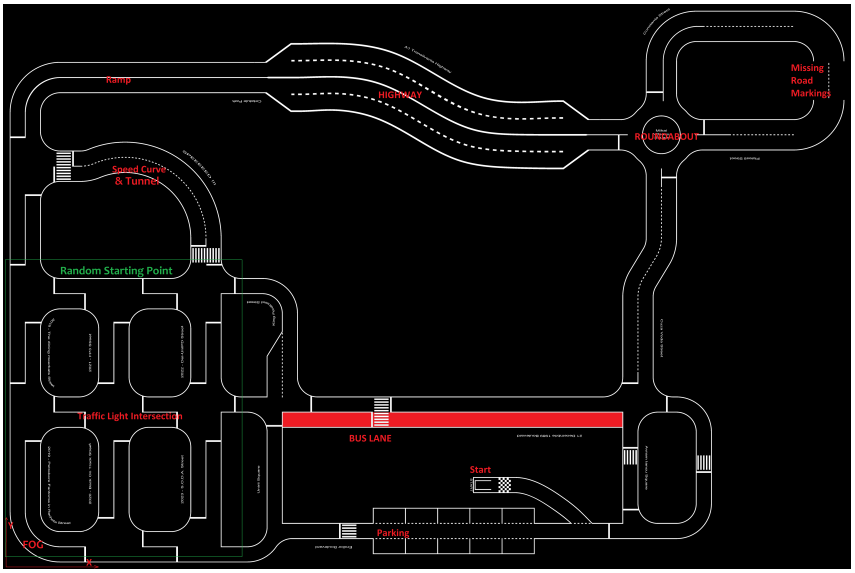


Figura 5.1.: Tracciato di gara.

Per quanto riguarda le corsie, esse hanno larghezza costante lungo tutto il tracciato e presentano le misure riportate in Figura 5.2 .

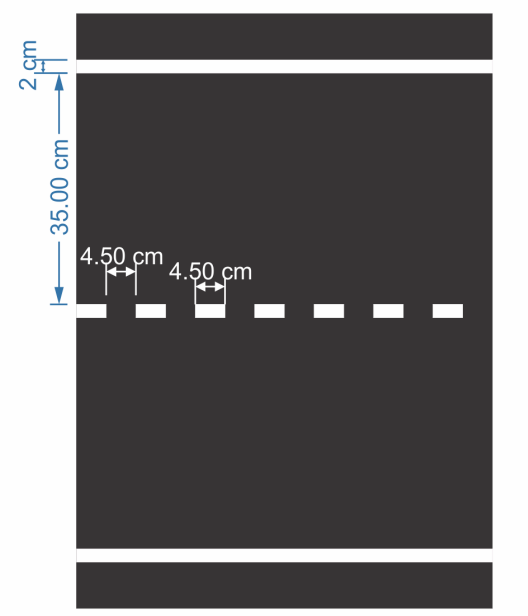


Figura 5.2.: Corsia e relative misure.

Queste misure permettono al veicolo di mantenere la corsia con un buon margine di distanza rispetto alla segnaletica orizzontale, richiedendo però precisione durante le manovre di sterzo, in quanto in curva il veicolo non si trova perfettamente centrato nella corsia che sta percorrendo.

Come introdotto in precedenza, al tracciato di gara è associato un grafo "virtuale" che viene utilizzato dal veicolo per ottenere la propria posizione in tempo reale.

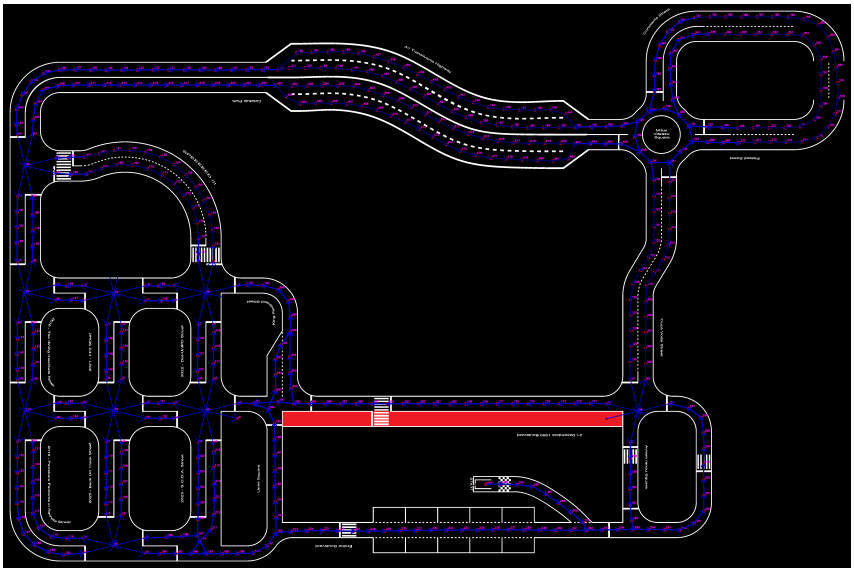


Figura 5.3.: Grafo del tracciato di gara.

Esso è formato da un numero finito di nodi, identificati da un numero univoco e dalle proprie coordinate rispetto all'origine. I nodi sono collegati tra di loro mediante archi direzionati, essendo il grafo orientato.

Infatti, gli archi sono identificati dal nodo di partenza, dal nodo di arrivo e dal tipo di linea di mezzeria, che può essere continua o tratteggiata.

Il grafo è codificato in GraphML ed è reperibile presso la repository GitHub della documentazione ufficiale [6].

5.3. Curve

Il tracciato di gara presenta curve di due raggi diversi, in quanto sono sempre presenti due corsie parallele. Per quanto riguarda la curva interna, il raggio della traiettoria che il veicolo deve seguire è di 66.50cm rispetto al centro della corsia, mentre la curva esterna presenta un raggio di curvatura pari a 103.50cm, sempre rispetto al centro della corsia.

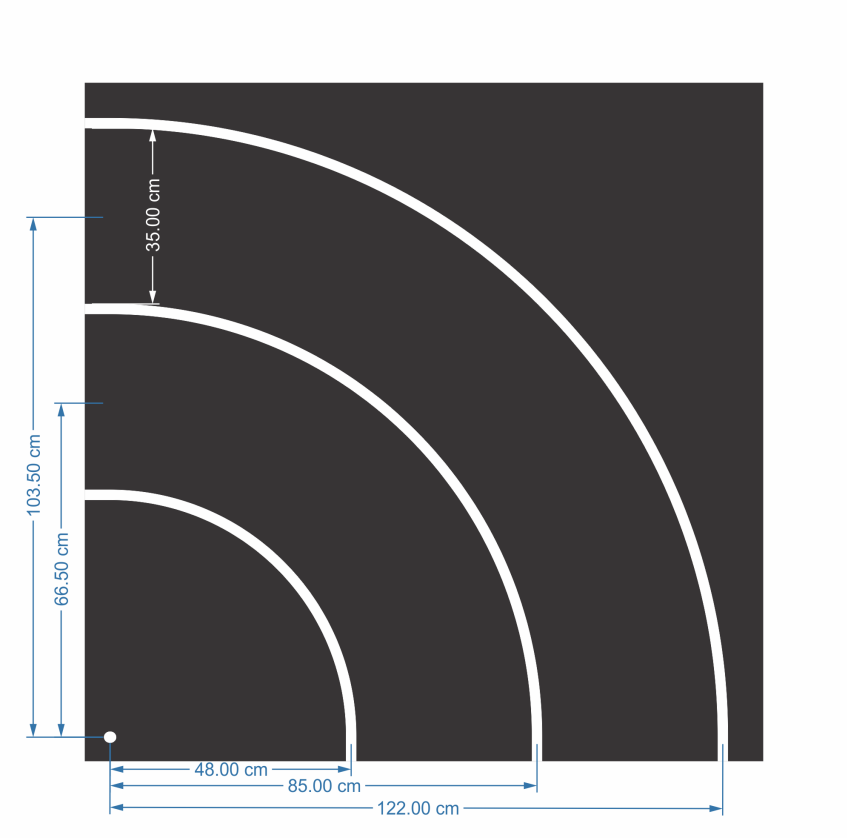


Figura 5.4.: Misure delle curve presenti nel tracciato di gara.

Entrambe le curve hanno raggi di curvatura maggiori rispetto al raggio minimo eseguibile dalla macchina, rendendo quindi possibile la corretta esecuzione della manovra e permettendo anche correzioni di traiettoria.

5.4. Incroci

Nel tracciato è presente una sezione densa di incroci, che presentano due configurazioni omogenee. La navigazione di essi è quindi piuttosto semplice, in quanto è possibile definire traiettorie "standard" in funzione del percorso scelto dagli algoritmi di path planning.

In Figura 5.5 sono mostrate le due configurazioni, rispettivamente a 3 e 4 vie.

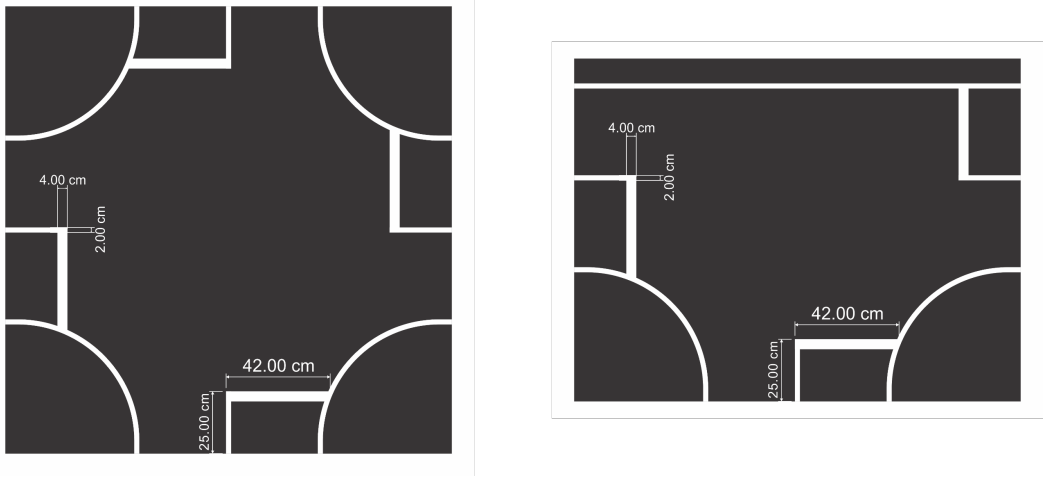


Figura 5.5.: Intersezioni, 4 vie (sx) e 3 vie (dx).

5.5. Parcheggi

Il tracciato di gara prevede anche una sezione adibita al parcheggio, il cui inizio e la cui fine sono indicati da un apposito cartello stradale. Questa sezione presenta una sola corsia adibita alla guida, affiancata su entrambi i lati da aree di parcheggio parallele alla corsia di marcia. Le misure delle aree di parcheggio sono riportate in Figura 5.6 e sono tali da permettere ai veicoli di sistemarsi in maniera autonoma, con un buon margine di scostamento da eventuali veicoli parcheggiati nelle aree adiacenti.

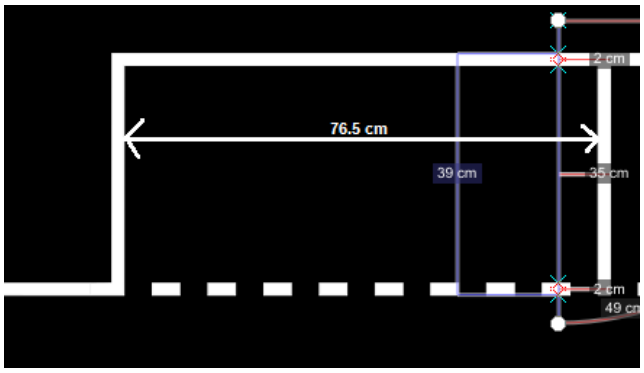


Figura 5.6.: Slot di parcheggio.

Inoltre, durante la competizione, alcune delle aree sono occupate da altri veicoli, per cui la vettura deve essere in grado di, innanzitutto, trovare uno slot libero per il parcheggio, e successivamente posizionarsi al suo interno senza urtare eventuali veicoli in sosta nelle vicinanze.

Capitolo 6.

Traiettorie di manovra

6.1. Introduzione

In riferimento alla sezione 3.2 ed in particolare alla Figura 3.1, il controllo laterale del veicolo è gestito da diversi moduli dell'architettura.

In particolare, i moduli possono essere divisi in due categorie:

- Mantenimento della corsia: in questa categoria ricade il modulo di lane keeping, il quale si occupa di mantenere il veicolo al centro della corsia durante la marcia;
- Manovre: questa categoria, contenente i moduli di lane switching (cambio corsia), parking (parcheggio) ed intersection navigation (navigazione degli incroci), raggruppa quelle azioni che il veicolo deve compiere in risposta a particolari stimoli esterni.

Una sostanziale differenza tra la prima e la seconda categoria risiede nel fatto che il mantenimento della corsia è in azione durante la maggior parte dell'autonomia del veicolo, mentre le manovre vengono eseguite in maniera discreta in risposta alle scelte del blocco decisionale, che si occupa di valutare le azioni che il veicolo deve compiere in reazione agli stimoli forniti dall'ambiente esterno.

Nel contesto di questo elaborato verranno analizzate a fondo solamente le traiettorie appartenenti alla seconda categoria, in quanto gli algoritmi di mantenimento della corsia richiedono un approfondimento ulteriore sui sistemi di visione del veicolo, i quali non sono pertinenti nella trattazione corrente

In questo capitolo verrà formalizzata la costruzione delle traiettorie di riferimento impiegate per il controllo laterale del veicolo, presentando la loro formulazione analitica e i parametri utilizzati a tale fine.

In particolare, verranno descritti gli scenari a curvatura costante, rappresentati dalle traiettorie necessarie per la navigazione degli incroci, per poi trattare gli scenari di parcheggio e di cambio corsia, che presentano delle traiettorie caratterizzate da curvature variabili.

Il primo scenario, sebbene di formulazione semplice, fornirà una base solida per la comprensione dei due scenari seguenti.

6.2. Metodologie operative

Per far sì che le traiettorie vengano eseguite correttamente, è necessario fornire al veicolo un riferimento che verrà poi utilizzato dagli algoritmi di controllo per attuare correttamente il servomotore di sterzo, per una durata di tempo idonea e per lo spazio percorso alla velocità desiderata.

Risulta quindi di facile comprensione la necessità di parametrizzare la manovra in funzione di queste due grandezze, che sono dunque:

- Funzione di riferimento: funzione continua che descrive l'andamento del riferimento nel tempo, durante l'esecuzione della manovra;
- Tempo di esecuzione: intervallo di tempo necessario al veicolo per percorrere la la traiettoria di manovra, spostandosi a velocità costante.

In particolare, per via dell'architettura impiegata durante la realizzazione del veicolo, la funzione di riferimento fornisce l'evoluzione ideale dello yaw rate durante l'esecuzione della manovra in funzione della posizione del veicolo. Questo riferimento può essere comparato con quello misurato dal sensore inerziale (IMU) e in base all'errore rispetto ad esso viene attuato il servomotore. Sebbene il ciclo di controllo verrà esplorato in un capitolo successivo, è necessario fornire questa breve introduzione per comprendere le motivazioni dietro a questa particolare implementazione.

È importante precisare il sistema di riferimento adottato per la costruzione delle traiettorie di manovra: in fase di formulazione, queste hanno origine coincidente con l'origine del piano cartesiano. In questo modo, considerando l'asse x del piano cartesiano come coincidente con l'asse longitudinale del veicolo e l'asse y come quello laterale, si ottiene una traiettoria indipendente dalla posizione effettiva del veicolo sulla mappa al momento t in cui essa viene inizializzata.

Le traiettorie che verranno descritte nelle prossime sezioni sono dunque delle curve continue e derivabili, e definiscono le coordinate del veicolo rispetto all'asse x ed y del sistema di riferimento, al variare del tempo t.

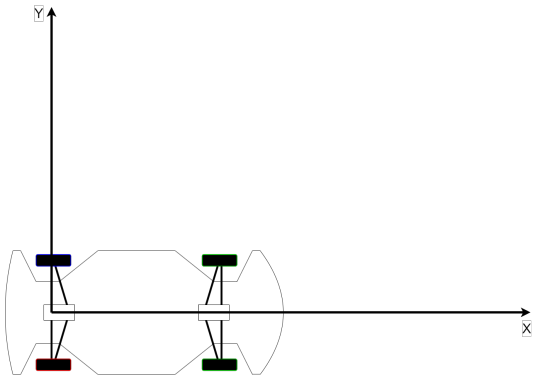


Figura 6.1.: Sistema di riferimento utilizzato per la costruzione delle traiettorie.

6.3. Traiettorie a curvatura costante

Per navigare le intersezioni, sono state costruite delle traiettorie ad arco di circonferenza, congiungenti i centri delle corsie percorse dal veicolo all'ingresso e all'uscita dell'incrocio. In Figura 6.2 è possibile osservare una delle varie traiettorie che il veicolo può percorrere nel contesto di un incrocio a T.

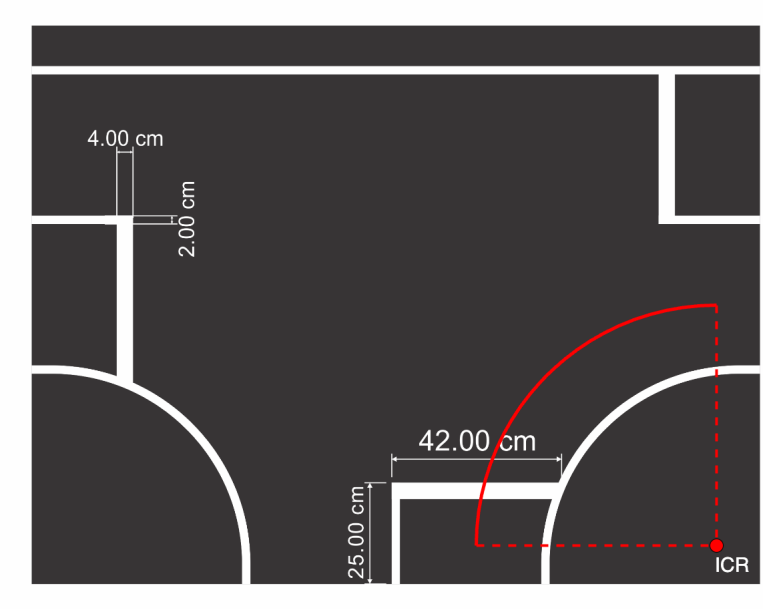


Figura 6.2.: Costruzione di una traiettoria su incrocio a T.

Ricordando la trattazione della sezione precedente, definiamo dunque in maniera analitica la traiettoria, la sua funzione di riferimento e il suo tempo di esecuzione.

6.3.1. Traiettoria

In relazione al sistema di riferimento sopra citato, la formulazione parametrica di questo tipo di traiettoria è quanto segue:

$$\begin{cases} x(t) = C_x + r \cdot \cos(\Theta(t)) \\ y(t) = C_y + r \cdot \sin(\Theta(t)) \end{cases} \tag{6.1}$$

Dove, in notazione vettoriale, il centro della circonferenza è dato dal punto C, ovvero da:

$$C = \begin{pmatrix} C_x \\ C_y \end{pmatrix} \tag{6.2}$$

mentre r rappresenta il raggio della circonferenza e $\Theta(t)$ la variabile che descrive l'angolo percorso.

Assunto che il veicolo si muova lungo il suo asse longitudinale a velocità costante v_x, ed essa sia tangente alla traiettoria, si ha che banalmente $v_x \equiv \omega$, dove ω corrisponde alla velocità tangenziale di un punto materiale che si sposta sulla traiettoria e dunque la velocità angolare attorno all'ICR. Alla luce di questa relazione, ed essendo $\Theta(t) = \omega \cdot t + \Theta_0$, è possibile sostituire quest'ultima nella formula iniziale, ottenendo dunque:

$$\begin{cases} x(t) = C_x + r \cdot \cos(\omega \cdot t + \Theta_0) \\ y(t) = C_y + r \cdot \sin(\omega \cdot t + \Theta_0) \end{cases} \tag{6.3}$$

Quest'ultima rappresenta la relazione che permette di descrivere la traiettoria sul piano cartesiano, su cui si baseranno, oltretutto, gli algoritmi di controllo per l'esecuzione di questa particolare manovra. Prendendo come esempio la traiettoria rappresentata in Figura 6.2, essa può essere rappresentata come un arco di circonferenza sotteso a un angolo di 90°, di raggio 60cm, come mostrato in Figura 6.3.

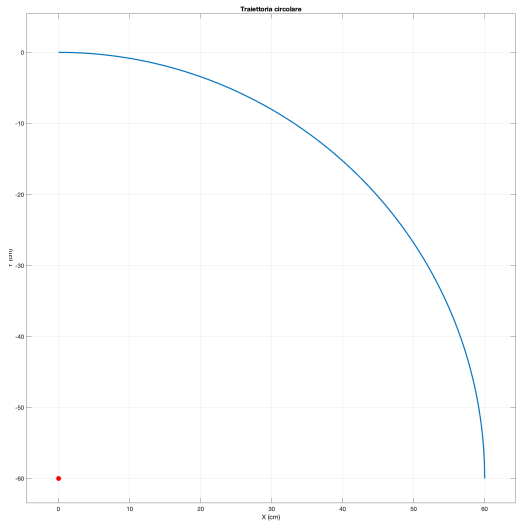


Figura 6.3.: Traiettoria ad arco di circonferenza.

Nella figura precedente è rappresentata una curva verso destra, ma lo stesso vale per una curva verso sinistra, che può essere facilmente ottenuta modificando i valori di raggio, angolo iniziale e tempo di esecuzione, oltre che modificando le coordinate del centro della circonferenza.

Per quanto riguarda le coordinate del centro, esse sono sempre posizionate sull'asse y delle ordinate e il valore della componente su tale asse ha modulo pari a quello del raggio di circonferenza e segno dipendente dalla direzione della curva.

6.3.2. Funzione di riferimento

Questo tipo di manovra è quello più semplice in termini di formulazione matematica della funzione di riferimento, in quanto deriva direttamente dalla definizione del modello cinematico del veicolo.

Poiché la funzione di riferimento per il controllo del veicolo è rappresentata dall'evoluzione nel tempo del valore dello yaw rate in funzione del punto della traiettoria in cui si trova il veicolo, e per una traiettoria circolare di raggio costante il veicolo ruota attorno al proprio ICR con uno yaw rate costante e pari a:

$$\dot{\psi} = \frac{v}{R} \tag{6.4}$$

Si ha che la relazione (6.4) rappresenta a tutti gli effetti la funzione di riferimento utilizzata nel ciclo di controllo per l'esecuzione di questo tipo di manovra.

Una rielaborazione di questa funzione si può ottenere considerando la relazione che sussiste tra il raggio della circonferenza descritta dalla traiettoria e dalla sua curvatura, che in generale è data da:

$$k = \frac{1}{R} \tag{6.5}$$

Dove k rappresenta la curvatura della traiettoria. Questa relazione è fondamentale per descrivere correttamente traiettorie più complesse, come quelle che verranno trattate nelle sezioni successive

In Figura 6.4, è fornita la rappresentazione grafica della funzione di riferimento per una traiettoria circolare con velocità costante di 20cm/s e di raggio 60cm.

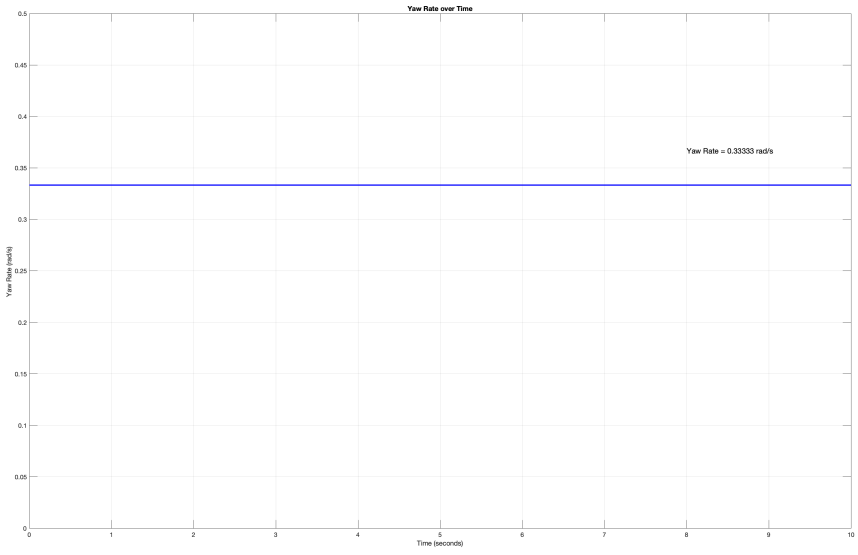


Figura 6.4.: Funzione di riferimento per traiettorie circolari.

6.3.3. Tempo di manovra

Poiché la traiettoria da percorrere descrive un arco di circonferenza, in questo caso la trattazione risulta essere molto semplice.

Infatti, è sufficiente ricavare la lunghezza dell'arco di circonferenza, ad esempio, in funzione dell'angolo a cui è sottesa. In questo caso si ha che:

$$L = \Theta \cdot R \tag{6.6}$$

dove si hanno:

- L: lunghezza dell'arco;
- Θ: angolo a cui è sotteso l'arco;
- R: raggio di circonferenza.

Poiché il veicolo si sposta a velocità tangenziale costante, il tempo di manovra si può facilmente ottenere applicando la relazione:

$$t_m = \frac{L}{v_x} \tag{6.7}$$

Prendendo come esempio la manovra descritta in Figura 6.2, si ha che per un raggio di 60cm, percorso ad una velocità di 20cm/s, il tempo di manovra corrisponde a:

$$t_m = \frac{\pi/2 \cdot 0.60}{0.20} = 4.71s \tag{6.8}$$

Da questo calcolo, è possibile adattare quindi la funzione di riferimento al tempo di manovra richiesto, ottenendo dunque graficamente il risultato mostrato in Figura 6.5.

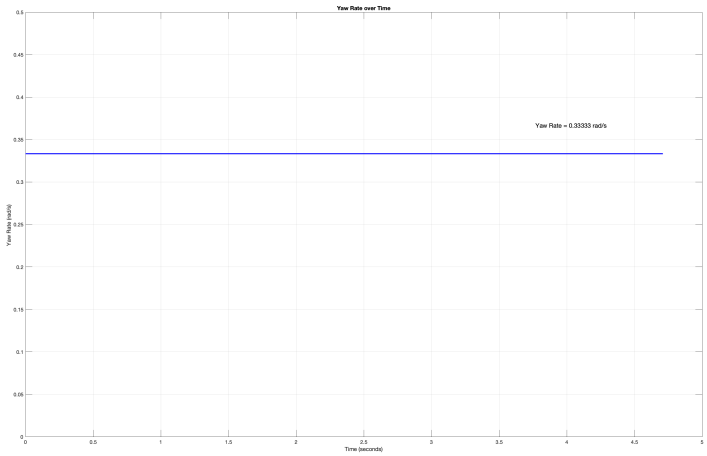


Figura 6.5.: Funzione di riferimento con tempo di manovra.

6.4. Traiettorie a curvatura variabile

Questo tipo di traiettoria è caratteristica delle manovre di sorpasso e di parcheggio, ed è in entrambi i casi riconducibile ad una traiettoria con un profilo sinusoidale. A causa di questa caratteristica, la curvatura di questa particolare traiettoria non ha un valore costante ed è dunque necessario analizzare in modo approfondito la sua formulazione.

La complessità di questo tipo di manovra è situata dunque nella sua costruzione e poiché alcuni aspetti di essa non possono essere ricavati in forma chiusa, sono state utilizzate alcune semplificazioni.

In Figura 6.6 è mostrato uno scenario di cambio corsia, caratterizzato da una traiettoria di tipo sinusoidale.

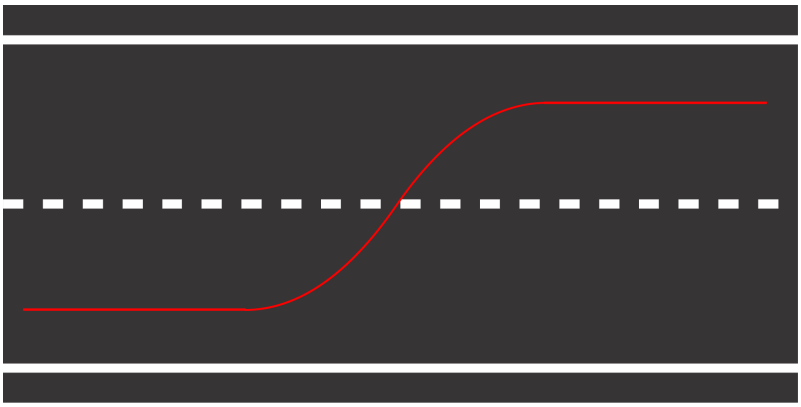


Figura 6.6.: Manovra di cambio corsia.

6.4.1. Traiettoria

A differenza della trattazione riguardante la traiettoria a curvatura costante, in questo caso conviene definire lo spostamento del veicolo inizialmente in termini del suo profilo di velocità, piuttosto che il suo spostamento.

Mediante operazioni di integrazione, è successivamente possibile ottenere le equazioni parametriche della traiettoria del veicolo.

Pertanto, le equazioni che definiscono l'evoluzione della velocità del veicolo nel tempo, in termini di velocità laterale e longitudinale, sono:

$$\begin{cases} \dot{y}(t) = v_y \cdot \sin\left(2\pi f t - \frac{\pi}{2}\right) + v_y \\ \dot{x}(t) = \sqrt{v^2 + \dot{y}(t)^2} \end{cases} \tag{6.9}$$

Nello specifico, poiché la frequenza f è esprimibile anche in termini della velocità v e della lunghezza d'onda λ, corrispondente allo scostamento longitudinale, si ottiene la seguente relazione:

$$f = \frac{v}{\lambda} \tag{6.10}$$

Inoltre, la velocità laterale v_y può essere espressa in termini dello scostamento laterale Δy, lunghezza del percorso l e velocità tangenziale v. Vale infatti la seguente relazione:

$$v_y = \frac{\Delta y \cdot v}{\lambda} \tag{6.11}$$

Sostituendo le relazioni 6.10 e 6.11 nella relazione 6.9 si ottiene dunque il seguente sistema di equazioni parametriche:

$$\begin{cases} \dot{y}(t) = \frac{\Delta y \cdot v}{\lambda} \cdot \sin\left(2\pi \frac{v}{\lambda} t - \frac{\pi}{2}\right) + \frac{\Delta y \cdot v}{\lambda} \\ \dot{x}(t) = \sqrt{v + \dot{y}(t)^2} \end{cases} \tag{6.12}$$

Integrando il sistema rispetto al tempo si ricava il sistema di equazioni che descrive la traiettoria del veicolo. Questa traiettoria è parametrizzata in funzione della velocità del veicolo, dallo scostamento laterale Δy, dello scostamento longitudinale λ e della velocità v del veicolo, assunta costante.

In Figura 6.7 è mostrata una traiettoria caratterizzata da $\Delta y = 0.35m$, $\lambda = 1.14m$ e da $v = 0.20m/s$.

