UNIVERSITÀ
POLITECNICA
DELLE MARCHE

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

Sviluppo di algoritmi su microcontrollore per il controllo di direzione e velocità per la guida autonoma di un veicolo in scala 1:10

Development of algorithms on microcontroller for direction and speed control for autonomous driving of a 1:10 scale vehicle

Relatore:
Andrea Bonci

Candidato:
Angjelo Libofsha

Anno Accademico 2023-2024

UNIVERSITÀ
POLITECNICA
DELLE MARCHE

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

Sviluppo di algoritmi su microcontrollore per il controllo di direzione e velocità per la guida autonoma di un veicolo in scala 1:10

Development of algorithms on microcontroller for direction and speed control for autonomous driving of a 1:10 scale vehicle

Relatore:
Andrea Bonci

Candidato:
Angjelo Libofsha

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

Sommario

La tesi che segue si pone come obiettivo quello di riportare tutti i passaggi necessari al fine di realizzare gli algoritmi su microcontrollore per il controllo in direzione e velocità per la guida autonoma di un veicolo in scala 1:10.

Il veicolo in questione è stato fornito dal Centro Ingegneristico della Bosch di Cluj-Napoca, in Romania, come strumentazione per partecipare alla competizione *Bosch Future Mobility Challenge 2024*. Tale gara prevede la progettazione e la realizzazione di diversi algoritmi per la gestione di una macchina a guida autonoma, che sia in grado, quindi, di simulare la guida in strada reagendo all'ambiente circostante tutelando la propria sicurezza e quella degli altri veicoli.

Per riuscire a completare le diverse task assegnate, la macchina è stata dotata di diversi sensori, in modo da avere un feedback sia sull'ambiente circostante che sul suo stato in modo da eseguire opportune regolazioni. I sensori più importanti sono telecamera, encoder, IMU e LIDAR.

Tuttavia l'obiettivo della tesi non è quello di spiegare in modo estensivo il funzionamento della macchina o di come le parti interagiscano tra di loro, bensì quello di focalizzare l'attenzione sulle tecniche utilizzate per il controllo del motore di trazione e del servo motore che regolano il moto.

Il focus non sarà solo quello del controllo in catena chiusa, ma verrà anche trattata l'implementazione nell'ambiente di sviluppo CubeIDE, in modo da poter replicare quanto trattato.

Indice

Elenco delle figure

Elenco delle tabelle

Listings

Capitolo 1.

Introduzione

1.1. Bosch Future Mobility Challenge (BFMC)

Bosch, fondata nel 1886 a Stoccarda, Germania, è una delle multinazionali più influenti e rispettate nel campo delle tecnologie e dei servizi. Con una presenza in oltre 60 paesi e un'ampia rete globale di produzione, ricerca e sviluppo, Bosch impiega circa 400.000 persone in tutto il mondo. L'azienda opera in diversi settori chiave, tra cui la mobilità, la tecnologia industriale, i beni di consumo e la tecnologia energetica e delle costruzioni. In ciascuno di questi settori, Bosch si distingue per la sua capacità di innovazione, qualità dei prodotti e impegno per la sostenibilità.

Nel campo della mobilità, Bosch è all'avanguardia nello sviluppo di tecnologie per veicoli a guida autonoma. La sua competenza si estende alla progettazione e produzione di una vasta gamma di sensori avanzati, essenziali per la percezione dell'ambiente circostante da parte dei veicoli autonomi. Tra questi sensori vi sono lidar, telecamere, encoder e IMU (Unità di Misura Inerziale), che consentono una rilevazione precisa e affidabile delle condizioni di guida. Questi dispositivi sono fondamentali per garantire che i veicoli possano operare in modo sicuro ed efficiente, identificando ostacoli, mantenendo la corsia e adattandosi dinamicamente alle condizioni del traffico.

Grazie alla sinergia tra hardware di alta qualità e software all'avanguardia, Bosch continua a ridefinire i limiti della tecnologia per la guida autonoma, rafforzando la sua posizione di leader del settore e partner di fiducia per le case automobilistiche di tutto il mondo.

Per quanto riguarda la challenge, questa è organizzata annualmente e quest'anno è giunta a termine la 5a edizione. Ogni edizione aggiunge nuove sfide e task da completare, in modo da avere sempre un livello di difficoltà crescente e far si che la competizione non risulti monotona. Ai partecipanti viene fornita una macchina in scala 1:10 con l'hardware necessario per completare le task. Tuttavia, ogni gruppo può effettuare delle modifiche alle componenti: queste sono permesse così da permettere ai partecipanti di affrontare le sfide in modo creativo e utilizzare intuizioni brillanti per trovare soluzioni ottimali ai problemi.

Le azioni principali che deve eseguire il veicolo sono:

- Lanekeeping;
- Affrontare incroci e rotatorie;
- Rilevare e rispettare eventuali cartelli stradali (es. stop, dare precedenza..);
- Evitare collisioni con altri veicoli o pedoni;
- Interpretare correttamente i semafori;
- Regolare la velocità in modo consono;
- Realizzare un parcheggio in parallelo.

Come si può notare, le task da completare sono diverse, e la difficoltà maggiore è riuscire a tenerle tutte in considerazione in ogni istante.

La novità di quest'anno è un sistema di localizzazione locale, composto da diverse ancore che consentono alla macchina di determinare la propria posizione in tempo reale sul tracciato.

Per poter fare ciò, il tracciato è stato accuratamente mappato con numerosi nodi collegati da archi, che restituiscono anche informazioni sul senso di marcia. Tuttavia, questa nuova variabile introduce un'ulteriore sfida: rendere la posizione il più precisa possibile utilizzando metodi che non verranno trattati in questa tesi.

1.2. Obiettivo della tesi

L'obiettivo di questa tesi è, quindi, quello di fornire inizialmente una panoramica dell'hardware utilizzato sulla macchina con una descrizione delle componenti che permetta di comprenderne il funzionamento. Ciò consentirà di comprendere le scelte effettuate durante la fase di sperimentazione sul motore.

Al termine della panoramica sull'hardware, si passa al primo argomento principale: il controllo del motore durante il movimento su una superficie piana.

La trattazione prosegue con l'analisi del comportamento del motore durante il movimento su una superficie inclinata (quindi prima in salita e poi in discesa).

Infine, la trattazione si conclude con la descrizione del controllo del servo motore, il cui scopo è gestire lo spostamento laterale del veicolo. In particolare, questa trattazione si limiterà alla parte di attuazione relativa allo spostamento laterale, poiché l'algoritmo di riconoscimento della corsia e il calcolo del riferimento da seguire non rientrano negli argomenti di questa tesi.

Capitolo 2.

Il veicolo: hardware

In questo capitolo verranno analizzate tutte le componenti presenti nella macchina e, per ognuna, ne verrà fornita una breve descrizione.
E' bene ricordare che non tutte le parti hardware sono state fornite dalla BOSCH all'interno del kit di partenza: infatti, alcune di esse sono state aggiunte in un secondo momento per poter implementare particolari funzioni.

Come si può vedere nella Figura 2.1 il veicolo fornito è un insieme di componenti che svolgono ruoli specifici:

- Nucleo F401RE;
- Raspberry PI 4;
- Coral TPU (aggiuta in seguito);
- QUICRUN Fusion SE;
- Encoder AMT 103 (aggiunto in seguito);
- Servo motore;
- IMU;
- Batteria;
- Camera;
- LIDAR (aggiunto in seguito);

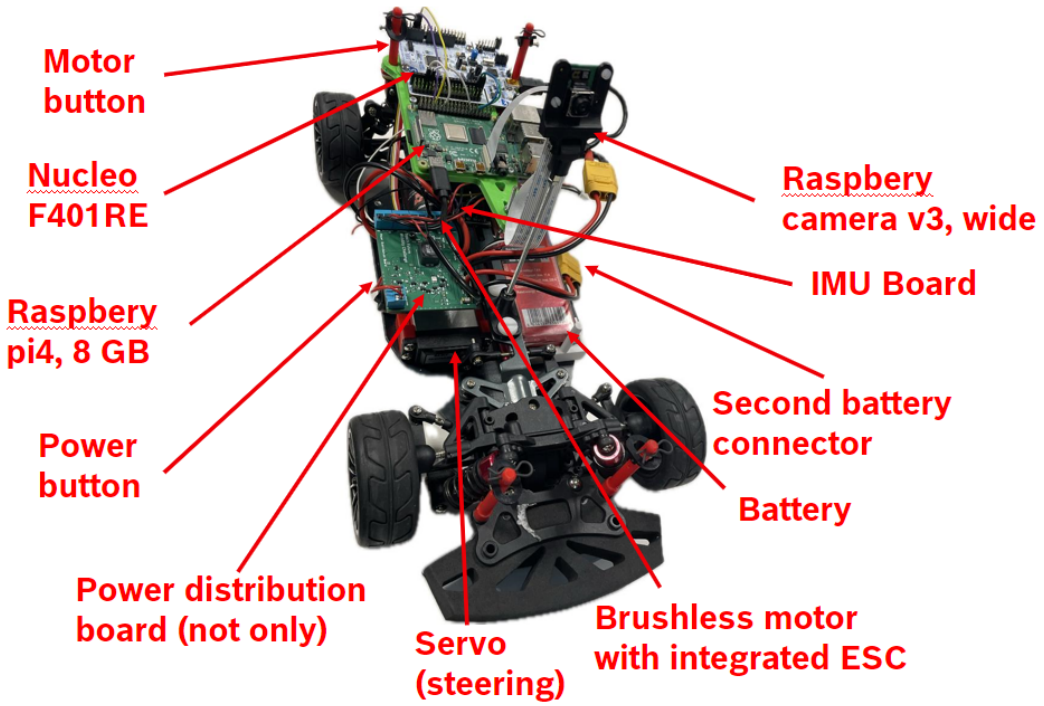


Figura 2.1.: Hardware della macchina

2.1. Nucleo F401RE

La scheda Nucleo F401RE, prodotta da STMicroelectronics, appartiene alla famiglia STM32 serie F4 e si basa sul processore Arm® Cortex®-M4 a 32 bit ed è capace di operare fino a 84 MHz. Questo processore include una Floating Point Unit (FPU) in grado di gestire operazioni in virgola mobile e garantisce consumi energetici estremamente ridotti sia in modalità di esecuzione che di arresto.

La Nucleo F401RE è dotata di 512 KB di memoria flash e 96 KB di RAM: in questo modo garantisce un'ampia capacità per l'esecuzione del codice e l'archiviazione dei dati. Oltre a ciò, offre una grande varietà di interfacce di comunicazione, come GPIO (General Purpose Input/Output), I2C (Inter-Integrated Circuit), SPI (Serial Peripheral Interface) e UART (Universal Asynchronous Receiver/Transmitter), che permettono di collegare facilmente sensori, attuatori e altri dispositivi esterni.

La scheda integra diverse funzionalità, tra cui:

- 3 USARTs che operano a 10.5 Mbit/s per la comunicazione seriale;
- 16 GPIOs per ingressi e uscite generiche;
- 10 TIMERs da 16 e 32 bit, con frequenze fino a 84 MHz;
- 1 porta USB 2.0 OTG.

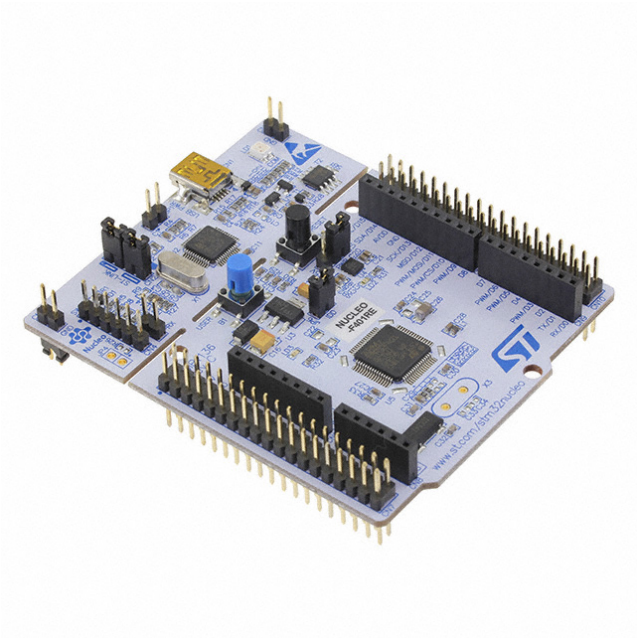


Figura 2.2.: NUCLEO F401RE

Come mostrato nella Figura 2.2, la scheda è equipaggiata con due pulsanti: uno blu programmabile e uno nero per il reset del microcontrollore. Sono presenti anche tre LED:

- LED1 indica la connessione USB;
- LED2 è programmabile e verrà usato per comunicare lo stato della macchina;
- LED3 si accende quando la scheda è alimentata;

La scheda può essere alimentata tramite USB o da un'alimentazione esterna ed è dotata di un debugger e di un programmatore integrato ST-Link/V2-1, che consente di caricare il codice direttamente sulla scheda. È compatibile con l'ambiente di sviluppo STM32CubeIDE, un IDE integrato per la programmazione e il debug del codice.

La piedinatura della scheda, illustrata nella Figura 2.3, comprende 64 pin suddivisi in due set principali:

1. Arduino Uno, con connettori femmina:
   - CN6 e CN8 per ingressi e uscite analogici;
   - CN5 e CN9 per ingressi e uscite digitali.
2. St Morphio, con connettori maschi CN7 e CN10, che estendono ulteriormente le funzionalità della scheda.

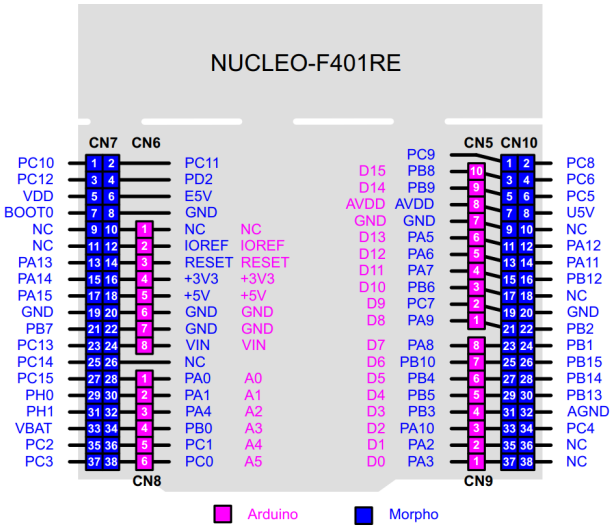


Figura 2.3.: Pinout della nucleo

Nell'ambito del lavoro svolto, la Nucleo F401RE è stata utilizzata come controllore per motori e sensori collegati, gestendo in modo efficace i comandi ricevuti dalla Raspberry Pi 4 tramite comunicazione seriale.

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

2.2. Raspberry PI 4

La Raspberry è un SBC (Single Board Computer) e, quindi, offre tutte le potenzialità e comodità di un computer fuse in un'unica scheda.

La Raspberry Pi 4 è dotata di un potente processore ARM Cortex-A72 quad-core a 64 bit con frequenza fino a 1.5 GHz. Inoltre, dispone di diverse porte USB 3.0 e 2.0 per collegare dispositivi esterni, insieme a porte display, DSI (Display Serial Interface) e CSI (Camera Serial Interface), per connettere display e telecamere, oltre ad utilizzare schede microSD per il sistema operativo e l'archiviazione dati.

La possibilità di cambiare il sistema operativo semplicemente sostituendo la scheda SD al boot consente di risparmiare molto tempo e di assegnare più persone alla risoluzione di vari task.

Inoltre, la Raspberry Pi 4 è dotata di un header GPIO a 40 pin che consente di collegare vari dispositivi elettronici (in questo caso ha permesso la comunicazione seriale con la NUCELO F401RE) e l'alimentazione avviene tramite un connettore USB-C saldato alla scheda di distribuzione.

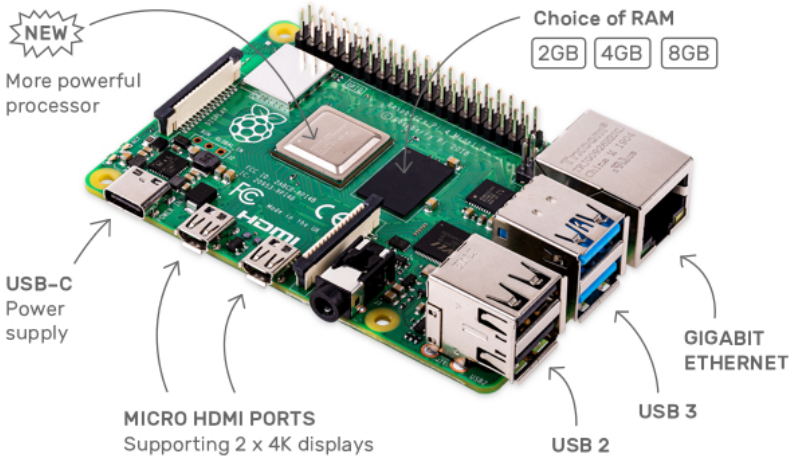


Figura 2.4.: Raspberry PI 4

Grazie all'hardware performante e ad un sistema operativo leggero basato su Kernel Linux, la Raspberry Pi 4 è ideale per l'elaborazione di immagini e video, compiti che richiedono elevate capacità computazionali.

La Raspberry PI 4 costituisce il "cervello" della macchina, acquisendo immagini tramite la camera per poi processarle e comunicare alla NUCELO i dati necessari per la correzione della traiettoria. Un altro compito fondamentale di questa scheda è quello del decision-making: comunica alla macchina l'azione successiva da compiere. Infine, è sempre attiva una rete neurale che riconosce la presenza di cartelli stradali o pedoni per reagire di conseguenza.

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

2.3. Coral TPU

La TPU (Tensor Processing Unit) Coral di Google è un sensore integrato progettato appositamente per accelerare l'esecuzione di modelli di machine learning direttamente sul dispositivo. Questo hardware specializzato è ottimizzato per eseguire operazioni di inferenza, ovvero il processo di esecuzione di un modello ML già addestrato su nuovi dati, in modo estremamente efficiente.

La TPU Coral offre prestazioni elevate a basso consumo energetico: ciò la rende ideale per applicazioni di edge computing dove la latenza e l'efficienza energetica sono critiche. Utilizzata in una varietà di dispositivi, come telecamere intelligenti, dispositivi IoT e applicazioni di automazione industriale, la TPU Coral permette di eseguire modelli complessi di deep learning in tempo reale senza il bisogno di inviare dati a un server remoto, e migliorando così la privacy, la sicurezza e la reattività del sistema.

Figura 2.5.: Google Coral

La Coral è una delle prime modifiche che sono state fatte alla macchina, per cercare di alleggerire il carico della rete neurale sulla Raspberry: in questo modo si migliora la performance dell'algoritmo di visione. Questo componente non è presente nella Figura 2.1 in quanto è stata aggiunta in seguito collegata direttamente alla Raspberry tramite cavo USB-A.

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

2.4. Motore

Il QUICRUN Fusion SE è il motore scelto dalla BOSCH e fornisce la trazione alla macchina. Questo dispositivo, sviluppato da Hobbywing, integra un motore brushless DC e un regolatore di velocità elettronico (ESC) in un'unica unità compatta, eliminando la necessità di cablaggi complessi e migliorando l'efficienza energetica.

Il QUICKRUN Fusion SE offre prestazioni elevate avando un funzionamento fluido e silenzioso: ciò lo rende ideale per applicazioni in veicoli RC crawler e scaler. Grazie alla tecnologia FOC (Field-Oriented Control), il sistema garantisce una risposta precisa e una coppia elevata a basse velocità, migliorando il controllo e la manovrabilità del veicolo in terreni difficili.

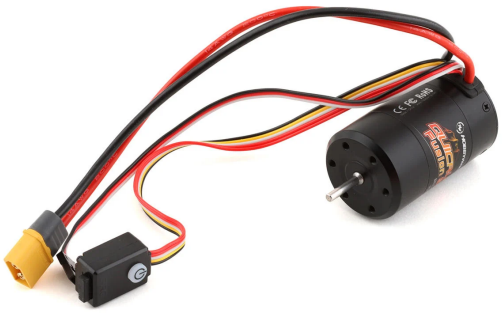


Figura 2.6.: Motore QUICKRUN Fusion SE

Il motore è molto capace e può raggiungere fino a 1800KV (1800 RPM per volt di tensione fornito) essendo progettato per veicoli RC da corsa. Questa tesi esplora i metodi utilizzati per massimizzare le prestazioni a basse velocità, poiché la guida su strada dovrebbe essere condotta mantenendo velocità ridotte in risposta ai pericoli stradali.

Il controllo viene effettuato mediante la tecnica del PWM, che verrà spiegata più nel dettaglio in seguito, e la velocità desiderata sarà mantenuta grazie ad un controllore PID. Tuttavia, per poterlo far funzionare è necessaria una misurazione in tempo reale della velocità, che viene fatta tramite l'uso di un Encoder.

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

2.5. Encoder

La spiegazione del funzionamento e dell'implementazione derivano dalle slide [2]. L'encoder è un dispositivo di controllo elettromeccanico che viene usato per ottenere un feedback ed è tale da generare un segnale ad onda quadra che viene interpretato da un controllore. In particolare, in questo lavoro di tesi, viene utilizzato l'AMT103, che è un encoder di tipo rotativo incrementale a doppio canale: segnala gli incrementi rispetto ad una posizione assunta come riferimento indipendentemente dal verso di rotazione, fornendo cioè informazioni sulla variazione di posizione, ma non sulla posizione assoluta. Il principio di funzionamento di questi tipi di encoder è piuttosto semplice ed è tale per cui il sensore è costituito da un disco rotante scanalato attaccato all'albero: ciò permette loro di ruotare come se fossero parte di un unico sistema (come si vede nella Figura 2.7).

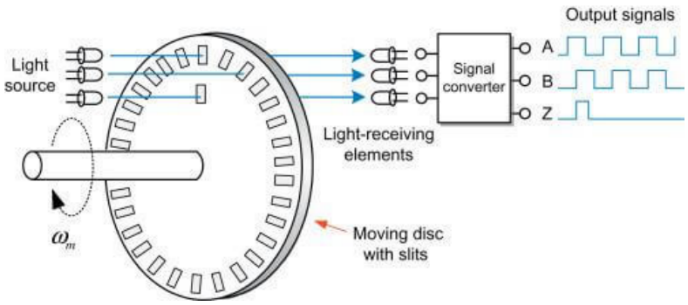


Figura 2.7.: Principio di funzionamento dell'encoder

Viene posizionata una sorgente luminosa (LED) da una parte del disco e un dispositivo ricevente (un fotodiodo/fototransistor) nella linea di vista dall'altra parte dello stesso. Quando il disco ruota, gli slot lasciano passare momentaneamente la luce: in questo modo essa raggiunge il ricevitore, che emette un segnale logico alto. Quando il disco ruota ulteriormente, il raggio di luce è ostruito: non arriva, quindi, al ricevitore e, di conseguenza, l'uscita del canale corrispondente diventa un segnale logico basso.

Gli encoder rotativi incrementali a doppio canale, come quello utilizzato in questa tesi, hanno due canali denominati canale A e B, che forniscono due segnali ad onda quadra sfasati tra loro di 90°elettrici. Con la lettura del solo canale A si riesce a calcolare l'informazione relativa alla velocità di rotazione (numero di impulsi nell'unita di tempo), mentre con anche la lettura del segnale B, si riesce a rilevare il senso di rotazione. Oltre a ciò è disponibile un ulteriore segnale chiamato canale Z, che fornisce una posizione assoluta di riferimento, ma che è inutilizzato.
L'encoder presenta 5 pin:

- B: corrisponde al canale B;
- 5V: corrisponde al pin per l'alimentazione a 5V;
- A: corrisponde al canale A;
- X: corrisponde al canale Z;
- G: corrisponde al ground.

Figura 2.8.: Encoder rotativo AMT103

In particolare, l'AMT103 è tale da fornire una risoluzione elevata consentendo una misurazione precisa dell'angolo di rotazione. Grazie all'utilizzo di questo encoder, è possibile misurare la velocità longitudinale a cui si sta muovendo il veicolo: ciò permette di effettuare un controllo corretto della velocità stessa e di conseguenza della curvatura.

Le informazioni sul componente sono state prese dal datasheet [7] rilasciato dalla casa madre.

2.6. Servo motore

Il modello di servo motore impiegato è il Reely RS610WP, comunemente utilizzato in diverse apparecchiature elettroniche e applicazioni che richiedono un controllo di posizione accurato: questo dispositivo riesce infatti a regolare il movimento in modo molto preciso.

Figura 2.9.: Servo motore Reely RS610WP

Controllato dal microcontrollore tramite tecnica di modulazione PWM, il sistema ha lo scopo, all'interno del progetto, di utilizzare un controllore PID e le informazioni dell'IMU per regolare la sterzata in modo fluido.

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

2.7. IMU

L'IMU, Inertial Measurement Unit, è un dispositivo che misura accelerazione, velocità angolare e orientamento di un oggetto in tre dimensioni nello spazio, dati fondamentali per determinarne i movimenti e la posizione in tempo reale. L'IMU utilizza:

- un accelerometro per misurare l'accelerazione lungo gli assi x, y e z: questa viene utilizzata per calcolare le variazioni di velocità e posizione nel tempo;
- un giroscopio per misurare la velocità angolare di rotazione intorno agli stessi assi: ciò permette di determinare i cambiamenti di orientamento dell'oggetto;
- alcuni IMU sono dotati anche di un magnetometro, che misura il campo magnetico terrestre e fornisce informazioni sull'orientamento dell'oggetto in questione rispetto ai punti cardinali.

Tutti i dati provenienti dai sensori vengono inviati a un'unita di elaborazione interna all'IMU: questa processa i dati grezzi per calcolare orientamento, velocità e posizione dell'oggetto nel tempo.
Oltre a ciò, spesso viene utilizzato un filtro di Kalman che migliora la precisione, combinando dati misurati e dati previsti. Questo dispositivo comunica con l'utente o con dispositivi esterni attraverso diverse interfacce come UART, I2C o SPI.

L'IMU utilizzato nel progetto è il BNO055, prodotto dalla BOSCH. La potenza di quest'ultimo è data della sua capacità di combinare i dati provenienti da tutti e tre i tipi di sensori: accelerometro, giroscopio e magnetometro in una modalità sensor fusion definita NDOF. Questo consente al sensore di fornire misurazioni di orientamento estremamente precise e stabili. Un'altra caratteristica notevole del BNO055 è la sua capacità di calibrarsi automaticamente: può compensare le variazioni ambientali, come i cambiamenti di temperatura e di inclinazione, senza richiedere interventi esterni.

Figura 2.10.: IMU BNO055

Lo scopo del BNO055 all'interno del progetto di tesi è quello di fornire dati riguardo l'accelerazione del veicolo lungo l'asse y, così da permettere al PID di svolgere un controllo corretto della velocità di sterzata, mantenendo al contempo fluida la guida. Oltre a ciò, viene utilizzato per rilevare il momento in cui la macchina si trova sulla rampa mediante il pitch.

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

2.8. Chassis

Lo chassis è una struttura di supporto su cui vengono montati i vari componenti di una macchina. E' costituito dal telaio principale su cui sono installate altre parti, come il motore o le sospensioni. Questa struttura fornisce supporto, protezione e integrità strutturale all'intero sistema.
Lo chassis utilizzato nel progetto è il modello Reely TC-04 Onroad-Chassis 1:10, RC model car Electric Road version 4WD ARR.

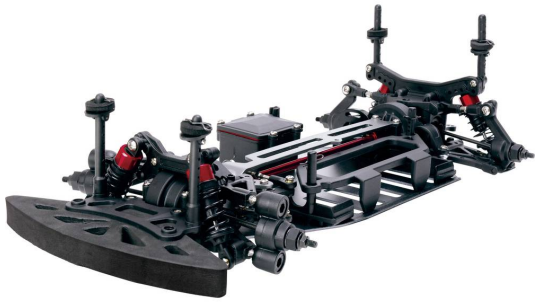


Figura 2.11.: Chassis

Le informazioni sul componente sono state prese dal datasheet [6] rilasciato dalla casa madre.

2.9. Batteria

La batteria impiegata è la *Conrad energy Scale model battery pack (LiPo) da 7.4 V e 5500 mAh, No. of cells: 2 20 C Softcase XT90*. E' una batteria ai polimeri di litio (LiPo) con una tensione nominale di 7,4 V e una capacita di 5500 mAh. Ciò significa che è in grado di fornire una corrente di 5500 mA (o 5,5 A) per un'ora prima che si esaurisca completamente. La batteria è composta da due celle collegate in serie per raggiungere la tensione di 7,4 V.

La specifica 20 C, presente su di essa, indica la capacità massima di scarica continua: in questo caso, la batteria può essere scaricata in modo continuo ad una corrente massima di 20 volte la sua capacita nominale (5500 mAh), che equivale a 110 A.

Figura 2.12.: Conrad energy Scale model battery pack

Infine, tale batteria è contenuta in un involucro morbido (Softcase): questo garantisce una maggiore flessibilità e facilità di installazione. Inoltre, utilizza un connettore XT90, ampiamente usato nel modellismo elettrico, per garantire una connessione elettrica adeguata e una trasmissione efficiente di corrente tra la batteria e il dispositivo o motore che alimenta.

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

2.10. Camera

La camera utilizzata è la Pi Camera V2.1, sviluppata da Raspberry Pi Foundation per essere utilizzata con le schede Raspberry Pi.
E' dotata di un sensore Sony IMX219 da 8 megapixel: ciò significa che può catturare immagini ad alta risoluzione con dettagli nitidi. Inoltre, ha anche la capacità di

registrare video fino a 1080p a 30 frame al secondo: ciò la rende ideale per progetti che richiedono la registrazione video di alta qualità. In aggiunta, offre una buona profondità di campo e una visione nitida degli oggetti sia da vicino che da lontano.

Altri aspetti notevoli di questa camera sono la sua dimensione compatta e il suo peso leggero, che la rendono facilmente integrabile in una varietà di progetti e applicazioni.
Infine, la connessione alla scheda Raspberry Pi avviene tramite un semplice cavo a 15 pin, che facilita l'integrazione e offre numerose opzioni di montaggio.

Figura 2.13.: Camera

Le informazioni sul componente sono state prese dal datasheet [14] rilasciato dalla casa madre.

2.11. LIDAR

Il LiDAR (Light Detection And Ranging) LD06 è un sensore di rilevamento laser compatto e ad alte prestazioni. Utilizzato principalmente per la mappatura e la navigazione di robot autonomi, funziona tramite l'emissione di impulsi laser e la misurazione del tempo impiegato da questi impulsi per riflettersi sugli oggetti circostanti e tornare al sensore. Ciò permette di calcolare con precisione la distanza dagli ostacoli.

l'LD06 offre una portata di rilevamento fino a 12 metri con una precisione di ± 2 cm, e un angolo di scansione di 360 gradi grazie a un sistema di rotazione interno. Il dispositivo opera ad una frequenza di scansione fino a 4500 Hz, garantendo una rapida acquisizione dei dati. Inoltre, come già accennato, il Lidar LD06 è progettato per essere leggero e di dimensioni contenute, facilitando l'integrazione in diverse piattaforme robotiche.

La sua interfaccia di comunicazione, che solitamente avviene tramite USB o UART, permette un facile collegamento con vari sistemi di controllo.
Nel caso in questione, si è optato per un collegamento USB direttamente sulle porte disponibili dalla Raspberry.

Il rilevamento grezzo degli ostacoli per ogni grado è poi stato filtrato ed elaborato in modo da riuscire a distinguere gli ostacoli filtrando eventuali errori di misurazione o disturbi.

Figura 2.14.: LiDAR LD06

Le informazioni sul componente sono state prese dal datasheet [11] rilasciato dalla casa madre.

2.12. Connessione tra le componenti

Di seguito è riportato il diagramma delle connessioni tra le differenti componenti hardware descritte fino ad ora. La Figura 2.15 rappresenta il diagramma effettivo, mentre la Figura 2.16 è la legenda:

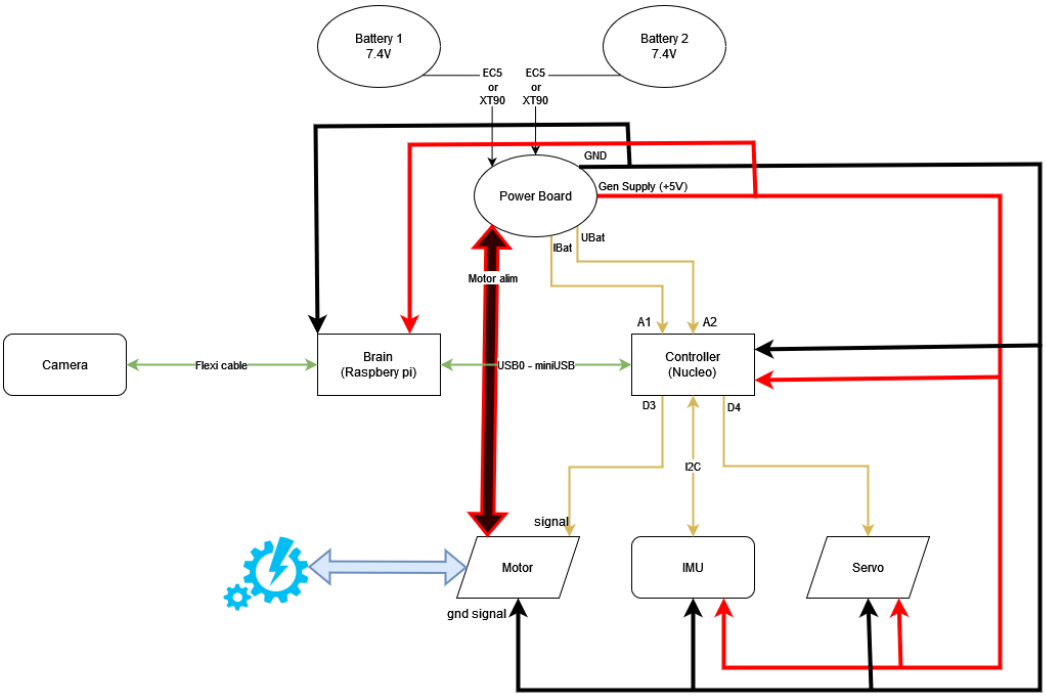


Figura 2.15.: Diagramma delle connessioni

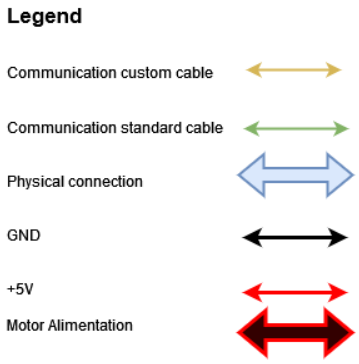


Figura 2.16.: Diagramma delle connessioni

2.13. Schema elettrico

Viene riportato in Appendice A lo schema elettrico di tutte le componenti. Tale configurazione è stata realizzata alla fine del progetto dopo aver definito tutte le componenti necessarie, in modo da rendere il veicolo il più pulito e manutenibile possibile.

Capitolo 3.

Preparazione del motore

Per poter mettere il motore in funzione ci sono alcuni accorgimenti da fare, come la gestione dell'alimentazione, la gestione dell'ESC e la lettura della velocità in modo da poter implementare un ciclo in catena chiusa.