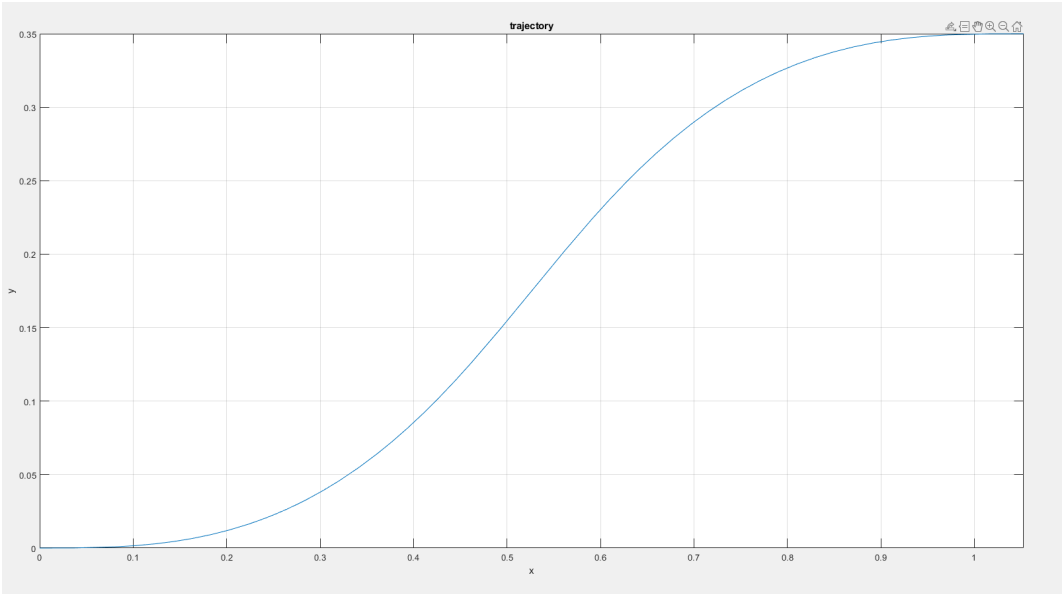


Figura 6.7.: Traiettoria di cambio corsia.

6.4.2. Funzione di riferimento

Come introdotto in precedenza, la funzione di riferimento utilizzata per il controllo del veicolo è basata sull'evoluzione ideale nel corso del tempo dello yaw rate. A differenza del caso a curvatura costante, in questo caso la funzione di riferimento utilizzata non è costante nel tempo e neppure di natura lineare. Questa funzione può essere ricavata in più modi, di cui ne verrà trattato uno solo. Dalla relazione:

$$\dot{\psi} = \frac{v}{R} \tag{6.13}$$

Ottenuta dal modello cinematico del veicolo riportato nella sezione 4.4.3, è possibile ottenere la funzione dello yaw rate nel tempo.

Ricordando che:

$$k = \frac{1}{R} \tag{6.14}$$

e sostituendola nella Relazione 6.13, si ottiene:

$$k(t) = \frac{\dot{\psi}(t)}{v} \tag{6.15}$$

La relazione 6.15, che lega la curvatura della traiettoria all'evoluzione dello yaw rate del veicolo nel tempo, permette di definire in modo rigoroso una funzione che correla la cinematica del veicolo con la sua dinamica laterale.

Poiché lo yaw del veicolo rappresenta la deviazione della direzione longitudinale del veicolo rispetto all'asse x del sistema di riferimento inerziale, esso può essere ottenuto dalla dinamica laterale del veicolo, ovvero invertendo la funzione $\dot{y}(t)$, ricavando:

$$\psi(t) = \arcsin\left(\frac{\dot{y}(t)}{v}\right) \tag{6.16}$$

Per ricavare la funzione di riferimento, è sufficiente derivare la funzione dello yaw, ottenendo:

$$\dot{\psi}(t) = \frac{d\psi(t)}{dt} = \frac{d\left(\arcsin\left(\frac{\dot{y}(t)}{v}\right)\right)}{dt} \tag{6.17}$$

Sviluppando la relazione appena ottenuta, si ottiene la seguente funzione:

$$\dot{\psi}(t) = \frac{2 \cdot \Delta y \cdot v \cdot \pi \cdot \sin\left(\frac{2 \cdot \pi \cdot t \cdot v}{\lambda}\right)}{\lambda^2 \cdot \sqrt{1 - \frac{\Delta y^2 \cdot \left(\cos\left(\frac{2 \cdot t \cdot v \cdot \pi}{\lambda} - 1\right)\right)^2}{\lambda^2}}} \tag{6.18}$$

È facilmente deducibile, che anche la funzione di riferimento dello yaw rate è parametrizzata relativamente allo scostamento laterale, a quello longitudinale e alla velocità longitudinale del veicolo, che è assunta costante.

Graficamente, in Figura 6.8 è riportato il grafico di $\dot{\psi}(t)$ relativo alla traiettoria descritta in Figura 6.7, avente quindi gli stessi valori di Δy, λ e v.

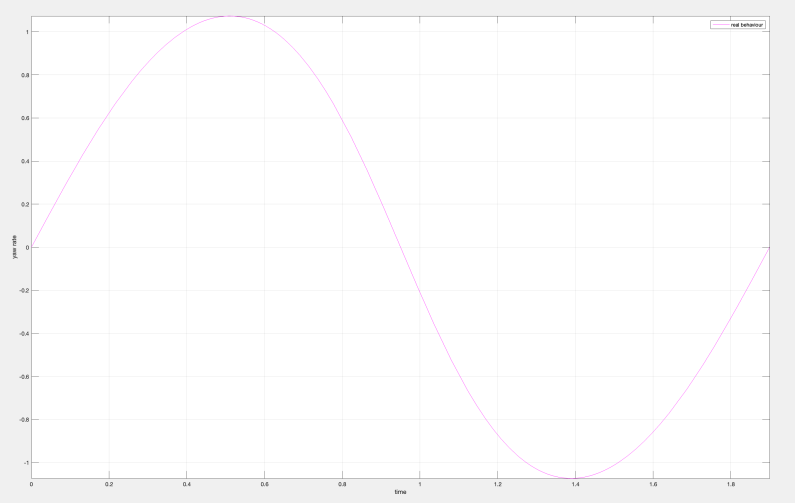


Figura 6.8.: Funzione di riferimento, curvatura variabile.

Analizzando la funzione $\dot{y}(t)$, si osserva che essa presenta un punto di discontinuità di prima specie se si verifica la seguente relazione:

$$\lambda \leq 2 \cdot \Delta y \tag{6.19}$$

ossia quando lo scostamento longitudinale risulta essere minore o uguale rispetto al doppio di quello laterale. Graficamente si ottiene quanto descritto in Figura 6.9.

Osservando la traiettoria ideale descritta dal veicolo, rappresentata in Figura 6.10, è possibile notare come in corrispondenza del punto di discontinuità della funzione di riferimento sia presente un punto di non derivabilità.

Un'osservazione che è possibile fare in merito alla condizione di inammissibilità è che la traiettoria risulta comunque essere definita nel piano, ma la funzione di riferimento ad essa associata, essendo discontinua, risulta in una brusca esecuzione della manovra, in quanto lo sterzo viene azionato in maniera estremamente rapida.

Per questa ragione, nell'affrontare gli scenari di guida della competizione, si è sempre cercato di costruire le traiettorie in modo da non presentare questo tipo di comportamento.

6.4.3. Tempo di manovra

A differenza delle traiettorie a curvatura costante, per questo tipo di traiettoria risulta complesso determinare il tempo di esecuzione. Tale difficoltà è dovuta alla

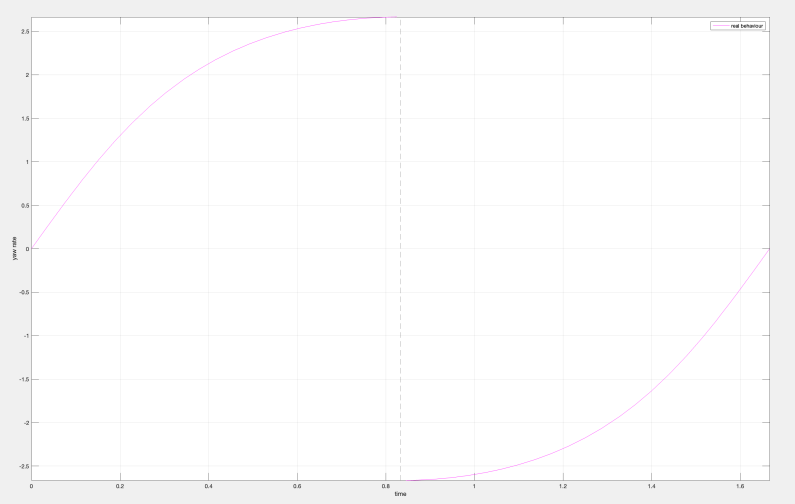


Figura 6.9.: Condizione inammissibile della funzione di riferimento.

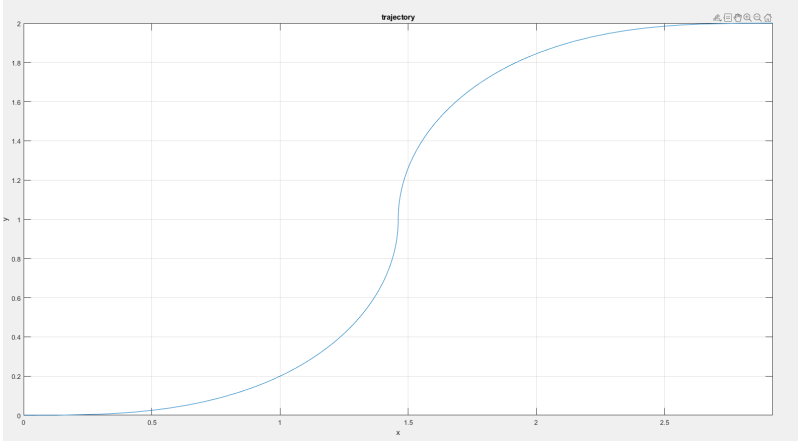


Figura 6.10.: Traiettoria, condizione inammissibile.

complessità nel calcolare in forma chiusa la lunghezza della curva che descrive la traiettoria del veicolo, la quale è espressa nella forma:

$$y(t) = a \cdot sin(\omega t + \phi) + c \tag{6.20}$$

Per calcolare la lunghezza di una curva L in $\mathbb{R}^2$ è sufficiente utilizzare la formula che segue:

$$L = \int_{t_0}^{t_1} \sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2}\, dt \tag{6.21}$$

Poiché, nel caso della traiettoria, lo spostamento lungo l'asse x è definito come costante, ovvero come:

$$x(t) = t \tag{6.22}$$

Calcolando le derivate delle funzioni $x(t)$ e $y(t)$, si ottiene rispettivamente:

$$\frac{dx}{dt} = 1 \tag{6.23}$$

$$\frac{dy}{dt} = a\omega \cdot \cos(\omega t + \phi) \tag{6.24}$$

Sostituendo i risultati ottenuti nelle relazioni (6.23) e (6.24) all'interno della relazione (6.21), si ottiene quanto segue:

$$L = \int_0^t \sqrt{1 + a^2\omega^2 \cdot \cos(\omega t + \phi)^2} dt \tag{6.25}$$

La relazione (6.25) rappresenta una generalizzazione dell'integrale ellittico di seconda specie, la cui soluzione non è esprimibile in forma chiusa in termini di funzioni elementari. La soluzione di questo integrale può essere ricavata numericamente con algoritmi di approssimazione quali quello dei trapezi o Simpson.

Poiché l'efficienza computazionale è cruciale nelle applicazioni di controllo in tempo reale, è stata adottata un'altra soluzione che, pur non fornendo un valore esatto, si è dimostrata comunque efficace nel controllo del veicolo.

Pertanto, per calcolare il tempo necessario per l'esecuzione della manovra, è stato costruito un rettangolo avente una base della stessa misura dello scostamento longitudinale (dx) ed un'altezza pari allo scostamento laterale (dy).

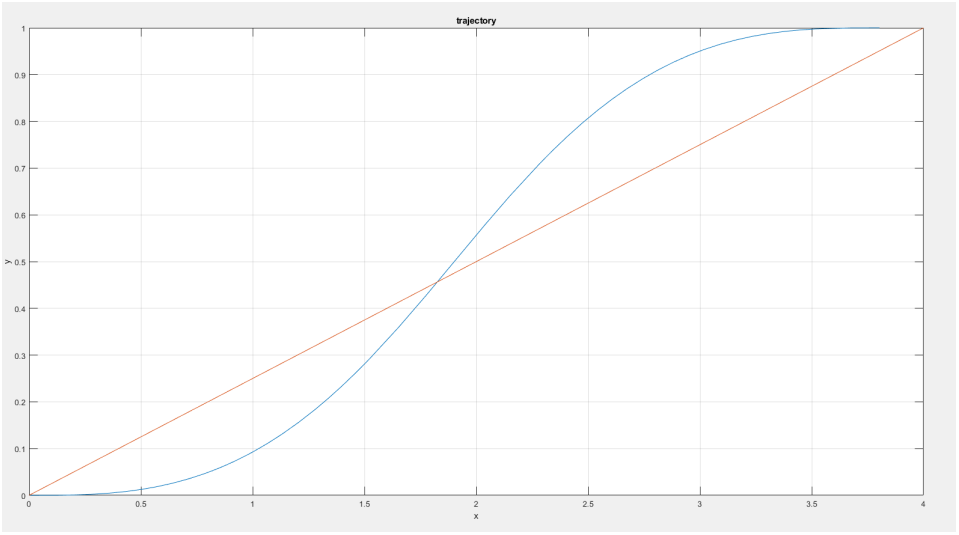


Figura 6.11.: Traiettoria semplificata.

La lunghezza della traiettoria semplificata è stata calcolata ricavando la lunghezza

della diagonale del rettangolo illustrato in Figura 6.11, utilizzando la seguente relazione:

$$L = \sqrt{dx^2 + dy^2} \tag{6.26}$$

A seguito di questa semplificazione, il tempo di manovra viene calcolato mediante la relazione:

$$t_m = \frac{L}{v} = \frac{\sqrt{dx^2 + dy^2}}{v} \tag{6.27}$$

6.4.4. Considerazioni

Nel contesto della progettazione di algoritmi di controllo per veicoli in scala, un'implementazione semplificata come quella descritta finora può risultare adeguata. Nonostante l'imprecisione dei parametri di controllo, questa risulta trascurabile poiché non incide in modo significativo sul comportamento effettivo del veicolo. Questo è vero perché la meccanica del veicolo utilizzato presenta tolleranze piuttosto lasche, che non permetterebbero in ogni caso un azionamento preciso in termini di ripetibilità sperimentale.

Nel caso di implementazione di questo tipo di algoritmo su un veicolo autonomo destinato all'uso civile, sarebbe indispensabile adottare tecniche di controllo avanzate e integrare misure di sicurezza aggiuntive per prevenire potenziali danni a persone e oggetti circostanti

Parte IV.

Implementazione

Capitolo 7.

Generalità sull'implementazione

Come trattato in precedenza, in particolare nel capitolo 3, il controllo del veicolo è stato delegato a due dispositivi, che si occupano di esso sotto due aree semantiche differenti.La componente decisionale è affidata in particolare alla SBC Rapsberry Pi 4 responsabile della generazione dei vari valori di riferimento necessari per il controllo del veicolo. D'altro canto, il microcontrollore STM32F401RE si occupa dell'interpretazione di tali riferimenti, generando segnali di controllo per i vari attuatori e permettendo così il moto del veicolo.

Nei capitoli successivi verranno illustrate le implementazioni degli algoritmi di controllo adibiti all'esecuzione delle manovre trattate fino ad ora, sia ad "alto livello" ovvero gli algoritmi implementati su Raspberry Pi 4, che a "basso livello", illustrando gli algoritmi implementati sul microcontrollore STM32F401RE.

In particolare, la trattazione continuerà utilizzando una strategia bottom-up, descrivendo dapprima gli algoritmi sviluppati su microcontrollore, per poi trattare l'implementazione su Raspberry Pi.

Questa scelta è data dal fatto che per comprendere appieno le motivazioni dietro alla formulazione degli algoritmi sviluppati su Raspberry è necessario comprendere il funzionamento degli algoritmi su microcontrollore, in quanto essi si occupano dell'attuazione del veicolo e necessitano di particolari input per svolgere tale compito.

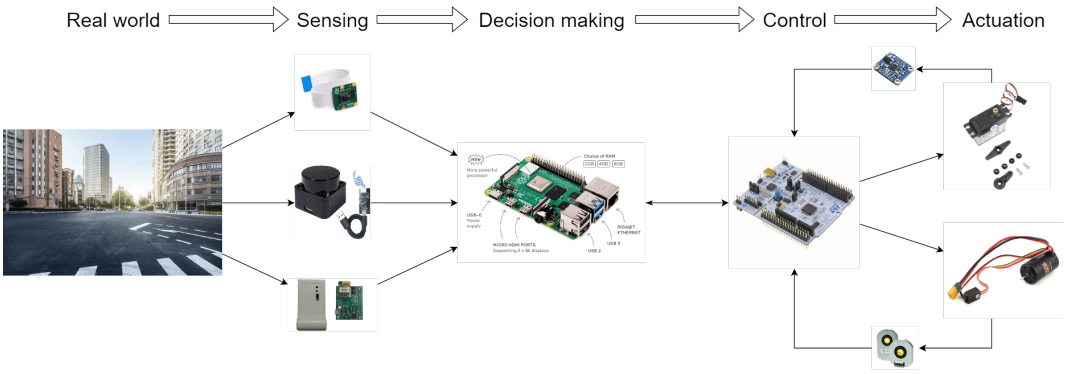


Figura 7.1.: Schema riassuntivo del flusso dei dati.

Capitolo 8.

Implementazione su microcontrollore

8.1. Introduzione

In questo capitolo verranno trattati i vari algoritmi di controllo sviluppati sul microcontrollore dal punto di vista della loro implementazione. Tutti gli algoritmi sono stati sviluppati utilizzando il linguaggio di programmazione C, utilizzando l'ambiente di sviluppo (IDE) STM32CubeIDE 1.13.2, fornito da STMicroelectronics NV. Questo IDE fornisce vari tool che facilitano lo sviluppo di codice per i microcontrollori della famiglia STM32, quali clock tree e un'interfaccia grafica che facilita l'attivazione e l'assegnamento delle varie funzionalità ai pin del microcontrollore (IOC).

Nelle sezioni successive verranno esaminati l'architettura del sistema sviluppato in ambito embedded, considerando sia la gestione di input e output esterni, sia il funzionamento interno, illustrato attraverso i vari algoritmi implementati.

Verrà dunque trattato il funzionamento del dispositivo dal punto di visto della sua macchina a stati, gli algoritmi sviluppati per la comunicazione, sia in ricezione che in trasmissione, il ciclo di controllo degli attuatori, con particolare focus sul controllo della trazione e quello del servosterzo.

8.2. Flusso di dati e macchina a stati

In questa sezione verrà esaminato il sistema costituito dal microcontrollore, sia dal punto di vista esterno con un'analisi dei suoi ingressi ed uscite, sia dal punto di vista interno, attraverso una valutazione del funzionamento come macchina a stati.

Dal punto di vista esterno, il microcontrollore può essere visto come una scatola nera che presenta i seguenti ingressi ed uscite:

- **Pulsante Blu**: questo pulsante, integrato al microcontrollore, controlla a livello hardware lo stato di funzionamento del veicolo. In particolare, permette di interrompere la comunicazione con la Raspberry Pi 4;
- **Valori di riferimento**: questo ingresso, fornito per comunicazione seriale dalla Raspberry Pi 4, è rappresentato dai valori di riferimento di velocità, curvatura, e una flag di controllo dello stato del veicolo;

- **Segnale di controllo per il motore di trazione**: questo segnale di controllo, rappresentato da un segnale elettrico di tipo PWM, viene generato dal microcontrollore per far si che la velocità del motore sia quella corretta;
- **Segnale di controllo per il servomotore**: questo segnale di controllo, anch'esso di tipo PWM, viene generato per assegnare il corretto angolo di sterzo al servomotore;
- **Telemetria**: messaggio generato periodicamente a frequenza costante, comunica per via seriale lo stato del veicolo, rappresentato dalle misurazioni fornite dai sensori a bordo;
- **LED di stato**: seppur non essendo una vera e propria uscita del sistema, lo stato del LED integrato della scheda offre un'indicazione visiva all'utente, permettendogli di comprendere lo stato attuale del veicolo. Questo aspetto verrà discusso nei prossimi paragrafi.

In Figura 8.1 è mostrato uno schema riassuntivo che evidenzia i vari input ed output in ingresso ed uscita dal microcontrollore.

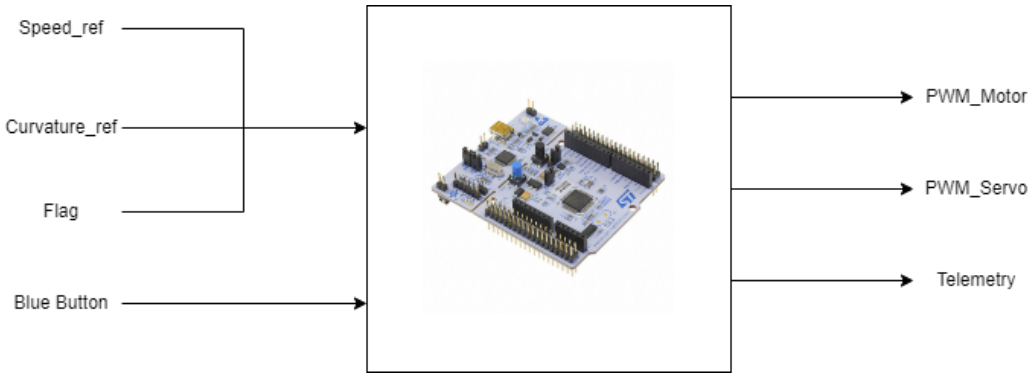


Figura 8.1.: Flusso di dati in ingresso ed uscita.

Analizzando invece il funzionamento interno del sistema, è possibile individuare in che modo esso sia strutturato come una macchina a stati finiti.

Come riportato in Figura 8.2, sono presenti cinque stati, divisi in tre aree funzionali. In particolare, si hanno:

- **Stato di Listening**: questo stato, in cui si trova il veicolo nel momento in cui viene fornita per la prima volta l'alimentazione, fa si che questo ascolti i messaggi inviati dalla Raspberry Pi 4, contenenti i valori di riferimento per l'azionamento, eseguendo i comandi da essa forniti. Questo stato è identificabile dal fatto che il LED di stato risulta essere spento;
- **Stato di Not Listening**: questo stato è raggiungibile dallo stato di listening mediante la pressione del pulsante blu per una durata inferiore ai 3s. Questo stato è identificabile dall'accensione del LED di stato ed è utile per interrompere

immediatamente il moto del veicolo in caso di erronea esecuzione delle manovre, in quanto arresta il motore di trazione e riporta alla posizione di riposo il servomotore di sterzo;

- **Procedura di calibrazione**: il motore brushless, a seguito dell'accensione, necessità di una calibrazione che permette di modificarne la curva caratteristica di ingresso-uscita. Per fare ciò è necessario fornire i valori minimi, neutri e massimi di PWM durante una procedura di calibrazione. L'insieme di questi tre stati rappresenta la sequenza di calibrazione, attivata tenendo premuto il pulsante blu per più di 3s, identificata dal LED di stato lampeggiante e dal movimento dello sterzo in corrispondenza delle tre fasi di calibrazione;

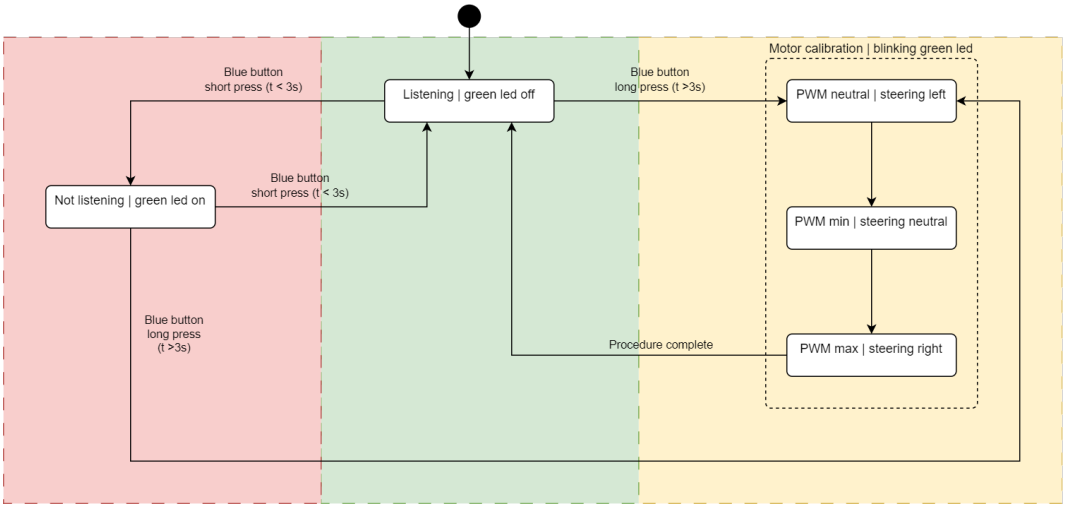


Figura 8.2.: Macchina a stati.

Seguendo la trattazione, è facilmente deducibile che il sistema presenta due tipi di ingressi, distinti tra hardware e software. A livello hardware si ha l'interazione dell'operatore umano con la scheda mediante il tasto blu, mentre a livello software sono presenti i segnali di controllo ricevuti in ingresso mediante comunicazione seriale. Questi due ingressi rappresentano una ridondanza, che serve a prevenire eventuali danni a persone o cose nel caso di malfunzionamento dei sistemi, in quanto l'operatore può arrestare manualmente il veicolo mediante la transizione allo stato di "Not listening".

8.3. Comunicazione

La comunicazione tra il microcontrollore e il mondo esterno è cruciale per il controllo del veicolo. In particolare, due concetti rivestono un'importanza particolare:

- **Ricezione dei riferimenti**: i riferimenti di velocità e curvatura, forniti dalla Raspberry Pi, devono essere processati tempestivamente e convertiti in segnali di controllo per gli attuatori;
- **Invio dei dati di telemetria**: questi dati, ricavati dalla lettura dei sensori a bordo, devono essere trasmessi a frequenza costante alla Raspberry Pi in quanto sono fondamentali per gli algoritmi di auto-localizzazione.

A causa di queste esigenze, è stato implementato un sistema di comunicazione seriale asincrono (UART) tra il microcontrollore e la Raspberry, sia in ricezione che in trasmissione, utilizzando la configurazione Full Duplex.

8.3.1. Implementazione

Per quanto riguarda il microcontrollore, il setup di questa periferica risulta essere molto semplice, in quanto l'IDE fornisce vari strumenti grafici per conseguire tale scopo. Di seguito verrà mostrata la procedura per inizializzare e configurare la comunicazione sia in ingresso che in uscita.

Configurazione dell'IOC

L'IDE utilizzato fornisce uno strumento di Configurazione di Input e Output (IOC), che permette di configurare il microcontrollore con semplicità, grazie ad un'interfaccia grafica. A seguito della configurazione all'interno di questo ambiente verrà generato automaticamente il codice necessario per il funzionamento richiesto in fase di configurazione.

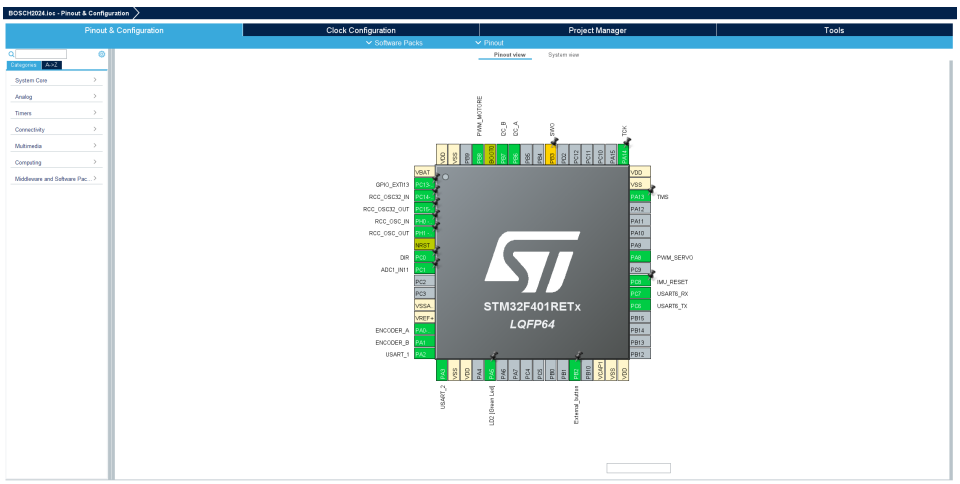


Figura 8.3.: IOC dell'IDE STM32CubeIDE.

Accedendo alla sezione "Connectivity", è possibile scegliere tra le varie porte di comunicazione del dispositivo. Selezionano la porta USART6, si ha la possibilità di attivarla e configurarne le impostazioni.

In particolare, per questa porta è stato utilizzato un Baud Rate di 115200 Bits/s, con una lunghezza di parola di 8 Bit, senza bit di parità e con un singolo bit di stop. Come anticipato, la porta è stata configurata in Full Duplex, abilitando dunque sia la lettura che la scrittura dei messaggi.

Una particolarità della configurazione è l'abilitazione del DMA in ingresso, per permettere una ricezione dei messaggi non bloccante, in quanto è necessaria la massima efficienza per l'esecuzione degli algoritmi di controllo; Questo aspetto verrà discusso nei paragrafi successivi.

Per quanto riguarda la configurazione hardware, sono stati assegnati i pin GPIO PC6 per la trasmissione dei dati e PC7 per la loro ricezione: a questi pin saranno poi collegati un jumper ciascuno, stabilendo dunque una connessione hardware con la Raspberry Pi.

Nelle Figure 8.4, 8.5, 8.6 e 8.7 sono riportate le impostazioni di configurazione dell'IOC.

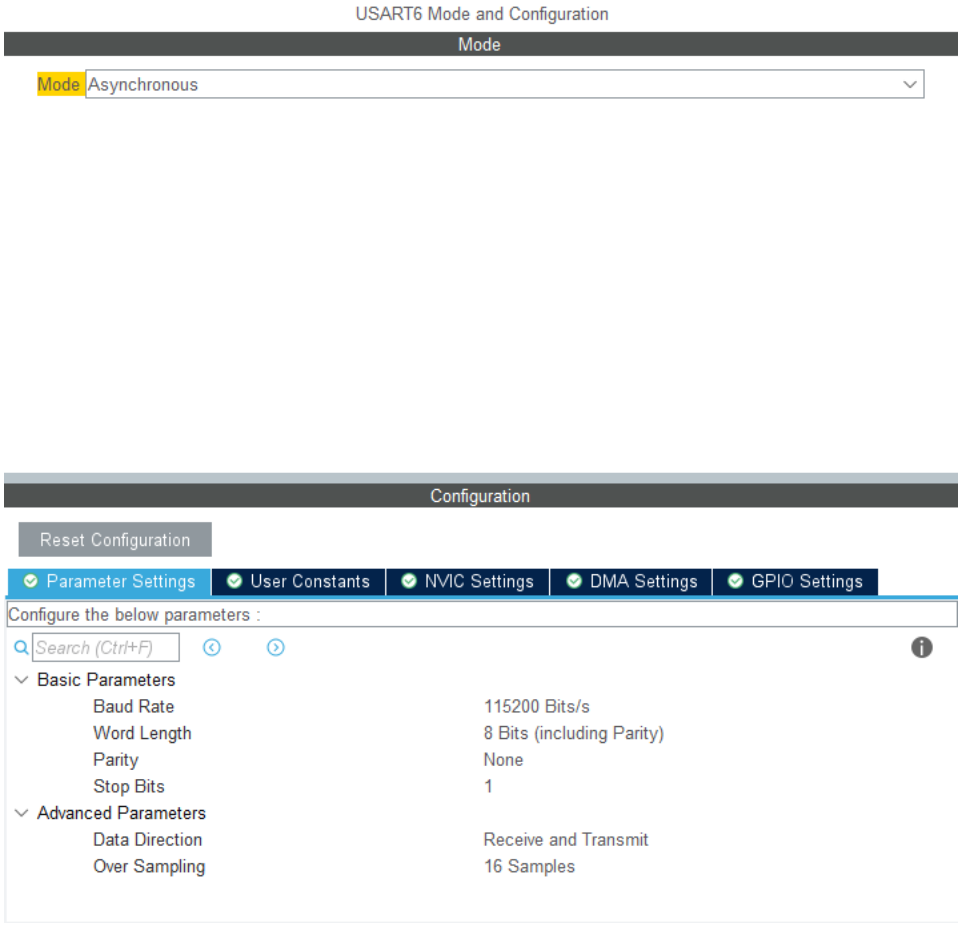


Figura 8.4.: USART6, configurazioni generali.