3.1. Analisi del motore

Come già introdotto nella Sezione 2.4, l'energia elettrica della batteria è trasformata in energia meccanica mediante il motore brushless QUICRUN Fusion SE, collegato direttamente sia all'asse posteriore che anteriore, rendendo la macchina a trazione integrale. Questo tipo di trazione sta diventando sempre più comune nelle auto reali tanto quanto in quelle radiocomandate in quanto permette di avere un maggiore controllo su differenti tipologie di terreni. La trazione integrale è un'ottima soluzione per evitare che della polvere sul pavimento faccia slittare la macchina in scala 1:10, la quale risente maggiormente di piccoli disturbi.

Andando ad analizzare più a fondo le connessioni del motore, come si può notare in Figura 3.1, questo è connesso a 3 componenti:

- una batteria o sorgente di alimentazione;
- un ricevitore che nel caso specifico è costituito unicamente da un cavo che trasporta il segnale in PWM all'ESC;
- un pulsante multifunzione che serve sia ad accendere/spegnere il motore che a definire un range personalizzato di funzionamento per l'ESC, come sarà spiegato più nel dettaglio nel Capitolo 3.3. Il pulsante è anche dotato di un LED che ne indica lo stato.

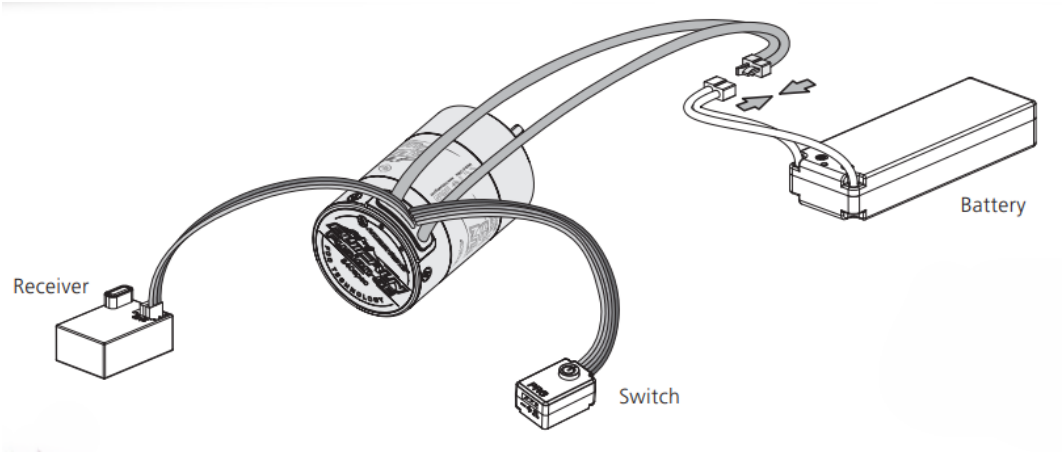


Figura 3.1.: Schema del motore

3.1.1. LED del pulsante

E' bene conoscere i vari stati che può assumere il LED del motore, in quanto può essere un mezzo utile durante un'eventuale fase di debugging oppure semplicemente per accertarsi che tutto proceda correttamente.

1. Fase di avvio
    - La luce rossa è sempre accesa: indica lo stato normale dopo l'accensione;
    - La luce rossa lampeggia continuamente e rapidamente: nessun segnale PWM è rilevato dall'ESC o il duty cycle, imposto come neutro nell'ESC, non corrisponde con quello ricevuto;
    - La luce verde lampeggia N volte: rilevamento del numero di celle Lipo (lampeggia N volte per indicare che ci sono N celle Lipo).
2. Fase di guida
    - Il PWM fornito è nell'intervallo neutrale, e la luce verde si spegne;
    - Durante l'avanzamento la luce verde lampeggia; quando viene fornito il duty cycle massimo accettabile la luce verde è sempre accesa;
    - Durante la retromarcia la luce verde lampeggia. In particolare, quando viene fornito il duty cycle minimo accettabile, la luce verde è sempre accesa.
3. Quando vengono attivate le funzioni di protezione dell'ESC:
    - Se lampeggia continuamente la luce rossa, l'ESC è entrato in stato di protezione, dovuto alla bassa tensione;
    - Se lampeggia continuamente la luce verde, la temperatura dell'ESC è troppo alta, quindi questo entra in stato di protezione dal surriscaldamento.

3.2. Tecnica di alimentazione: PWM

3.2.1. Funzionamento dei timer

Per comprendere il processo di generazione di un segnale PWM è fondamentale conoscere il funzionamento dei timer. I concetti che verranno spiegati di seguito sono stati ripresi dalle slide [1].

I timer sono utilizzati per applicazioni in cui è necessario tener traccia in modo preciso e affidabile dello scorrere del tempo, azione richiesta frequentemente quando si fa uso dei microcontrollori. E' molto importante denotare la loro affidabilità, in quanto utilizzano il clock della CPU per aggiornare un loro registro interno: ciò permette loro di agire in parallelo con tutte le altre task. Così facendo, si è certi che un software pesante o mal ottimizzato non interferisca con il conteggio del tempo.

Oltre a ciò, il timer è dotato di un registro di comparazione (*CCR: Catch Compare Register*) al quale si può assegnare un valore: al raggiungimento del contatore di tale valore corrisponde una determinata azione (ad esempio la chiamata di un interrupt).

Di seguito nella Figura 3.2 è riportato un grafico che rappresenta il funzionamento di un timer e si nota come il contatore si resetti continuamente una volta raggiunto il valore massimo consentito, denominato *counter period* e immagazzinato nel registro *ARR* (*Auto Reload Register*).

Questo metodo di funzionamento del timer è definito *free running* poiché questo continua la sua azione indefinitamente. Esistono diverse modalità operative dei timer che, tuttavia, non verranno trattate in quanto esulano dall'obiettivo prefissato.

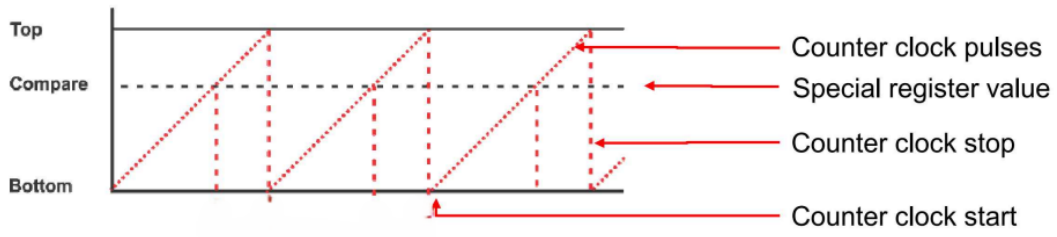


Figura 3.2.: Funzionamento timer

Riassumendo, un timer è un contatore che aumenta fino al raggiungimento del suo valore massimo ad ogni colpo di clock ricevuto, sia esso interno o esterno al microcontrollore, e il tempo trascorso viene ricavato fattorizzando il valore raggiunto dal contatore e la frequenza di clock che si sta utilizzando.

Infine, ai fini dello studio, è bene conoscere quali sono i passaggi per calcolare la frequenza di un timer. Innanzitutto sono necessari 3 parametri, che sono:

- $Timer_{clock}$: frequenza che il timer riceve dalla CPU;
- *Prescaler*: circuito che permette di ridurre la frequenza di clock percepita dal timer in un suo sottomultiplo;

- *Counter period*: valore massimo che il contatore può raggiungere prima di resettarsi.

Questi parametri verranno settati in base al caso specifico ed utilizzati nella seguente formula:

$$\text{Frequenza} = \frac{\text{Timer}_{\text{clock}}}{(\text{Prescaler} + 1)(\text{Period} + 1)} \tag{3.1}$$

3.2.2. Come funziona il PWM

Il PWM, Pulse Width Modulation (Modulazione a Larghezza di Impulso), è una tecnica ampiamente utilizzata in elettronica per effettuare il controllo di motori, LED, e altri dispositivi in modo preciso. Il materiale originario di questi concetti è il [3].

Il PWM funziona modulando la larghezza degli impulsi di un segnale digitale. Questo significa che è possibile variare la durata (o larghezza) di ogni impulso mentre viene mantenuta costante la frequenza complessiva del segnale. La percentuale di tempo durante la quale il segnale è alto si chiama duty cycle e si misura in percentuale: perciò può variare da 0% (segnale sempre basso) a 100% (segnale sempre alto).

Per meglio comprendere la sua generazione si fa riferimento alla teoria dei timer spiegata sopra e, osservando la Figura 3.3, è facile notare che per far variare il duty cycle è sufficiente modificare il valore del registro di compare. Così facendo, la prima volta che il contatore raggiunge questo valore il segnale entra in uno stato di valore alto, mentre quando il contatore si resetta questo assume nuovamente per un istante il valore di compare, portando così il segnale ad un valore basso.

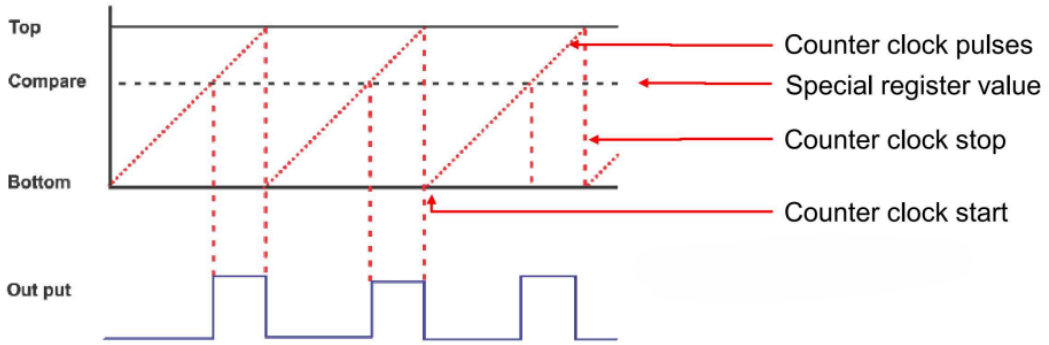


Figura 3.3.: Generazione del segnale PWM

Per controllare l'ampiezza del segnale di uscita, è essenziale regolare il duty cycle, ovvero la percentuale di tempo in cui il segnale di uscita si trova in uno stato alto rispetto al periodo totale.
In particolare, un duty cycle più elevato comporta un segnale di uscita con un'ampiezza maggiore, mentre un duty cycle piu basso produce un segnale con un'ampiezza

inferiore.

Tra i vantaggi apportati dal PWM mostra una notevole:

1. Efficienza: il PWM è molto efficiente perché i dispositivi di commutazione sono completamente accesi o spenti. Ciò riduce le perdite di potenza;
2. Precisione: il PWM permette un controllo preciso della potenza fornita ai dispositivi;
3. Flessibilità: il PWM può essere utilizzato in una vasta gamma di applicazioni, dalla regolazione della velocità dei motori al controllo della luminosità dei LED.

3.2.3. Generazione del PWM con MCU

Dopo aver capito la teoria dietro il PWM e la sua generazione, andiamo a vedere concretamente come quest'ultimo viene generato utilizzando il microcontrollore.

Analizzando il datasheet del motore si vede che l'attuatore è costruito su un PWM con una frequenza di $50Hz$, quindi all'interno del file *.ioc* utilizzato per definire un nuovo timer devono essere inseriti i seguenti parametri come si può osservare in Figura 3.4:

- Prescaler $= 840 - 1$;
- Counter period $= 2000 - 1$.

Questi valori sono stati ottenuti conoscendo la frequenza desiderata, infatti:

$$\text{Frequenza} = \frac{84.000.000}{(840)(2.000)} = 50Hz \tag{3.2}$$

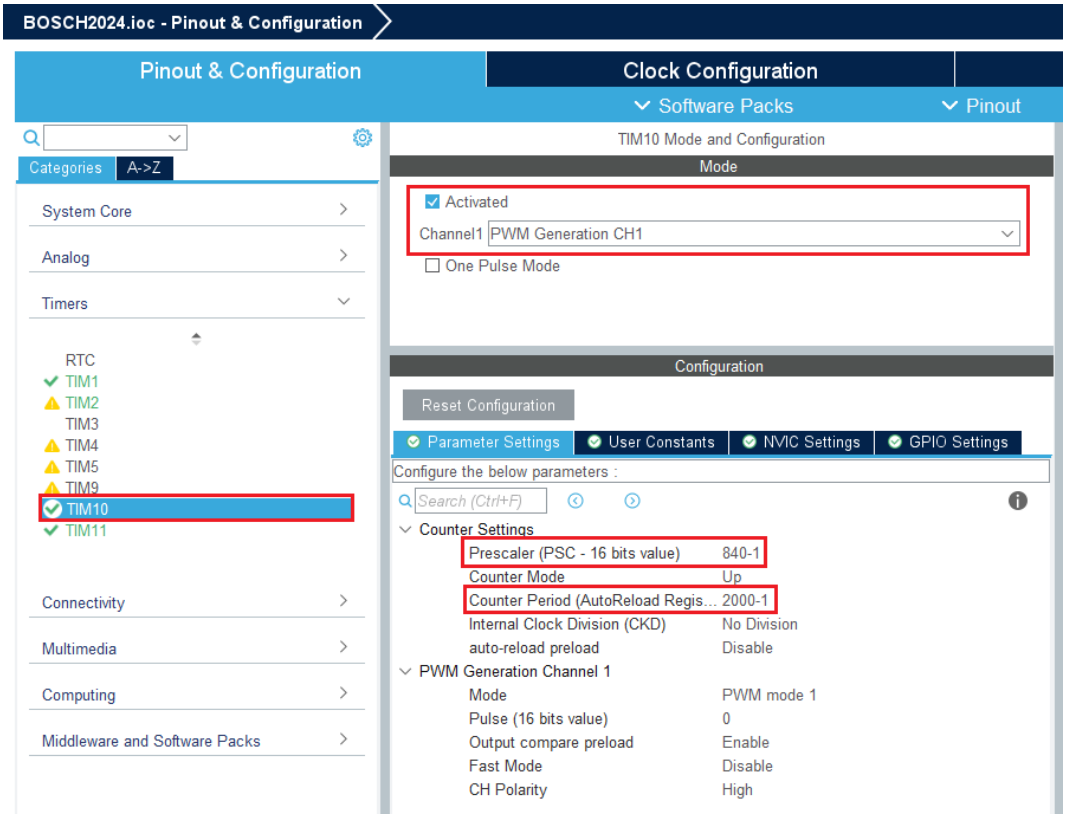


Figura 3.4.: Generazione del PWM nell'ioc

Per quanto riguarda il codice in C sarà sufficiente inizializzare il timer una volta entrati nel main nel modo che segue:

```
1         HAL_TIM_PWM_Start(&htim10, TIM_CHANNEL_1);
2
```

Listing 3.1: Inizializzazione timer

Successivamente, per assegnare il PWM al PIN definito in precedenza (PB2 in questo caso), è stata creata una funzione che richiede come parametro un float dove andrà specificato il duty cycle:

```
1     void BL_set_PWM(float duty){
2      TIM10->CCR1 = duty*TIM10->ARR;
3     }
4
```

Listing 3.2: Assegnazione PWM

Nella formula presente nel Codice 3.2 si pone il valore del registro di compare (*TIM10->CCR1*) ad una percentuale del valore massimo del contatore (*TIM10->ARR*) definita dal valore *duty* che viene passato alla funzione.

3.3. Alimentazione tramite ESC

3.3.1. Cos'è e come funziona

Un ESC (Electronic Speed Controller) è un dispositivo elettronico utilizzato per controllare la velocità, la direzione e, talvolta, la frenatura di un motore brushless. Di seguito vengono elencati concetti derivanti dalle slide [4].

I motori brushless (BLDC - Brushless DC Motors) funzionano senza spazzole, utilizzando una serie di magneti permanenti sul rotore e un avvolgimento elettrico sullo statore. La commutazione della corrente negli avvolgimenti dello statore genera un campo magnetico rotante, che interagisce con i magneti permanenti del rotore, producendo movimento.

Il compito principale dell'ESC è quindi quello di commutare la corrente tra gli avvolgimenti del motore in modo sincronizzato con la posizione del rotore. Questo processo è noto come commutazione elettronica e può essere di due tipi:

- Commutazione sensorizzata: utilizza sensori (come Hall effect sensors) per rilevare la posizione del rotore e commutarla di conseguenza;
- Commutazione senza sensori: rileva la posizione del rotore attraverso la back EMF (Electromotive Force) generata negli avvolgimenti del motore.

Per variare la velocità del motore, l'ESC utilizza la modulazione della larghezza dell'impulso (PWM - Pulse Width Modulation). Regolando la durata degli impulsi di tensione applicati agli avvolgimenti, l'ESC controlla la quantità media di potenza fornita al motore, influenzando così la velocità di rotazione.

Come abbiamo già visto, il nostro ESC è dotato di diversi meccanismi di protezione per prevenire danni dovuti a sovraccarichi, sovratensioni, surriscaldamento, e cortocircuiti.

In Figura 3.5 è riportato lo schema elettrico di un ESC collegato ad un motore brushless trifase, che possiede quindi 3 avvolgimenti di cui controllerà l'eccitazione:

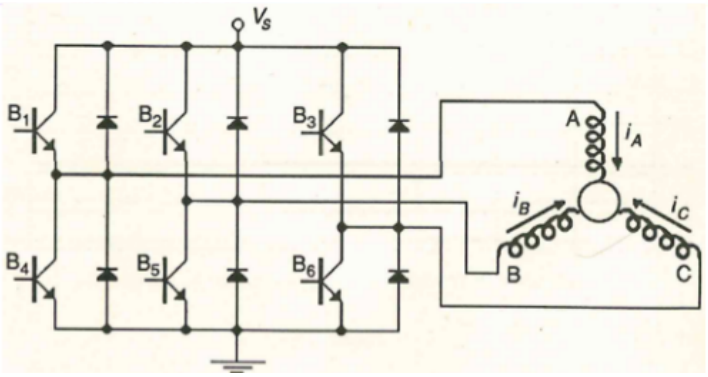


Figura 3.5.: Schema elettrico ESC

3.3.2. Perché tarare l'ESC

La taratura dell'ESC è di fondamentale importanza, in quanto permette all'ESC di interpretare il segnale PWM che gli viene mandato nel modo corretto.

Ogni volta che avviene un POR (power on reset), ovvero quando viene acceso il veicolo (più nello specifico il motore) è necessario calibrare l'ESC in quanto gli algoritmi di controllo sono stati tarati in funzione della caratteristica ottenuta a seguito della calibrazione scelta. Non calibrare il motore condurrà ad un controllo instabile della trazione.

3.3.3. Procedura di taratura

Il codice é stato sviluppato in modo tale da rendere il veicolo una macchina a stati, che possono essere navigati mediante la pressione del tasto blu presente sulla scheda. Gli stati sono 3 e sono i seguenti:

- Listening (DEFAULT): è lo stato iniziale a seguito dell'accensione/reset della scheda. In questo stato il veicolo riceve mediante comunicazione seriale i setpoint di velocità e curvatura da seguire, che vengono successivamente forniti agli algoritmi di controllo;
- Not listening: pone il veicolo in uno stato di idle, ignorando i messaggi in arrivo dalla seriale (ideale per debugging);
- Procedura di calibrazione del motore: questo stato permette di calibrare l'ESC del motore brushless per ottenere una curva caratteristica meno sensibile alla variazione dei segnali PWM forniti dagli algoritmi di controllo

In Figura 3.6 è riportato un diagramma di macchine a stati esplicativo:

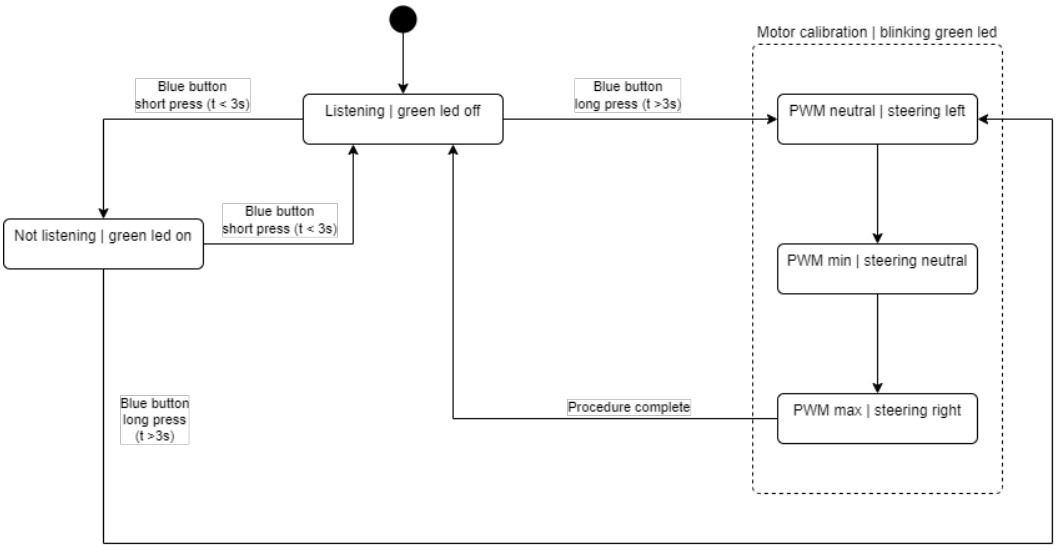


Figura 3.6.: Macchina a stati del veicolo

La calibrazione del motore richiede alcuni semplici passi, che sono riportato di seguito:

1. Verificare che l'ESC del motore sia spento (led ESC spento). Se risulta essere acceso occorre spegnerlo tenendo premuto il tasto dell'ESC;
2. Accendere il motore in modalità di calibrazione (tenere premuto il tasto dell'ESC finché il motore non inizia a produrre segnali acustici intermittenti, rilasciare il tasto in seguito);
3. Attivare lo stato di calibrazione del veicolo seguendo le istruzioni sopra riportate;
4. Nel momento in cui le ruote sterzano verso sinistra, premere il tasto dell'ESC fino a che il motore non produce un suono distinto (-);
5. Premere il tasto dell'ESC nel momento in cui le ruote sterzano verso la posizione neutra, fino a quando il motore non produce un suono distinto (- -);
6. Premere il tasto dell'ESC nel momento in cui le ruote sterzano verso destra, fintantoché il motore non produce un suono distinto (- - -).

Se la procedura di calibrazione é avvenuta correttamente, quando alla fine della procedura le ruote sterzano verso la posizione neutra, il motore dovrebbe produrre un suono distinto (—-)

3.3.4. Implementazione procedura di taratura

Dopo aver compreso la procedura dal punto di vista teorico, si passa ad trattare la sua implementazione che sarà divisa in 2 parti:

1. Configurazione user button: permette di entrare in modalità calibrazione, come descritto nel Capitolo 3.3.3;
2. Procedura di calibrazione: il metodo vero e proprio che temporizza la calibrazione.

Per quanto riguarda lo user button questo è stato configurato in modalità interrupt, in modo da rilevare la pressione indipendentemente dal codice che sta venendo eseguito.

In Figura 3.7 è riportata la configurazione che si ha abilitando il pin *PC13* e impostando la *GPIO mode* in *External Interrupt Mode with Rising/Falling edge trigger detection*, in modo da mandare un segnale di interrupt sia quando si preme il pulsante che quando lo si rilascia. Questo è fondamentale in quanto si sta cercando di percepire una pressione prolungata del pulsante.

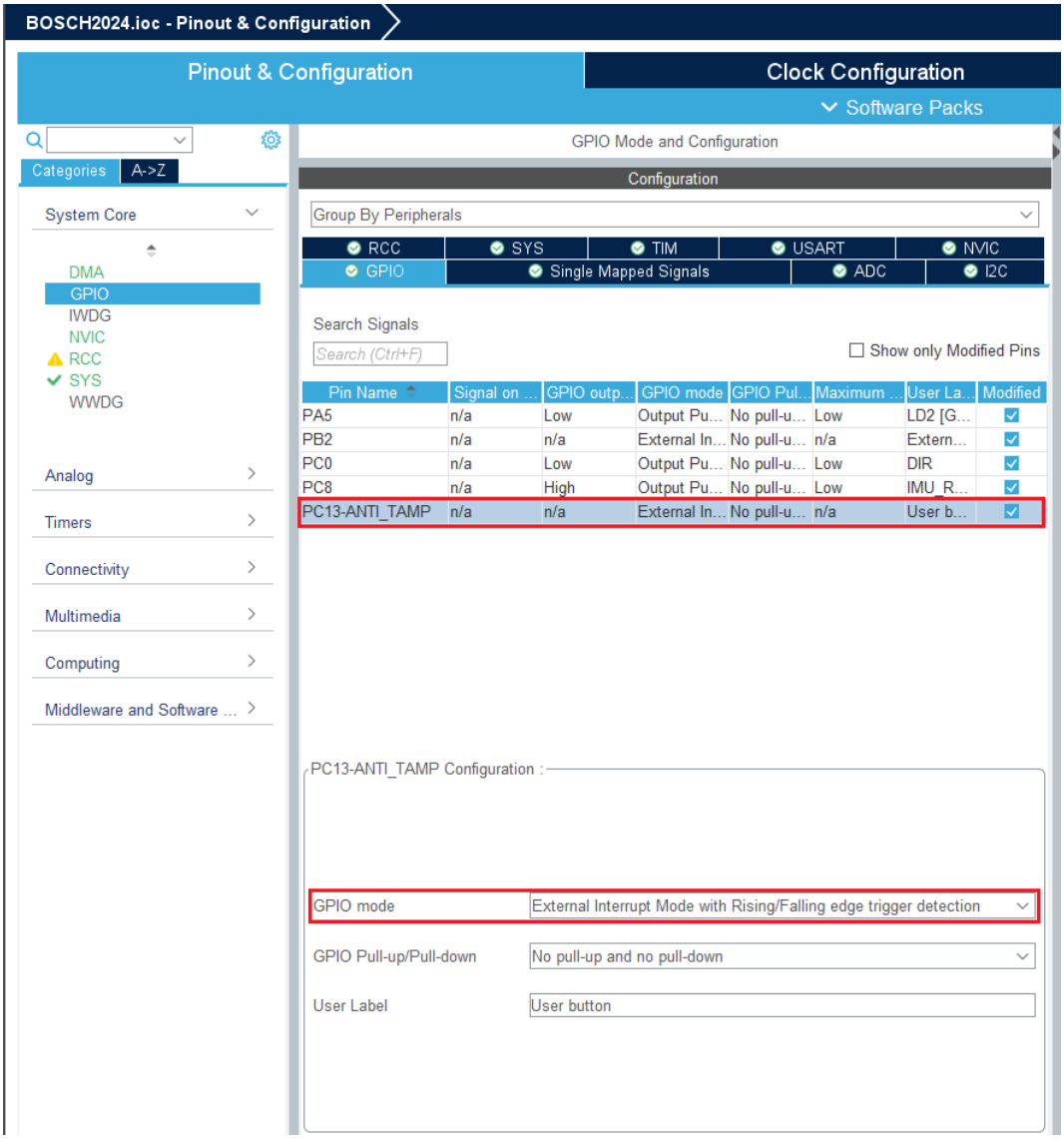


Figura 3.7.: ioc user button GPIO

Infine, per far si che il tasto funzioni in modalità interrupt sarà necessario abilitarlo nella schermata *NVIC* come in Figura 3.8

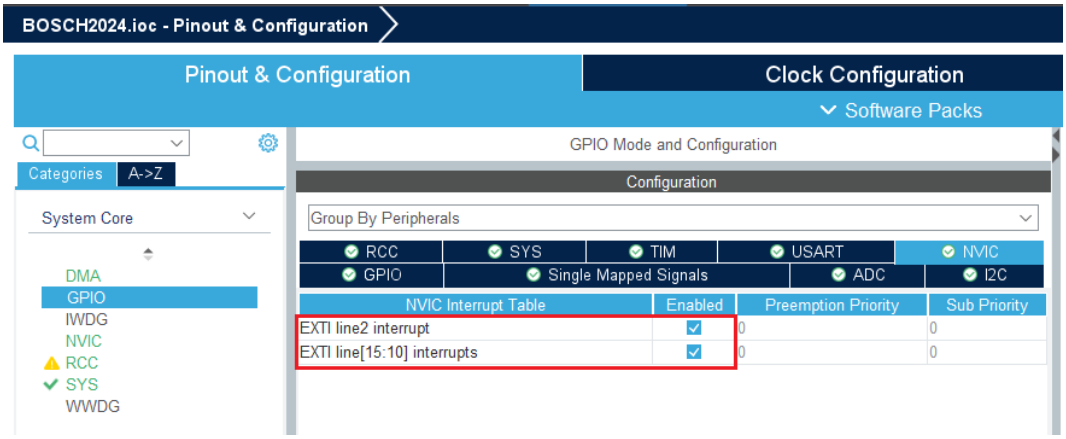


Figura 3.8.: ioc user button NVIC

Per quanto riguarda il codice questo è strutturato come si riportadi seguito (Codice 3.3). E' bene ricordare che:

- cnt_10ms_button: incrementa il suo valore di 1 ogni 10ms;
- SHORT_PRESS_THRESHOLD: ha un valore di 100, e simboleggia che una pressione breve deve durare meno di 1 secondo;
- LONG_PRESS_THRESHOLD: ha un valore di 300, e simboleggia che una pressione prolungata deve durare più di 3 secondi;
- buttonPressStartTime: immagazzina il valore in ms di quando viene premuto il pulsante;
- buttonPressEndTime: immagazzina il valore in ms di quando viene rilasciato il pulsante.
- flag_button: rappresenta lo stato della macchina come espresso dalla Figura 3.6

```
1   void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
2     if (GPIO_Pin == GPIO_PIN_13){
3       if ((HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) == GPIO_PIN_RESET)){
4         // Button pressed
5         buttonPressStartTime = cnt_10ms_button;
6
7       } else { // Button released
8         buttonPressEndTime = cnt_10ms_button;
9
10        //Verifico quanto tempo ho tenuto premuto il tasto
11        pressDuration = buttonPressEndTime - buttonPressStartTime;
12        if (pressDuration < SHORT_PRESS_THRESHOLD){
13          if(flag_button >= 0 && flag_button < max_flag_button){
14            flag_button++;
15          }
```

```
16              else{
17                flag_button = 0;
18              }
19            } else if (pressDuration >= LONG_PRESS_THRESHOLD){
20              flag_button = -1;
21              counter_cal_ESC = 0;
22            }
23          }
24        }
25      }
26
```

Listing 3.3: User button

Si passa ora a discutere della procedura effettiva di temporizzazione per la calibrazione. Questa ha l'obiettivo di passare al motore il PWM che si desidera registrare, e come descritto nel Capitolo 3.3.3 sarà necessario premere il pulsante sull'ESC ogni volta. Sono state adoperate quindi 2 tecniche per comunicare quando premere il pulsante. Queste sono:

- Cambio di stato del LED del MCU;
- Sterzo delle ruote impostato verso sinistra e destra in modo alternato.

Nel Listing 3.4 è riportato il metodo che gestisce la temporizzazione.

```
1   void ProceduraCalibrazione(){
2     if(counter_cal_ESC < 5){
3       HAL_GPIO_WritePin(GPIOE, GPIO_PIN_1, GPIO_PIN_RESET);
4     }
5     if(counter_cal_ESC <= 300){
6       if(!(counter_cal_ESC % 15)){
7         HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
8       }
9     }
10    else if(counter_cal_ESC <= 600){
11      duty = NEUTRAL_PWM;
12      BL_set_PWM(duty);
13      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
14      servo_motor(-20);
15    }
16    else if(counter_cal_ESC <= 900){
17      duty = MAX_PWM;
18      BL_set_PWM(duty);
19      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
20      servo_motor(0);
21    }
22    else if(counter_cal_ESC <= 1100){
23      duty = MIN_PWM;
24      BL_set_PWM(duty);
25      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
26      servo_motor(20);
```

```
27        }
28        else if(counter_cal_ESC <= 1200){
29          if(!(counter_cal_ESC % 15)){
30            HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
31          }
32        }
33        else if (counter_cal_ESC <= 1300){
34          flag_button = 0;
35        }
36      }
37
```

Listing 3.4: Procedura calibrazione

3.4. Lettura della velocità tramite Encoder

Per un controllo efficace del motore è necessario un feedback sull'uscita del motore (quindi la velocità erogata) in base agli ingressi che vengono forniti (quindi la velocità di riferimento impostata). Questo concetto è alla base del controllo in catena chiusa ed è rappresentato dallo schema a blocchi in Figura 3.9

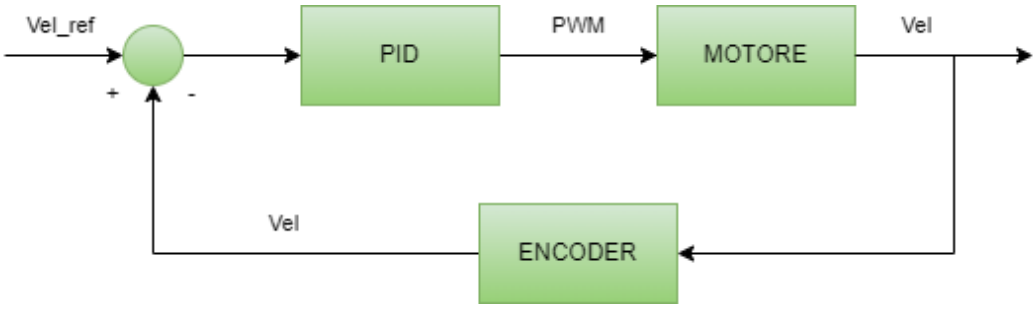


Figura 3.9.: Catena chiusa encoder

Nel Capitolo 2.5 è stato spiegato il funzionamento teorico di questo sensore. Si passa, ora, a vedere come viene implementato all'atto pratico.

3.4.1. Montaggio dell'encoder

Durante la fase di testing su banco è stato stampato sia un supporto in 3D realizzato su misura per il motore che l'encoder utilizzati. In questo modo è stato possibile isolare queste due componenti al fine di affrontare un solo problema alla volta.

In Figura 3.10 viene riportata una vista del modello.

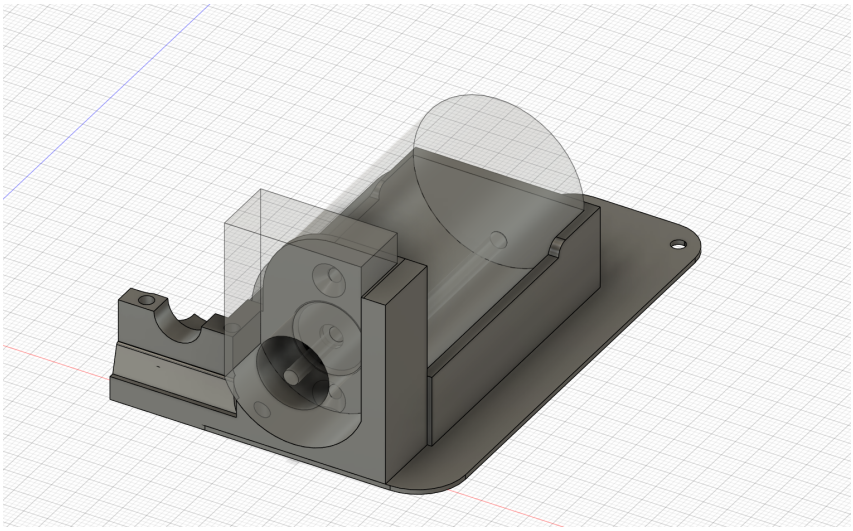


Figura 3.10.: Modello 3D del supporto per l'encoder

Lo stesso lavoro è stato fatto sulla macchina per permettere di simulare un tachimetro, in modo da poter regolare in tempo reale al velocità del veicolo in corsa.