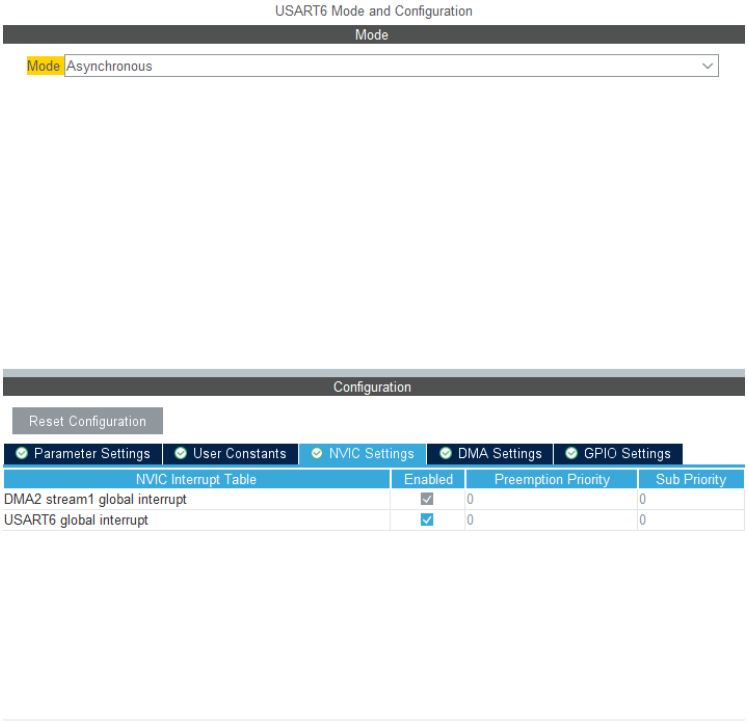


Figura 8.5.: USART6, configurazione degli interrupt.

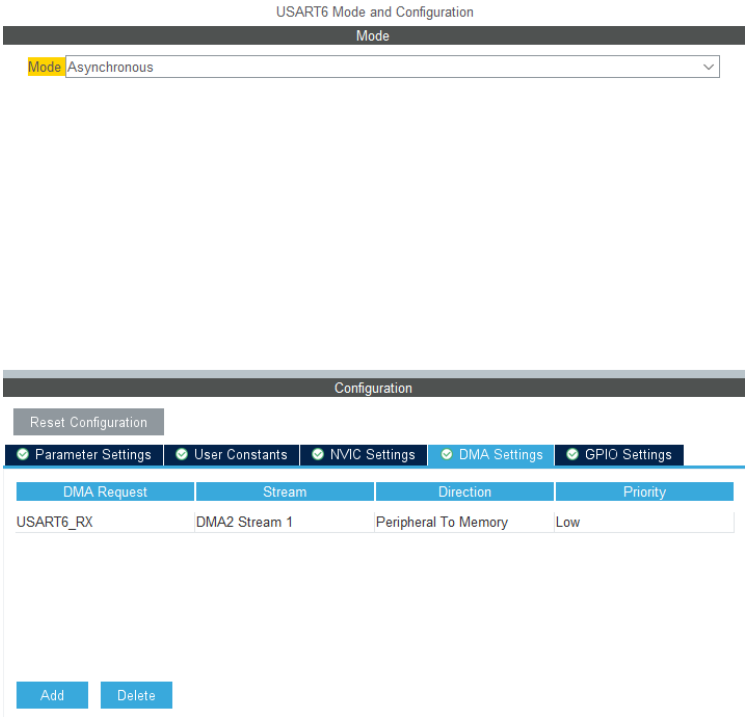


Figura 8.6.: USART6, configurazione del DMA.

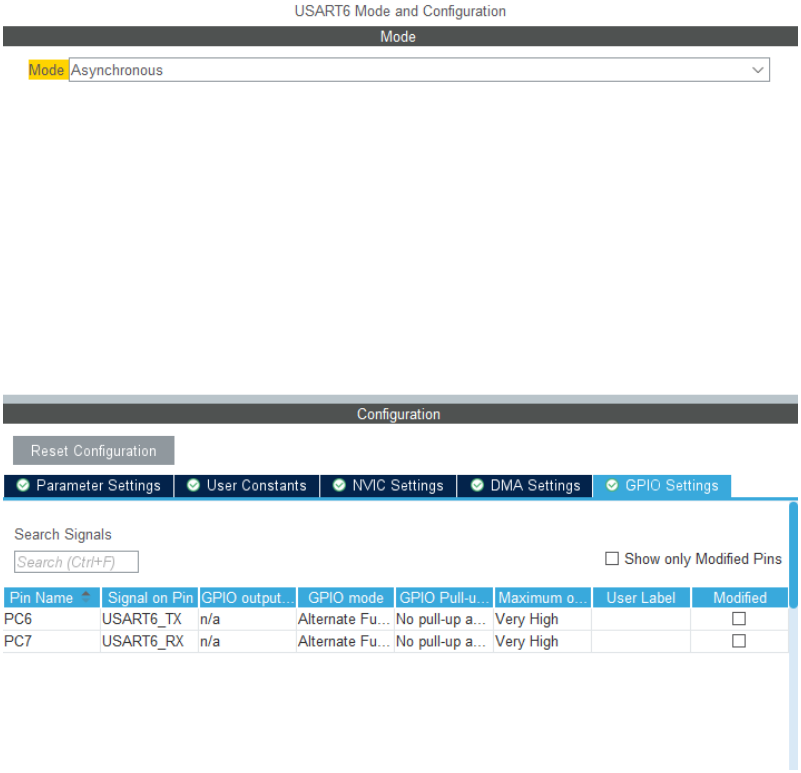


Figura 8.7.: USART6, pinout

Codice sorgente

Il codice sorgente sviluppato per implementare la comunicazione seriale tra i due dispositivi è stato implementato nel file main.c, assieme ad una libreria ideata ad hoc per l'interpretazione delle stringhe ricevute in input, denominata RX_UART.c, con il suo rispettivo header file. Verranno analizzate, nei prossimi paragrafi, le varie strutture dati, funzioni e metodi progettati per la realizzazione della comunicazione.

Strutture dati

Come introdotto in precedenza, il microcontrollore deve ricevere in ingresso i riferimenti e le flag di controllo, comunicando a sua volta i dati telemetrici misurati dai propri sensori.

Per i dati ricevuti in ingresso è stata creata una struct denominata SerialDataRx. Di seguito se ne riporta il codice:

```
typedef struct SerialDataRX {
  float curvature_radius_ref_m; // [m]
  float linear_speed_ref_m_s; //[m/s]
  int enable; //Start Flag [bool]
  float offset;
} serialDataRX;
```

Listing 8.1: SerialDataRX

In particolare, questa struttura dati contiene:

- **curvature_radius_ref_m**: misura, espressa in metri, del raggio di curvatura istantaneo della traiettoria da percorrere;
- **linear_speed_ref_m_s**: velocità longitudinale di riferimento a cui si deve spostare la macchina;
- **enable**: flag di controllo software che abilita o disabilita il moto del veicolo;
- offset: misura, espressa in metri, del discostamento della traiettoria corrente da quella ideale.

Per quanto riguarda i dati trasmessi dal microcontrollore, anch'essi sono stati raggruppati in una struttura dati denominata SerialDataTX, di cui è riportato di seguito il codice:

```
typedef struct SerialDataTX {
  float current_speed_rpm; // [m]
  float current_servo_angle_deg; // [m/s]
  float current_yaw_rate_deg_sec; // [deg/s]
  float quaternion_x;
  float quaternion_y;
  float quaternion_z;
  float quaternion_w;
  float magne_x;
  float magne_y;
  float magne_z;
  float accel_x;
  float accel_y;
  float accel_z;
  float angle_x;
  float angle_y;
  float angle_z;
} serialDataTX;
```

Listing 8.2: SerialDataTX

In questa struttura dati sono racchiusi tutti i dati telemetrici forniti dal veicolo, utili agli algoritmi di auto-localizzazione implementati su Raspberry Pi per computare la posizione e l'orientazione corrente del veicolo. Essi sono costituiti da:

- Velocità attuale;
- Angolo di sterzo attuale;
- Yaw rate attuale;
- Quaternioni;
- Intensità del campo magnetico lungo i 3 assi;
- Accelerazione lungo i 3 assi;
- Angoli di Eulero lungo i 3 assi, forniti dal giroscopio.

Ricezione dei dati

Come già anticipato, la ricezione dei dati avviene mediante l'utilizzo del Direct Memory Access (DMA), che si occupa di svolgere in parallelo al ciclo di controllo la ricezione dei dati forniti in ingresso sulla porta seriale. Questo aspetto è fondamentale in quanto la lettura dei dati sulla porta seriale è un processo bloccante e interrompe la normale esecuzione del codice di controllo degli attuatori. Infatti, il codice di controllo dei sistemi di attuazione verrebbe interrotto alla ricezione di ogni carattere, allungando considerevolmente il tempo di esecuzione delle operazioni che determinano il movimento del veicolo. L'uso del DMA, invece, permette al codice di controllo di proseguire senza interruzioni, aggiornando i dati ricevuti ogni volta che un nuovo messaggio è disponibile, ossia ogni 10ms.

Questa funzionalità è implementata attraverso una funzione disponibile solo a seguito della corretta configurazione della porta seriale, denominata HAL_UARTEx_ReceiveToIdle_DMA. Questa funzione prende 3 argomenti, che sono la porta seriale di interesse, un buffer di ricezione dati costituito da un vettore di char, e la grandezza massima del buffer, ovvero la sua lunghezza espressa con un intero.

Quando il buffer si riempie o il il microcontrollore rileva un evento di idle, ossia la fine della ricezione di un messaggio, viene generato un segnale di interrupt. Quesro segnale è gestito dall'interrupt handler, che esegue il codice presente nel metodo di callback associato alla porta seriale. Il codice relativo a tale metodo è riportato di seguito.

```
// COMUNICAZIONE (RICEZIONE DATI)
void HAL_UARTEx_RxEventCallback(UART_HandleTypeDef *huart, uint16_t
    Size){
  if (huart->Instance == USART6){
    cnt_DMA = 0; //reset del counter per il watchdog del timer
    memcpy(msg, RxBuf, Size); //trasferimento dei dati
    float floatArray[MAX_VALUES];  //definizione del vettore dei
    riferimenti
    parseCSV(msg, floatArray); //interpretazione del messaggio

    //eliminazione condizioni inammissibili
    if(floatArray[0] == 0 || floatArray[0] == 1){
      //assegnamento della flag
      dataRX.enable = floatArray[0];
      //assegnamento riferimento di velocita'
      dataRX.linear_speed_ref_m_s = floatArray[1];

      //prevenzione di assegnamento di valori inammissibili
      if(dataRX.curvature_radius_ref_m != 0)
        //salvataggio in caso di inammissibilita'
        last_curvature_radius_ref_m = dataRX.curvature_radius_ref_m;
      if(floatArray[2] == 0)
        //assegnamento dell'ultimo valore ammissibile ricevuto
        dataRX.curvature_radius_ref_m = last_curvature_radius_ref_m;
      else
```

```
24          //assegnamento del nuovo valore di curvatura
25          dataRX.curvature_radius_ref_m = floatArray[2];
26        //condizione di stop
27        if(floatArray[0] == 3){
28          // Reset dei pid
29          resetPID(&pid_steering);
30          resetPID(&pid_traction);
31          resetPID(&pid_traction_RWD);
32          resetPID(&pid_traction_DESC);
33        }
34        //ricorsione e riattivazione dell'ascolto
35        HAL_UARTEx_ReceiveToIdle_DMA(&huart6, RxBuf, RxBuf_SIZE);
36      }
37    }
38 }
```

Listing 8.3: Ricezione Dati

In particolare, alla riga 7 viene utilizzato un metodo che interpreta la stringa in ingresso, trasformando in numeri, i caratteri letti ed assegnandoli ciascuno in maniera sequenziale nel vettore FloatArray. Di seguito è riportato il codice sorgente della funzione parseCSV:

```
1  #define MAX_VALUES 3
2
3
4  void parseCSV(const char *csvString, float *values) {
5      char *token;
6      char *copy = strdup(csvString); // Make a copy of the string to
       avoid modifying the original
7      int index = 0;
8
9      token = strtok(copy, ",");
10     while (token != NULL && index < MAX_VALUES) {
11         values[index++] = strtof(token, NULL); // Convert token to
       float and store in the array
12         token = strtok(NULL, ",");
13     }
14
15     free(copy); // Free the dynamically allocated memory for the
       copied string
16 }
```

Listing 8.4: ParseCSV

In sunto, questo metodo si occupa di suddividere in token il messaggio fornito in input, dividendolo rispetto ai segni di ";", convertendo i char in float ed assegnandoli sequenzialmente al vettore di output.

Questa funzione è stata ideata perché il messaggio ricevuto in ingresso presenta una struttura ben definita, del tipo: "flag; velocità_di_riferimento; curvatura_di_riferimento; offset;", a cui l'algoritmo si adatta perfettamente.

Poiché la ricezione dei dati è fondamentale, è inoltre stato implementato manualmente un meccanismo simile ad un watchdog che si occupa di controllare che avvenga la ricezione dei dati in maniera corretta. Questo algiritmo, riportato di seguito, verifica il valore di un counter che viene incrementato ogni 10ms e nel momento in cui il suo valore supera una soglia, in questo caso pari a 10, chiama il metodo che attiva la ricezione dei dati in modo forzato.

```
//Timer11 for temporization
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
  if (htim == &htim11) {
    Flag_10ms = 1;
    //incremento counter di cotrollo della ricezione
    cnt_DMA++;
    if(cnt_DMA >= 10){
      //bruteforce dell'ascolto della seriale
      HAL_UARTEx_ReceiveToIdle_DMA(&huart6, RxBuf, RxBuf_SIZE);
      cnt_DMA = 0;
    }
    ...
    }
}
```

Listing 8.5: Watchdog Seriale

In questo caso, il tempo massimo che può trascorrere tra la lettura di due messaggi in caso di errore è di 100ms, ottenuta secondo la seguente relazione:

$$t = (timer\_period) \cdot (counter\_threshold) = 10ms \cdot 10 = 100ms \tag{8.1}$$

Nel caso in cui avvenisse una corretta ricezione del messaggio, all'interno del metodo HAL_UARTEx_RxEventCallback è presente una riga di codice che assegna al counter il valore 0, facendo ripartire il conteggio dall'inizio.

Invio dei dati

Per quanto riguarda l'invio dei dati telemetrici, esso viene eseguito alla fine del ciclo di controllo, fornendo dunque i dati ad una frequenza di 10ms. Come è possibile osservare nel seguente blocco di codice, alla fine di ogni iterazione del ciclo infinito viene invocato il metodo TransmitTelemetry() che si occupa dell'invio dei messaggi sulla porta seriale.

```
    while(1) {
    ...
    TransmitTelemetry();
    }
```

Listing 8.6: Invio dati telemetrici

Il metodo TransmitTelemetry() è a sua volta implementato come segue:

```
// COMUNICAZIONE (TRASMISSIONE DATI)
```

```
void TransmitTelemetry(){
  dataTX.current_speed_rpm = vehicleState.motor_speed_RPM;
  dataTX.current_yaw_rate_deg_sec = vehicleState.yaw_rate_deg_sec;

  bno055_vector_t accel = bno055_getVectorAccelerometer();
  bno055_vector_t angle = bno055_getVectorGyroscope();
  bno055_vector_t magne = bno055_getVectorMagnetometer();
  bno055_vector_t quat = bno055_getVectorQuaternion();
  bno055_calibration_state_t cal = bno055_getCalibrationState();

  collectedData[0][pointer]=accel.x;
  collectedData[1][pointer]=accel.y;
  collectedData[2][pointer]=accel.z;
  collectedData[3][pointer]=cal.mag;
  collectedData[4][pointer]=angle.y;
  collectedData[5][pointer]=angle.z;
  collectedData[6][pointer]=magne.x;
  collectedData[7][pointer]=magne.y;
  collectedData[8][pointer]=magne.z;
  collectedData[9][pointer]=tempRPM * RPM_2_m_s;
  pointer++;
  if (pointer==NUM_OF_SAMPLE_TO_AVERAGE){
    pointer=0;
    for(int j=0;j<NUM_DATA_TO_AVERAGE;j++){
      double sum = 0.0;
      for(int i=0;i<NUM_OF_SAMPLE_TO_AVERAGE;i++){
        sum+=collectedData[j][i];
      }
      averagedData[j]=sum/NUM_OF_SAMPLE_TO_AVERAGE;
    }

    printf("%+2.4f, %+2.4f, %+2.4f,"
        " %+2.4f, %+2.4f, %+2.4f,"
        " %+2.4f, %+2.4f, %+2.4f,"
        " %+2.4f\r\n",
        averagedData[0],averagedData[1],averagedData[2],
        averagedData[3],averagedData[4],averagedData[5],
        averagedData[6],averagedData[7],averagedData[8],
        averagedData[9]);
  }
}
```

Listing 8.7: Metodo TransmitTelemetry()

Questa funzione preleva, mediante gli appositi metodi get forniti dal costruttore dell'IMU, i dati relativi alle accelerazioni, agli angoli di Eulero e alle intensità del campo magnetico, oltre che ai quaternioni e ai valori di velocità e di angolo di sterzo correnti. Questi dati vengono poi elaborati grazie ad un filtro a media mobile e inviati in un unica stringa, separati da una virgola.

Occorre fare una precisazione sull'uso del metodo printf(), che richiede la particolare implementazione di una funzione affinché i caratteri vengano indirizzati sulla porta

seriale corretta. L'implementazione della funzione ___io_putchar(), necessaria a questo scopo, è riportata di seguito:

```
1 //USART2 -> ST_Link UART for DEBUG with USB (e.g. PUTTY)
2 int __io_putchar(int ch) {
3   HAL_UART_Transmit(&huart2, (uint8_t*) &ch, 1, 0xFFFF); //putty
4   HAL_UART_Transmit(&huart6, (uint8_t*) &ch, 1, 0xFFFF); //rpi
5   return ch;
6 }
```

Listing 8.8: Funzione ___io_putchar()

In questo caso le stringhe inviate in stampa mediante la funzione printf() vengono reindirizzate sia alla porta seriale 6, connessa alla Raspberry Pi, che alla porta seriale 2, collegata ad un eventuale computer per scopi di debugging.

8.4. Ciclo di controllo

Lo scopo principale del microcontrollore è quello di tradurre i riferimenti ricevuti via comunicazione seriale in segnali elettrici di controllo degli attuatori. Per far ciò è necessario un ciclo di controllo ben strutturato, che implementa le funzionalità sopra citate. In particolare, questo ciclo si deve occupare di:

- **Leggere i dati in ingresso**: il microcontrollore deve interpretare i messaggi ricevuti dalla Raspberry Pi 4, poiché forniscono i riferimenti per gli algoritmi di controllo;
- **Attuare il motore di trazione**: il microcontrollore deve calcolare e generare i segnali di controllo necessari per spostare il veicolo ad una velocità pari a quella del riferimento, controllandone quindi la dinamica longitudinale;
- **Attuare il motore di sterzo**: il microcontrollore deve calcolare e generare i segnali di controllo necessari per azionare il servomotore dello sterzo, in modo da eseguire una curva pari a quella del riferimento, controllando quindi la dinamica laterale del veicolo;
- **Elaborare ed inviare i dati telemetrici**: il microcontrollore deve ottenere i dati telemetrici dai sensori a bordo ed inviarli alla Raspberry Pi mediante comunicazione seriale a frequenza costante.

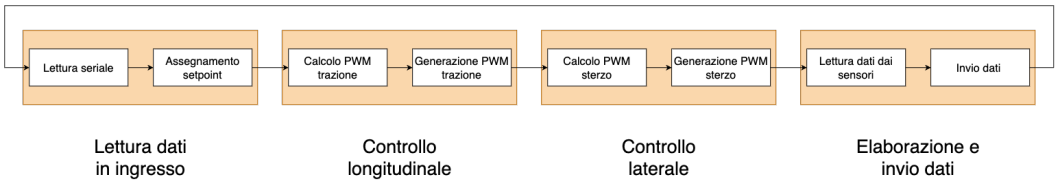


Figura 8.8.: Pseudo-algoritmo del ciclo di controllo.

In Figura 8.8 è riportato lo pseudo-algoritmo del ciclo di controllo del veicolo, evidenziandone i passaggi principali.

La comunicazione in ingresso è stata ampiamente discussa nella sezione precedente, mentre nelle successive sezioni verranno analizzati gli algoritmi di controllo longitudinale e laterale. L'interezza del ciclo di controllo viene completata ad una frequenza di 100Hz, ovvero un'iterazione ogni 10ms. La temporizzazione del ciclo di controllo è fondamentale e ne verrà esplicitata l'implementazione nei successivi paragrafi.

8.4.1. Implementazione

La temporizzazione del ciclo di controllo riveste un ruolo cruciale nella sua implementazione, poiché gli algoritmi utilizzati necessitano di operare con una frequenza costante. Per questo motivo, la sua implementazione è stata realizzata seguendo questo ordine:

- **Definizione della frequenza di funzionamento**: questo aspetto è fondamentale, in quanto una frequenza troppo bassa risulta essere insufficiente per un controllo efficace degli attuatori in risposta alle variazioni dei riferimenti;
- **Scelta dei timer e configurazione**: lo strumento hardware utilizzato per la temporizzazione del ciclo è rappresentato dai timer, che possono essere configurati per lavorare a frequenze arbitrarie. Un approfondimento sul funzionamento dei timer è riportato in Appendice B;
- **Implementazione degli algoritmi di controllo**: per controllare gli attuatori sono necessari appositi algoritmi di controllo, analizzati nelle sezioni successive;

In questa sezione verrà analizzata l'implementazione della temporizzazione del ciclo di controllo, con particolare riguardo all'inizializzazione e alla configurazione del timer utilizzato.

Configurazione dell'IOC

Il primo passaggio fondamentale per configurare un timer è la configurazione dell'albero dei clock di sistema (Clock Tree). Per temporizzare il ciclo di controllo è stato utilizzato il timer TIM11, che ha come sorgente APB2, come riportato nel datasheet fornito dal costruttore. Per ottenere il massimo delle prestazioni, la sorgente è stata impostata ad una frequenza di 84Mhz, utilizzando prescaler con fattori di riduzione unitari. In Figura 8.9 è riportato il clock tree configurato per il corretto funzionamento del microcontrollore.

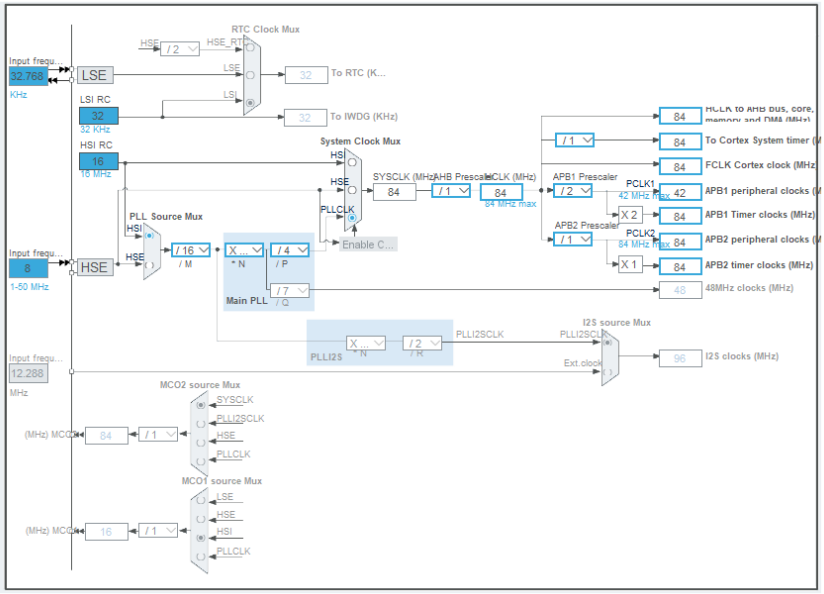


Figura 8.9.: Clock Tree, STM32 Nucleo F401-RE

Il timer TIM11 è stato ulteriormente configurato, in modo da ottenere un funzionamento in modalità free-running ad una frequenza di 100Hz, ovvero con un periodo di 10ms tra un fronte di salita ed il successivo.

I valori di prescaler e counter period, oltre ad altri parametri di inizializzazione sono riportati in Figura 8.10.

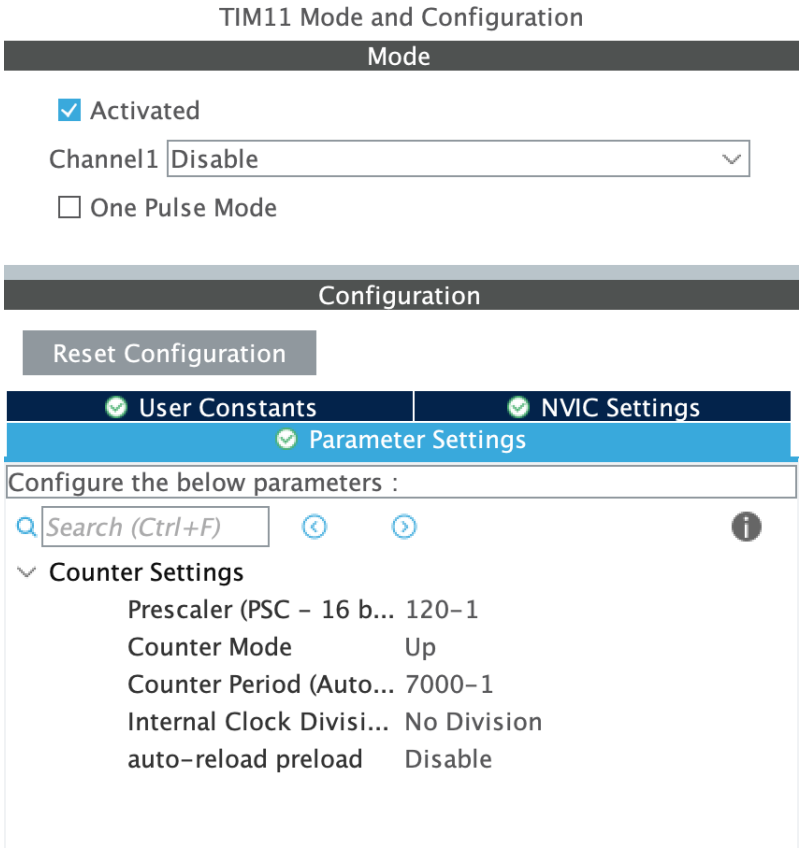


Figura 8.10.: Configurazione Timer TIM11.

Difatti, calcolando la frequenza di aggiornamento si ottiene il seguente risultato:

$$UEV(Hz) = \frac{84000000}{(119+1)\cdot(6999+1)} = 100Hz \tag{10}$$

Codice sorgente

In termini di codice, è stata implementata una logica a flag che regola l'esecuzione del ciclo di controllo. Il valore della flag viene portato a 1 ogni qualvolta avviene un evento di overflow del timer ed a 0 quando viene eseguita la porzione di codice contenente il ciclo di controllo. La gestione degli eventi di overflow avviene mediante interrupt. In particolare, nel caso dei timer la Interrupt Service Routine (ISR) è rappresentata da un particolare blocco di codice denominato Callback. Nel caso di interesse, la callback si occupa di portare a 1 il valore della flag.

Di seguito è riportato il segmento di codice che implementa la funzione di callback:

```
//Timer11 for temporization
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
  if (htim == &htim11) {
    Flag_10ms = 1;

    //other code

  }
}
```

Listing 8.9: Callback TIM11

Come anticipato, nel ciclo di controllo è presente una condizione sentinella che verifica lo stato della flag e la riporta a 0 ogni qualvolta venga eseguito il ciclo.

Inoltre, sono presenti altre due flag di controllo le cui funzioni sono:

- **HardwareEnable**: flag di controllo hardware che viene portata a 1 solo se viene premuto un pulsante sulla scheda STM;
- **dataRX.enable**: flag di controllo software che assume il valore 1 esclusivamente quando viene impostata a tale valore dagli algoritmi decisionali implementati su Raspberry.

Queste due flag rappresentano un ulteriore livello di sicurezza che impedisce al veicolo di compiere azioni impredicibili.

```
while(1){

    //STATE MACHINE CONTROL

    if (HardwareEnable == 1 && dataRX.enable == 1) {
    if (Flag_10ms == 1) {
        Flag_10ms = 0;

            if(dataRX.linear_speed_ref_m_s != 0){

            //-------------------------------------------------------

            //TRACTION CONTROL

            //-------------------------------------------------------

            //STEERING CONTROL

            //-------------------------------------------------------
            }
        }
    }
}
```

Listing 8.10: Struttura del ciclo di controllo.

Difatti, se almeno una delle due condizioni non è verificata, il veicolo si porta automaticamente in uno stato di stop, come suggerito dal blocco di codice riportato di seguito.

```
while(1){

    //STATE MACHINE CONTROL

    if (HardwareEnable == 1 && dataRX.enable == 1) {
    if (Flag_10ms == 1) {
        Flag_10ms = 0;
            if(dataRX.linear_speed_ref_m_s != 0){

            //VEHICLE CONTROL

            }
            else {
                BL_set_PWM(NEUTRAL_PWM);
                servo_motor(0);

                resetPID(&pid_steering);
                resetPID(&pid_traction);
                resetPID(&pid_traction_RWD);
                resetPID(&pid_traction_DESC);
            }
        }
    }
}
```

Listing 8.11: Stato di arresto del veicolo.

8.5. Controllo della trazione

Il controllo della trazione, anche chiamato controllo longitudinale, ha come scopo quello di abilitare lo spostamento del veicolo lungo l'asse x del proprio sistema di riferimento. Questo controllo è ovviamente fondamentale per la navigazione dell'ambiente, in quanto rappresenta uno dei due gradi di libertà del veicolo.

Il controllo della trazione è stato realizzato in riferimento alla sua velocità, in quanto è cruciale che essa sia costante per la corretta esecuzione delle varie manovre, discusse ampiamente nei capitoli 5 e 6.

Per realizzare il controllo vero e proprio, è stato utilizzato un controllore Proporzionale, Integrale, Derivativo, o anche detto PID, che si adatta perfettamente al contesto applicativo in quanto riesce a produrre uno sforzo di controllo indipendentemente dalla formulazione matematica del processo da controllare. Un approfondimento sulle nozioni teoriche associate a questo tipo di controllore è riportato nell'Appendice C.

8.5.1. Controllore PID

Il controllore PID è solo un elemento dell'intero ciclo di controllo, e può essere descritto sia da un punto di vista esterno che da un punto di vista interno.

Dal punto di vista esterno, ovvero considerando gli ingressi e le uscite del sistema di controllo della trazione, in questo caso si ha un solo ingresso ed una sola uscita, rappresentati da:

- **Riferimento in velocità**: valore reale fornito in ingresso al ciclo di controllo, che rappresenta la velocità alla quale il veicolo deve traslare sul piano stradale;
- **Moto del veicolo**: valore reale prodotto in uscita dal ciclo di controllo, rappresentante la velocità effettiva alla quale si sposta il veicolo a seguito dello sforzo di controllo.