3.4.2. Configurazione dell'encoder

Dopo aver montato l'encoder si passa a vedere come questo possa funzionare utilizzando la NUCLEO F401RE e l'ambiente di sviluppo CubeIDE.

Per prima cosa, è necessario inizializzare un timer in modalità encoder attraverso il *.ioc*. Come si vede in Figura 3.11 è stato utilizzato il TIM2, specificando nella voce *Combined channels* la modalità *Encoder mode*. Successivamente, nei parametri nella sezione *Encoder -> Encoder mode* sono stati selezionati entrambi i canali.

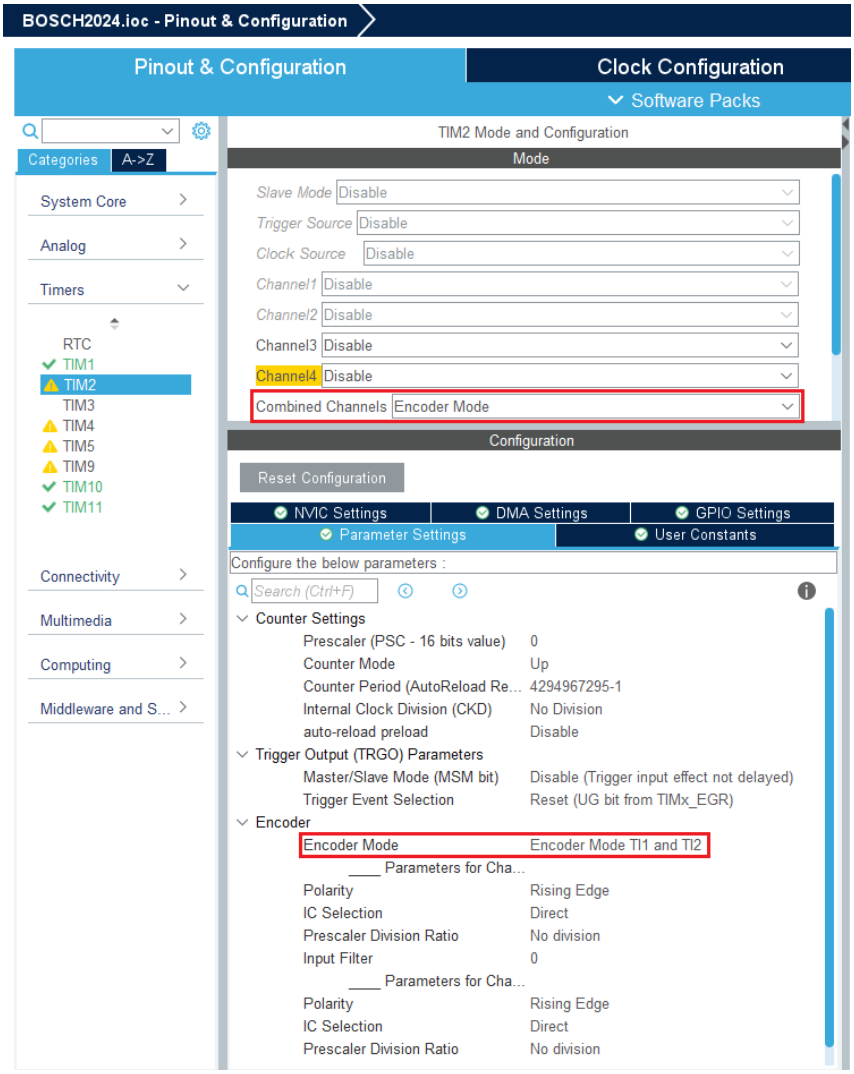


Figura 3.11.: Configurazione ioc encoder

Passando al codice sono stati inizialmente definiti i parametri dell'encoder in una libreria apposita, seguendo il datasheet e la configurazione effettuata

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

```

Listing 3.5: Parametri encoder

I 3 parametri utilizzati sono:

- **PPR (Pulses Per Revolution)**: Indica il numero di impulsi generati dall'encoder per ogni rivoluzione completa dell'albero a cui è collegato. Questo valore determina la risoluzione dell'encoder: un valore più alto di PPR significa una risoluzione maggiore e quindi una misura più precisa della posizione angolare e della velocità;

- **Rapporto di Riduzione (Gearbox Ratio)**: Indica il rapporto di trasmissione tra l'albero del motore e l'albero dell'encoder, ovvero il numero di giri che l'albero del motore deve eseguire per far compiere una rivoluzione completa all'albero dell'encoder. Un valore di 1 significa che non c'è riduzione o moltiplicazione (rapporto 1:1), quindi un giro dell'albero del motore corrisponde esattamente ad un giro dell'albero dell'encoder.
- **Modalità di Conteggio dell'Encoder (Encoder Counting Mode)**: Indica come vengono contati gli impulsi dall'encode: un valore pari a 4, in questo contesto, solitamente si riferisce al conteggio quadraturale (quadrature counting), dove vengono contati tutti i fronti di salita e discesa dei segnali A e B dell'encoder. Questo quadruplica la risoluzione effettiva dell'encoder, perciò un encoder con PPR di 2048 e una modalità di conteggio 4 quadruplicherà il numero di impulsi per giro, risultando in 8192 conteggi per rivoluzione.

3.4.3. Implementazione delle letture tramite l'encoder

Il ciclo di controllo principale viene chiamato, nel caso specifico, ogni 10ms dal TIM1, che ha configurazione analoga ai timer precedenti ma con parametri differenti per ottenere la frequenza desiderata.

In Figura 3.12 è riportato uno schema che rappresenta come viene fatta la lettura dell'encoder, in modo da poter comprendere i passaggi chiave.

Il *valore standard* al quale il contatore viene posto ogni volta è la metà del suo valore massimo, ovvero:

$$\frac{\text{TIM2->ARR}}{2} = \frac{4294967295}{2} = 2147483647 \tag{3.3}$$

Questo serve per evitare che il contatore vada in overflow durante una iterazione del ciclo di controllo, e quindi portare ad una lettura falsata della velocità. E' stato scelto questo valore in particolare visto che il numero di conteggi disponibili dell'encoder è sufficientemente alto per la velocità massima del veicolo e per quella del ciclo di controllo.

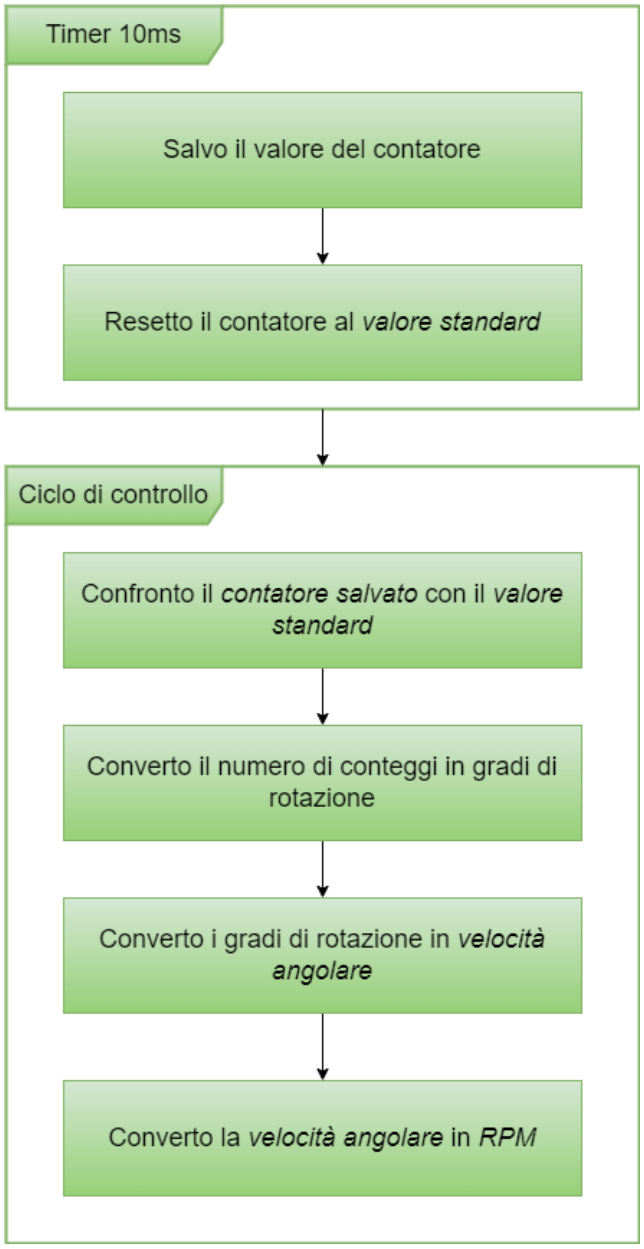


Figura 3.12.: Schema funzionamento encoder

Nel Codice 3.6 è riportato il timer di 10ms, con all'interno solo le componenti che riguardano l'encoder:

```
1    void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
2      if (htim == &htim11) {
3        Flag_10ms = 1;
4
5        vehicleState.counts = TIM2->CNT;
6        TIM2->CNT = TIM2->ARR / 2;
7      }
8    }
```

```

```

Listing 3.6: Timer 10ms

Nel Codice 3.7, invece, è riportata la parte del ciclo di controllo che riguarda la conversione del numero di letture in rotazioni al minuto del motore, salvata nella variabile *tempRPM*

```
        vehicleState.ref_count = TIM2->ARR / 2;
        vehicleState.delta_count = vehicleState.counts - vehicleState.
    ref_count;

        vehicleState.delta_angle_deg = (vehicleState.delta_count *
    360) / ((double)(ENCODER_PPR * ENCODER_COUNTING_MODE *
    GEARBOX_RATIO));
        vehicleState.motor_speed_deg_sec = vehicleState.
    delta_angle_deg / ENCODER_SAMPLING_TIME;
        tempRPM = BL_DegreeSec2RPM(vehicleState.motor_speed_deg_sec);

```

Listing 3.7: Lettura encoder

3.5. Filtraggio delle letture della velocità

Il passaggio necessario per implementare la velocità mediante l'encoder è l'elaborazione dei dati di velocità. In particolare, è necessario utilizzare un filtro per rendere le misurazioni meno affette da rumore.

Il rumore, infatti, è un disturbo presente in questo tipo di encoder e il metodo scelto per rimuoverlo è una media mobile. Questo procedimento consiste nel salvare i valori di velocità letti in un vettore (in questo caso di 10 elementi). Man mano che vengono eseguite delle letture quelle più vecchie vengono sostituite con le nuove, utilizzando un ordinamento di First In First Out.

La velocità effettiva sarà quindi la media di questi 10 valori salvati: in questo modo si evitano picchi inaspettati. Tuttavia, a seguito di questa riduzione di rumore, si assiste ad un ritardo di 100 ms nelle letture, poiché viene effettuata una lettura ogni 10ms e servono 10 letture per riempire il vettore allocato. Tuttavia, considerando la dinamica del veicolo, tale ritardo non porta a nessun comportamento indesiderabile.

Di seguito, nel Codice 3.8 è riportata l'implementazione del filtro per il disturbo a media mobile:

```
        ArrayRPM[PtrRPM] = tempRPM;
        MeanRPM = 0;
        for(int i = 0; i < MAX_RPM_VALUES; i++){
            MeanRPM += ArrayRPM[i];
        }
        MeanRPM /= MAX_RPM_VALUES;

```

```
8           if(PtrRPM == MAX_RPM_VALUES-1)
9               PtrRPM = 0;
10          else
11              PtrRPM++;
12          vehicleState.motor_speed_RPM = MeanRPM;
13
```

Listing 3.8: Filtro a media mobile

Si è visto quindi come si possa implementare un filtro a media mobile così da poter avere delle misurazioni di velocità del motore più precise. Di seguito sono riportati i risultati. In particolare, in rosso è rappresentata la velocità desiderata (in RPM), mentre in blu quella misurata con l'encoder attaccato al motore. La Figura 3.13 riporta le letture eseguite dall'encoder alla velocità di 0.2m/s, mentre in Figura 3.14 è stata impostata una velocità di riferimento di 1.0m/s.

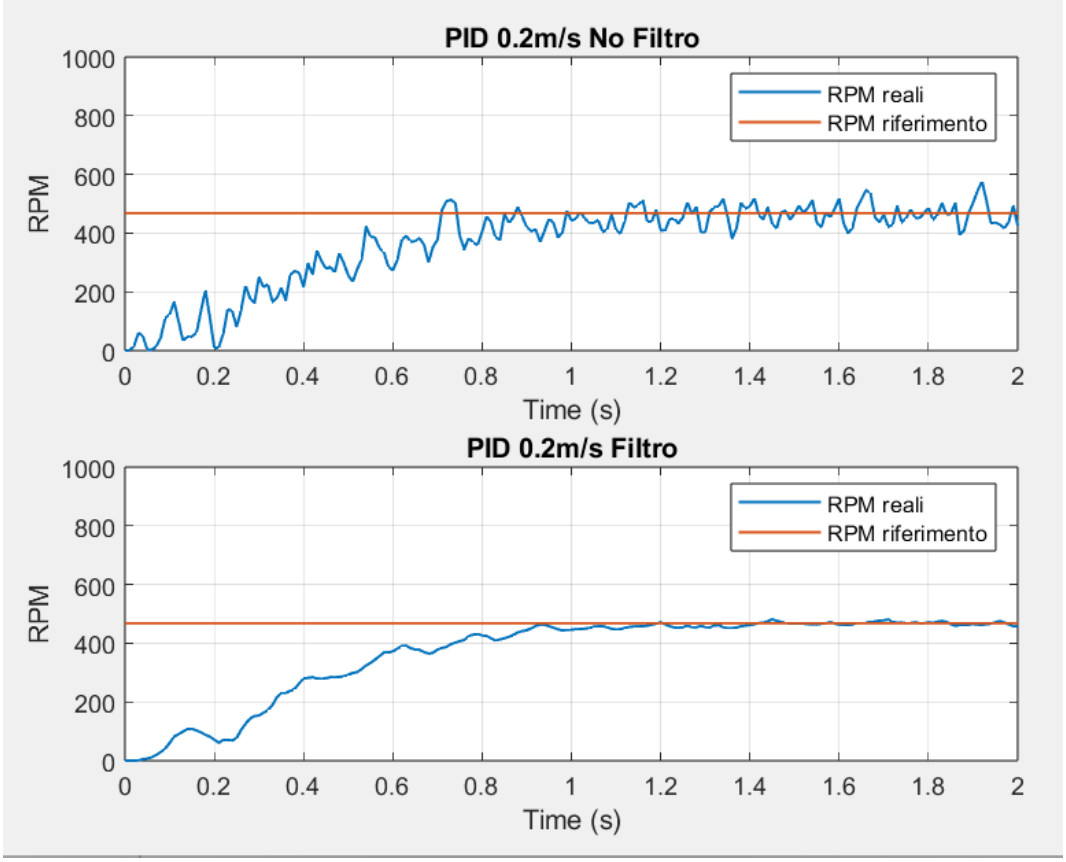


Figura 3.13.: Velocità filtrata 0.2m/s

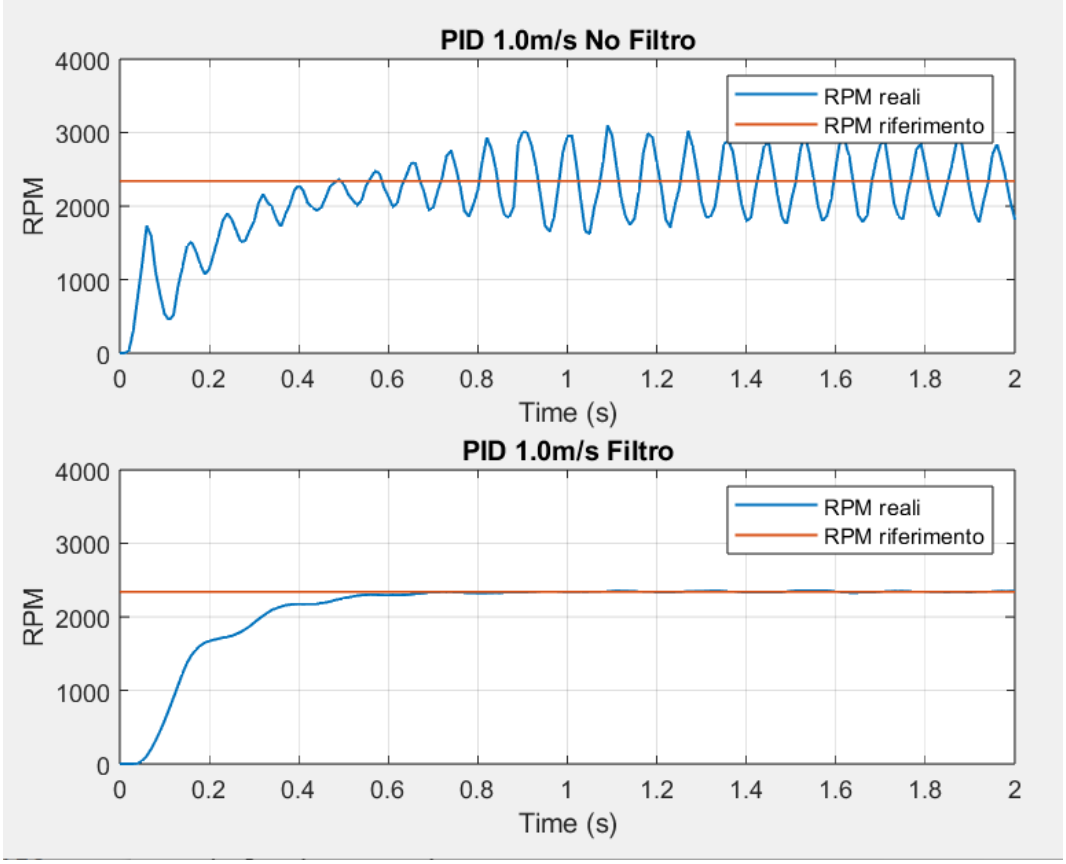


Figura 3.14.: Velocità filtrata 1.0m/s

Osservando le figure riportate si nota il risultato ottenuto dopo il filtraggio: questo ha permesso di eliminare i picchi di errore nella misurazione quasi del tutto, soprattutto alle alte velocità.

Oltre a ciò, tuttavia, si nota il ritardo introdotto poiché, per raggiungere la velocità di riferimento, il motore impiega più tempo.

D'ora in poi ogni volta che verrà riportato un grafico si utilizzerà la velocità filtrata senza comunicarlo esplicitamente, e bisogna anche considerare che i grafici riportati sono stati presi in condizioni ideali, ovvero quando all'albero motore non era attaccato nessun carico. ciò potrebbe comportare una loro variazione durante l'effettivo moto del veicolo.

Capitolo 4.

Trazione

In questo capitolo si analizza come è stato eseguito il controllo del motore di trazione.

4.1. Una prima mappatura

Nel Capitolo 3.3.2 è stato accennato che è fondamentale definire tre percentuali di duty cycle fisse per ottenere la taratura. Queste sono:

- **Duty cycle neutro**: valore per il quale il motore resterà fermo;
- **Duty cycle massimo**: valore per il quale il motore girerà in avanti;
- **Duty cycle minimo**: valore per il quale il motore girerà all'indietro.

Così facendo è possibile definire un range di funzionamento. Ciò è di fondamentale importanza, altrimenti l'ESC non saprebbe come interpretare il segnale che gli viene mandato.

Inizialmente è stata fatta una prova senza effettuare nessuna calibrazione particolare, quindi usando il comportamento standard del motore. Sono stati forniti tutti i valori di duty cycle possibili (da 0% a 100%), ed è stata ottenuta la curva che si può vedere in Figura 4.1.

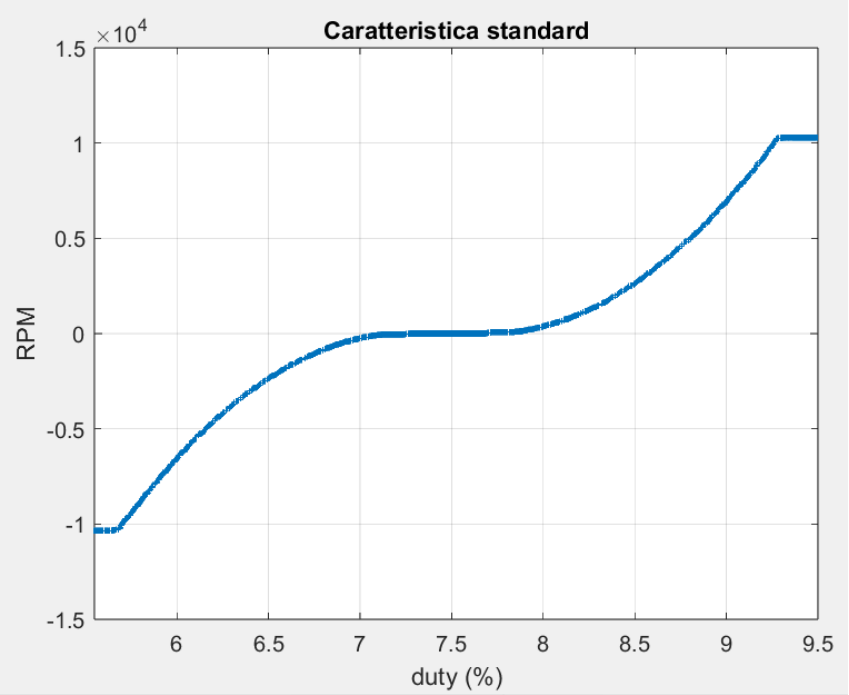


Figura 4.1.: Caratteristica standard del motore

Di conseguenza, sono stati ottenuti i seguenti valori critici:

Duty cycle neutro:	7.5%
Duty cycle massimo:	9.5%
Duty cycle minimo:	5.5%

Tabella 4.1.: Valori critici di duty cycle

Per facilità di lettura, il grafico è stato limitato lungo l'asse x così da rappresentare solo i valori significativi, e ignorando, quindi, i valori di duty cycle minori di 5.5 e maggiori di 9.5 in quanto, in questo range, il motore si trova in uno stato di saturazione.

La Figura 4.1 mostra il tipico comportamento di un motore brushless: questo funziona grazie alla commutazione elettronica che genera un campo magnetico rotante, sincronizzato con i magneti permanenti del rotore. La velocità del motore è proporzionale alla frequenza di commutazione dei segnali applicati agli avvolgimenti dello statore.

Il duty cycle del segnale PWM determina la tensione media applicata al motore. Con un aumento del duty cycle, si ha un incremento della tensione media che alimenta il motore, che a sua volta comporta un aumento della corrente che circola tra gli avvolgimenti. A sua volta, questa corrente genera il campo magnetico necessario per produrre una coppia e quindi movimento.

Per un motore brushless, la velocità di rotazione è all'incirca proporzionale alla tensione applicata. Tuttavia, la tensione effettiva in un circuito PWM è una funzione quadratica del duty cycle. Questo perché la potenza elettrica ($P = V * I$) ha una relazione non lineare con tensione e corrente, in quanto entrambe aumentano con il duty cycle.

Quando il duty cycle aumenta, la tensione media applicata non cresce linearmente, ma in modo quadratico. Di conseguenza, la velocità del motore tende a seguire una curva quadratica in relazione al duty cycle. Questo comportamento è influenzato anche dalle perdite nel sistema, come quelle dovute alla resistenza degli avvolgimenti e alle perdite nel circuito di commutazione, che possono ulteriormente accentuare la natura non lineare della risposta.

4.2. Implicazioni dell'andamento quadratico

L'andamento quadratico della riposta ha una serie di implicazioni, come mostrato in seguito:

- **Controllo di Precisione**: il controllo della velocità richiede una maggiore precisione e linearizzazione del segnale di comando. Sistemi di controllo avanzati come PID possono essere utilizzati per compensare la non linearità e ottenere una risposta più lineare;
- **Efficienza Energetica**: poiché la velocità del motore aumenta rapidamente con il duty cycle, piccoli incrementi di quest'ultimo possono portare a grandi variazioni di velocità. Questo fatto può essere sfruttato per ottimizzare il consumo energetico, ma richiede un attento controllo per evitare che ci sia instabilità;
- **Progettazione del Sistema di Controllo**: la progettazione di algoritmi di controllo deve tenere conto della caratteristica quadratica per garantire una risposta stabile e prevedibile del motore. In molte applicazioni, vengono utilizzati modelli matematici del motore per prevedere e compensare questo comportamento.

Il comportamento quadratico tra il duty cycle e la velocità di un motore brushless è una conseguenza della relazione non lineare tra la tensione media applicata tramite PWM e la velocità del motore. Ciò richiede tecniche di controllo avanzate per garantire prestazioni ottimali, stabilità e precisione nel controllo della velocità, che hanno particolare importanza in applicazioni in cui è richiesta alta precisione e risposta dinamica.

4.3. Miglioriamo la taratura

Un comportamento come quello visto nel Capitolo 4.1 è poco desiderabile in un veicolo che deve essere in grado di spostarsi a velocità basse e controllate. Bisogna perciò, cercare di linearizzare il più possibile la funzione.

Dopo diversi tentativi si è giunti alla conclusione che la migliore caratteristica si ha per:

Duty cycle neutro:	7.5%
Duty cycle massimo:	99%
Duty cycle minimo:	1%

Tabella 4.2.: Valori critici di duty cycle

Si è deciso di utilizzare una configurazione del genere, sensibilmente sbilanciata a favore del moto in avanti, visto che per la maggior parte del tempo sarà la direzione preferenziale della macchina, mentre la retromarcia verrà usata solamente per le manovre di parcheggio.

Analizzando la Figura 4.2 si osserva che durante l'avanzamento la caratteristica è stata notevolmente linearizzata, mentre in retromarcia mostra ancora un andamento fortemente quadratico.

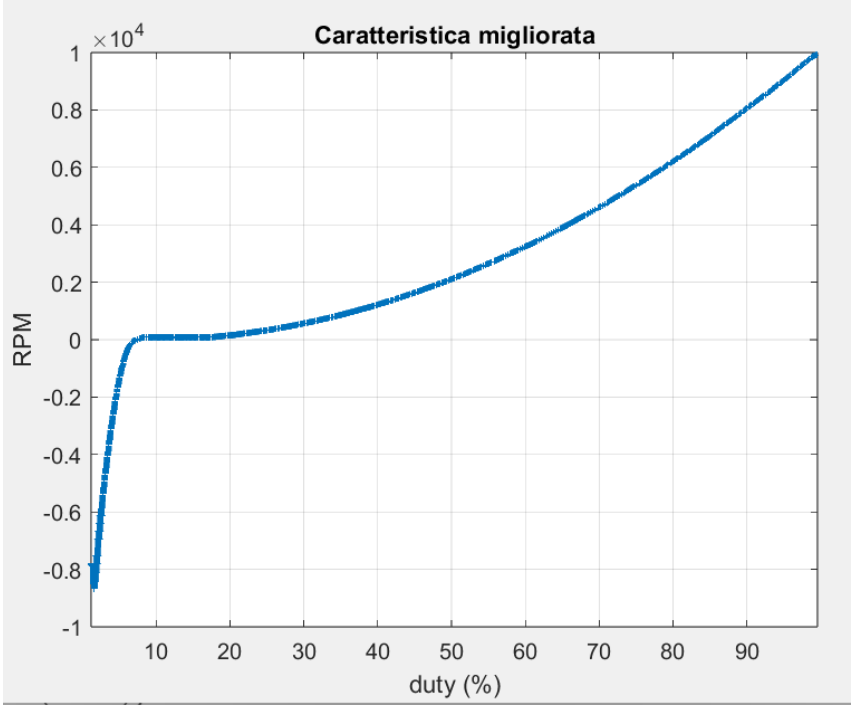


Figura 4.2.: Caratteristica migliorata del motore

4.4. Scheda programmabile

Un nuovo strumento molto utile allo scopo della tesi, è un programmatore per ESC. Si è deciso di scegliere questo poiché era fortemente consigliato durante la fase di acquisto del motore.

La scheda programmabile permette di modificare alcuni comportamenti intrinsechi dell'ESC come la sua risposta in presenza di resistenza esterna, oltre che vari meccanismi di sicurezza, verso di rotazione del motore, potenza di frenata e di retromarcia.

Come si nota dalla Figura 4.3 la connessione avviene direttamente con l'ESC mediante apposito cavo da 3 PIN e presenta un'interfaccia molto semplice. Deve essere collegata quando l'ESC si trova nello stato OFF: deve essere acceso tenendo premuto il tasto di accensione e successivamente appaiono sulla scheda programmabile tutti i parametri che possono essere modificati.

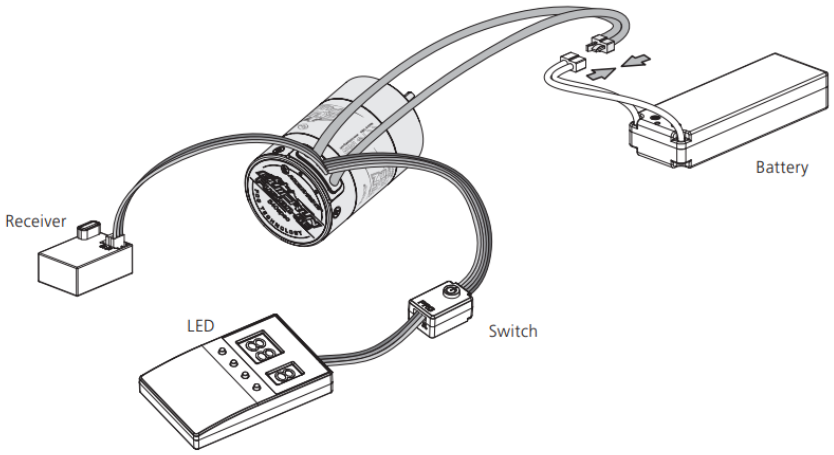


Figura 4.3.: Scheda programmabile

La scheda è dotata di 3 pulsanti:

- **ITEM**: permette di iterare tra i parametri modificabili;
- **VALUE**: permette di iterare tra i valori dei parametri;
- **OK**: permette di confermare la modifica e salvare il parametro selezionato.

In Figura 4.4 viene riportata la tabella che specifica tutti i parametri del motore che possono essere modificati tramite questo strumento. Le opzioni evidenziate sono quelle di default.

No.	Setting item	Option 1	Option 2	Option 3	Option 4	Option 5	Option 6	Option 7	Option 8	Option 9
1	RPM/Throttle Matching	*Enabled*	Disabled							
2	LiPo Cells	*Auto*	2S	3S						
3	Cutoff Voltage	Disabled	Low	*Medium*	High					
4	ESC Thermal Protection	*105℃/221℉*	125℃/257℉							
5	Motor Rotation	*CCW*	CW							
6	BEC Voltage	*6.0V*	7.4V							
7	Drag Brake Force	Disabled	Level 1	Level 2	Level 3	*Level 4*	Level 5	Level 6	Level 7	Level 8
8	Drag Brake Rate	Level 1	Level 2	Level 3	Level 4	*Level 5*	Level 6	Level 7	Level 8	Level 9
9	Max. Reverse Force	25%	50%	75%	*100%*					

Figura 4.4.: Parametri del motore programmabili

La funzionalità che è necessario cambiare è la numero 9 (riportata in Figura 4.4), ovvero quella che indica come il motore si comporta durante il movimento il retromarcia. Analizzando anche il datasheet del motore si nota che, di default, la massima forza rotazionale in retromarcia è impostata al 100%, ma, nel caso studio, è stata impostata al 25%, in modo da rendere la curva più lineare possibile.

4.5. Taratura finale

Mettendo insieme tutte le modifiche che sono state fatte alle proprietà del nostro motore si può ottenere la seguente caratteristica visualizzata in Figura 4.5.

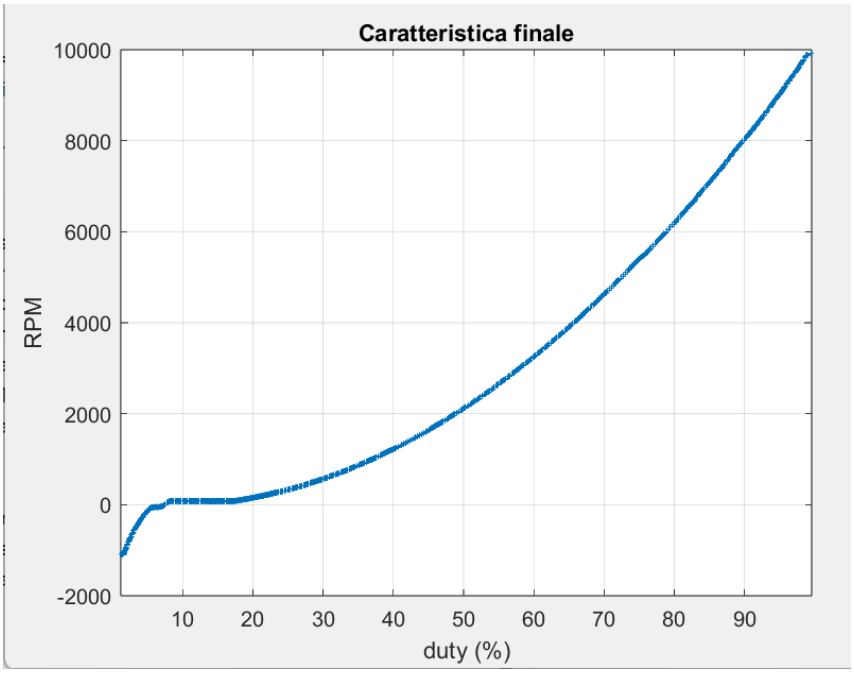


Figura 4.5.: Caratteristica finale del motore

Si nota che, il veicolo in scala, esibisce il comportamento desiderato sia nella marcia normale, avendo velocità accettabili anche con un duty cycle del 50%, sia durante la retromarcia, dove la velocità massima è relativamente bassa e controllabile.

4.6. Riepilogo tarature

Di seguito, nella Figura 4.6 sono state messe a confronto le tre diverse tarature analizzate, in modo da poter osservare quanto il comportamento del motore in relazione al duty cycle sia cambiato.

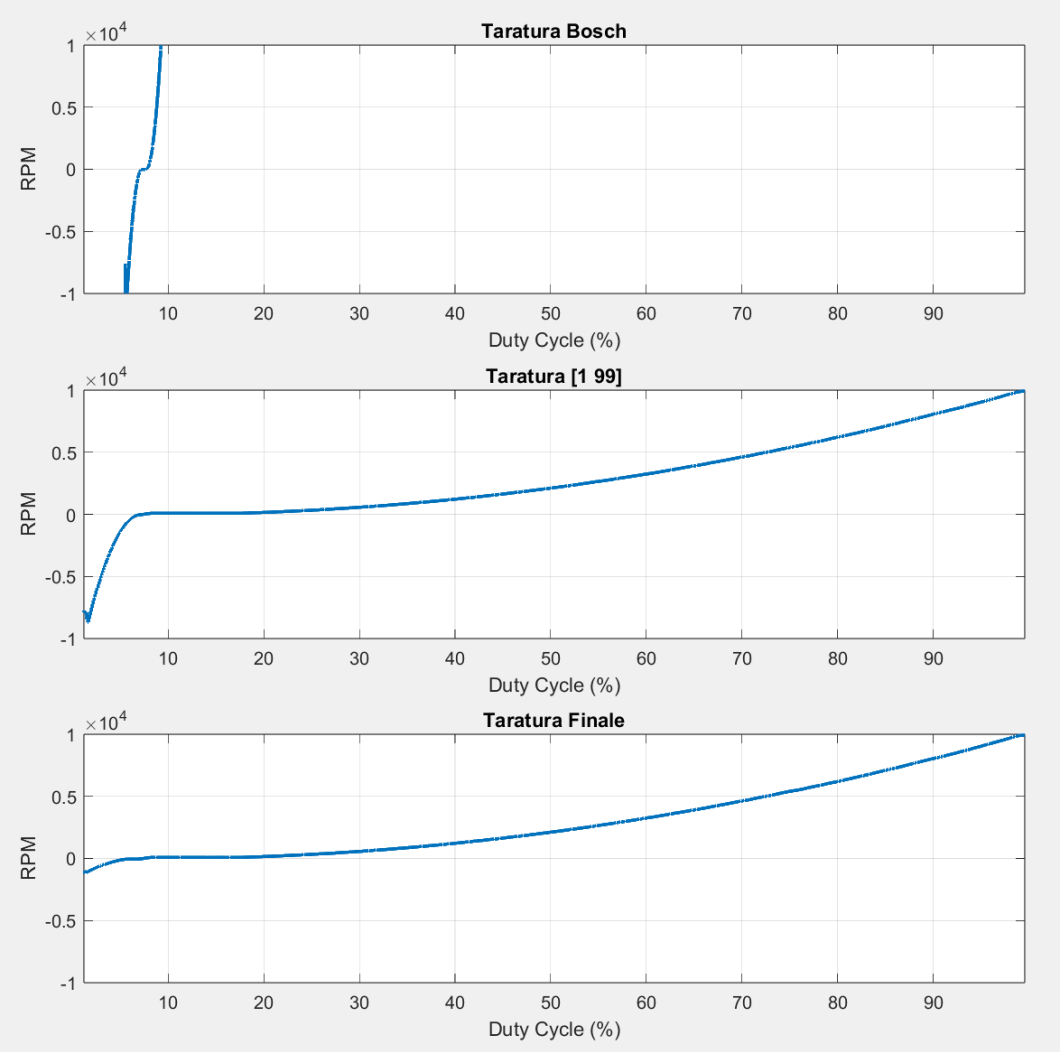


Figura 4.6.: Confronto delle diverse caratteristiche

Si nota come il comportamento sia stato ampiamente linearizzato, permettendo anche un controllo più granulare in quanto il motore reagisce in modo meno brusco alle variazioni del duty cycle.

4.7. Controllore PID

I PID (Proporzionale-Integrativo-Derivativo) sono la tipologia di controllori lineari più utilizzata in ingegneria e automazione per la regolazione dei sistemi dinamici. Grazie alla loro capacità di controllare efficacemente un'ampia gamma di processi,

trovano largo impiego in vari settori industriali per far si che variabili come temperatura, pressione, velocità e posizione mantengano i valori desiderati. I controllori PID calcolano l'uscita del sistema di controllo utilizzando tre componenti principali:

- **Proporzionale (P)**: componente proporzionale all'errore e tra il segnale di riferimento, ovvero il valore desiderato, e il valore misurato dal sistema. Un valore proporzionale più alto comporta una risposta più reattiva alle variazioni dell'errore attuale;
- **Integrativo (I)**: componente proporzionale all'integrale dell'errore e, cioè al suo valor medio. Tiene conto degli errori passati accumulati nel tempo: è richiesto per imporre che l'errore si annulli asintoticamente a fronte di segnali di riferimento o disturbi additivi costanti;
- **Derivativo (D)**: questa componente è proporzionale alla derivata di e. Tiene conto del tasso di cambiamento dell'errore nel tempo e perciò ha il compito di provare ad anticipare l'andamento dell'errore negli istanti futuri, aiutando quindi a prevenire oscillazioni e a migliorare la stabilità del sistema.

La combinazione di questi tre contributi consente al controllore PID di adattarsi dinamicamente alle variazioni del sistema e di mantenere la variabile controllata il più vicino possibile al valore desiderato. I parametri del controllore PID ovvero Kp, Ki e Kd, rispettivamente proporzionale, integrativo e derivativo, devono essere ottimizzati per adattarsi al sistema specifico che si vuole controllare e ciò può richiedere un'analisi dettagliata del sistema e dei test sperimentali per ottenere le migliori prestazioni di controllo. Non tutte e tre le azioni, però, devono essere contemporaneamente presenti: in particolare, è possibile impiegare soltanto una di esse o combinazioni di due. Trascurando i regolatori caratterizzati unicamente dalla presenza dell'azione proporzionale, integrativa o derivativa, dal generico PID si possono ottenere i seguenti casi particolari:

- **Regolatori PI**: si ricavano ponendo Kd = 0 e possono essere visti come reti ritardatrici. Vengono utilizzati quando l'azione integrale è indispensabile per le prestazioni statiche, ma è necessaria anche una banda passante più ampia rispetto a quella ottenibile con un semplice regolatore integrale;
- **Regolatori PD**: Si ottengono ponendo Ki = 0 e si possono considerare come reti anticipatrici. Vengono utilizzati nei casi in cui non vi siano problemi di stabilità o di prestazioni statiche, ma sia invece necessario ottenere la banda passante il più ampia possibile;
- **Regolatori PID**: hanno il contributo di tutte e tre le componenti, Kp, Ki e Kd.

4.8. Banco di pid per movimento in piano

Il controllo del movimento longitudinale viene effettuato mediante due controllori PID, uno per il movimento frontale e uno per la retromarcia. Questa distinzione è stata necessaria a causa della natura diversa tra i due movimenti, evidenziata dalla Figura 4.5, e dalla diversa configurazione dell'ESC mediante la scheda programmabile.

Per questo si è deciso di impiegare un banco di PID, che permetta l'uso di uno piuttosto che dell'altro in base ad una condizione specifica che è il segno della velocità di riferimento. Nella Figura 4.7 è riportato il diagramma di flusso che permette di comprendere facilmente la condizione di switch da un PID all'altro.

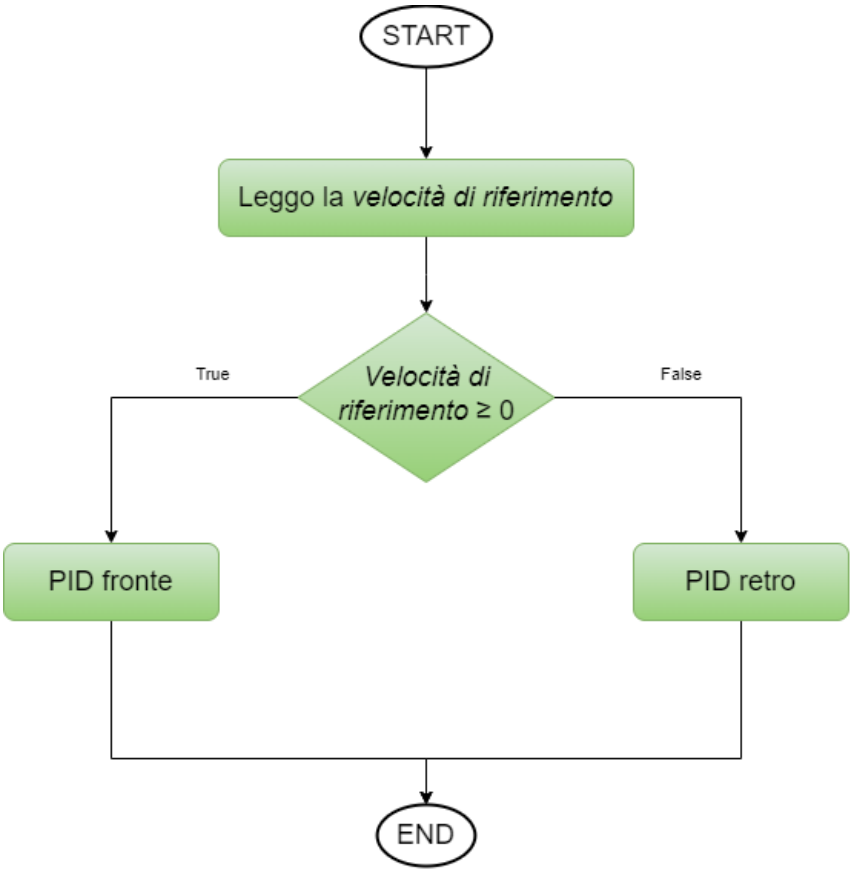


Figura 4.7.: Diagramma di flusso banco di pid in piano

4.9. Implementazione PID

Dopo aver spiegato come funziona un controllore PID nel capitolo precedente, si passa a vedere la sua implementazione in C. Questa volta non sarà necessario effettuare alcuna configurazione nel file *.ioc*, in quanto il PID è puramente un calcolo matematico e non ha bisogno di funzionalità particolari del microcontrollore.

Il codice si divide in tre parti:

1. Header file;
2. Source file;
3. Parte nel file main.

4.9.1. Header file PID

Partendo dalla prima viene illustrata la dichiarazione della struct PID che possiede al suo interno tutti i dati necessari, così come le varie funzioni:

```
1    typedef struct PID{
2      float Kp; //guadagno proporzionale
3      float Ki; //guadagno integrale
4      float Kd; //guadagno derivativo
5      float Kb; //guadagno del back-calculation per l'anti-windup
6
7      float Tc; //periodo
8      float u_max; //limite superiore
9      float u_min; //limite inferiore
10
11     float e_old;
12     float Iterm;
13
14     float offset;
15   }PID;
16
17   void init_PID(PID*, float, float, float, float);
18   void tune_PID(PID*,float,float,float, float);
19   float PID_controller(PID*,float,float);
20   void resetPID(PID*);
21
```

Listing 4.1: Header file PID

Di seguito si riporta l'analisi e la descrizione delle funzionalità di ciascuna delle funzioni:

- **void init_PID(PID*, float, float, float, float)**: serve ad inizializzare un PID con i parametri Tc, u_max, u_min e offset;
- **void tune_PID(PID*,float,float,float, float)**: serve per tarare il pid, modificando i parametri Kp, Ki, Kd e Kb;
- **float PID_controller(PID*,float,float)**: è la funzione che modifica e corregge lo sforzo di controllo effettuato dal PID;
- **void resetPID(PID*)**: è la funzione che permette un reset della memoria del PID.

4.9.2. Source file PID

In questa sezione si esegue un'analisi del Souce file, che include le funzioni precedentemente definite:

```
1   void init_PID(PID* p, float Tc, float u_max, float u_min, float
    offset){
2     p->Tc = Tc;
3     p->u_max = u_max;
4     p->u_min = u_min;
5     p->Iterm = 0;
6     p->e_old = 0;
7     p->offset = offset;
8   }
9
10  void tune_PID(PID*p, float Kp, float Ki, float Kd, float Kb){
11    p->Kp = Kp;
12    p->Ki = Ki;
13    p->Kd = Kd;
14    p->Kb = Kb;
15  }
16
17  void resetPID(PID* p){
18    p->Iterm = 0;
19    p->e_old = 0;
20  }
21
22  float PID_controller(PID* p , float y, float r){
23    float u;
24    float newIterm;
25    float e = 0;
26
27    e = r-y;
28
29    if (isinf(p->Iterm) || isnan(p->Iterm)) {
30      p->Iterm = 0;
31      p->e_old = 0;
32    }
33
34    float Pterm = p->Kp*e;
35    newIterm = p->Iterm + (p->Ki)*p->Tc*p->e_old;
36    float Dterm = (p->Kd/p->Tc)*(e - p->e_old);
37      u = Pterm + newIterm + Dterm + p->offset;
38
39    p->e_old = e;
40
41    // saturazione con back-calculation
42    float saturated_u = u;
43    if(saturated_u > p->u_max){
44      saturated_u = p->u_max;
45    }
46    else if(saturated_u < p->u_min){
```

```
47          saturated_u = p->u_min;
48        }
49        float correction = p->Kb * (saturated_u - u) * p->Ki * p->Tc;
50        p->Iterm = newIterm + correction;
51
52        u = saturated_u;
53        return u;
54      }
55
```

Listing 4.2: Source file PID

E' bene focalizzare l'attenzione alle righe 42-50 del Codice 4.2, in quanto queste servono come anti-windup in caso di saturazione dell'attuatore. Per quanto riguarda i PID di trazione non vengono usati questi comandi in quanto il motore non raggiunge mai la saturazione, quindi verranno analizzate in modo approfondito nel Capitolo 6.5.

4.9.3. Main file PID

Infine, viene mostrato come il PID venga gestito all'interno del file main. Inizialmente è necessario inizializzare il PID e tararlo mediante la chiamata delle seguenti funzioni:

```
1     //PID traction FWD
2   init_PID(&pid_traction, TRACTION_SAMPLING_TIME, MAX_U_TRACTION,
      MIN_U_TRACTION, NEUTRAL_PWM);
3   tune_PID(&pid_traction, KP_TRACTION, KI_TRACTION, KD_TRACTION, -1);
4
5   //PID traction RWD
6   init_PID(&pid_traction_RWD, TRACTION_SAMPLING_TIME, MAX_U_TRACTION,
      MIN_U_TRACTION, NEUTRAL_PWM);
7   tune_PID(&pid_traction_RWD, KP_TRACTION_RWD, KI_TRACTION_RWD,
      KD_TRACTION_RWD, -1);
8
```

Listing 4.3: Main file PID: inizializzazione

Successivamente, all'interno del ciclo di controllo, subito dopo aver effettuato la lettura della velocità utilizzando l'encoder (come spiegato nel Capitolo 3.4.3), verrà eseguito il Codice 4.4. Così facendo, verrà calcolato il duty cycle da assegnare al motore per avvicinarsi sempre di più alla velocità di riferimento *vehicleState.motor_speed_ref_RPM*, oltre alla condizione dell'if, che permette di scegliere se utilizzare il PID per la marcia frontale o per la retromarcia.

```
1     if(vehicleState.motor_speed_ref_RPM >= 0){
2         u_trazione = PID_controller(&pid_traction, vehicleState.
    motor_speed_RPM, vehicleState.motor_speed_ref_RPM);
3     } else{
4         u_trazione = PID_controller(&pid_traction_RWD, vehicleState.
    motor_speed_RPM, vehicleState.motor_speed_ref_RPM);
```