Graficamente, il ciclo di controllo, visto esternamente, può essere rappresentato dalla Figura 8.11.

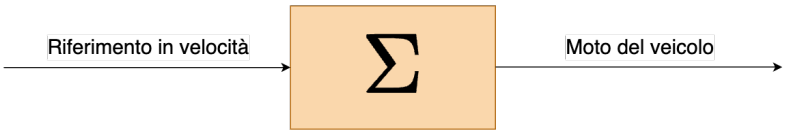


Figura 8.11.: Ciclo di controllo di trazione, ingressi ed uscite.

Da un punto di vista interno, si hanno invece più componenti che interagiscono tra di loro, con un funzionamento analogo a quello esterno. Queste componenti sono:

- **Processo**: in questo caso è rappresentato dal motore di trazione, che trasforma il segnale di controllo fornitogli in ingresso in rotazione e di conseguenza in spostamento longitudinale del veicolo;
- **Trasduttore**: questo componente, fondamentale per il controllo in catena chiusa, svolge il ruolo cruciale di misurare la stato attuale del sistema da controllare. In questo caso il trasduttore è rappresentato da un encoder rotativo, che si occupa di rilevare la velocità di rotazione dell'asse del motore;
- **Controllore PID**: questo componente si occupa del calcolo dello sforzo di controllo necessario per far sì che il valore di uscita del sistema, ossia la velocità del motore, corrisponda al valore di riferimento impostato in ingresso. Infatti, lo sforzo di controllo viene calcolato in base alla differenza tra il valore di riferimento e quello effettivo misurato dal trasduttore.

Una rappresentazione del ciclo di controllo è mostrata in Figura 8.12.

Nelle prossime sezioni, per far sì che risulti più chiara l'implementazione degli algoritmi su microcontrollore, verranno brevemente analizzate le diverse componenti che, nel loro insieme, compongono il ciclo di controllo della trazione.

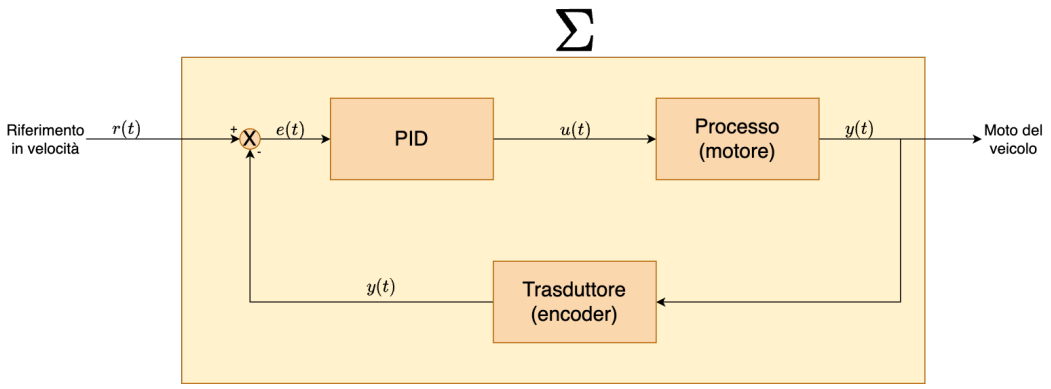


Figura 8.12.: Ciclo di controllo di trazione, vista interna.

Motore brushless

Il motore brushless, come già anticipato, rappresenta il processo da controllare. In particolare, si vuole realizzare un controllo della sua velocità.

Durante la realizzazione di un sistema di controllo è fondamentale, ove possibile, conoscere il processo che si vuole controllare. Per questo motivo, nel caso del motore di trazione, è stato realizzato un grafico, noto anche come curva caratteristica, che rappresenta la velocità del rotore in funzione del segnale di controllo fornito in ingresso.

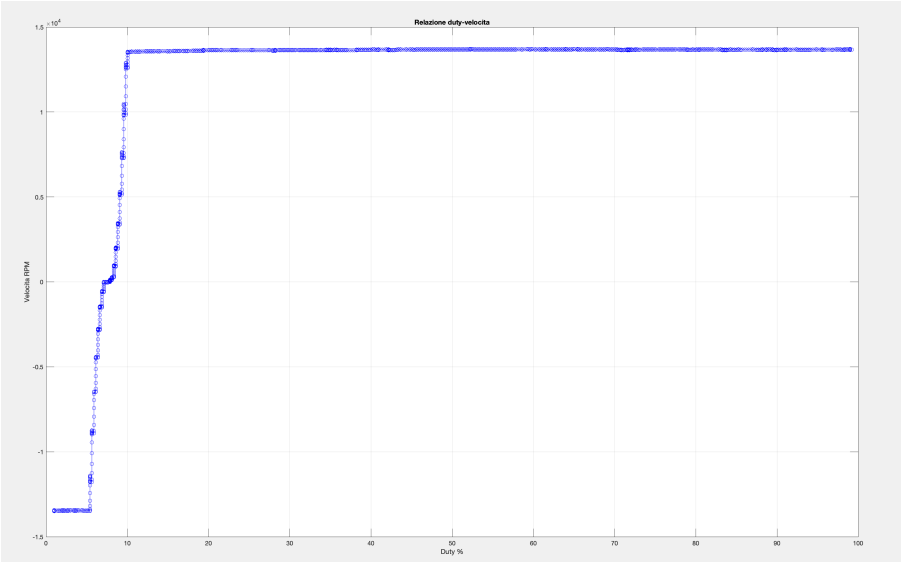


Figura 8.13.: Relazione duty cycle - velocità, impostazioni di fabbrica.

Come è possibile osservare in Figura 8.13, con le impostazioni di fabbrica, la velocità evolve in maniera fortemente non lineare rispetto ad una variazione lineare del segnale in ingresso, e satura a un valore di velocità significativamente inferiore rispetto al range disponibile. Queste condizioni non risultano essere minimamente

ideali per il controllo del motore, in quanto una minima variazione del segnale di controllo provoca un cambiamento brusco della velocità del motore.

Fortunatamente, è possibile configurare manualmente sia i valori di duty cycle a cui corrispondono la velocità massima e minima, che la velocità massima in retromarcia. Impostando un valore di duty cycle massimo al 99%, il minimo all'1% e una velocità massima in retromarcia pari al 25% di quella totale, si ottiene un andamento decisamente più desiderabile, riportato in Figura 8.14.

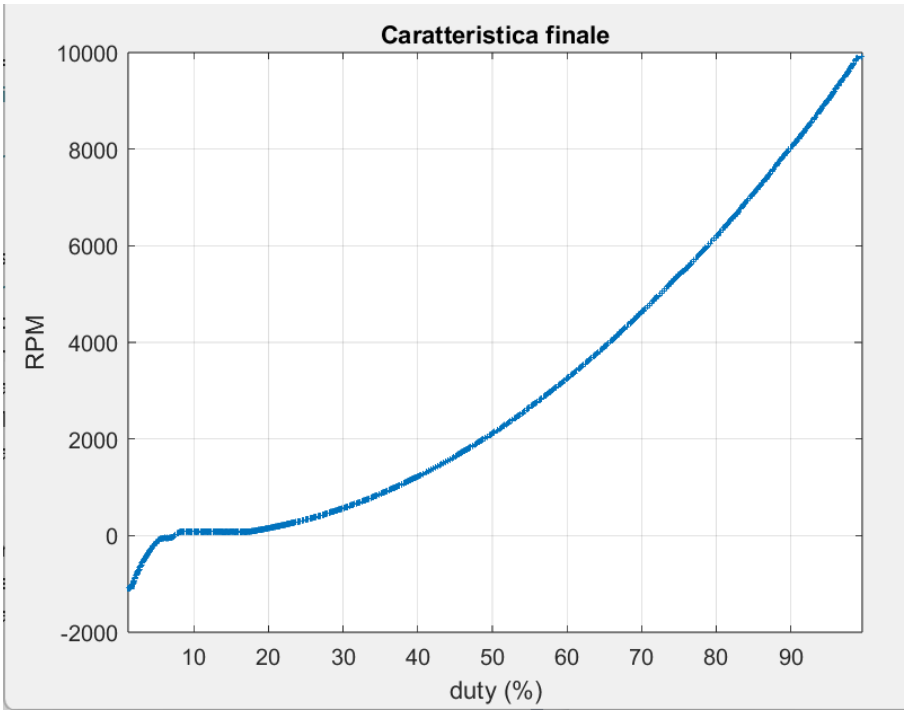


Figura 8.14.: Relazione duty cycle - velocità, impostazioni finali.

Dopo le modifiche alla configurazione del motore, è stato ottenuto un processo molto più favorevole al controllo, in quanto risulta molto meno sensibile alle variazioni di un segnale di ingresso, il quale è soggetto a fluttuazioni dovute al calcolo numerico.

In sintesi, dunque, il controllo del motore avviene mediante un segnale generato dal microcontrollore, di tipo Pulse Width Modulation (PWM), al quale è associato un valore di velocità di rotazione univoco.

È importante specificare che il modello matematico del motore è ininfluente per quanto riguarda il suo controllo, in quanto il controllore, come verrà trattato nelle sezioni successive, agisce solo in funzione dell'errore rispetto alla velocità di riferimento.

Encoder

L'encoder rotativo rappresenta il blocco di trasduzione del ciclo di controllo. Questo blocco è fondamentale in quanto si occupa della traduzione dell'uscita del processo,

in questo caso della velocità di rotazione del motore, in un segnale comprensibile ed interpretabile dal microcontrollore.

Questo ruolo è fondamentale in quanto il funzionamento del controllore PID è fondato sulla funzione errore $e(t)$, data dalla differenza tra il valore di riferimento e il valore effettivo che assume l'uscita del processo, misurato dunque dall'encoder.

L'encoder utilizzato nel contesto applicativo, di tipo incrementale e a quadratura, genera difatti un treno di impulsi su due canali, dai quali è possibile ricavare la sua velocità e la direzione di rotazione in funzione della frequenza dei segnali da esso generati.

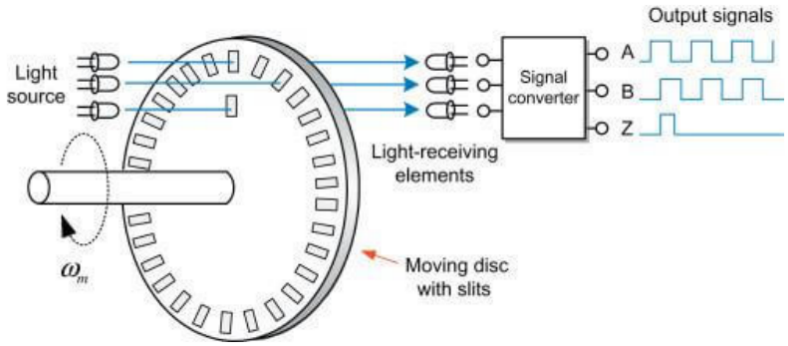


Figura 8.15.: Principio di funzionamento dell'encoder rotativo.

Nelle sezioni successive verranno dunque trattati gli algoritmi sviluppati su microcontrollore per leggere correttamente la velocità del motore, e di conseguenza la velocità longitudinale del veicolo.

Controllore PID

Il controllore PID rappresenta il componente che si occupa del calcolo dello sforzo di controllo, in risposta al segnale che viene fornito ad esso in ingresso. Questo controllore è un algoritmo che applica una tecnica di controllo a controreazione, generando uno sforzo di controllo che corregge l'errore commesso dal processo rispetto al valore di riferimento.

Il nome PID è rappresentativo delle tre azioni che esso svolge in maniera parallela, ovvero:

- **Azione Proporzionale (P)**: il controllore si oppone all'errore in maniera direttamente proporzionale ad esso. Questa azione è identificata dal parametro K_p, che in termini pratici rappresenta la velocità con cui il controllore si avvicina al riferimento, nel tempo;
- **Azione Integrale (I)**: il controllore si oppone all'errore in maniera direttamente proporzionale al suo integrale, ovvero alla sua evoluzione nel tempo. Questa azione è identificata dal parametro K_i;

- **Azione Derivativa (D)**: il controllore si oppone all'errore in maniera direttamente proporzionale alla derivata della sua evoluzione. Questa azione è identificata dal parametro K_d e può essere interpretata come uno smorzamento del comportamento del controllore, in quanto riduce le variazioni brusche dell'uscita del processo.

In Figura 8.15 è mostrato uno schema espanso che evidenzia i vari costituenti del controllore PID e la loro formulazione matematica.

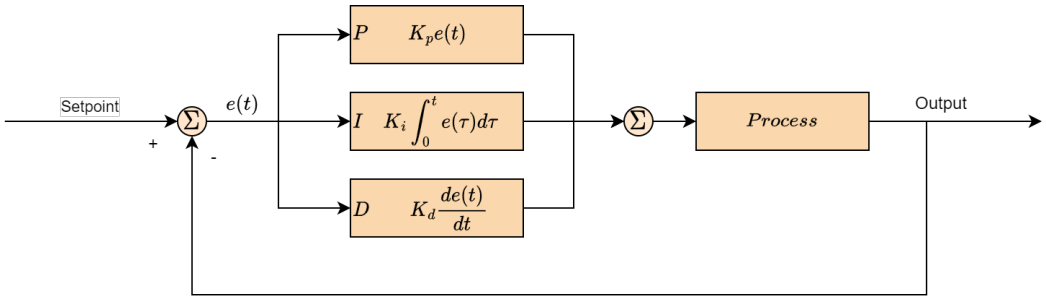


Figura 8.16.: Controllore PID espanso in configurazione parallela.

Scelta dei parametri di controllo e banco di PID

Il processo di scelta dei parametri di controllo è di tipo iterativo, ma si può definire una procedura standard per semplicità. Per scegliere in modo efficiente i parametri $K_p,\ K_i\ K_d$ si procede nel seguente modo:

1. Scelta del parametro K_p: l'azione proporzionale è quella che influisce in modo maggiore sulla performance del controllore, per cui questo parametro va scelto per primo. Un buon parametro K_p garantisce una rapida risposta al gradino, con un errore a regime diverso da 0;

2. Scelta del parametro K_i: l'azione integrale permette di annullare l'errore a regime, per cui è fondamentale. Un buon parametro K_i garantisce un errore a regime nullo, senza oscillazioni a regime e senza un sovraelungamento eccessivo;

3. Scelta del parametro K_d: l'azione derivative permette di ridurre sforzi di controllo eccessivi in risposta a variazioni brusche del riferimento. Tuttavia, questo parametro deve essere mantenuto molto piccolo rispetto agli altri due, in quanto a regime può risultare in un comportamento instabile del processo.

A seguito di questa premessa, bisogna specificare che nel contesto dell'implementazione del controllo mediante PID del veicolo è stato usato un banco di PID. Questa configurazione prevede l'utilizzo di più di un controllore, ciascuno caratterizzato dai suoi parametri, che agiscono sul processo in funzione del riferimento fornito.

Questa scelta è stata fatta in quanto il moto del veicolo deve essere controllato sia in avanti che in retromarcia, e le caratteristiche duty cycle - velocità differiscono di

molto nelle due casistiche. Come illustrato in Figura 8.15, il fattore decisionale che regola l'uso di un controllore piuttosto che di un altro è infatti il segno del riferimento di velocità in ingresso.

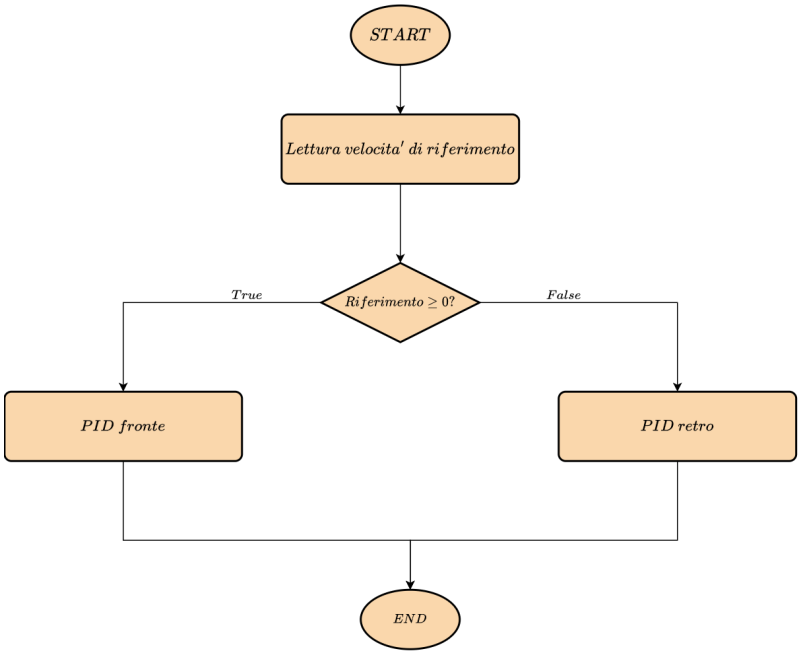


Figura 8.17.: Banco di PID per il controllo longitudinale

Per il controllore per la marcia in avanti, a seguito di più iterazioni sperimentali e modifiche ai parametri, sono stati assegnati ai coefficienti del controllore i seguenti valori:

K_p	0.0001
K_i	0.001
K_d	0.000001

Tabella 8.1.: Coefficienti del PID di trazione in avanti.

Per quanto riguarda il movimento in retromarcia, invece, sono stati utilizzati i seguenti valori:

K_p	0.000025
K_i	0.00025
K_d	0.00000025

Tabella 8.2.: Coefficienti del PID di trazione in retromarcia.

E possibile osservare come esista una relazione di proporzionalità tra i rispettivi valori dei due controllori, rappresentata da un coefficiente moltiplicativo pari a 0.,25.

Questa relazione è giustificata dal fatto che il rapporto tra le velocità massime in avanti e in retromarcia, a seguito della configurazione del motore, è dato dalla

seguente relazione:

$$V_{maxF} = 4 \cdot V_{maxR} \tag{8.2}$$

dove V_{maxF} rappresenta la velocità massima in avanti, mentre V_{maxR} quella in retromarcia. Questa relazione è inoltre evidente osservando la Figura 8.14.

8.5.2. Implementazione

Per implementare il controllo della trazione, è necessario implementare le seguuenti funzionalità:

- Lettura della velocità mediante l'encoder rotativo;
- Realizzazione dell'algoritmo di controllo PID;
- Controllo dell'attuatore, ovvero del motore brushless.

Configurazione dell'IOC per l'encoder

La lettura di un encoder incrementale viene effettuata usando un particolare timer a due canali, che prende il nome di "Encoder Mode Timer" dal contesto applicativo.

Per implementare la lettura della velocità è stato dunque utilizzato il timer TIM2, in modalità combinata Encoder Mode. Entrambi i canali del timer sono stati impostati con una condizione di trigger rising edge, ovvero vengono attivati ogni qualvolta rilevano un fronte di salita in ingresso sul canale. Al timer in questione non è stato applicato alcun prescaler e il counter period è stato impostato al suo valore massimo.

Di seguito, in Figura 8.18 è mostrata la schermata di configurazione dell'IOC relativa al timer TIM2.

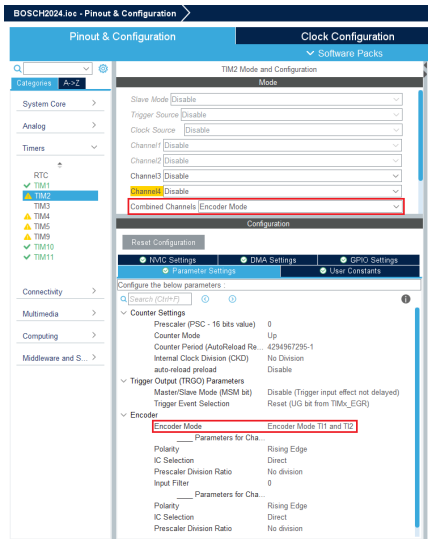


Figura 8.18.: Configurazione dell'IOC relativa al TIM2.

Inoltre, sono stati definiti, in un file di configurazione, i parametri caratteristici dell'encoder, rappresentati da:

```
//ENCODER
#define ENCODER_PPR 2048
#define GEARBOX_RATIO 1
#define ENCODER_COUNTING_MODE 4
```

I 3 parametri sopra riportati, nello specifico, sono:

- **PPR (Pulses Per Revolution)**: rappresenta la quantità di impulsi generata da un singolo canale per ogni rotazione effettuata dal sensore. Definisce, quindi, la granularità della misurazione effettuata, in quanto un valore di PPR maggiore risulta in una variazione angolare minima rilevabile minore;
- **Rapporto di Riduzione(Gearbox Ratio)**: rappresenta un fattore di riduzione presente tra la velocità dell'asse di rotazione su cui è montato il sensore e la velocità di rotazione effettiva del rotore del motore. Un fattore unitario rappresenta l'assenza di un meccanismo di riduzione;
- **Modalità di Conteggio dell'Encoder (Encoder Counting Mode)**: questo valore indica il supersampling degli impulsi dell'encoder, ovvero quanti fronti di salita e/o discesa vengono considerati per la stima della rotazione del sensore. Un fattore pari a 4 indica che per ogni canale vengono utilizzati sia i fronti di salita che di discesa, quadruplicando virtualmente il PPR del sensore.

Codice sorgente per la lettura della velocità

Il ciclo di controllo, come discusso nelle sezioni precedenti, viene eseguito ogni 10ms in corrispondenza del clock TIM1. Prima di presentare il codice sorgente sviluppato, è importante spiegare il funzionamento dell'algoritmo, che può essere rappresentato mediante un diagramma di flusso evidenziando i passaggi principali, come mostrato in Figura 8.19. In riferimento a tale figura, definiamo il *valore standard*, rappresentato da un valore pari alla metà della dimensione del contatore del timer, ovvero definito nel seguente modo:

$$\frac{TIM2->ARR}{2} = \frac{4294967295}{2} = 2147483647 \tag{8.3}$$

Questo valore è stato scelto poiché, a causa della rapida rotazione del motore, la quantità di impulsi e di conseguenza di conteggi risulta essere molto elevata per la singola unità di tempo utilizzata per il calcolo della velocità. Nel caso in cui il valore di conteggi dovesse superare quello massimo del registro di conteggio, avverrebbe un evento di overflow, che porterebbe ad una lettura errata della velocità di rotazione, compromettendo dunque il ciclo di controllo e causando instabilità nel moto del veicolo.

Impostando però il valore massimo del registro di conteggio al valore massimo possibile, ed il valore di riferimento ad una cifra pari alla metà del precedente, è stato verificato che la condizione di overflow non viene mai raggiunta.

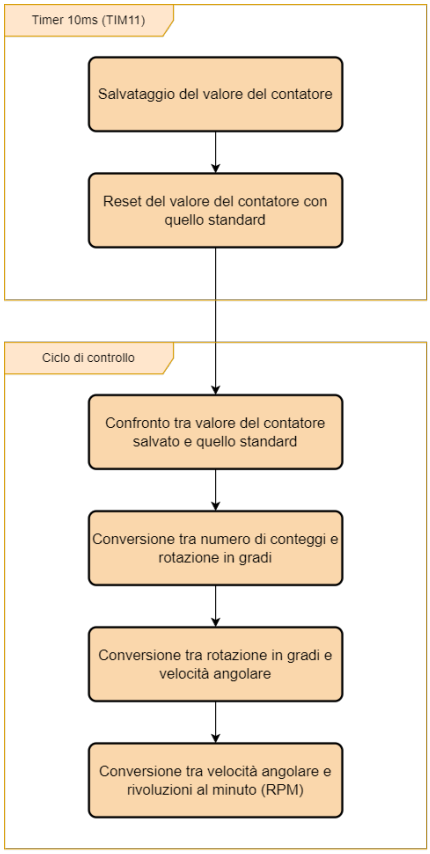


Figura 8.19.: Diagramma di flusso per la lettura della velocità.

Nel seguente blocco di codice è riportata dunque l'implementazione della callback del timer TIM11, con periodo 10ms, e le righe di codice che eseguono il salvataggio e il reset del valore del contatore.

```
1  //Timer11 for temporization
2  void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
3    if (htim == &htim11) {
4      Flag_10ms = 1;
5
6      //DMA control code
7
8      //Encoder
9      vehicleState.counts = TIM2->CNT;
10     TIM2->CNT = TIM2->ARR / 2;
11     }
12 }
```

Listing 8.12: TIM11 Callback

Nel seguente blocco di codice è invece riportata la sezione di codice appartenente al ciclo di controllo principale adibita al calcolo della velocità in termini di rotazioni al minuto (RPM) del motore. Questo valore viene salvato nella variabile *tempRPM*.

```
1     //TRACTION control
2
3     //Measure speed with encoder
4     vehicleState.ref_count = TIM2->ARR / 2;
5     vehicleState.delta_count = vehicleState.counts - vehicleState.
      ref_count;
6
7     vehicleState.delta_angle_deg = (vehicleState.delta_count * 360) /
      ((double) (ENCODER_PPR * ENCODER_COUNTING_MODE * GEARBOX_RATIO));
8     vehicleState.motor_speed_deg_sec = vehicleState.delta_angle_deg /
      ENCODER_SAMPLING_TIME;
9     tempRPM = BL_DegreeSec2RPM(vehicleState.motor_speed_deg_sec);
```

Listing 8.13: Interpretazione dei dati forniti dall'encoder

Infine, per ottenere una stima ancora migliore della velocità istantanea del veicolo, il suo valore grezzo è stato elaborato mediante un filtro a media mobile, che è risultato essere molto efficace nell'eliminazione di rumore ad alte frequenze.

La tecnica utilizzata consiste nel salvare i valori di velocità letti in un vettore (nello specifico di 10 elementi) che rappresenta la finestra di scorrimento. Con il progredire delle iterazioni, il vettore viene inizialmente popolato con le prime 10 misurazioni, se appena inizializzato. In seguito, segue una logica Last In First Out (LIFO), in cui tutti i valori vengono spostati di una posizione a sinistra, eliminando il primo valore salvato e aggiungendo il valore più recentente all'ultima posizione.

In seguito viene eseguita una media aritmetica dei valori presenti nel vettore, in funzione della quantità di elementi attualmente presenti, poiché nei primi 10 intervalli temporali essi sono minori della lunghezza massima del vettore. Il valore così ottenuto viene poi salvato in una variabile *motor_speed_RPM*, che è propria dello stato del veicolo.

Di seguito è riportato il blocco di codice, che implementa il filtro a media mobile. Nel contesto del codice sorgente, esso è scritto nel ciclo di controllo principale, di seguito al codice riportato nella Listing 8.13.

```
1     //Measure speed with encoder
2
3     //Moving average filter for speed
4         //-----------------------------------------
5         ArrayRPM[PtrRPM] = tempRPM;
6         MeanRPM = 0;
7         for(int i = 0; i < MAX_RPM_VALUES; i++){
8             MeanRPM += ArrayRPM[i];
9         }
10        MeanRPM /= MAX_RPM_VALUES;
11
12        if(PtrRPM == MAX_RPM_VALUES-1)
```

```
13              PtrRPM = 0;
14          else
15              PtrRPM++;
16          vehicleState.motor_speed_RPM = MeanRPM;
17          //----------------------------------------
```

Listing 8.14: Filtro a media mobile per la velocità

Implementazione del controllore PID

Come introdotto fin'ora, l'algoritmo utilizzato per l'automatizzazione del controllo della trazione del veicolo è rappresentato dal controllore PID. Questo tipo di controllore, nonostante la formulazione matematica piuttosto complessa, è relativamente facile da implementare, in quanto le operazioni da esso svolte sono scomponibili in operazioni algebriche elementari.

L'implementazione del controllore è stata suddivisa su tre file, in modo da non appesantire il file main.c, contenente il ciclo di controllo. Nel file PID.c ed il suo file header PID.h, sono infatti implementate le funzioni che garantiscono il corretto funzionamento dell'algoritmo, che vengono chiamate all'occorrenza nel file main.c.

Di seguito è riportato il contenuto del file PID.h .

```
1  #ifndef INC_PID_H_
2  #define INC_PID_H_
3
4  #include <main.h>
5  #include <stdio.h>
6
7  typedef struct PID{
8    float Kp; //guadagno proporzionale
9    float Ki; //guadagno integrale
10   float Kd; //guadagno derivativo
11   float Kb; //guadagno del back-calculation per l'anti-windup
12
13   float Tc; //tempo di campionamento
14   float u_max; //limite superiore
15   float u_min; //limite inferiore
16
17   float e_old;
18   float Iterm;
19
20   float offset; //valore neutro diverso da zero
21 }PID;
22
23 void init_PID(PID*, float, float, float, float);
24 void tune_PID(PID*,float,float,float, float);
25 float PID_controller(PID*,float,float);
26 void resetPID(PID*);
27
```

```
28 #endif /* INC_PID_H_ */
```

Listing 8.15: Header file PID.h

Questo header definisce la struttura di una struct generica di un controllore PID, oltre alle funzioni necessarie per il suo funzionamento. A seguito di ciascuna variabile è inoltre spiegato a commento il suo significato.

Passando all'implementazione delle funzioni definite nel file header, di seguito è riportato il contenuto del file PID.c .

```
1  #include "PID.h"
2  #include <stdio.h>
3
4  void init_PID(PID* p, float Tc, float u_max, float u_min, float offset
       ){
5    p->Tc = Tc;
6    p->u_max = u_max;
7    p->u_min = u_min;
8    p->Iterm = 0;
9    p->e_old = 0;
10   p->offset = offset;
11 }
12
13 void tune_PID(PID*p, float Kp, float Ki, float Kd, float Kb){
14   p->Kp = Kp;
15   p->Ki = Ki;
16   p->Kd = Kd;
17   p->Kb = Kb;
18 }
19
20 void resetPID(PID* p){
21   p->Iterm = 0;
22   p->e_old = 0;
23 }
24
25 float PID_controller(PID* p , float y, float r){
26   float u;
27   float newIterm;
28   float e = 0;
29
30   e = r-y;
31
32   if (isinf(p->Iterm) || isnan(p->Iterm)) {
33     p->Iterm = 0;
34     p->e_old = 0;
35   }
36
37   float Pterm = p->Kp*e;
38   newIterm = p->Iterm + (p->Ki)*p->Tc*p->e_old;
39   float Dterm = (p->Kd/p->Tc)*(e - p->e_old);
40
41   p->e_old = e;
```

```
42
43   u = Pterm + newIterm + Dterm + p->offset;
44
45   // saturazione con back-calculation
46   float saturated_u = u;
47
48   if(saturated_u > p->u_max){
49     saturated_u = p->u_max;
50   }
51   else if(saturated_u < p->u_min){
52     saturated_u = p->u_min;
53   }
54
55   float correction = p->Kb * (saturated_u - u) * p->Ki * p->Tc;
56   p->Iterm = newIterm + correction;
57
58   u = saturated_u;
59
60   return u;
61 }
```

Listing 8.16: Source file PID.c

Come è possibile osservare nella Listing 8.16, sono presenti 4 funzioni, che sono:

- **init_PID(p, Tc, u_max, u_min, offset)**: questo metodo permette di istanziare un nuovo controllore, assegnandogli tempo di campionamento, sforzo di controllo massimo, minimo e offset;
- **tune_PID(p, Kp, Ki, Kd, Kb)**: questo metodo permette di modificare i parametri di un dato controllore PID istanziato in precedenza;
- **resetPID(p)**: questo metodo permette di annullare il contributo integrale e l'errore di un dato controllore;
- **PID_controller(p, y, r)** questa funzione permette di calcolare lo sforzo di controllo u, calcolato base al riferimento r e all'uscita corrente y del sistema, misurata da appositi sensori.

Per quanto riguarda l'implementazione nel file main.c, sono presenti due sezioni di interesse: una inerente all'inizializzazione del controllore, ed una nel ciclo di controllo principale che si occupa del calcolo e della produzione dello sforzo di controllo.

Per quanto riguarda l'inizializzazione dei PID di trazione, il codice è riportato di seguito.

```
1 //PID traction FWD
2 init_PID(&pid_traction, TRACTION_SAMPLING_TIME, MAX_U_TRACTION,
3         MIN_U_TRACTION, NEUTRAL_PWM);
4 tune_PID(&pid_traction, KP_TRACTION, KI_TRACTION, KD_TRACTION, -1);
5
6 //PID traction RWD
```

```
init_PID(&pid_traction_RWD, TRACTION_SAMPLING_TIME, MAX_U_TRACTION,
        MIN_U_TRACTION, NEUTRAL_PWM);
tune_PID(&pid_traction_RWD, KP_TRACTION_RWD, KI_TRACTION_RWD,
    KD_TRACTION_RWD, -1);
```

Listing 8.17: Main.c: inizializzazione PID di trazione

Per quanto riguarda gli argomenti delle funzioni, essi sono stati definiti costanti nel file di configurazione *configuration.h*, secondo quanto analizzato nelle sezioni precedenti.

Passando al codice riguardante il controllo della trazione, esso è incapsulato nel ciclo di controllo principale trattato nella sezione 8.4, ed è riportato di seguito.

```
//-----------------------------------------
//TRACTION CONTROL

////Measure speed with encoder
//Moving average filter for speed

//Speed reference for motor
vehicleState.motor_speed_ref_RPM = dataRX.linear_speed_ref_m_s /
    RPM_2_m_s;

//Control effort calculation (u_trazione)
if(vehicleState.motor_speed_ref_RPM >= 0){
    u_trazione = PID_controller(&pid_traction,
                                vehicleState.motor_speed_RPM,
                                vehicleState.motor_speed_ref_RPM);
}
else{
    u_trazione = PID_controller(&pid_traction_RWD,
                                vehicleState.motor_speed_RPM,
                                vehicleState.motor_speed_ref_RPM);
}
```

Listing 8.18: Main.c: controllo della trazione

COnfigurazione dell'IOC per il controllo del motore brushless

Per controllare la velocità del motore è stato utilizzato un controllo di tipo Pulse Width Modulation (PWM). Il segnale di controllo viene infatti generato dal microcontrollore mediante l'uso di un timer, nello specifico TIM10, con una frequenza dell'onda portante pari a 50Hz, e con un'onda modulante variabile, in funzione dello sforzo di controllo generato dal controllore PID. Come anticipato, il timer utilizzato è il TIM10, configurato in modalitá di generazione PWM e con i parametri riportati di seguito:

- **Prescaler**: 840-1;
- **Counter period**: 2000-1.

Questi parametri sono stati scelti in funzione della frequenza del clock di sistema, che è pari a 84MHz. La frequenza finale viene infatti ottenuta secondo la seguente relazione:

$$Frequenza = \frac{84.000.000}{(840) \cdot (2.000)} = 50Hz$$

Per quanto riguarda la configurazione dell'IOC, di seguito è riportata la schermata di configurazione relativa al timer TIM10, al quale sono stati assegnati i parametri sopracitati.

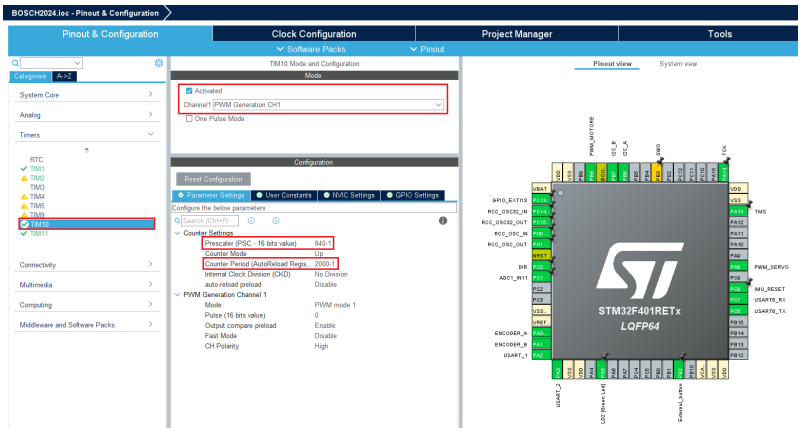


Figura 8.20.: Configurazione IOC per generazione di PWM con TIM10.

Codice sorgente per il controllo del motore brushless

Per quanto riguarda il codice sorgente sviluppato con lo scopo di controllare il motore brushless, in primis è necessario inizializzare il timer, usando il codice riportato di seguito.

```
1    HAL_TIM_PWM_Start(&htim10, TIM_CHANNEL_1);
```

Listing 8.19: Inizializzazione timer TIM10

Successivamente, per assegnare il PWM calcolato in precedenza dagli algoritmi di controllo al pin desiderato, è stata creata una funzione che trasforma il valore numerico del duty cycle, compreso tra 0 ed 1, in una variazione del registro di compare del timer. Di seguito è riportato il codice che implementa questa funzionalità.

```
1    void BL_set_PWM(float duty){
2    TIM10->CCR1 = duty * TIM10->ARR;
3    }
```

Listing 8.20: Funzione che genera il segnale PWM.

Questa funzione si occupa difatti di ridimensionare il valore del registro di compare in modo tale da essere pari al valore massimo del contatore, moltiplicato per la percentuale che rappresenta il duty cycle assegnato.

Risultati sperimentali

A seguito dell'implementazione del codice, sono stati effettuati vari test sperimentali per valutare la qualità degli algoritmi sviluppati. In particolare, nelle seguenti figure sono riportati i risultati ottenuti impostando il riferimento a 0.2m/s e a 1.0m/s.

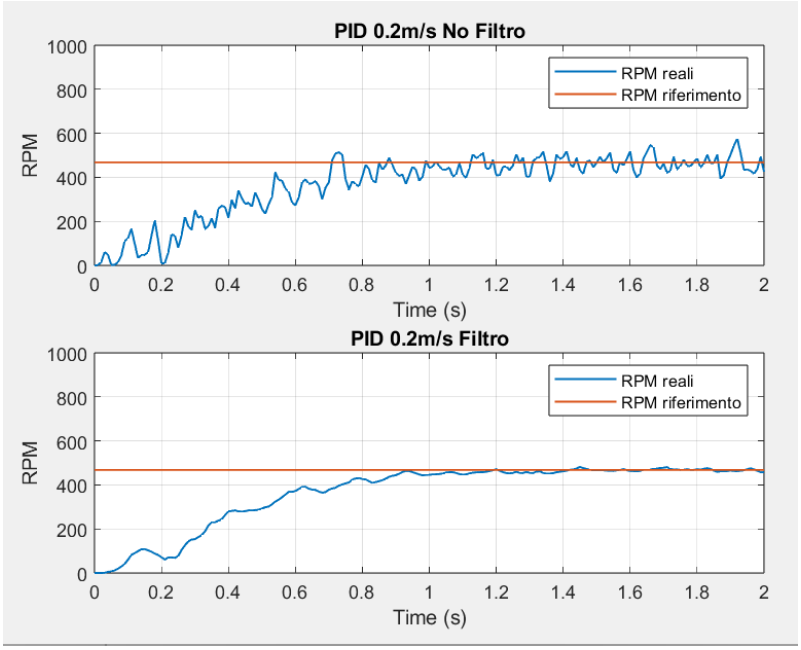


Figura 8.21.: Confronto delle velocità misurate, riferimento 0.2m/s.

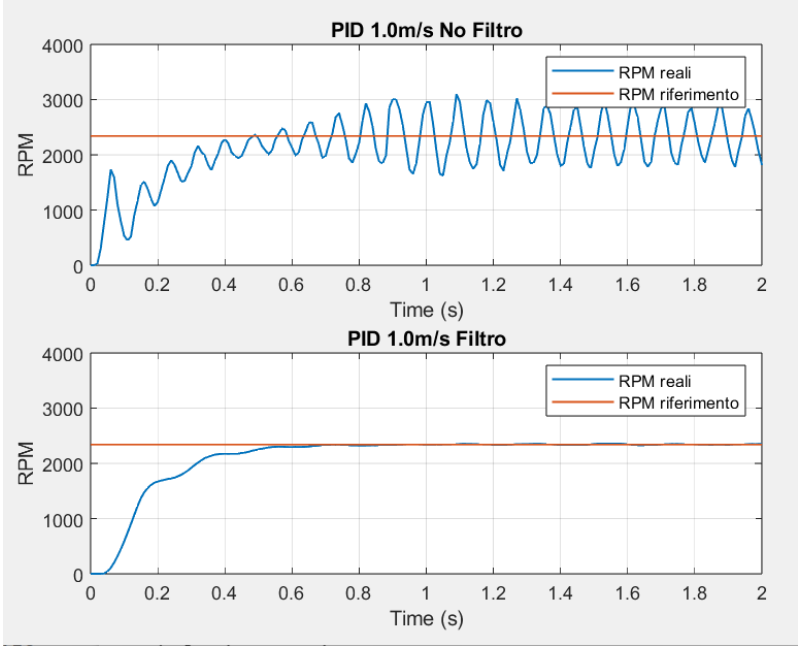


Figura 8.22.: Confronto delle velocità misurate, riferimento 1.0m/s.

Osservando le due figure, si può facilmente dedurre quanto il filtro a media mobile sia utile al fine di ottenere un miglioramento delle letture, soprattutto per velocità pari a 1.0m/s. Una stabilità numerica del valore letto dal sensore è infatti fondamentale per ridurre l'oscillazione a regime del controllore che, se tarato in maniera molto reattiva, è molto sensibile a questo tipo di variazioni.

8.6. Controllo del servosterzo

Il controllo della servosterzo, anche chiamato controllo laterale, ha come scopo quello di abilitare la rotazione del veicolo rispetto all'asse z del proprio sistema di riferimento, risultante nelle manovre di sterzata. Questo controllo è ovviamente fondamentale per la navigazione dell'ambiente, in quanto rappresenta il secondo grado di libertà del veicolo.

Il controllo del servosterzo è stato realizzato in riferimento al raggio di curvatura istantaneo, e di conseguenza del valore di yaw rate nel medesimo istante. Questo tipo di controllo è stato basato su quanto discusso nei capitoli 5 e 6.

8.6.1. Controllore PID

Anche per il controllo del servosterzo è stato utilizzato un controllore PID, e la struttura del ciclo di controllo è analoga a quella del controllo della trazione. Nel caso del controllo laterale ovviamente variano il processo e il blocco di trasduzione, in quanto il primo è rappresentato dal servomotore di sterzo e il secondo è dato dal sensore inerziale (IMU) BNO055.

Di seguito verrà trattato il ciclo di controllo dal punto di vista esterno ed interno, così come è stato fatto per il controllo della trazione.

Da un punto di vista esterno, in questo caso si hanno due ingressi ed una uscita, rappresentati da:

- **Curvatura di riferimento**: valore reale fornito in ingresso al ciclo di controllo, che rappresenta la curvatura istantanea della traiettoria che il veicolo deve percorrere;
- **Velocità del veicolo**: valore reale fornito in ingresso al ciclo di controllo, che rappresenta la velocità longitudinale istantanea del veicolo;
- **Yaw rate**: valore reale prodotto in uscita, risultante dall'azione sterzante del meccanismo di servosterzo.

Una visualizzazione del ciclo di controllo del servosterzo, visto dall'esterno, è riportata in Figura 8.23.

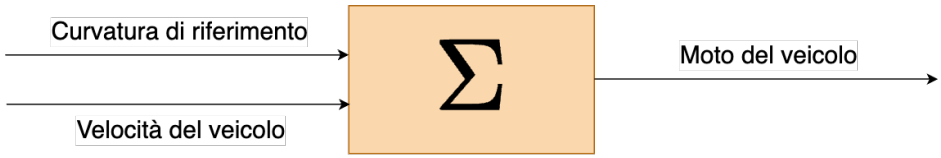


Figura 8.23.: Ciclo di controllo del servosterzo, vista esterna con ingressi ed uscite.

Da un punto di vista interno, la situazione risulta essere invece leggermente diversa rispetto al caso della trazione, in quanto il processo in sé è il risultato di un ciclo di controllo innestato. Le parti costituenti del ciclo di controllo della trazione sono rappresentate da:

- **Processo**: in questo caso è rappresentato dal servosterzo, che trasforma il segnale di controllo fornito in ingresso in una rotazione di un angolo θ dell'asse del motore, portando così all'attivazione del meccanismo di sterzo. Il funzionamento del servomotore è il risultato di un ciclo di controllo integrato all'attuatore, che verrà descritto in seguito;
- **Trasduttore**: questo componente è rappresentato dall'IMU BNO055, che in questo ciclo di controllo fornisce la misura dello yaw rate nel corso del tempo;
- **Controllore**: anche nel caso del controllo laterale è stato utilizzato un controllore PID, opportunamente tarato.

Uno schema che sintetizza il ciclo di controllo da un punto di vista interno, evidenziando il ciclo innestato, è riportato in Figura 8.24 .

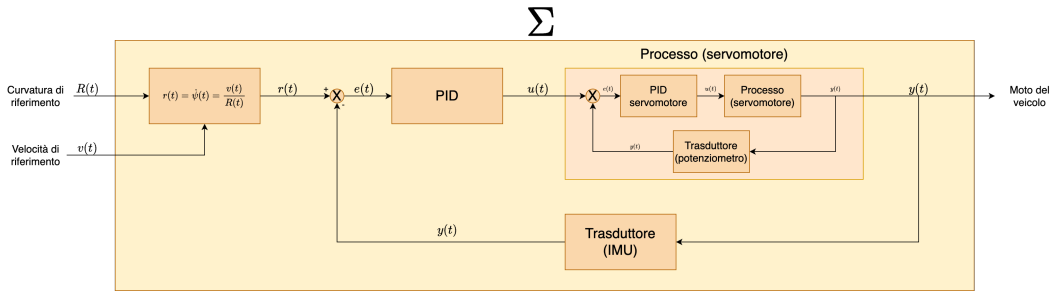


Figura 8.24.: Ciclo di controllo del servosterzo, vista interna.

Nelle prossime sezioni, come già fatto per il controllo della trazione, verranno brevemente analizzati i componenti del ciclo di controllo, in modo da comprendere in modo chiaro l'implementazione degli algoritmi di controllo utilizzati.

Servomotore

Il servomotore rappresenta il processo da controllare. Per il controllo del servomotore è stato impiegato un approccio analogo al controllo del motore di trazione.

Una differenza sostanziale risiede però nel tipo di controllo: invece di un controllo in velocità, come nel caso del motore brushless, il servomotore viene controllato in posizione, facendo riferimento all'angolo compreso tra la posizione di riposo e quella di massimo scostamento. Nel contesto applicativo è stato utilizzato un servomotore con un angolo di rotazione compreso tra 0^0 e 180^0.

La relazione tra il duty cycle e l'angolo di rotazione è di tipo lineare e risulta essere:

$$\theta = \delta_\theta \cdot 180^0 \tag{8.4}$$

Questo tipo di controllo è adatto al contesto applicativo in quanto, collegando l'asse del servomotore al meccanismo di sterzo è possibile controllare l'angolo di sterzo delle ruote anteriori del veicolo. Come introdotto in precedenza, il servomotore presenta un ciclo di controllo integrato, che implementa il controllo in posizione sopracitato. Questo ciclo di controllo può essere di tipologia analogica o digitale, in funzione della tecnologia, ma il principio di funzionamento è lo stesso di un controllore a controreazione di tipo PID. Le componenti di questo ciclo sono le seguenti:

- **Processo**: è un motore DC che può ruotare in entrambe le direzioni di rotazione, solitamente connesso ad un moto-riduttore che diminuisce la velocità dell'asse in uscita dal servomotore, aumentandone di un fattore considerevole la coppia;
- **Trasduttore**: generalmente è un potenziometro rotativo, che opportunamente collegato all'asse di rotazione finale, fornisce una misurazione della sua rotazione, espressa come un valore di tensione;
- **Controllore**: questo, che può essere di natura analogica o digitale, confronta il segnale fornito in ingresso dall'utente, solitamente di tipo PWM, con il valore fornito dal potenziometro. Esso si occupa di ridurre a 0 l'errore commesso dal servomotore, attuando il motore DC in maniera appropriata.

Grazie a questo ciclo di controllo integrato, il servomotore può essere gestito mediante un segnale di controllo digitale di tipo PWM, regolando l'ampiezza dell'angolo del suo asse rispetto alla sua posizione di riposo.

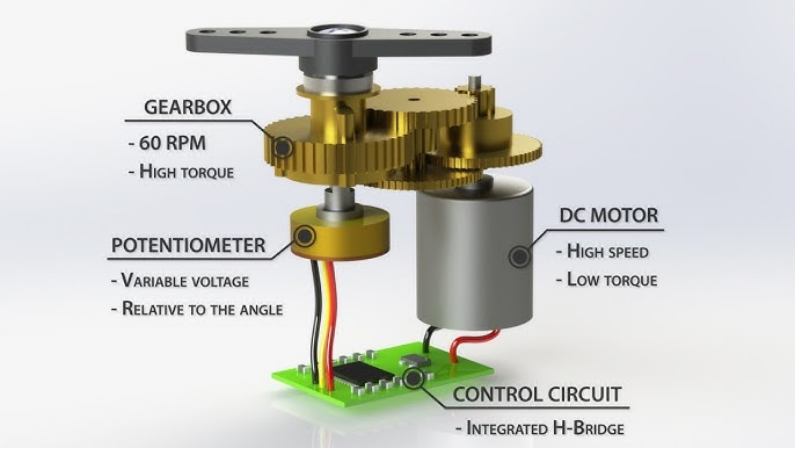


Figura 8.25.: Componenti di un generico servomotore.

IMU BNO055

L'IMU BNO055 rappresenta il blocco di trasduzione del ciclo di controllo. In questo caso, il sensore inerziale ha il compito di fornire una misurazione dello yaw rate del veicolo, che ne permette il controllo laterale secondo quanto discusso in precedenza.

Figura 8.26.: IMU BNO055.

I dati di interesse vengono calcolati a bordo del sensore da un microcontrollore integrato, e vengono prelevati mediante protocollo Inter Integrated Circuit (I^2C) dal microcontrollore STM32, ad intervalli temporali regolari.

Nelle sezioni successive verranno trattati gli algoritmi sviluppati su microcontrollore per la lettura e l'interpretazione dei dati forniti dall'IMU, che sono fondamentali per la valutazione della funzione errore $e(t)$ e il conseguente controllo in retroazione.

Controllore PID, limiti funzionali e scelta dei parametri

Anche nel caso del controllo laterale del veicolo, come in quello longitudinale, è stato utilizzato un controllore PID, in quanto si presta bene all'applicazione di interesse.

A differenza del caso del controllo della trazione, per il controllo del servosterzo è necessario limitare il campo di azione del controllore PID, in quanto l'intervallo degli angoli di sterzo ammissibili è ridotto rispetto al campo operativo del servomotore. Difatti, sebbene l'asse del servomotore possa ruotare da un angolo minimo di $-90°$ ad un massimo di $90°$, il meccanismo di sterzo presenta dei limiti meccanici, precedentemente calcolati, che riducono il campo operativo ad un intervallo compreso tra $-23°$ e $23°$.

Per questo motivo, è stato inserito un algoritmo di clamping al controllore, di cui verrà discussa l'implementazione nelle prossime sezioni. Questo algoritmo ha lo scopo di limitare lo sforzo di controllo in un certo intervallo, prevenendo dunque eventuali danni al meccanismo dovuti ad uno sforzo meccanico eccessivo.

Per quanto riguarda la scelta dei parametri, essi sono stati ricavati sperimentalmente a seguito di più iterazioni, e sono riportati nella Tabella 8.3 .

Kp	20
Ki	250
Kd	0
Kb	50

Tabella 8.3.: Taratura PID di sterzo

Un'altra differenza rispetto al PID di trazione, generata dalla presenza dei limiti fisici del processo è rappresentata dalla presenza di un ulteriore parametro K_b. Questo parametro rappresenta il coefficiente dell'azione di back calculation, che è un algoritmo di anti-windup del termine integrale. Gli algoritmi di anti-windup appartengono ad una classe di soluzioni progettate per attenuare l'azione integrale, che tende a far crescere lo scorzo di controllo in modo eccessivo quando il campo operativo del processo è limitato e lo sforzo massimo non è sufficiente ad eliminare l'errore nel ciclo di controllo.

Questo fenomeno, nel caso del controllo laterale del veicolo, si verifica se viene assegnata una curvatura di riferimento maggiore rispetto a quella che il meccanismo di sterzo può fisicamente seguire.

Di seguito, in Figura 8.27, è riportato lo schema a blocchi del controllore PID con meccanismo di anti-windup.

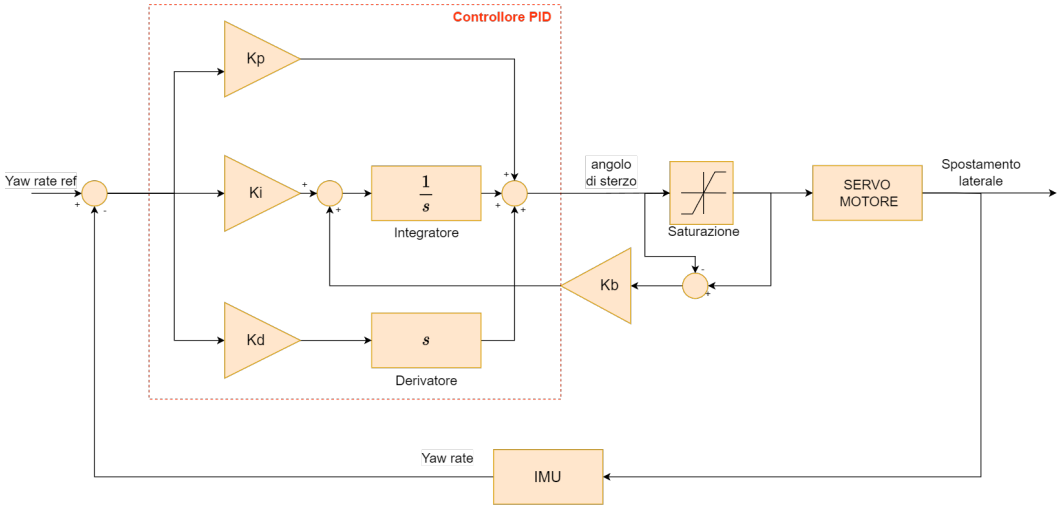


Figura 8.27.: Ciclo di controllo con anti windup.

8.6.2. Implementazione

Per implementare il controllo laterale del veicolo, è necessario sviluppare le seguenti funzionalità:

- Calcolo del riferimento di yaw rate e del valore corrente fornito dal sensore inerziale;

- Implementazione del controllore PID, valutando la funzione errore e producendo lo sforzo di controllo adeguato;
- Attuazione del servomotore di sterzo, utilizzando lo sforzo di controllo prodotto dal controllore.

Nei prossimi paragrafi verranno dunque discusse le implementazioni di queste funzionalità, mostrando infine i risultati sperimentali ottenuti.

Configurazione dell'IOC per la lettura dei dati forniti dall'IMU

La lettura dei dati calcolati a bordo del sensore BNO055 avviene prelevando i dati di interesse dai registri del microcontrollore presente sul sensore. Questa operazione avviene mediante protocollo Inter Integrated Circuit (I^2C), che è un particolare protocollo di comunicazione che si basa sull'interazione di più dispositivi, classificati in master e slave.

Nel caso di interesse, il dispositivo master è il microcontrollore STM32 F401RE, mentre lo slave è il sensore inerziale. Il ruolo del master è quello di richiedere i dati sulla rete di comunicazione mediante l'invio di determinati messaggi, mentre quello dello slave è quello di analizzare i messaggi pubblicati dal master e agire di conseguenza, in risposta ai segnali di controllo ricevuti.

Per implementare la comunicazione I^C è necessario configurare la porta sul microcontrollore, mediante l'IOC fornito dall'IDE. Nel caso di interesse, è stata abilitata la porta I2C1, con i parametri di configurazione mostrati in Figura 8.28.

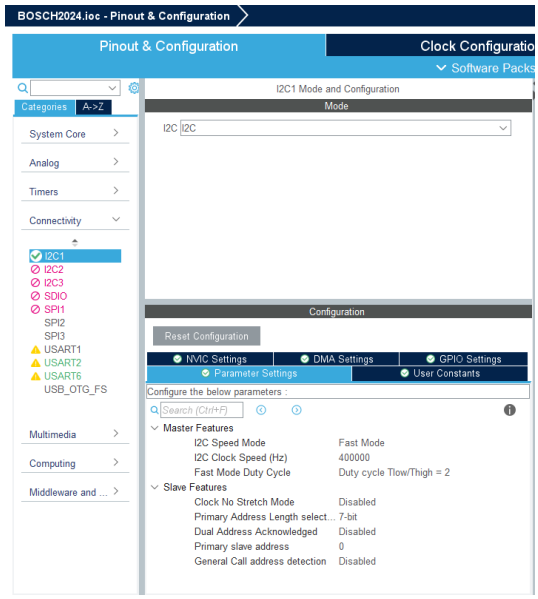


Figura 8.28.: Configurazione dell'IOC relativa alla porta I2C1.

In termini di hardware, sono state configurate le porte GPIO come riportato in Figura 8.28, in quanto per comunicare, i due dispositivi devono essere collegati

fisicamente mediante due canali, denominati SCL (Serial CLock line) e SDA (Serial DAta line). Come è facile intuire, il protocollo è sincrono, in quanto il canale SCL detta il clock della comunicazione, mentre il canale SDA trasporta i dati tra i dispositivi.

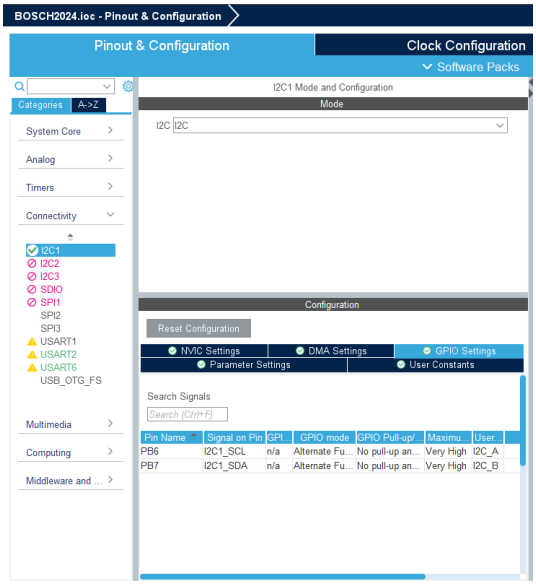


Figura 8.29.: Configurazione porte GPIO della comunicazione I2C1.

Codice sorgente per la lettura dei dati di yaw rate

Prima di essere in grado di leggere i dati forniti dal sensore, è necessario configurare sia la comunicazione I^2C che il sensore. Queste operazioni vengono effettuate prima dell'inizio del ciclo di controllo, nella sezione di inizializzazione delle periferiche, e sono seguite nel seguente blocco di codice.

```
1     MX_I2C1_Init();
2
3     resetBNO055();  //IMU BNO055 Configuration
4   HAL_I2C_IsDeviceReady(&hi2c1, BNO055_I2C_ADDR << 1, 5, 1000);
5   bno055_assignI2C(&hi2c1);
6   bno055_setup();
7   bno055_setOperationModeNDOF();
8     printf("Initialization Completed!\r\n");
```

Listing 8.21: Codice di inizializzazione del sensore BNO055.

Una volta inizializzato il sensore, è possibile ricavare i dati telemetrici desiderati mediante le funzioni fornite dal driver del costruttore, riportato in Appendice D.

Nel ciclo di controllo sono state quindi usate le suddette funzioni, e per quanto riguarda la lettura del dato relativo allo yaw rate corrente è sufficiente utilizzare la funzione **bno055_getVectorGyroscope()**, che fornisce in output un vettore di 3 elementi contenente le velocità di rotazione attorno ai 3 assi cartesiani, ovvero quelle

di Roll Rate, Pitch Rate e Yaw Rate. Successivamente viene salvato il valore di Yaw Rate, convertito successivamente in radianti al secondo. Come ultimo passo, viene memorizzato l'ultimo valore di riferimento di curvatura, fornito dalla comunicazione seriale con la Raspberry Pi. Di seguito è riportato il codice che implementa le funzionalità appena trattate.

```
1  //-------------------------------------------------------------
2
3      //STEERING control
4
5      //Get and process yaw rate from IMU sensor
6      bno055_vector_t v = bno055_getVectorGyroscope();
7      vehicleState.yaw_rate_deg_sec = v.z;
8      vehicleState.yaw_rate_rad_sec = (vehicleState.yaw_rate_deg_sec *
       M_PI) / 180;
9      last_read = dataRX.curvature_radius_ref_m;
10
11     //Servo control using PID
```

Listing 8.22: Lettura ed interpretazione dello yaw rate.

A differenza della lettura del valore di velocità del motore di trazione, in questo caso non è stato utilizzato un filtro, poiché avrebbe introdotto un ritardo ulteriore al ciclo di controllo, che è già rallentato dalla dinamica del processo. Nella sezione dei risultati sperimentali verrà ulteriormente discussa la questione una volta ottenuti i dati sperimentali.

Configurazione dell'IOC per l'attuazione del servomotore

Così come per l'attuazione del motore di trazione, anche per attuare il servomotore di sterzo è stato utilizzato un timer in modalità PWM, nello specifico il timer TIM1.

Il controllo richiede una portante di frequenza 50Hz e quindi di periodo 20ms; l'angolo di rotazione minimo si ottiene con un impulso di 0.5ms mentre l'angolo di rotazione massimo si ottiene con un impulso di 2.5ms. La relazione che permette di assegnare correttamente il valore di duty cycle al servomotore risulta quindi essere la seguente:

$$\delta = \frac{0.5 + \left(\frac{\theta}{180^0}\right) \cdot 2}{20} \qquad \theta \in (0^0, 180^0) \tag{8.5}$$

A livello di implementazione, è necessario inizializzare un timer in modalità PWM (TIM1) con parametri del tutto analoghi al timer utilizzato per la generazione del PWM per il motore brushless. I valori di prescaler e counter period sono riportati in Figura 8.30 .

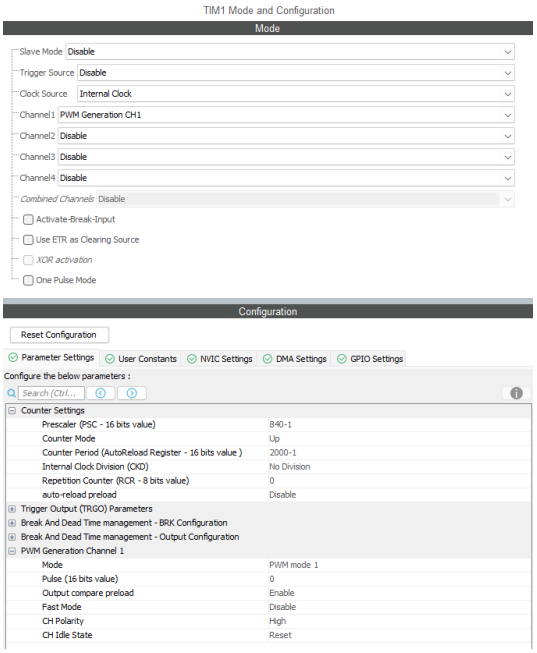


Figura 8.30.: Configurazione TIM1

Attuazione del servomotore

Applicando la Relazione in funzione della dimensione del registro di autoreload (ARR) e ricordando che sussiste una relazione lineare tra la durata di un ciclo del timer e la dimensione del registro sopra menzionato, si ottiene il metodo riportato di seguito, contenuto nell'apposito file servo_motor.c .

```
#include "servo_motor.h"

void servo_motor(float angolo){

  if(angolo < MIN_ANGOLO)
    angolo = MIN_ANGOLO;
  else if (angolo > MAX_ANGOLO)
    angolo = MAX_ANGOLO;

  float angolo_corretto = angolo + DRITTO;

  int min_duty = TIM1->ARR/40; //0.5ms

  float duty = (angolo_corretto / 180) * (TIM1->ARR/10); //0ms - 2ms

  TIM1->CCR1 = min_duty + (int)duty;

}
```

Listing 8.23: FIle sorgente servo_motor.c .

Le costanti MIN_ANGOLO, MAX_ANGOLO e DRITTO sono definite nel file header servo_motor.h, che è così strutturato:

```

#include <main.h>

#ifndef INC_SERVO_MOTOR_H_
#define INC_SERVO_MOTOR_H_

#define DRITTO 90
#define MAX_ANGOLO 23
#define MIN_ANGOLO -23
#define PI 3.141592654

void servo_motor(float);

#endif /* INC_SERVO_MOTOR_H_ */
```

Listing 8.24: File header servo_motor.h .

Chiamando il metodo **servo_motor(float angolo)** è quindi possibile generare il segnale PWM necessario per controllare in posizione il servomotore.

Implementazione del controllo mediante PID

Nelle sezioni precedenti, nonché nei capitoli riguardanti la formulazione teorica della soluzione, è stata introdotta la metodologia operativa utilizzata per il controllo laterale del veicolo. In riferimento a quanto detto in precedenza, è stato sviluppato il codice di controllo riportato di seguito.

```
//------------------------------------------------------------

    //STEERING control

    //Get yawrate from IMU

    // Check for curvature values that define a straight line
    if (dataRX.curvature_radius_ref_m >=
    MAX_CURVATURE_RADIUS_FOR_STRAIGHT) {

        vehicleState.yaw_rate_ref_rad_sec = 0;

        u_sterzo = PID_controller(&pid_steering, vehicleState.
    yaw_rate_rad_sec, vehicleState.yaw_rate_ref_rad_sec);

        servo_motor(-u_sterzo); //Minus because yawrate and steering
    are opposite
    } else {

        vehicleState.linear_speed_m_s = vehicleState.motor_speed_RPM *
     RPM_2_m_s;
        if (dataRX.curvature_radius_ref_m == 0)
```

```
19              vehicleState.yaw_rate_ref_rad_sec = 0;
20          else
21              vehicleState.yaw_rate_ref_rad_sec = vehicleState.
    linear_speed_m_s / dataRX.curvature_radius_ref_m;
22
23          float yaw_rate_ref_rad_sec_abs = vehicleState.
    yaw_rate_ref_rad_sec;
24          float yaw_rate_rad_sec_abs = vehicleState.yaw_rate_rad_sec;
25          if (vehicleState.yaw_rate_ref_rad_sec < 0)
26              yaw_rate_ref_rad_sec_abs = -vehicleState.
    yaw_rate_ref_rad_sec;
27          if (vehicleState.yaw_rate_rad_sec < 0)
28              yaw_rate_rad_sec_abs = -vehicleState.yaw_rate_rad_sec;
29
30          u_sterzo = PID_controller(&pid_steering, yaw_rate_rad_sec_abs,
     yaw_rate_ref_rad_sec_abs);
31
32          //Minus because yawrate and steering are opposite
33          if (dataRX.curvature_radius_ref_m >= 0 && u_sterzo > 0)
34              u_sterzo *= -1.0;
35          if (dataRX.curvature_radius_ref_m < 0 && u_sterzo < 0)
36              u_sterzo *= -1.0;
37
38          servo_motor(u_sterzo);
39      }
40      dataTX.current_servo_angle_deg = u_sterzo;
```

Listing 8.25: Iterazione del ciclo di controllo laterale del veicolo.

In questo segmento di codice, che rappresenta una singola iterazione del ciclo di controllo, vengono eseguite le seguenti azioni:

- Lettura dei valori di riferimento e di yaw rate reale;
- Controllo per valori di curvatura superiori ad una soglia che definisce una linea retta: questo controllo viene effettuato per ridurre le oscillazioni a regime durante i tratti rettilinei, limitando il controllo eseguito dal PID solo alle curve e alle manovre;
- Calcolo del riferimento in yaw rate, utilizzando la relazione che lo lega alla velocità e al raggio di curvatura istantaneo;
- Calcolo del segno del riferimento, in funzione della direzione della curva;
- Calcolo dello sforzo di controllo, mediante l'algoritmo PID;
- Attuazione del servomotore.

Al termine del ciclo di controllo viene salvato il valore dello stato del servosterzo, il quale viene successivamente inviato mediante comunicazione seriale alla Raspberry Pi.

Risultati sperimentali

Per quanto riguarda il controllo laterale, di seguito è riportato un grafico ottenuto durante una manovra di sorpasso: viene evidenziata la differenza tra il valore di riferimento in yaw rate e quello misurato dai sensori a bordo. Come è possibile osservare in Figura 8.31, è presente un notevole sfasamento tra il riferimento e il valore effettivo, causato dal tempo di salita del processo di attuazione del servomotore, che è dato dal ciclo di controllo integrato all'attuatore stesso.

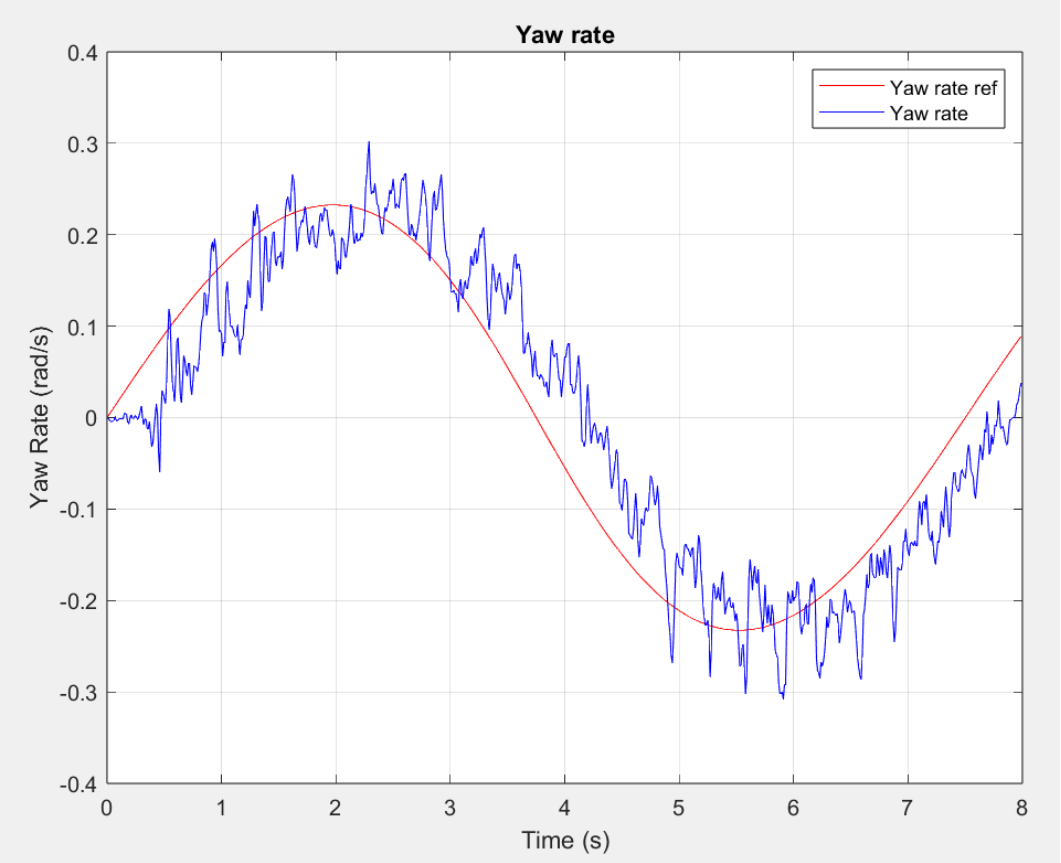


Figura 8.31.: Risposta del sistema rispetto ad un riferimento sinusoidale.

A causa di questo ritardo intrinseco, si è scelto di non implementare un filtro a media mobile per lo smoothing del segnale letto, in quanto avrebbe introdotto un ulteriore ritardo direttamente proporzionale alla finestra di scorrimento.

Capitolo 9.

Implementazione su Raspberry Pi 4

9.1. Introduzione

In questo capitolo verranno esplorati gli algoritmi relativi alle manovre di parcheggio e di sorpasso implementati ad alto livello, ovvero su Raspberry Pi 4. Verrà inizialmente mostrata una panoramica dell'architettura utilizzata, con particolare focus sul middleware impiegato, ovvero ROS2.

Di seguito verrà esplicitata l'implementazione della comunicazione tra la Raspberry Pi e il microcontrollore, mediante protocollo seriale.

Verrà infine illustrata l'implementazione degli algoritmi adibiti alla generazione dei riferimenti in velocità e curvatura che vengono poi processati dal microcontrollore, come trattato fino ad ora.

9.2. Middleware ROS2

Il middleware ROS2 (Robot Operating System 2) è un framework software progettato per facilitare lo sviluppo e l'integrazione di applicazioni robotiche distribuite. Si basa su un'architettura orientata ai messaggi, che consente la comunicazione tra nodi software attraverso una rete, garantendo un'interazione efficiente e scalabile. A differenza del suo predecessore ROS1, ROS2 incorpora una serie di miglioramenti significativi, tra cui il supporto per la comunicazione in tempo reale, la sicurezza e l'adozione di standard come DDS (Data Distribution Service).

Figura 9.1.: Logo del middleware ROS2.

ROS2 fornisce un insieme di librerie e strumenti che semplificano la creazione di componenti riutilizzabili e modulabili, permettendo agli sviluppatori di concentrarsi sulla logica applicativa piuttosto che sulla gestione della comunicazione. Inoltre, la sua architettura decentralizzata favorisce l'implementazione di sistemi robotici

complessi e distribuiti, rendendo ROS2 adatto per applicazioni in vari contesti, dalla robotica di servizio, all'industria e alla ricerca.

In sintesi, il middleware ROS2 rappresenta una soluzione avanzata per la progettazione di sistemi robotici interoperabili e adattabili, promuovendo la collaborazione tra diversi componenti e facilitando l'innovazione nel campo della robotica.

9.2.1. Distribuzione utilizzata

Esistono diverse distribuzioni del middleware ROS2. Nel particolare contesto applicativo è stata utilizzata la distribuzione Foxy in quanto è una delle release più supportate e stabili di ROS2. Per implementare la struttura si è fatto riferimento alla documentazione ufficiale [14] e a tutorial [1].

Questa distribuzione è pensata per essere utilizzata in congiunzione al sistema operativo Ubuntu 20.04, rendendola quindi adatta all'uso sulla Raspberry Pi 4.

9.3. Architettura ROS2

L'architettura utilizzata è quella descritta in Figura 3.1 del capitolo 3. In particolare, ogni blocco dell'architettura rappresenta in ROS2 un package, ovvero una unità di base del funzionamento del sistema.

Ogni package contiene uno o più eseguibili, che a loro volta possono contenere uno o più nodi. Un nodo rappresenta l'unità fondamentale funzionale di ROS, che a sua volta può comunicare con uno o più nodi, in modo da dividere un processo in più sotto-processi isolati che lavorano in maniera coordinata.

I nodi implementano dunque le singole funzionalità del package, comunicando mediante un sistema di publisher/subscriber proprio del funzionamento di ROS2. Un nodo publisher, quindi, "pubblica" dei messaggi su di un certo topic, che rappresenta un costrutto a cui i nodi subscriber, se "iscritti" ad esso possono accedere.

In ROS2, il codice dei nodi può essere scritto in C++ o in Python, a discrezione dello sviluppatore. Nel contesto della robotica è però consigliato utilizzare, ove possibile, C++ in quanto offre prestazioni migliori rispetto a Python. Ovviamente questa scelta va fatta anche in funzione della funzionalità da sviluppare, che potrebbe richiedere un alto livello di astrazione, proprio del linguaggio Python.

9.4. Comunicazione seriale

Come nel capitolo dedicato all'implementazione su microcontrollore, il primo blocco funzionale che verrà trattato è quello che implementa la comunicazione seriale tra i due dispositivi, sia in lettura che in scrittura.

9.4.1. Serial handler

Questo pacchetto contiene i metodi utilizzati per leggere e scrivere messaggi sulla porta seriale della Raspberry, permettendo dunque la comunicazione bidirezionale col microcontrollore.

Per stabilire una comunicazione, la porta seriale va innanzitutto abilitata e le vanno forniti i permessi necessari per essere usata correttamente. Nell'implementazione specifica sono stati riscontrati numerosi problemi utilizzando la porta UART default della Raspberry (/dev/ttyS0), per cui è stata utilizzata una porta alternativa, rappresentata da /dev/ttyAMA2.

Dopo aver configurato la porta AMA2, si accede ad essa tramite la libreria PySerial, impostandone i parametri e configurando il baud rate. Nello specifico, è stato utilizzato un baud rate di 115200 bit/s. Di seguito è riportato il codice di inizializzazione della seriale.

```
import serial

speed = 115200
self.ser = serial.Serial ("/ dev/ ttyAMA2 ",
                                  speed ,
                                  timeout =60)
```

Per quanti riguarda l'invio di messaggi, questo vengono prima costruiti seguendo una struttura ben precisa, del tipo "flag, velocità, raggio_di_curvatura". Ad ogni iterazione degli algoritmi decisionali i tre valori sopramenzionati vengono modificati ed inviati sotto forma di stringhe, utilizzando il metodo write(). Di seguito è riportato il codice relativo alla scrittura dei messaggi contenenti i riferimenti di controllo.

```

def listener_callback(self, msg):
    try :
        data = [msg.flag, msg.vel, msg.radius]
        serial_msg = self.string_formatter(data)
        self.ser.write(bytes(serial_msg, 'ascii') )
    except Exception as e:
        self.get_logger().error(f  'Writing on serial error { e } ')
```

Per quanto riguarda la ricezione dei dati, anch'essa è una funzionalità fondamentale, poiché gli algoritmi di sensor fusion la utilizzano per ottenere una stima della posizione corrente del veicolo. Questa lettura viene fatta ad una frequuenza costante di 50Hz, per cui all'inizio del codice viene istanziato un timer che regola il flusso di dati in ingresso. Di seguito è riportato il codice relativo alla lettura dei dati telemetrici forniti dal microcontrollore.

```
# Timer per la lettura dall'IMU
self.timer_period = 1.0 / 5
3 self.timer = self.create_timer(self.timer_period, self.
    publish_imu_msg )

```

```
def publish_imu_msg ( self ) :
    while self.ser.in_waiting > 0:
        raw_data = self.ser.readline()
        try :
            line = raw_data.decode('utf-8').strip()
            data = line.split(', ')
            if len(data) == 10 and verify_values(data):
                imuMsg = Imu()
                imuMsg.header.stamp = self.get_clock().now().to_msg()
                imuMsg.header.frame_id = 'imu_link'
                imuMsg.orientation.w = float(data[9]) # encoder
                imuMsg.orientation.x = float(data[6]) # magnetometer
                imuMsg.orientation.y = float(data[7]) # magnetometer
                imuMsg.orientation.z = float(data[8]) # magnetometer
                imuMsg.angular_velocity.x = float(data[3]) # calib.
                imuMsg.angular_velocity.y = float(data[4])
                imuMsg.angular_velocity.z = float(data[5])
                imuMsg.linear_acceleration.x = float(data[0])
                imuMsg.linear_acceleration.y = float(data[1])
                imuMsg.linear_acceleration.z = float(data[2])
                self.imu_output.publish(imuMsg)
                # Scrittura dei dati sul file di log
                with open(self.log_file_path, 'a') as log_file:
                    log_file.write (f"{imuMsg.header.stamp},
                                      {imuMsg.orientation.w},
                                      {imuMsg. orientation.x},
                                      {imuMsg.orientation.y},
                                      {imuMsg.orientation.z},
                                      {imuMsg.angular_velocity.x},
                                      {imuMsg.angular_velocity.y},
                                      {imuMsg.angular_velocity.z},
                                      {imuMsg.linear_acceleration.x},
                                      {imuMsg.linear_acceleration.y},
                                      {imuMsg.linear_acceleration.z}
                                      \n")
            else:
                print (" Data not valid ")
        except UnicodeDecodeError :
            continue
        except ValueError :
            continue
```

9.5. Manovre di sorpasso e parcheggio

Sulla Raspberry Pi sono stati implementati gli algoritmi che rendono possibili le manovre di sorpasso e di parcheggio, entrambi basati sulla funzione di riferimento ricavata analiticamente nel capitolo 6. Per quanto riguarda la logica decisionale, questa verrà esplorata solo in termini di pseudo algoritmi, in quanto troppo complessa per il contesto di questa trattazione.

Verrà invece approfondito il pacchetto Maneuver Handler, che si occupa della generazione del riferimento in curvatura e velocità per la corretta esecuzione di entrambe le manovre.

9.5.1. Logica di sorpasso

Quella di sorpasso è una manovra di object avoidance, che viene eseguita ogni qualvolta, durante la marcia del veicolo, se ne incontri un altro davanti, sia nella condizione stazionaria che nella condizione di moto a velocità minore rispetto a quella di crociera.

In entrambi i casi, il veicolo deve rilevare l'ostacolo, seguirlo ad una distanza di sicurezza (eventualmente fermandosi) e dopo un certo lasso di tempo eseguire la manovra di sorpasso, dopo aver controllato il tipo di linea di mezzeria. Difatti, se la linea di mezzeria risulta essere continua, la manovra di sorpasso non può essere eseguita ed il veicolo deve seguire quello che lo precede fintantoché non raggiunge un tratto con linea di mezzeria tratteggiata, eseguendo dunque il sorpasso, portandosi dapprima nella corsia di sinistra e poi rientrando in quella originaria.

Il riconoscimento dell'eventuale veicolo ostacolante viene effettuato mediante il sensore LiDAR 2D posto sul fronte del veicolo autonomo, che individua eventuali ostacoli su una circonferenza giacente sul piano orizzontale del veicolo. L'algoritmo appena descritto può essere rappresentato attraverso uno schema a blocchi, che ne illustra il funzionamento, come mostrato in Figura 9.2 .

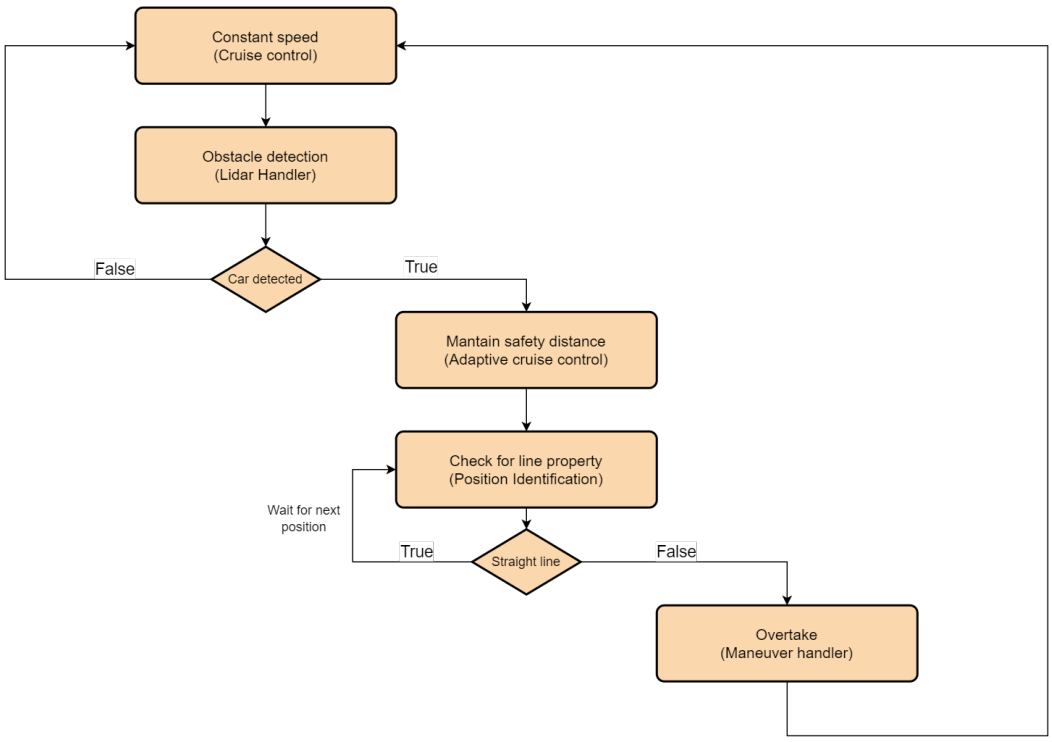


Figura 9.2.: Algoritmo di sorpasso, passaggi essenziali.

9.5.2. Logica di parcheggio

Per quanto riguarda la logica di parcheggio, che verrà anch'essa trattata riportandone i passaggi essenziali, la situazione è analoga a quella della logica di sorpasso, a meno di qualche dettaglio.

La prima considerazione da fare riguarda il campo visivo del LiDAR, che è in grado di rilevare la presenza di ostacoli attorno a sé con un campo "visivo" di 270°, pari a 135° in entrambe le direzioni rispetto a quella longitudinale. Questa caratteristica del sensore permette non solo di rilevare ostacoli frontali, ma anche quelli laterali, permettendo quindi anche di individuare la presenza di veicoli ai lati di esso. Questa proprietà viene sfruttata, come verrà successivamente descritto, per individuare gli slot di parcheggio liberi, dove quindi è possibile parcheggiare.

Un'altra caratteristica del percorso su cui si basa questo algoritmo è la costruzione del grafo rispetto al percorso di gara. Nell'area destinata ai parcheggi, il nodi sono disposti in modo alternato: inizialmente nei centri degli slot di parcheggio, seguiti da nodi posizionati tra due slot adiacenti. Questa configurazione è cruciale ed è illustrata in Figura 9.3.

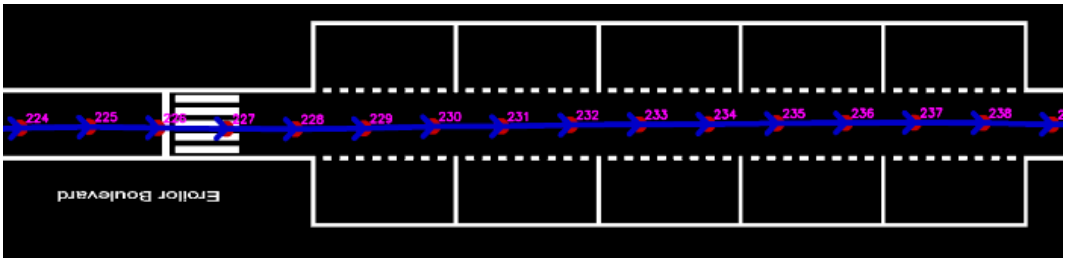


Figura 9.3.: Area di parcheggio e struttura del grafo associato ad essa.

Inoltre, la terza ed ultima caratteristica del percorso è la delimitazione dell'area di parcheggio da appositi cartelli stradali, posti su entrambi i lati della corsia, sia in ingresso che in uscita da essa. Il cartello stradale di parcheggio presenta la seguente grafica, riportata in Figura 9.4 .

Figura 9.4.: Cartello di parcheggio.

Per quanto riguarda l'algoritmo che implementa la logica di parcheggio, nel diagramma a blocchi riportato di seguito sono presentati i passaggi essenziali.

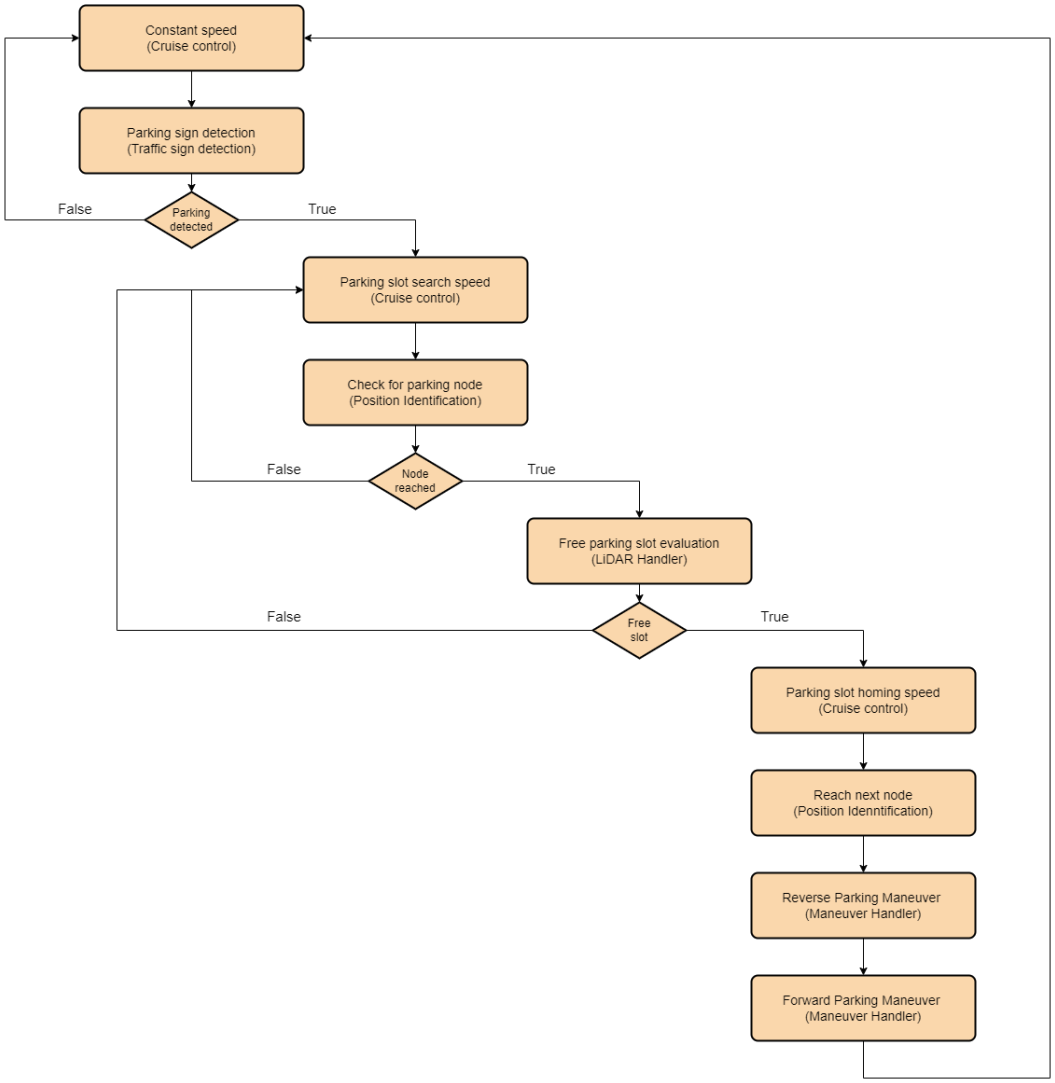


Figura 9.5.: Algoritmo di parcheggio, passaggi essenziali.

Per quanto riguarda l'algoritmo di parcheggio, a differenza di quello di sorpasso, esso fa uso anche delle informazioni ricavate dall'identificazione dei cartelli stradali mediante reti neurali, che si occupano di analizzare il feed fornito dalla videocamera a bordo.

9.5.3. Maneuver Handler

L'algoritmo che si occupa del calcolo del riferimento in velocità e curvatura per le manovre di sorpasso e parcheggio, è gestito da un unico pacchetto ROS, denominato Maneuver Handler. Questo pacchetto, scritto in C++, implementa sia un forwarding dei messaggi generati dagli algoritmi di Lane Keeping, durante la marcia del veicolo,

che la generazione dei riferimenti in caso di esecuzione di manovre di sorpasso e parcheggio. La funzione che genera i riferimenti risulta essere unica per entrambe le manovre, in quanto differiscono solo per la direzione del moto, ovvero per il segno della velocità a cui vengono eseguite.

Dunque, è possibile identificare gli ingressi e le uscite del Maneuver Hander, che sono così definite:

- **Riferimenti in velocità e curvatura**: questi riferimenti, forniti in ingresso al nodo dagli algoritmi di lane keeping, in condizioni normali di marcia, vengono pubblicati dal nodo stesso ed inviati mediante comunicazione seriale;
- **Flag di controllo**: questa flag identifica la tipologia di manovra da eseguire e viene fornita dal blocco decisionale;
- **Riferimenti in velocità e curvatura**: questi dati vengono prodotti in uscita dal blocco, in riferimento alla manovra da eseguire.

Di seguito è riportato il codice relativo alla gestione della manovra.

```
void choose_maneuver ( const std::shared_ptr < custom_msgs::msg::
    Maneuver > &msg) {
    if (msg->flag == 1) {
        output_msg.vel = msg->vel;
        output_msg.radius = msg->radius;
    } else if (msg->flag == 6) {
        float maneuver_time = calculate_time(msg->dy,msg->dx, msg->vel
    );
        // float maneuver_time = calculate_distance(msg->dy, msg->dx)/
    msg->vel;
        auto start = std::chrono::system_clock::now();
        // Ha visto il cartello di parcheggio o deve sorpassare
        while(true) {
            auto end = std::chrono::system_clock::now();
            auto elapsed = std::chrono::duration_cast <
            std::chrono::milliseconds>(end-start).count()/1000.0 f;
            if (elapsed >= maneuver_time) {
                break ;
            }
        float yaw_rate = calculate_yaw_rate (msg->dy, msg->dx, msg->
    vel, elapsed);
        output_msg.vel = msg->vel;
        output_msg.radius = msg->vel/yaw_rate;
        publisher_ ->publish (output_msg);
        std::this_thread::sleep_for(std::chrono::milliseconds(100));
        }
    } else {
        RCLCPP_INFO(this ->get_logger(), "Wrong flag");
    }
    publisher_ ->publish(output_msg);
}
```

Listing 9.1: Scelta della manovra

Se dal blocco decisionale viene, ad esempio, inviata una flag con valore '6', viene eseguita la manovra di parcheggio. La lunghezza del percorso, e dunque la durata della manovra, viene stimata con le metodologie discusse nel capitolo 6, e il codice che implementa questo calcolo è riportato di seguito.

```
1
2 float calculate_time ( float dy , float dx , float vel) {
3     return (LAMBDA_SCALAR*sqrt(dx*dx + dy*dy) / std::fabs( vel)) +
4     TEMPO_SALITA_PID ;
5 }
```

Listing 9.2: Calcolo del tempo di esecuzione della manovra.

Questo metodo tiene anche conto del tempo di salita del controllore PID, che deve portare il veicolo da una velocità nulla ad una costante di manovra.

Per il calcolo vero e proprio del riferimento in yaw rate, viene utilizzata la funzione descritta analiticamente nei capitoli precedenti, con i dovuti accorgimenti di ammissibilità. Inoltre, sperimentalmente, sono stati ricavati dei fattori di correzione che tengono conto dell'attenuazione e dello sfasamento della traiettoria descritta dal veicolo, a causa dei tempi di salita degli attuatori e dalle imprecisioni nel moto del veicolo.

Di seguito sono riportati i valori dei fattori di correzione e la funzione che calcola il riferimento in yaw rate per l'esecuzione della manovra.

```
1 static constexpr float MIN_DX = 2.00;
2 static constexpr float DY_ADJUST_FACTOR = -0.04;
3 static constexpr float DY_ADJUST_OFFSET = 0.08;
4 static constexpr float LAMBDA_SCALAR = 1.08;
5 static constexpr float TEMPO_SALITA_PID = 0.0;
6
7 float calculate_yaw_rate ( float dy, float dx, float v, float t) {
8     float lambda = dx;
9     if (dx < MIN_DX ) {
10         dy += DY_ADJUST_FACTOR*dx + DY_ADJUST_OFFSET;
11         lambda = LAMBDA_SCALAR*sqrt(dx*dx+dy*dy);
12     }
13     if (lambda*lambda <= 4*dy*dy) {
14         lambda = 2*dy;
15     }
16
17     float angle = M_PI/2-(2*M_PI*t*v)/lambda;
18
19     return (2*dy*v*M_PI*cos(angle))/
20     (lambda*lambda*sqrt(1-pow((dy*v/
21     lambda-(dy*v*sin( angle))/lambda),2)/(v*v)));
22 }
```

Listing 9.3: Fattori di correzione e calcolo dello yaw rate di riferimento.

9.5.4. Risultati sperimentali, criticità e miglioramenti

Per quanto riguarda i risultati sperimentali, in Figura 9.6 è mostrato uno dei risultati sperimentali ottenuti durante la fase di validazione dell'algoritmo. Come è possibile notare, sono evidenti lo sfasamento e l'attenuazione caratteristici dell'esecuzione della manovra, che vengono dunque compensati a monte con i fattori di correzione illustrati in precedenza. Un possibile miglioramento dell'algoritmo potrebbe consistere in una valutazione dinamica dei fattori di correzione fatta, ad esempio, mediante tecniche di machine learning.

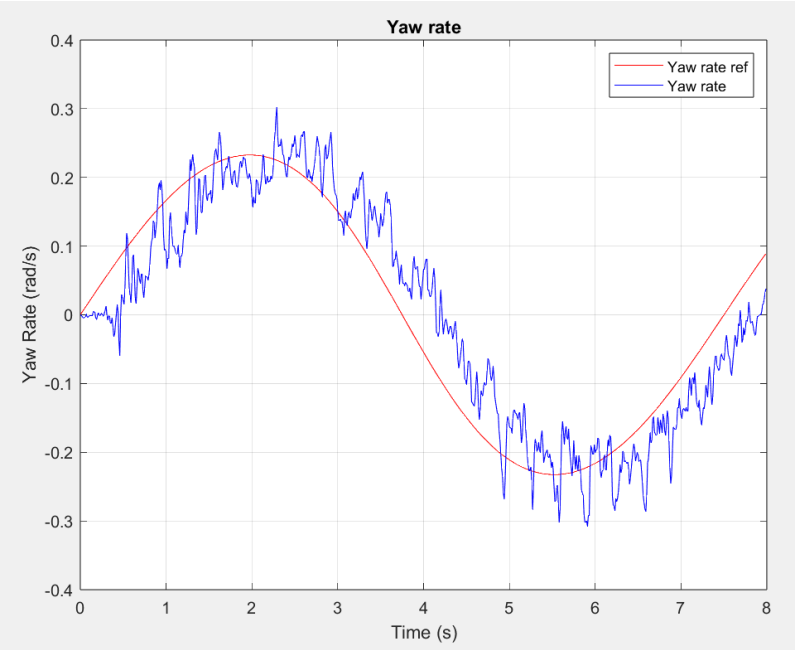


Figura 9.6.: Riferimento e dati telemetrici forniti durante una manovra di parcheggio.

Una seconda criticità dell'algoritmo risiede nel fatto che durante l'esecuzione esso ignora tutti i segnali di controllo provenienti dal blocco decisionale, che potrebbe, ad esempio, rilevare una condizione d'arresto del veicolo. Questo comportamento non permette difatti di implementare variazioni del comportamento del veicolo durante le manovre e rappresenta un rischio per la sicurezza durante la guida.

Parte V.

Conclusione

Capitolo 10.

Conclusioni

Per concludere questo lavoro di tesi, si può affermare che il lavoro svolto ha raggiunto con successo gli obiettivi prefissati. La partecipazione alla Bosch Future Mobility Challenge ha rappresentato un'opportunità significativa per lo sviluppo di competenze avanzate nell'ambito della guida autonoma su veicoli in scala, sia dal punto di vista software che hardware. Attraverso l'implementazione di algoritmi di controllo complessi e l'integrazione di sensori avanzati, è stato possibile realizzare un sistema di navigazione autonomo efficiente, in grado di affrontare scenari reali simulati in ambienti competitivi.

Le sfide affrontate durante il progetto hanno permesso di esplorare soluzioni innovative e di perfezionare le capacità di problem solving in situazioni di incertezza e variabilità.

L'integrazione tra i vari componenti hardware, come il sensore LiDAR e l'IMU, e i moduli software sviluppati per il controllo del veicolo hanno dimostrato una buona robustezza ed efficacia, ottenendo risultati sperimentali soddisfacenti.

Infine, questo progetto rappresenta un punto di partenza per futuri sviluppi nel campo della guida autonoma, aprendo la strada a ulteriori miglioramenti in termini di precisione e sicurezza, con potenziali applicazioni su scala reale.

Bibliografia

[1] A. Isidori. *Sistemi di controllo I*. Ed. Scient. SIDEREA, ROMA, 1993.

[2] *S. Macenski, T. Foote, B. Gerkey, C. Lalancette, W. Woodall. Robot operating system 2: Design, architecture, and uses in the wild.* 2022. URL: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

Sitografia

[1] *"Ros2 tutorial for beginners (foxy)"*. 2021. URL: https://www.youtube.com/watch?v=bFDfvKctvV8&list=PLRE44FoOoKf7NzWwxt3W2taZ7BiWyfhCp.

[2] *1:10 Electric Car Model "Deathwatcher EVO" 4WD ARR*. URL: https://asset.conrad.com/media10/add/160267/c1/-/gl/001406735ML02/manual-1406735-reely-tc-04-onroad-chassis-110-rc-model-car-electricroad-version-4wdarr.pdf.

[3] *AMT10 MODULAR INCREMENTAL ENCODER*. URL: https://www.sameskydevices.com/product/resource/amt10.pdf.

[4] *BNO055 Intelligent 9-axis absolute orientation sensor*. URL: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bno055-ds000.pdf.

[5] *Coral Accelerator Module datasheet*. URL: https://coral.ai/static/files/Coral-Accelerator-Module-datasheet.pdf.

[6] *Grafo tracciato di gara*. URL: https://github.com/ECC-BFMC/Documentation/blob/master/source/racetrack/Competition_track_graph.graphml.

[7] *LD06 Datasheet*. URL: https://www.inno-maker.com/wp-content/uploads/2020/11/LDROBOT_LD06_Datasheet.pdf.

[8] *Material safety datasheet*. URL: https://asset.conrad.com/media10/add/160267/c1/-/en/001344152SD01/security-datasheet-1344152-conrad-energy-scale-model-battery-pack-lipo-74-v-5500-mah-no-of-cells-2-20-c-softcase-xt90.pdf.

[9] *MDEK1001 Kit User Manual*. URL: https://www.qorvo.com/products/d/da007995.

[10] *QUICKRUN Fusion SE*. URL: https://www.hobbywing.com/en/uploads/file/20221206/634e3900be06055a474a384cff420fe7.pdf.

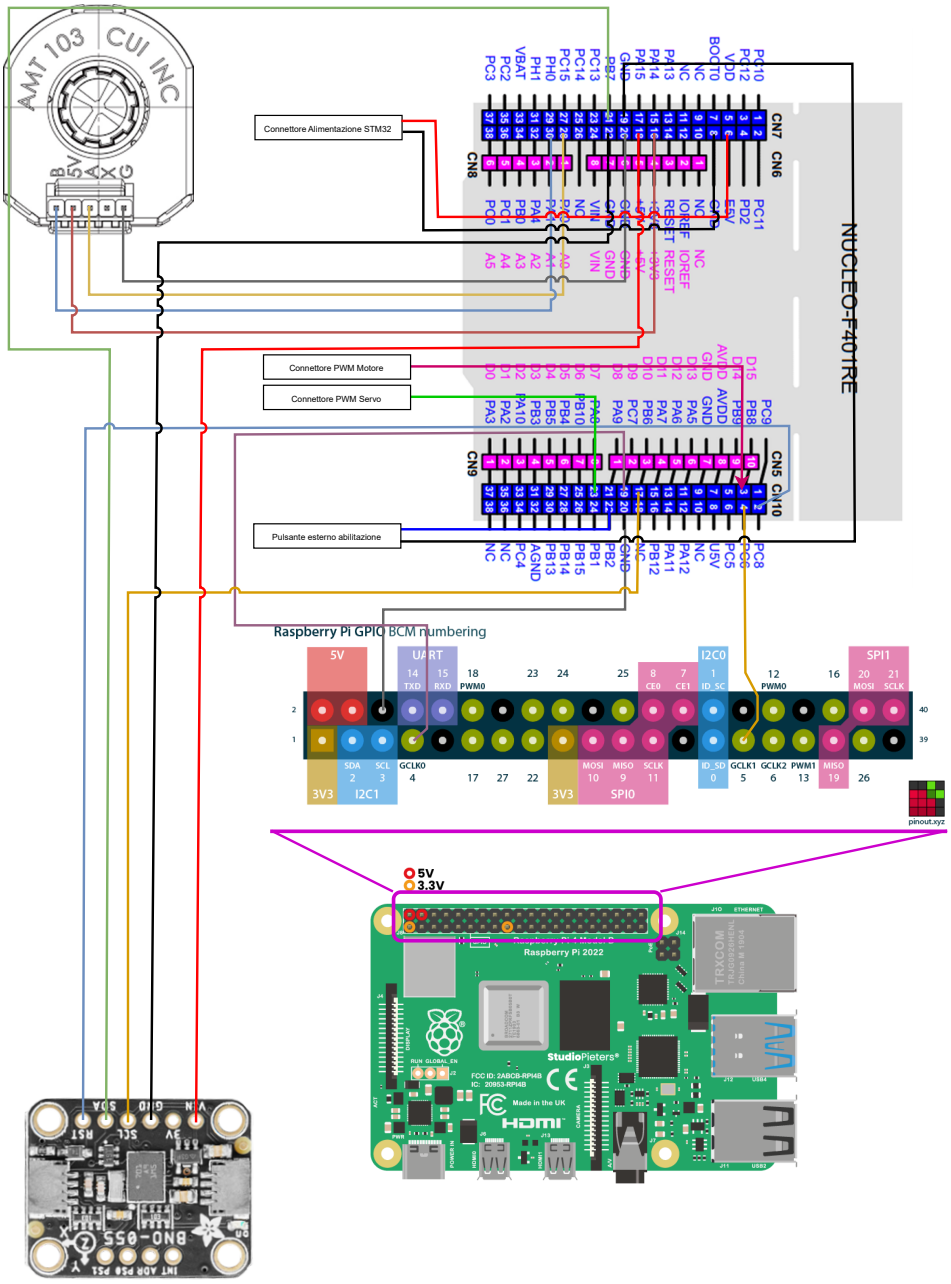
[11] *Raspberry Pi 4 Model B*. URL: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf.

[12] *Raspberry Pi Documentation*. URL: https://www.raspberrypi.com/documentation/accessories/camera.html.

[13] *Servo Motore.* URL: https://asset.conrad.com/media10/add/160267/c1/-/en/001365926DS01/datasheet-1365926-reely-standard-servo-rs-610wp-mg-analogue-servo-gear-box-material-metal-connector-system-jr.pdf.

[14] *UM1724 User manual.* URL: https://www.st.com/resource/en/user_manual/um1724-stm32-nucleo64-boards-mb1136-stmicroelectronics.pdf.

Appendice A: Schema delle connessioni



Appendice B: Cenni teorici sui timer

1. Introduzione

Lo scopo di un timer è quello di fornire una misura accurata del tempo, che può essere utilizzata per i seguenti scopi:

- Temporizzare le azioni del sistema e creazione di routine;
- Misurare l'intervallo di tempo trascorso tra due eventi;
- Generare segnali di controllo per i sistemi di attuazione.

Ovviamente l'utilizzo dei timer non è limitato alle sopracitate applicazioni, che rappresentano però gli utilizzi che sono stati fatti della tecnologia nel contesto applicativo preso in esame.

1.1. Principio di funzionamento dei timer

Dal punto di vista circuitale, un timer é un contatore binario che incrementa il valore di un registro ogni volta che riceve un segnale di ingresso da parte del clock del microcontrollore, traducendo quindi una variazione temporale nella variazione di un dato. Al raggiungimento del valore massimo del registro viene generata una Interupt Request (IRQ) che viene poi gestita dallo scheduler del sistema.
A seguito della definizione appena fornita risulta intuitivo pensare che l'intervallo di tempo misurato da un timer sia interamente dipendente dalla frequenza del core del microcontrollore. Un'implementazione del genere sarebbe però altamente inefficiente e poco flessibile in quanto le applicazioni pratiche non richiedono frequenze dell'ordine di grandezza di quelle del clock del core.
Per questo motivo sono stati implementati meccanismi di modulazione della frequenza, rappresentati da:

- **Preload register**: valore iniziale del registro di conteggio a seguito di un reset;
- **Prescaler**: circuito che permette di ridurre la frequenza di clock percepita dal timer in un suo sottomultiplo, riducendo dunque la velocità di incremento del registro del contatore;
- **Counter period**: valore costante che rappresenta una soglia oltre la quale avviene un reset del conteggio. Il valore del counter period é compreso tra 0 e il valore massimo del registro di count, dato dalla risoluzione del timer.

In Figura 1 è mostrato lo schema di principio di un timer 16-bit:

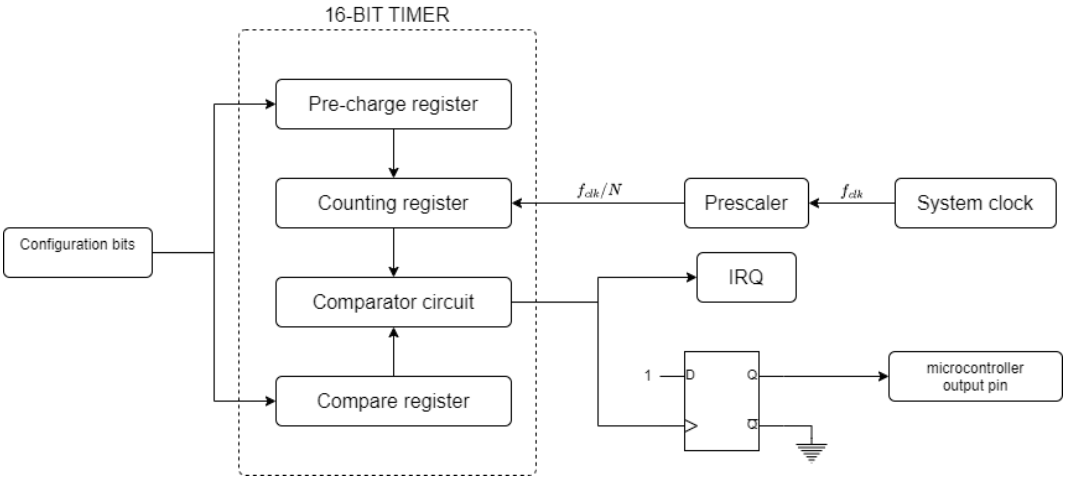


Figura 1.: Schema di principio di un timer a 16 bit.

Mediante l'utilizzo di prescaler e counter period è possibile modulare la frequenza del timer, ovvero la frequenza con la quale vengono generate le IRQ. Un possibile schema di funzionamento è riassunto in Fig. 2:

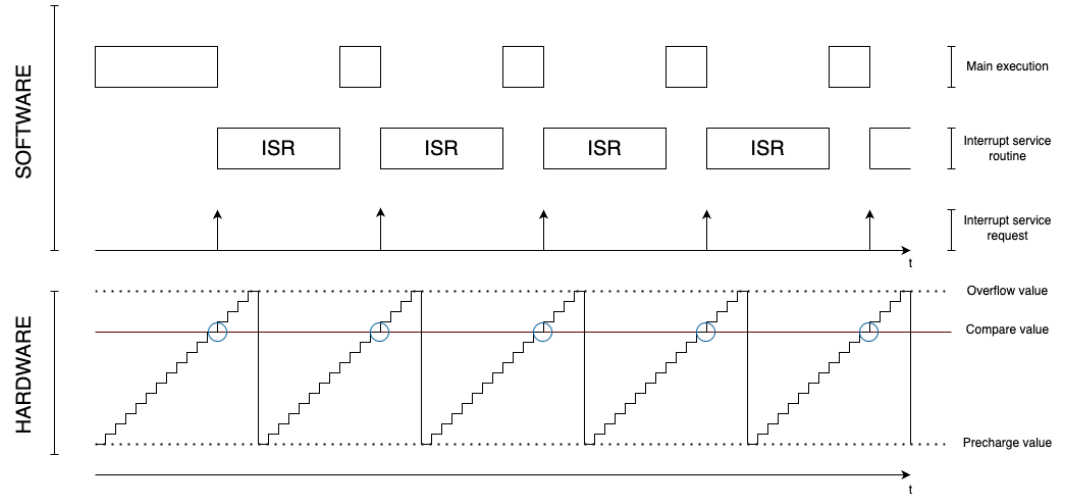


Figura 2.: Generazione IRQ ed esecuzione ISR

Come è possibile osservare in figura, a seguito del raggiungimento da parte del registro del timer del valore di compare, viene generata una IRQ che a livello software viene gestita in interrupt mediante l'esecuzione di un particolare blocco di codice denominato Interrupt Service Routine (ISR). È importante notare come la ISR abbia priorità rispetto alla normale esecuzione del codice, ed è proprio questo livello di priorità a permettere l'esecuzione temporizzata di routine a livello software.

1.2. Modulazione della frequenza

Prescaler e Counter Period assumono un ruolo fondamentale nella modulazione della frequenza con la quale vengono generate le IRQ. La frequenza di aggiornamento del timer (UEV), la frequenza del clock di sistema, il valore di prescaler e quello di counter period sono legati dalla seguente relazione:

$$UEV = \frac{Clock_{sys}}{(Prescaler + 1)(CounterPeriod + 1)} \tag{1}$$

È fondamentale notare che UEV e $Clock_{sys}$ in questo caso sono entrambi espressi in Hz, quindi per ottenere il periodo trascorso tra una generazione di IRQ e la successiva è necessario eseguire la conversione riportata di seguito:

$$UEV(s) = \frac{1}{UEV(Hz)} \tag{2}$$

Come è possibile osservare, clock di sistema, prescaler e counter period sono tutte variabili. Bisogna quindi stabilire un metodo sistematico per il setup del timer che permetta di ottenere le frequenze desiderate. Un buon metodo consiste nell'impostare rispettivamente $Clock_{sys}$, Prescaler e Counter Period. La motivazione che risiede dietro a questa scelta è molto semplice ed è una conseguenza dell'architettura del microcontrollore stesso. Difatti, in ordine di generalità abbiamo innanzitutto il clock di sistema, che detta la temporizzazione delle operazioni del sistema in toto; dal clock di sistema possono derivare altri clock la cui funzione è quella di orchestrare altre componenti del sistema, che potrebbero a loro volta essere soggette ad azioni di prescaler. È importante quindi tener conto delle altre funzionalità del microcontrollore e non abbassare eccessivamente la frequenza del clock di sistema (a meno che non sia esplicitamente richiesto dall'applicazione).

Successivamente, individuato il clock che genera la base dei tempi del timer di interesse, si può agire sul prescaler ad esso associato, in modo da ottenere una buona granularità dei tick che vanno ad incrementare il valore del registro del contatore del timer. Nella selezione del valore di prescaling è buona norma scegliere un valore che sia nella forma 2^n perchè il prescaler non è altro che un contatore binario che preprocessa il segnale del clock di sistema. Ad esempio, nel caso di un prescaler a 8-bit, un buon valore di prescaling potrebbe essere 2^n con $n = \{0, 1, ..., 8\}$.

In extremis si va a scegliere il valore del registro di compare (Counter Period), che deve essere un valore compreso tra 1 e la risoluzione del timer. Ad esempio, per un timer 16-bit, il valore del registro deve essere compreso tra 2^0 e 2^{16}.

Di seguito è mostrato un esempio pratico.

Supponiamo di voler ottenere un timer che genera un IRQ ad una frequenza di 50Hz. Sia $Clock_{sys} = 48MHz$ e si assuma di disporre un prescaler 16-bit e un timer 16-bit. Un buon valore di prescaling potrebbe essere 2^8, ovvero 256. Sostituendo i

valori nell'equazione (1) si ricava un valore di Counter Period di:

$$CounterPeriod + 1 = \frac{48000000}{50 \cdot (255 + 1)} \tag{3}$$

da cui:

$$CounterPeriod = \frac{48000000}{50 \cdot (255 + 1)} - 1 = 3749 \tag{4}$$

Ovviamente la soluzione non è univoca in quanto esistono piú combinazioni di parametri che permettono di ottenere la stessa frequenza di refresh del timer.

2. Modalità operative

I timer sono dispositivi molto versatili in quanto possono essere utilizzati per molteplici scopi. Nella sezione precedente è stato discusso l'utilizzo più comune e più intuitivo del dispositivo (Free running mode), mentre in questa sezione verranno illustrate modalità operative alternative a quella standard che sono state utilizzate anche nel contesto del controllo del veicolo autonomo.

2.1. Compare mode

Questa modalitá operativa permette di generare impulsi di durata variabile. Difatti, facendo riferimento a Figura 1 é possibile osservare come, oltre alla generazione di una IRQ, il circuito di compare è capace di generare impulsi a livello hardware mediante le porte General Purpose Input/Output (GPIO).

Quando il registro di conteggio viene resettato, è possibile impostare un pin su un valore alto, che viene riportato ad un livello basso ogni volta che il registro di conteggio raggiunge il valore di confronto. Uno schema di funzionamento é mostrato in Figura 3.

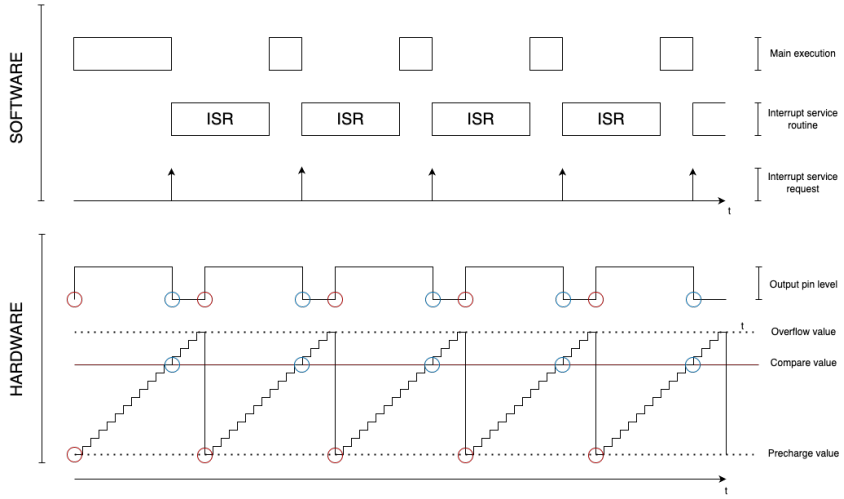


Figura 3.: Timer in modalità compare.

Utilizzando un timer in questa modalità é quindi possibile generare impulsi ad onda quadra di lunghezza variabile, modificando il valore del registro di compare.

2.2. PWM generation mode

Questa modalità utilizza gli stessi principi della modalità compare per generare dei segnali di controllo del tipo Pulse Width Modulation (PWM). Un segnale PWM é un impulso ad onda quadra a periodo costante modulato sul rapporto tra il periodo totale e la durata dell'impulso (Duty Cycle).

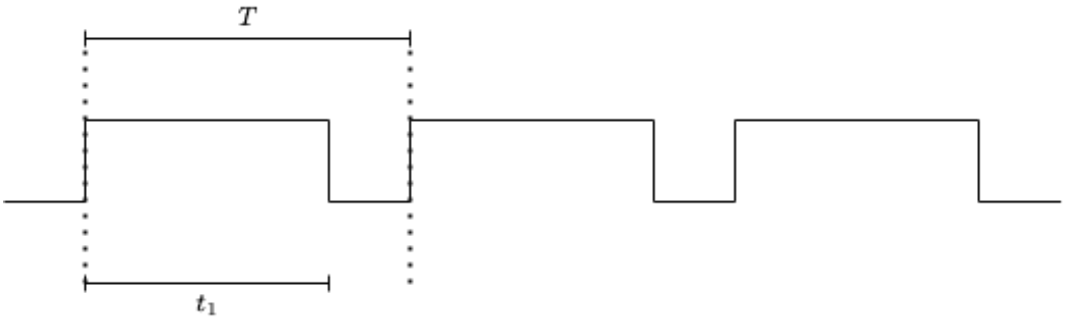


Figura 4.: Periodo e durata dell'impulso di un segnale PWM.

In riferimento alla Figura 4, il Duty Cycle δ è dato dalla relazione:

$$\delta = \frac{t_1}{T}[\%] \tag{5}$$

Un segnale di controllo PWM è generato da due onde:

- **Onda portante**: onda periodica di frequenza costante (solitamente triangolare o a dente di sega) che genera il periodo del segnale in uscita;
- **Onda modulante**: segnale costante o variabile che permette di regolare la larghezza dell'impulso generato.

Nel contesto dei timer, l'onda portante è generata da un timer in modalità free-running con periodo costante, mentre l'onda modulante si ottiene variando il valore del registro di confronto, ovvero modificando il periodo del contatore. In maniera del tutto analoga alla modalità compare, il segnale in uscita viene posto ad un livello alto ogni qualvolta avviene un reset del registro di count del timer e viene riportato ad un livello basso ogni volta che il registro di count raggiunge il valore di compare.

Il duty cycle è espresso come il rapporto tra il valore del registro di compare e quello della dimensione del registro di count, secondo la seguente relazione:

$$\delta = \frac{Counter\ Period}{Count\ Register}[\%] \tag{6}$$

Un possibile schema di funzionamento è mostrato in Figura 5.

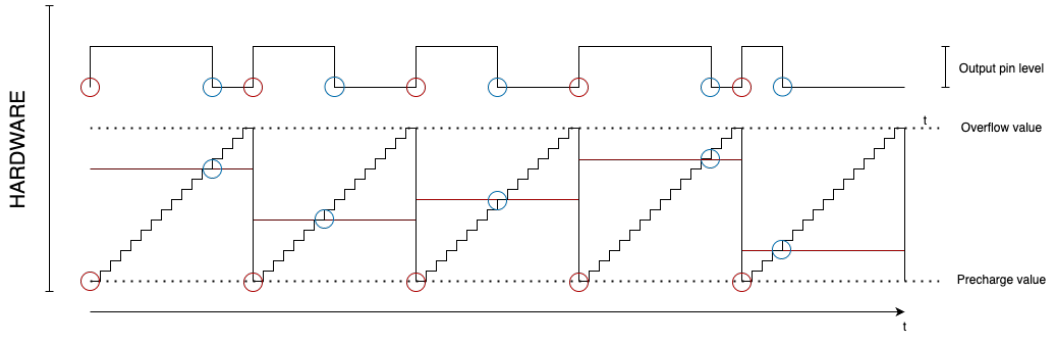


Figura 5.: Generazione di segnali PWM mediante timer.

I principali vantaggi del controllo in PWM sono l'efficienza termica e l'ottima resa in termini di controllo. Difatti, il contenuto energetico medio di un segnale PWM su un periodo è pari ad un segnale costante di ampiezza proporzionale al duty cycle, nonostante periodi di tempo in cui il circuito è a riposo e quindi non disperde energia per effetto ohmico. Questo risultato deriva dall'integrazione del valore fornito in output dal segnale lungo un periodo.

Difatti, detto E_{max} il valore massimo di tensione generabile in output dal microcontrollore, il valor medio della tensione sul carico ($\overline{V_c}$) è ricavabile dalla relazione:

$$\overline{V_c} = \int_0^T E_{max} dt = \frac{1}{T} \int_{t_0}^{t_0+t_1} E_{max} dt \tag{7}$$

da cui:

$$\overline{V_c} = E_{max} \cdot \frac{t_1}{T} = E_{max} \cdot \delta \tag{8}$$

L'uso più comune del controllo in PWM è quello dell'attuazione di motori di vario tipo, che puó essere realizzato in vari modi. Le due modalità di controllo più utilizzate sono:

- Signal-magnitude(S-M);
- Locked Anti-Phase(LAP);

La tecnica di controllo S-M consiste nel separare il segno e l'ampiezza della corrente in due segnali differenti. L'ampiezza viene fornita in valore assoluto in proporzione al duty cycle ed è adibita alla regolazione della velocità, mentre un ulteriore segnale logico (0-5V) è utilizzato per definire la direzione di rotazione. Questa tecnica di controllo é spesso utilizzata per il controllo in velocità dei motori DC con spazzole.

D'altro canto, la tecnica di controllo LAP combina le informazioni di velocità e direzione in un unico segnale PWM, che è divisa in due range: un primo range, per esempio quello compreso tra un duty cycle di 0% e 50% viene riservato per il controllo in un verso, mentre il range compreso tra il 50% e il 100% controlla il motore nell'altro verso di rotazione. Il duty cycle che congiunge i due range costituisce il

valore di riposo del motore. Questa tecnica di controllo è spesso utilizzata per il controllo di motori brushless.

2.3. Input capture mode

Questa modalità permette di misurare l'intervallo temporale trascorso tra due eventi esterni. In particolare, in modalità Input Capture il timer inizia il conteggio in risposta ad un evento esterno, registrato su un pin GPIO. Tipicamente l'evento trigger è un fronte di salita o di discesa di un segnale logico (0-3.3V). Il conteggio del timer si interrompe nel momento in cui viene registrato un ulteriore evento sullo stesso pin. Il contatore del timer procede solitamente in modalitá free-running e i valori registrati vengono salvati in appositi registri. La distanza temporale tra i due eventi può essere dunque ricavata mediante la sottrazione dei valori registrati, tenendo conto della frequenza di aggiornamento del registro di count.

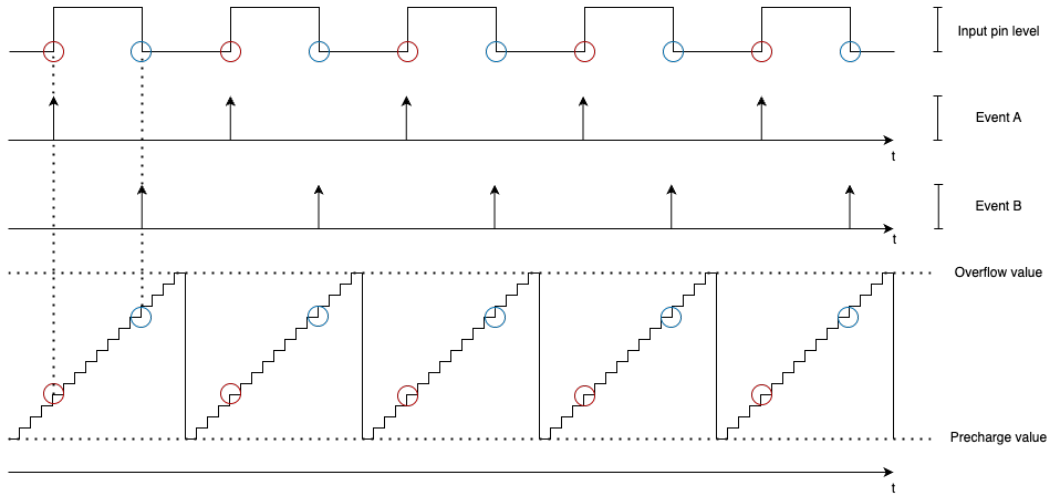


Figura 6.: Timer in modalità input capture.

In Figura 6 è illustrato il funzionamento del timer in modalità input capture: in corrispondenza degli eventi A e B vengono aggiornati i valori dei registri contenenti il numero di tick trascorsi nel momento in cui viene registrato l'evento corrispondente. Il Δt tra i due eventi è facilmente ottenibile utilizzando la relazione riportata di seguito:

$$\Delta t = Timer(B) - Timer(A) \tag{9}$$

dove $Timer(A)$ e $Timer(B)$ sono rispettivamente i tick del timer espressi nel dominio del tempo.

La modalità input capture può essere utilizzata su più canali e una delle applicazioni più rilevanti che fa uso di questa tecnologia è la lettura della velocità di rotazione assiale mediante l'uso di encoder a quadratura.

Appendice C: Cenni teorici sui controllori PID

Nella seguente appendice verranno trattati cenni teorici sulla formulazione dei controllori PID, derivanti dalla teoria del controllo. In particolare, tutte le relazioni utilizzate sono state attinte dal libro di testo [7].

Per un dato controllore $G(s)$, caratterizzato da una funzione di trasferimento della forma:

$$G(s) = \frac{K}{s} \cdot \frac{(1 + \tau_a s)}{\left(1 + \frac{\tau_a}{m_a} s\right)} \tag{1}$$

con $m_a > 1$. Se il valore di m_a risulta essere sufficientemente elevato, si ha che:

$$\omega \frac{\tau_a}{m_a} << 1 \tag{2}$$

Per cui per tutti i valori ω della banda di frequenza di interesse si può approssimare la funzione $G(s)$ con la funzione:

$$G(s) = \frac{K}{s} \cdot (1 + \tau_a s) \tag{3}$$

Un controllore caratterizzato da una funzione di trasferimento di questo tipo viene definito come controllore ad azione proporzionale ed integrale (abbr. controllore PI).

La funzione di trasferimento può essere riscritta nella forma:

$$G(s) = K_P + \frac{K_I}{s} \tag{4}$$

dove:

$$K_P = K\tau_a \qquad K_I = K$$

Evidenziando i termini K_P e K_I, il controllore così costituito, se inserito in uno schema a controreazione della forma riportata in Figura 1, impone al processo una legge di controllo pari alla somma di azioni proporzionali all'errore e al suo integrale rispetto al tempo.

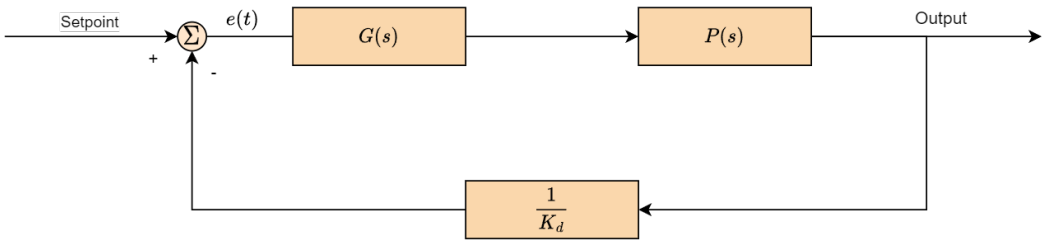


Figura 1.: Schema di controllo a controreazione.

Difatti, l'ingresso al blocco del controllore risulta pari a

$$\frac{e(t)}{K_d} \tag{5}$$

Per cui, se $G(s)$ ha la forma (4), l'uscita da questo blocco nel dominio del tempo risulta essere:

$$m(t) = \frac{K_P}{K_d} \cdot e(t) + \frac{K_I}{K_d} \int_0^t e(\tau) d\tau \tag{6}$$

I parametri K_P e K_I prendono rispettivamente il nome di guadagno dell'azione proporzionale e guadagno dell'azione integrale.

Se si utilizza un controllore della forma (1), il contributo alla fase del sistema in toto risulta essere sempre negativo, a causa dell'aggiunta del polo rappresentato dall'azione integratrice. Per conseguire il margine di fase desiderato, di conseguenza spesso il guadagno K_P risulta avere un valore troppo basso. In questi casi è necessario introdurre nel controllore una rete anticipatrice, considerando dunque un controllore caratterizzato da una funzione di trasferimento della forma:

$$G(s) = \left(K_P + \frac{K_I}{s}\right) \cdot \frac{(1+\tau s)}{\left(1 + \frac{\tau}{m} s\right)} \tag{7}$$

con $m > 1$. Supponendo anche in questo caso $m >> 1$, la funzione (7) può essere approssimata con:

$$G(s) = \left(K_P + \frac{K_I}{s}\right) \cdot (1+\tau s) \tag{8}$$

e riscritta nella forma:

$$G(s) = \hat{K}_P + \frac{K_I}{s} + K_D s \tag{9}$$

dove:

$$\hat{K}_P = K_P + K_I \tau \qquad K_D = K_P \tau$$

Un controllore caratterizzato da una funzione di trasferimento della forma (8) prende il nome di controllore ad azione proporzionale, integrale e derivativa (abbr. controllore PID). Il parametro K_D prende il nome di guadagno dell'azione derivativa.

Appendice D: Header e Source File per l'utilizzo del sensore BNO055

1. Header file

Di seguito è riportato il codice contenuto nel file bno055.h .

```
1
2  #ifndef BNO055_H_
3  #define BNO055_H_
4
5  #ifdef __cplusplus
6    extern "C" {
7  #endif
8  // #define FREERTOS_ENABLED true
9
10 #include <stdbool.h>
11 #include <stdint.h>
12 #include <stdio.h>
13
14 #define START_BYTE 0xAA
15 #define RESPONSE_BYTE 0xBB
16 #define ERROR_BYTE 0xEE
17
18 #define BNO055_I2C_ADDR_HI 0x29
19 #define BNO055_I2C_ADDR_LO 0x28
20 #define BNO055_I2C_ADDR    BNO055_I2C_ADDR_LO
21
22 #define BNO055_READ_TIMEOUT 100
23 #define BNO055_WRITE_TIMEOUT 10
24
25 #define ERROR_WRITE_SUCCESS 0x01  // Everything working as expected
26 #define ERROR_WRITE_FAIL
                \
27   0x03  // Check connection, protocol settings and operation more of
       BNO055
28 #define ERROR_REGMAP_INV_ADDR 0x04   // Invalid register address
29 #define ERROR_REGMAP_WRITE_DIS 0x05  // Register is read-only
30 #define ERROR_WRONG_START_BYTE 0x06  // Check if the first byte
31 #define ERROR_BUS_OVERRUN_ERR
                \
32   0x07  // Resend the command, BNO055 was not able to clear the
       receive buffer
```

```
33 #define ERROR_MAX_LEN_ERR
               \
34   0x08  // Split the command, max fire size can be up to 128 bytes
35 #define ERROR_MIN_LEN_ERR 0x09  // Min length of data is less than 1
36 #define ERROR_RECV_CHAR_TIMEOUT
               \
37   0x0A  // Decrease the waiting time between sending of two bytes of
       one frame
38
39 #define REG_WRITE 0x00
40 #define REG_READ 0x01
41
42 // Page 0
43 #define BNO055_ID (0xA0)
44 #define BNO055_CHIP_ID 0x00        // value: 0xA0
45 #define BNO055_ACC_ID 0x01         // value: 0xFB
46 #define BNO055_MAG_ID 0x02         // value: 0x32
47 #define BNO055_GYRO_ID 0x03        // value: 0x0F
48 #define BNO055_SW_REV_ID_LSB 0x04  // value: 0x08
49 #define BNO055_SW_REV_ID_MSB 0x05  // value: 0x03
50 #define BNO055_BL_REV_ID 0x06      // N/A
51 #define BNO055_PAGE_ID 0x07
52 #define BNO055_ACC_DATA_X_LSB 0x08
53 #define BNO055_ACC_DATA_X_MSB 0x09
54 #define BNO055_ACC_DATA_Y_LSB 0x0A
55 #define BNO055_ACC_DATA_Y_MSB 0x0B
56 #define BNO055_ACC_DATA_Z_LSB 0x0C
57 #define BNO055_ACC_DATA_Z_MSB 0x0D
58 #define BNO055_MAG_DATA_X_LSB 0x0E
59 #define BNO055_MAG_DATA_X_MSB 0x0F
60 #define BNO055_MAG_DATA_Y_LSB 0x10
61 #define BNO055_MAG_DATA_Y_MSB 0x11
62 #define BNO055_MAG_DATA_Z_LSB 0x12
63 #define BNO055_MAG_DATA_Z_MSB 0x13
64 #define BNO055_GYR_DATA_X_LSB 0x14
65 #define BNO055_GYR_DATA_X_MSB 0x15
66 #define BNO055_GYR_DATA_Y_LSB 0x16
67 #define BNO055_GYR_DATA_Y_MSB 0x17
68 #define BNO055_GYR_DATA_Z_LSB 0x18
69 #define BNO055_GYR_DATA_Z_MSB 0x19
70 #define BNO055_EUL_HEADING_LSB 0x1A
71 #define BNO055_EUL_HEADING_MSB 0x1B
72 #define BNO055_EUL_ROLL_LSB 0x1C
73 #define BNO055_EUL_ROLL_MSB 0x1D
74 #define BNO055_EUL_PITCH_LSB 0x1E
75 #define BNO055_EUL_PITCH_MSB 0x1F
76 #define BNO055_QUA_DATA_W_LSB 0x20
77 #define BNO055_QUA_DATA_W_MSB 0x21
78 #define BNO055_QUA_DATA_X_LSB 0x22
79 #define BNO055_QUA_DATA_X_MSB 0x23
80 #define BNO055_QUA_DATA_Y_LSB 0x24
```

```
81 #define BNO055_QUA_DATA_Y_MSB 0x25
82 #define BNO055_QUA_DATA_Z_LSB 0x26
83 #define BNO055_QUA_DATA_Z_MSB 0x27
84 #define BNO055_LIA_DATA_X_LSB 0x28
85 #define BNO055_LIA_DATA_X_MSB 0x29
86 #define BNO055_LIA_DATA_Y_LSB 0x2A
87 #define BNO055_LIA_DATA_Y_MSB 0x2B
88 #define BNO055_LIA_DATA_Z_LSB 0x2C
89 #define BNO055_LIA_DATA_Z_MSB 0x2D
90 #define BNO055_GRV_DATA_X_LSB 0x2E
91 #define BNO055_GRV_DATA_X_MSB 0x2F
92 #define BNO055_GRV_DATA_Y_LSB 0x30
93 #define BNO055_GRV_DATA_Y_MSB 0x31
94 #define BNO055_GRV_DATA_Z_LSB 0x32
95 #define BNO055_GRV_DATA_Z_MSB 0x33
96 #define BNO055_TEMP 0x34
97 #define BNO055_CALIB_STAT 0x35
98 #define BNO055_ST_RESULT 0x36
99 #define BNO055_INT_STATUS 0x37
100 #define BNO055_SYS_CLK_STATUS 0x38
101 #define BNO055_SYS_STATUS 0x39
102 #define BNO055_SYS_ERR 0x3A
103 #define BNO055_UNIT_SEL 0x3B
104 #define BNO055_OPR_MODE 0x3D
105 #define BNO055_PWR_MODE 0x3E
106 #define BNO055_SYS_TRIGGER 0x3F
107 #define BNO055_TEMP_SOURCE 0x40
108 #define BNO055_AXIS_MAP_CONFIG 0x41
109 #define BNO055_AXIS_MAP_SIGN 0x42
110 #define BNO055_ACC_OFFSET_X_LSB 0x55
111 #define BNO055_ACC_OFFSET_X_MSB 0x56
112 #define BNO055_ACC_OFFSET_Y_LSB 0x57
113 #define BNO055_ACC_OFFSET_Y_MSB 0x58
114 #define BNO055_ACC_OFFSET_Z_LSB 0x59
115 #define BNO055_ACC_OFFSET_Z_MSB 0x5A
116 #define BNO055_MAG_OFFSET_X_LSB 0x5B
117 #define BNO055_MAG_OFFSET_X_MSB 0x5C
118 #define BNO055_MAG_OFFSET_Y_LSB 0x5D
119 #define BNO055_MAG_OFFSET_Y_MSB 0x5E
120 #define BNO055_MAG_OFFSET_Z_LSB 0x5F
121 #define BNO055_MAG_OFFSET_Z_MSB 0x60
122 #define BNO055_GYR_OFFSET_X_LSB 0x61
123 #define BNO055_GYR_OFFSET_X_MSB 0x62
124 #define BNO055_GYR_OFFSET_Y_LSB 0x63
125 #define BNO055_GYR_OFFSET_Y_MSB 0x64
126 #define BNO055_GYR_OFFSET_Z_LSB 0x65
127 #define BNO055_GYR_OFFSET_Z_MSB 0x66
128 #define BNO055_ACC_RADIUS_LSB 0x67
129 #define BNO055_ACC_RADIUS_MSB 0x68
130 #define BNO055_MAG_RADIUS_LSB 0x69
131 #define BNO055_MAG_RADIUS_MSB 0x6A
```

```
132 //
133 // BNO055 Page 1
134 #define BNO055_PAGE_ID 0x07
135 #define BNO055_ACC_CONFIG 0x08
136 #define BNO055_MAG_CONFIG 0x09
137 #define BNO055_GYRO_CONFIG_0 0x0A
138 #define BNO055_GYRO_CONFIG_1 0x0B
139 #define BNO055_ACC_SLEEP_CONFIG 0x0C
140 #define BNO055_GYR_SLEEP_CONFIG 0x0D
141 #define BNO055_INT_MSK 0x0F
142 #define BNO055_INT_EN 0x10
143 #define BNO055_ACC_AM_THRES 0x11
144 #define BNO055_ACC_INT_SETTINGS 0x12
145 #define BNO055_ACC_HG_DURATION 0x13
146 #define BNO055_ACC_HG_THRESH 0x14
147 #define BNO055_ACC_NM_THRESH 0x15
148 #define BNO055_ACC_NM_SET 0x16
149 #define BNO055_GYR_INT_SETTINGS 0x17
150 #define BNO055_GYR_HR_X_SET 0x18
151 #define BNO055_GYR_DUR_X 0x19
152 #define BNO055_GYR_HR_Y_SET 0x1A
153 #define BNO055_GYR_DUR_Y 0x1B
154 #define BNO055_GYR_HR_Z_SET 0x1C
155 #define BNO055_GYR_DUR_Z 0x1D
156 #define BNO055_GYR_AM_THRESH 0x1E
157 #define BNO055_GYR_AM_SET 0x1F
158
159 enum bno055_system_status_t {
160   BNO055_SYSTEM_STATUS_IDLE = 0x00,
161   BNO055_SYSTEM_STATUS_SYSTEM_ERROR = 0x01,
162   BNO055_SYSTEM_STATUS_INITIALIZING_PERIPHERALS = 0x02,
163   BNO055_SYSTEM_STATUS_SYSTEM_INITIALIZATION = 0x03,
164   BNO055_SYSTEM_STATUS_EXECUTING_SELF_TEST = 0x04,
165   BNO055_SYSTEM_STATUS_FUSION_ALGO_RUNNING = 0x05,
166   BNO055_SYSTEM_STATUS_FUSION_ALOG_NOT_RUNNING = 0x06
167 };
168
169 typedef enum {  // BNO-55 operation modes
170   BNO055_OPERATION_MODE_CONFIG = 0x00,
171   // Sensor Mode
172   BNO055_OPERATION_MODE_ACCONLY,
173   BNO055_OPERATION_MODE_MAGONLY,
174   BNO055_OPERATION_MODE_GYRONLY,
175   BNO055_OPERATION_MODE_ACCMAG,
176   BNO055_OPERATION_MODE_ACCGYRO,
177   BNO055_OPERATION_MODE_MAGGYRO,
178   BNO055_OPERATION_MODE_AMG,  // 0x07
179                               // Fusion Mode
180   BNO055_OPERATION_MODE_IMU,
181   BNO055_OPERATION_MODE_COMPASS,
182   BNO055_OPERATION_MODE_M4G,
```

```
183   BNO055_OPERATION_MODE_NDOF_FMC_OFF,
184   BNO055_OPERATION_MODE_NDOF   // 0x0C
185 } bno055_opmode_t;
186
187 typedef struct {
188   uint8_t mcuState;
189   uint8_t gyrState;
190   uint8_t magState;
191   uint8_t accState;
192 } bno055_self_test_result_t;
193
194 typedef struct {
195   uint8_t sys;
196   uint8_t gyro;
197   uint8_t mag;
198   uint8_t accel;
199 } bno055_calibration_state_t;
200
201 typedef struct {
202   int16_t x;
203   int16_t y;
204   int16_t z;
205 } bno055_vector_xyz_int16_t;
206
207 typedef struct {
208   bno055_vector_xyz_int16_t gyro;
209   bno055_vector_xyz_int16_t mag;
210   bno055_vector_xyz_int16_t accel;
211 } bno055_calibration_offset_t;
212
213 typedef struct {
214   uint16_t mag;
215   uint16_t accel;
216 } bno055_calibration_radius_t;
217
218 typedef struct {
219   bno055_calibration_offset_t offset;
220   bno055_calibration_radius_t radius;
221 } bno055_calibration_data_t;
222
223 typedef struct {
224   double w;
225   double x;
226   double y;
227   double z;
228 } bno055_vector_t;
229
230 typedef struct {
231   uint8_t x;
232   uint8_t x_sign;
233   uint8_t y;
```

```
234   uint8_t y_sign;
235   uint8_t z;
236   uint8_t z_sign;
237 } bno055_axis_map_t;
238
239 typedef enum {
240   BNO055_VECTOR_ACCELEROMETER = 0x08,  // Default: m/s
241   BNO055_VECTOR_MAGNETOMETER = 0x0E,   // Default: uT
242   BNO055_VECTOR_GYROSCOPE = 0x14,      // Default: rad/s
243   BNO055_VECTOR_EULER = 0x1A,          // Default: degrees
244   BNO055_VECTOR_QUATERNION = 0x20,     // No units
245   BNO055_VECTOR_LINEARACCEL = 0x28,    // Default: m/s
246   BNO055_VECTOR_GRAVITY = 0x2E         // Default: m/s
247 } bno055_vector_type_t;
248
249 enum bno055_system_error_t {
250   BNO055_SYSTEM_ERROR_NO_ERROR = 0x00,
251   BNO055_SYSTEM_ERROR_PERIPHERAL_INITIALIZATION_ERROR = 0x01,
252   BNO055_SYSTEM_ERROR_SYSTEM_INITIALIZATION_ERROR = 0x02,
253   BNO055_SYSTEM_ERROR_SELF_TEST_FAILED = 0x03,
254   BNO055_SYSTEM_ERROR_REG_MAP_VAL_OUT_OF_RANGE = 0x04,
255   BNO055_SYSTEM_ERROR_REG_MAP_ADDR_OUT_OF_RANGE = 0x05,
256   BNO055_SYSTEM_ERROR_REG_MAP_WRITE_ERROR = 0x06,
257   BNO055_SYSTEM_ERROR_LOW_PWR_MODE_NOT_AVAILABLE_FOR_SELECTED_OPR_MODE
        = 0x07,
258   BNO055_SYSTEM_ERROR_ACCEL_PWR_MODE_NOT_AVAILABLE = 0x08,
259   BNO055_SYSTEM_ERROR_FUSION_ALGO_CONF_ERROR = 0x09,
260   BNO055_SYSTEM_ERROR_SENSOR_CONF_ERROR = 0x0A
261 };
262
263 enum bno055_axis_map_representation_t {
264   BNO055_AXIS_X = 0x00,
265   BNO055_AXIS_Y = 0x01,
266   BNO055_AXIS_Z = 0x02
267 };
268
269 enum bno055_axis_map_sign_t {
270   BNO055_AXIS_SIGN_POSITIVE = 0x00,
271   BNO055_AXIS_SIGN_NEGATIVE = 0x01
272 };
273
274 void bno055_writeData(uint8_t reg, uint8_t data);
275 void bno055_readData(uint8_t reg, uint8_t *data, uint8_t len);
276 void bno055_delay(int time);
277
278 void bno055_reset();
279 bno055_opmode_t bno055_getOperationMode();
280 void bno055_setOperationMode(bno055_opmode_t mode);
281 void bno055_setOperationModeConfig();
282 void bno055_setOperationModeNDOF();
283 void bno055_enableExternalCrystal();
```

```
284 void bno055_disableExternalCrystal();
285 void bno055_setup();
286
287 int8_t bno055_getTemp();
288
289 uint8_t bno055_getBootloaderRevision();
290 uint8_t bno055_getSystemStatus();
291 uint8_t bno055_getSystemError();
292 int16_t bno055_getSWRevision();
293
294 bno055_self_test_result_t bno055_getSelfTestResult();
295 bno055_calibration_state_t bno055_getCalibrationState();
296 bno055_calibration_data_t bno055_getCalibrationData();
297 void bno055_setCalibrationData(bno055_calibration_data_t calData);
298 bno055_vector_t bno055_getVectorAccelerometer();
299 bno055_vector_t bno055_getVectorMagnetometer();
300 bno055_vector_t bno055_getVectorGyroscope();
301 bno055_vector_t bno055_getVectorEuler();
302 bno055_vector_t bno055_getVectorLinearAccel();
303 bno055_vector_t bno055_getVectorGravity();
304 bno055_vector_t bno055_getVectorQuaternion();
305 void bno055_setAxisMap(bno055_axis_map_t axis);
306
307 #ifdef __cplusplus
308   }
309 #endif
310 #endif  // BNO055_H_
```

Listing 10.1: Contenuto del file bno055.h .

2. Source file bno055.c

Di seguito è riportato il contenuto del source file bno055.c .

```
1
2 #include "bno055.h"
3 #include <string.h>
4
5 uint16_t accelScale = 100;
6 uint16_t tempScale = 1;
7 uint16_t angularRateScale = 16;
8 uint16_t eulerScale = 16;
9 uint16_t magScale = 16;
10 uint16_t quaScale = (1<<14);    // 2^14
11
12 void bno055_setPage(uint8_t page) { bno055_writeData(BNO055_PAGE_ID,
     page); }
13
14 bno055_opmode_t bno055_getOperationMode() {
15   bno055_opmode_t mode;
16   bno055_readData(BNO055_OPR_MODE, &mode, 1);
```

```
17   return mode;
18 }
19
20 void bno055_setOperationMode(bno055_opmode_t mode) {
21   bno055_writeData(BNO055_OPR_MODE, mode);
22   if (mode == BNO055_OPERATION_MODE_CONFIG) {
23     bno055_delay(19);
24   } else {
25     bno055_delay(7);
26   }
27 }
28
29 void bno055_setOperationModeConfig() {
30   bno055_setOperationMode(BNO055_OPERATION_MODE_CONFIG);
31 }
32
33 void bno055_setOperationModeNDOF() {
34   bno055_setOperationMode(BNO055_OPERATION_MODE_NDOF);
35 }
36
37 void bno055_setExternalCrystalUse(bool state) {
38   bno055_setPage(0);
39   uint8_t tmp = 0;
40   bno055_readData(BNO055_SYS_TRIGGER, &tmp, 1);
41   tmp |= (state == true) ? 0x80 : 0x0;
42   bno055_writeData(BNO055_SYS_TRIGGER, tmp);
43   bno055_delay(700);
44 }
45
46 void bno055_enableExternalCrystal() { bno055_setExternalCrystalUse(
     true); }
47 void bno055_disableExternalCrystal() { bno055_setExternalCrystalUse(
     false); }
48
49 void bno055_reset() {
50   bno055_writeData(BNO055_SYS_TRIGGER, 0x20);
51   bno055_delay(700);
52 }
53
54 int8_t bno055_getTemp() {
55   bno055_setPage(0);
56   uint8_t t;
57   bno055_readData(BNO055_TEMP, &t, 1);
58   return t;
59 }
60
61 void bno055_setup() {
62   bno055_reset();
63
64   uint8_t id = 0;
65   bno055_readData(BNO055_CHIP_ID, &id, 1);
```

```
66    if (id != BNO055_ID) {
67      printf("Can't find BNO055, id: 0x%02x. Please check your wiring.\r
       \n", id);
68    }
69    bno055_setPage(0);
70    bno055_writeData(BNO055_SYS_TRIGGER, 0x0);
71
72    // Select BNO055 config mode
73    bno055_setOperationModeConfig();
74    bno055_delay(10);
75  }
76
77  int16_t bno055_getSWRevision() {
78    bno055_setPage(0);
79    uint8_t buffer[2];
80    bno055_readData(BNO055_SW_REV_ID_LSB, buffer, 2);
81    return (int16_t)((buffer[1] << 8) | buffer[0]);
82  }
83
84  uint8_t bno055_getBootloaderRevision() {
85    bno055_setPage(0);
86    uint8_t tmp;
87    bno055_readData(BNO055_BL_REV_ID, &tmp, 1);
88    return tmp;
89  }
90
91  uint8_t bno055_getSystemStatus() {
92    bno055_setPage(0);
93    uint8_t tmp;
94    bno055_readData(BNO055_SYS_STATUS, &tmp, 1);
95    return tmp;
96  }
97
98  bno055_self_test_result_t bno055_getSelfTestResult() {
99    bno055_setPage(0);
100   uint8_t tmp;
101   bno055_self_test_result_t res = {
102       .mcuState = 0, .gyrState = 0, .magState = 0, .accState = 0};
103   bno055_readData(BNO055_ST_RESULT, &tmp, 1);
104   res.mcuState = (tmp >> 3) & 0x01;
105   res.gyrState = (tmp >> 2) & 0x01;
106   res.magState = (tmp >> 1) & 0x01;
107   res.accState = (tmp >> 0) & 0x01;
108   return res;
109 }
110
111 uint8_t bno055_getSystemError() {
112   bno055_setPage(0);
113   uint8_t tmp;
114   bno055_readData(BNO055_SYS_ERR, &tmp, 1);
115   return tmp;
```

```
116 }
117
118 bno055_calibration_state_t bno055_getCalibrationState() {
119   bno055_setPage(0);
120   bno055_calibration_state_t cal = {.sys = 0, .gyro = 0, .mag = 0, .
      accel = 0};
121   uint8_t calState = 0;
122   bno055_readData(BNO055_CALIB_STAT, &calState, 1);
123   cal.sys = (calState >> 6) & 0x03;
124   cal.gyro = (calState >> 4) & 0x03;
125   cal.accel = (calState >> 2) & 0x03;
126   cal.mag = calState & 0x03;
127   return cal;
128 }
129
130
131 bno055_calibration_data_t bno055_getCalibrationData() {
132   bno055_calibration_data_t calData;
133   uint8_t buffer[22];
134   bno055_opmode_t operationMode = bno055_getOperationMode();
135   bno055_setOperationModeConfig();
136   bno055_setPage(0);
137
138   bno055_readData(BNO055_ACC_OFFSET_X_LSB, buffer, 22);
139
140   // Assumes little endian processor
141   memcpy(&calData.offset.accel, buffer, 6);
142   memcpy(&calData.offset.mag, buffer + 6, 6);
143   memcpy(&calData.offset.gyro, buffer + 12, 6);
144   memcpy(&calData.radius.accel, buffer + 18, 2);
145   memcpy(&calData.radius.mag, buffer + 20, 2);
146
147   bno055_setOperationMode(operationMode);
148
149   return calData;
150 }
151
152 void bno055_setCalibrationData(bno055_calibration_data_t calData) {
153   uint8_t buffer[22];
154   bno055_opmode_t operationMode = bno055_getOperationMode();
155   bno055_setOperationModeConfig();
156   bno055_setPage(0);
157
158   // Assumes litle endian processor
159   memcpy(buffer, &calData.offset.accel, 6);
160   memcpy(buffer + 6, &calData.offset.mag, 6);
161   memcpy(buffer + 12, &calData.offset.gyro, 6);
162   memcpy(buffer + 18, &calData.radius.accel, 2);
163   memcpy(buffer + 20, &calData.radius.mag, 2);
164
165   for (uint8_t i=0; i < 22; i++) {
```

```
166     // TODO(oliv4945): create multibytes write
167     bno055_writeData(BNO055_ACC_OFFSET_X_LSB+i, buffer[i]);
168   }
169
170   bno055_setOperationMode(operationMode);
171 }
172
173 bno055_vector_t bno055_getVector(uint8_t vec) {
174   bno055_setPage(0);
175   uint8_t buffer[8];    // Quaternion need 8 bytes
176
177   if (vec == BNO055_VECTOR_QUATERNION)
178     bno055_readData(vec, buffer, 8);
179   else
180     bno055_readData(vec, buffer, 6);
181
182   double scale = 1;
183
184   if (vec == BNO055_VECTOR_MAGNETOMETER) {
185     scale = magScale;
186   } else if (vec == BNO055_VECTOR_ACCELEROMETER ||
187              vec == BNO055_VECTOR_LINEARACCEL || vec ==
       BNO055_VECTOR_GRAVITY) {
188     scale = accelScale;
189   } else if (vec == BNO055_VECTOR_GYROSCOPE) {
190     scale = angularRateScale;
191   } else if (vec == BNO055_VECTOR_EULER) {
192     scale = eulerScale;
193   } else if (vec == BNO055_VECTOR_QUATERNION) {
194     scale = quaScale;
195   }
196
197   bno055_vector_t xyz = {.w = 0, .x = 0, .y = 0, .z = 0};
198   if (vec == BNO055_VECTOR_QUATERNION) {
199     xyz.w = (int16_t)((buffer[1] << 8) | buffer[0]) / scale;
200     xyz.x = (int16_t)((buffer[3] << 8) | buffer[2]) / scale;
201     xyz.y = (int16_t)((buffer[5] << 8) | buffer[4]) / scale;
202     xyz.z = (int16_t)((buffer[7] << 8) | buffer[6]) / scale;
203   } else {
204     xyz.x = (int16_t)((buffer[1] << 8) | buffer[0]) / scale;
205     xyz.y = (int16_t)((buffer[3] << 8) | buffer[2]) / scale;
206     xyz.z = (int16_t)((buffer[5] << 8) | buffer[4]) / scale;
207   }
208
209   return xyz;
210 }
211
212 bno055_vector_t bno055_getVectorAccelerometer() {
213   return bno055_getVector(BNO055_VECTOR_ACCELEROMETER);
214 }
215 bno055_vector_t bno055_getVectorMagnetometer() {
```

```
216   return bno055_getVector(BNO055_VECTOR_MAGNETOMETER);
217 }
218 bno055_vector_t bno055_getVectorGyroscope() {
219   return bno055_getVector(BNO055_VECTOR_GYROSCOPE);
220 }
221 bno055_vector_t bno055_getVectorEuler() {
222   return bno055_getVector(BNO055_VECTOR_EULER);
223 }
224 bno055_vector_t bno055_getVectorLinearAccel() {
225   return bno055_getVector(BNO055_VECTOR_LINEARACCEL);
226 }
227 bno055_vector_t bno055_getVectorGravity() {
228   return bno055_getVector(BNO055_VECTOR_GRAVITY);
229 }
230 bno055_vector_t bno055_getVectorQuaternion() {
231   return bno055_getVector(BNO055_VECTOR_QUATERNION);
232 }
233
234 void bno055_setAxisMap(bno055_axis_map_t axis) {
235   uint8_t axisRemap = (axis.z << 4) | (axis.y << 2) | (axis.x);
236   uint8_t axisMapSign = (axis.x_sign << 2) | (axis.y_sign << 1) | (
      axis.z_sign);
237   bno055_writeData(BNO055_AXIS_MAP_CONFIG, axisRemap);
238   bno055_writeData(BNO055_AXIS_MAP_SIGN, axisMapSign);
239 }
```

Listing 10.2: Contenuto del source file bno055.c .