```
5       }
6
7       //Assegno il duty al motore
8       if (vehicleState.motor_speed_ref_RPM == 0)
9           BL_set_PWM(NEUTRAL_PWM);
10      else
11          BL_set_PWM(u_trazione);
12
```

Listing 4.4: Main file PID: ciclo di controllo

4.10. Analisi della taratura

Durante la fase di taratura del controllore PID per la regolazione della velocità longitudinale del veicolo, è fondamentale considerare diversi aspetti affinchè sia garantita una risposta del sistema stabile e precisa. Innanzitutto, è necessario iniziare con la determinazione dei parametri proporzionale (Kp), integrale (Ki) e derivativo (Kd) che influenzeranno la reattività e la stabilità del controllo.

In particolare, un valore troppo alto di Kp può portare a oscillazioni e instabilità, mentre un valore troppo basso può condurre ad una risposta lenta e inefficace. L'integrazione di Ki permette di eliminare l'errore a regime: tuttavia, valori eccessivi possono introdurre ritardi e instabilità. Inoltre, il termine derivativo Kd contribuisce a smorzare le oscillazioni, ma un valore troppo elevato può amplificare il rumore del segnale.

Durante la taratura, è consigliabile adottare un approccio iterativo, partendo da valori bassi e incrementando gradualmente i parametri ed osservando attentamente la risposta del sistema ad ogni modifica. Due aspetti particolarmente importanti da monitorare sono il tempo di salita e il comportamento a regime. In particolare, il tempo di salita deve essere sufficientemente rapido per garantire una risposta pronta del veicolo ai comandi, ma allo stesso tempo non deve introdurre oscillazioni significative. Il comportamento a regime, invece, deve assicurare che la velocità desiderata venga mantenuta con precisione, minimizzando l'errore stazionario e garantendo stabilità.

Infine, è essenziale testare il sistema in diverse condizioni operative per garantire la robustezza del controllo PID in tutte le situazioni di guida, assicurando così prestazioni ottimali sia in condizioni normali che in presenza di perturbazioni.

4.10.1. Moto frontale

Il PID che regola il movimento frontale è stato tarato con i seguenti parametri:

Kp:	0.0001
Ki:	0.001
Kd:	0.000001

Tabella 4.3.: Taratura PID marcia avanti

Come si può notare, dopo un'estensiva taratura sperimentale, si è deciso di attribuire loro valori molto bassi. L'azione integrale è più alta delle altre in quanto il motore funzionerà quasi sempre a regime: pertanto, è di fondamentale importanza annullare le oscillazioni in questa condizione. Successivamente, è stata inclusa anche la componente derivativa per evitare che durante la fase di salita la risposta avesse una sovraelongazione eccessiva, che avrebbe potuto causare un comportamento indesiderato nel motore.

In Figura 4.8 è riportato il comportamento che il controllore ha quando gli viene fornita una velocità di riferimento prima di 0.2m/s e successivamente di 0.5m/s.

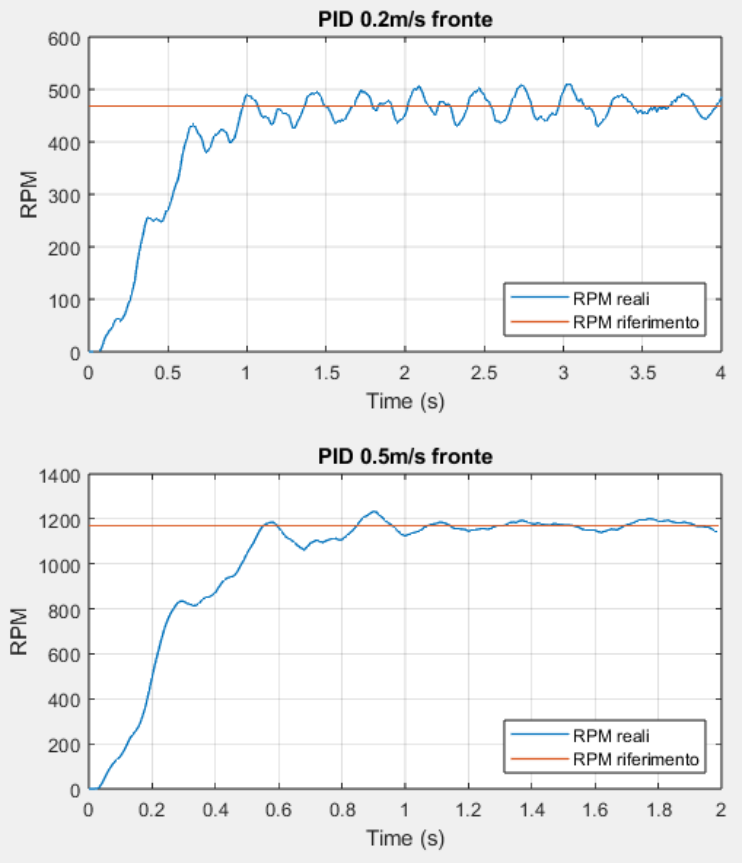


Figura 4.8.: Risposta controllore PID fronte

4.10.2. Retromarcia

Il PID che regola il movimento posteriore è stato tarato con i seguenti parametri:

Kp:	0.000025
Ki:	0.00025
Kd:	0.00000025

Tabella 4.4.: Taratura PID marcia indietro

Si può notare che i parametri sono gli stessi del movimento frontale ma moltiplicati per un fattore di $0,25$ poichè, come spiegato nel Capitolo 4.4, la potenza massima erogabile nella retromarcia è stata impostata al 25% rispetto a quella frontale.

In Figura 4.9 è riportato il comportamento del controllore quando gli viene fornita una velocità di riferimento prima a -0.2m/s e successivamente a -0.5m/s.

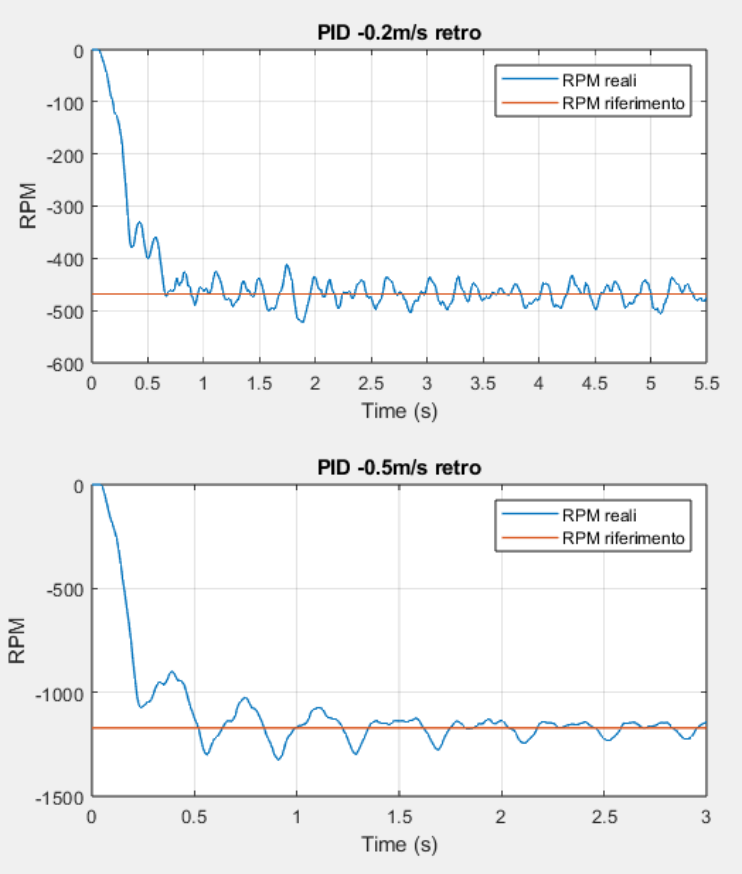


Figura 4.9.: Risposta controllore PID retro

Capitolo 5.

Trazione in pendenza

In questo capitolo verrà trattato il problema della trazione in situazioni di pendenza, quindi sia in salita che in discesa.

E' stato necessario affrontare tale questione visto che nel percorso della Bosch Future Mobility Challenge era presente una rampa, e il veicolo doveva essere in grado di percorrerla senza problemi, ma soprattutto garantendo una guida confortevole. E' stato necessario quindi assicurarsi che il PID fosse in grado di aumentare lo sforzo di controllo in modo lineare e progressivo, evitando brusche accelerazioni o oscillazioni.

5.1. Potenziali problematiche

Si passa ora ad analizzare le possibili problematiche che possono verificarsi durante l'attraversamento di questo ostacolo.

5.1.1. Variazione del carico

In salita, il motore deve fornire più coppia per superare la forza di gravità, mentre in discesa può essere necessario frenare per evitare l'accelerazione eccessiva.

Soluzione: la taratura del PID deve considerare le variazioni di carico. Un guadagno proporzionale Kp elevato può aiutare a rispondere rapidamente ai cambiamenti di pendenza, mentre il guadagno integrale Ki può essere regolato per eliminare l'errore a lungo termine.

5.1.2. Instabilità e Oscillazioni

Eseguire una rampa può introdurre oscillazioni nella velocità se il PID non è tarato correttamente. Questo è particolarmente problematico in salita, dove un controllo inadeguato può causare vibrazioni o addirittura stalli.

Soluzione: l'incremento del guadagno derivativo Kd può aiutare a smorzare le oscillazioni. Tuttavia, è importante bilanciare questo indicatore con Kp e Ki per evitare l'amplificazione del rumore.

5.1.3. Errore a Regime

Mantenere una velocità costante è cruciale per il comfort e la sicurezza del veicolo. L'errore a regime deve essere minimizzato per evitare variazioni di velocità indesiderate.

Soluzione: l'integrazione Ki deve essere tarata per correggere l'errore a regime senza introdurre eccessivi ritardi o oscillazioni. Inoltre, è utile testare il sistema su diverse pendenze per ottimizzare i parametri.

5.2. Struttura della rampa

Prima di proseguire con la descrizione delle soluzioni adottate per affrontare questo ostacolo presente sul percorso di gara, è bene analizzare la struttura della rampa. Innanzitutto, questa era ricoperta dello stesso materiale del resto del percorso, e possedeva le strisce laterali che delimitavano la carreggiata. Per ciò, l'unico problema su cui è stato necessario concentrarsi è stata la regolazione dello sforzo di controllo, ignorando altri eventuali disturbi.

In Figura 5.1 sono riportate le misure della rampa in metri.

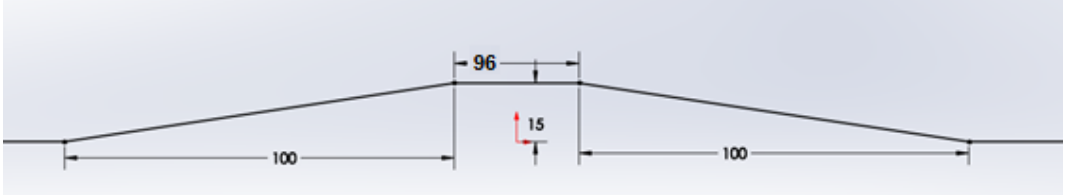


Figura 5.1.: Misure della rampa

5.3. Banco di PID

Si è quindi scelto di adottare un banco di controllori PID, in modo da poter tarare indipendentemente la parte di salita e quella di discesa.

In questo modo si ha la possibilità di regolare il comportamento in base alla correzione desiderata, in modo da avere, come già visto nel Capitolo 5.1, un comportamento più reattivo nella fase iniziale di salita e, al contrario, un comportamento meno reattivo nella fase finale di discesa evitando così oscillazioni e garantire una corretta azione frenante.

Aggiungendo altri due controllori PID si arriva ad un totale di quattro. Nella Figura 5.2 è riportato il diagramma di flusso che esprime la priorità di valutazione della condizione di un determinato PID.

Visto che le condizioni di pendenza sono l'eccezione alla regola, si è scelto di verificarle prima, per poi passare alla trazione in piano una volta rilevata assenza di pendenza.

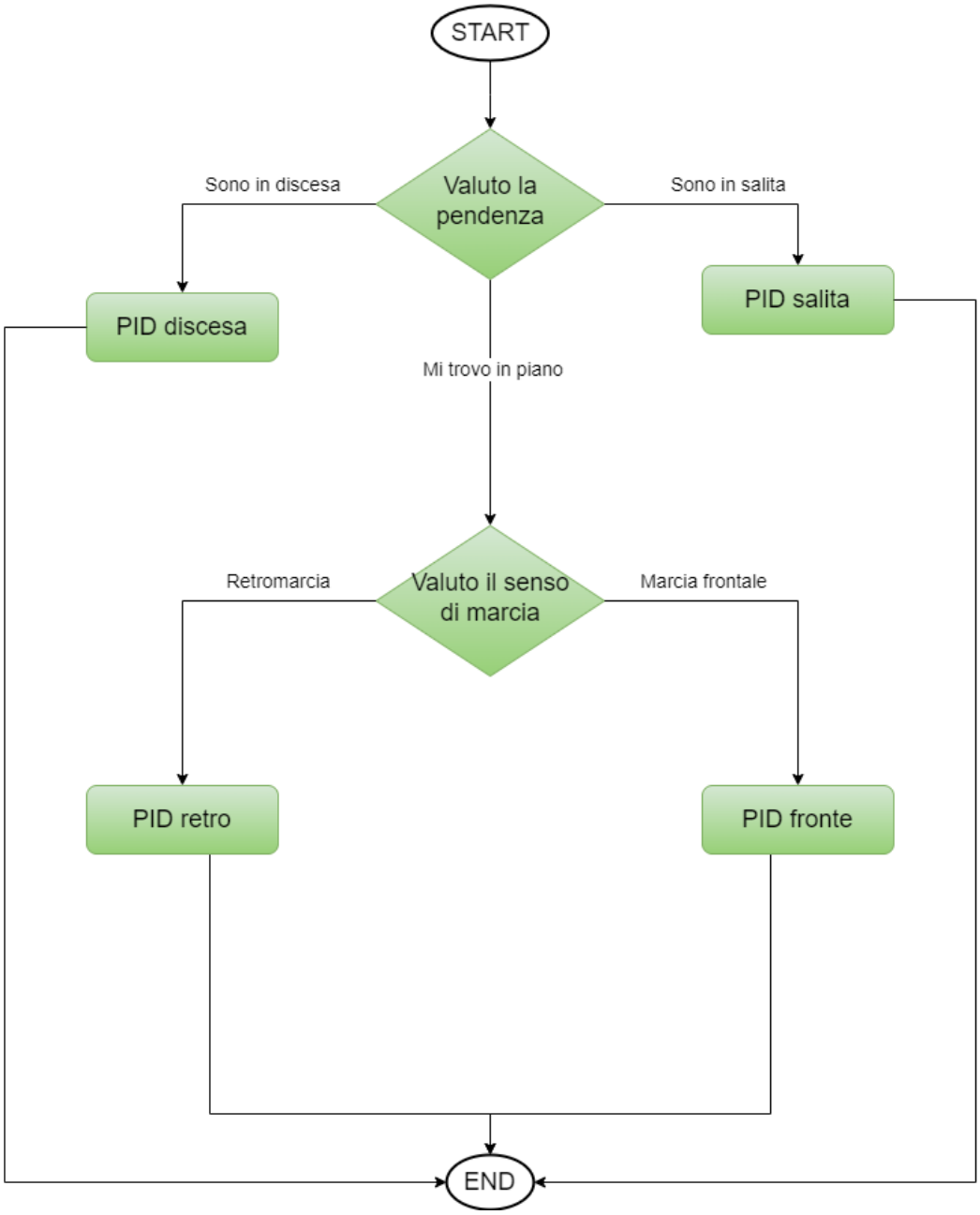


Figura 5.2.: Banco di PID completo

Detto ciò non resta altro che trattare la condizione di swtich da un PID all'altro.

5.4. Condizione di switch in pendenza

In questo capitolo verrà affrontato uno dei problemi principali che può verificarsi durante l'attraversamento della rampa, ovvero la percezione della condizione di pendenza.

Nel Capitolo 2.7 è stato descritto il funzionamento del modulo IMU presente sulla macchina, oltre ai dati che esso riesce a misurare. Al fine di rilevare la condizione di pendenza sono stati presi in considerazione due differenti casi:

1. Accelerazione gravitazionale lungo l'asse X;
2. Valore del pitch della macchina.

5.4.1. Soluzione 1: Accelerazione gravitazionale lungo l'asse X

L'IMU è in grado di restituire le componenti dell'accelerazione gravitazionale presente su tutti e tre gli assi del suo sistema di riferimento, che d'ora in poi sarà considerato solidale a quello del veicolo, in quanto l'IMU viene saldamente fissata ad esso. In Figura 5.3 vengono definite le componenti dell'accelerazione gravitazionale relative alla macchina.

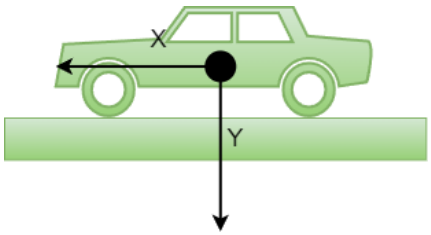


Figura 5.3.: Accelerazioni in piano

A questo punto, è possibile osservare come variano le accelerazioni quando la macchina si trova a dover compiere la salita, come riportato in Figura 5.4.

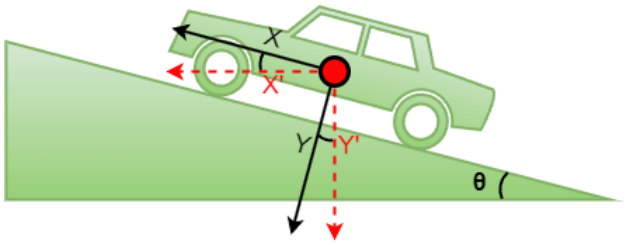


Figura 5.4.: Accelerazioni sulla rampa

E' possibile notare che la pendenza inserita nel sistema suddivide l'accelerazione gravitazionale, che prima era solo lungo l'asse Y, anche sull'asse X. Definendo la pendenza della salita pari a θ, è possibile calcolare l'accelerazione gravitazionale lungo l'asse X della macchina come riportato di seguito:

$$a_x = g \cdot cos(\theta) = 9.8m/s^2 \cdot cos(15°) = 2.5m/s^2 \tag{5.1}$$

Sostituendo i valori del caso specifico, è stata ottenuta un'accelerazione di $2.5m/s^2$. Tuttavia, per sicurezza, si è deciso di scegliere una soglia di $2m/s^2$.

Nella Figura 5.5 è riportato un grafico rappresentate l'accelerazione lungo l'asse x. Il grafico comprende tutta la rampa: in particolare è così suddiviso:

- **Regione 1**: la macchina non ha ancora intrapreso la rampa, e si trova in piano;
- **Regione 2**: la macchina si trova nella salita, dopo essersi stabilizzata, quindi è completamente inclinata;
- **Regione 3**: la macchina si trova in piano, però in cima alla rampa;
- **Regione 4**: la macchina si trova nella discesa, dopo essersi stabilizzata, quindi è completamente inclinata;
- **Regione 5**: la macchina ha terminato la rampa ed è tornata in piano.

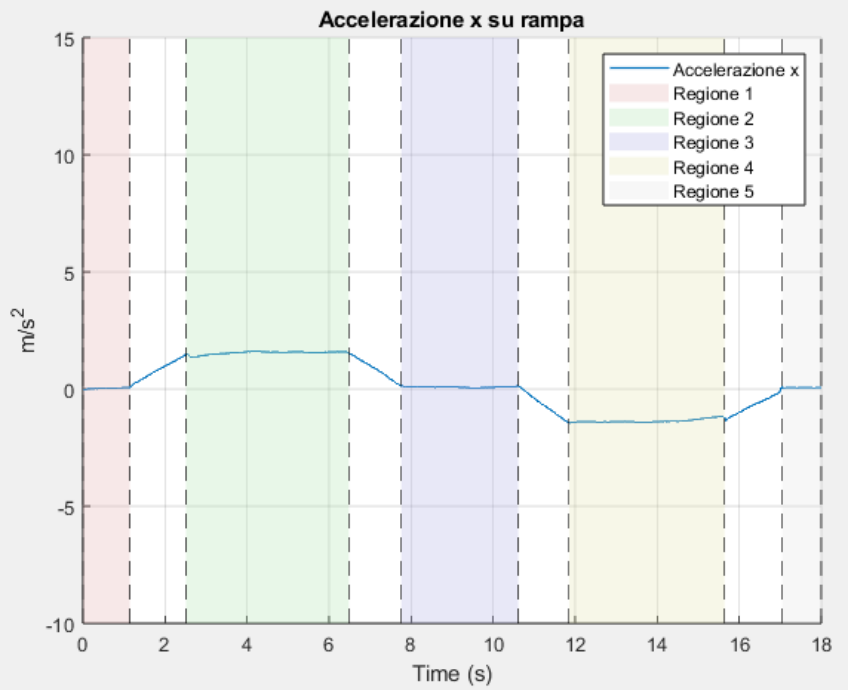


Figura 5.5.: Accelerazione asse x lungo la rampa

Dal grafico appena visionato si può, quindi, ottenere la posizione del veicolo in relazione alla rampa. E' da notare che le zone non evidenziate sono delle zone di transizione, ovvero quando la macchina non ha assunto la totale inclinazione della rampa ma, ad esempio, l'asse anteriore si trova sulla rampa mentre l'asse posteriore si trova ancora nella parte di piano antecedente.

A questo punto, per implementarne la logica non bisognerà far altro che considerare il segno di questo valore come riportato nel diagramma di flusso rappresentato in Figura 5.6.

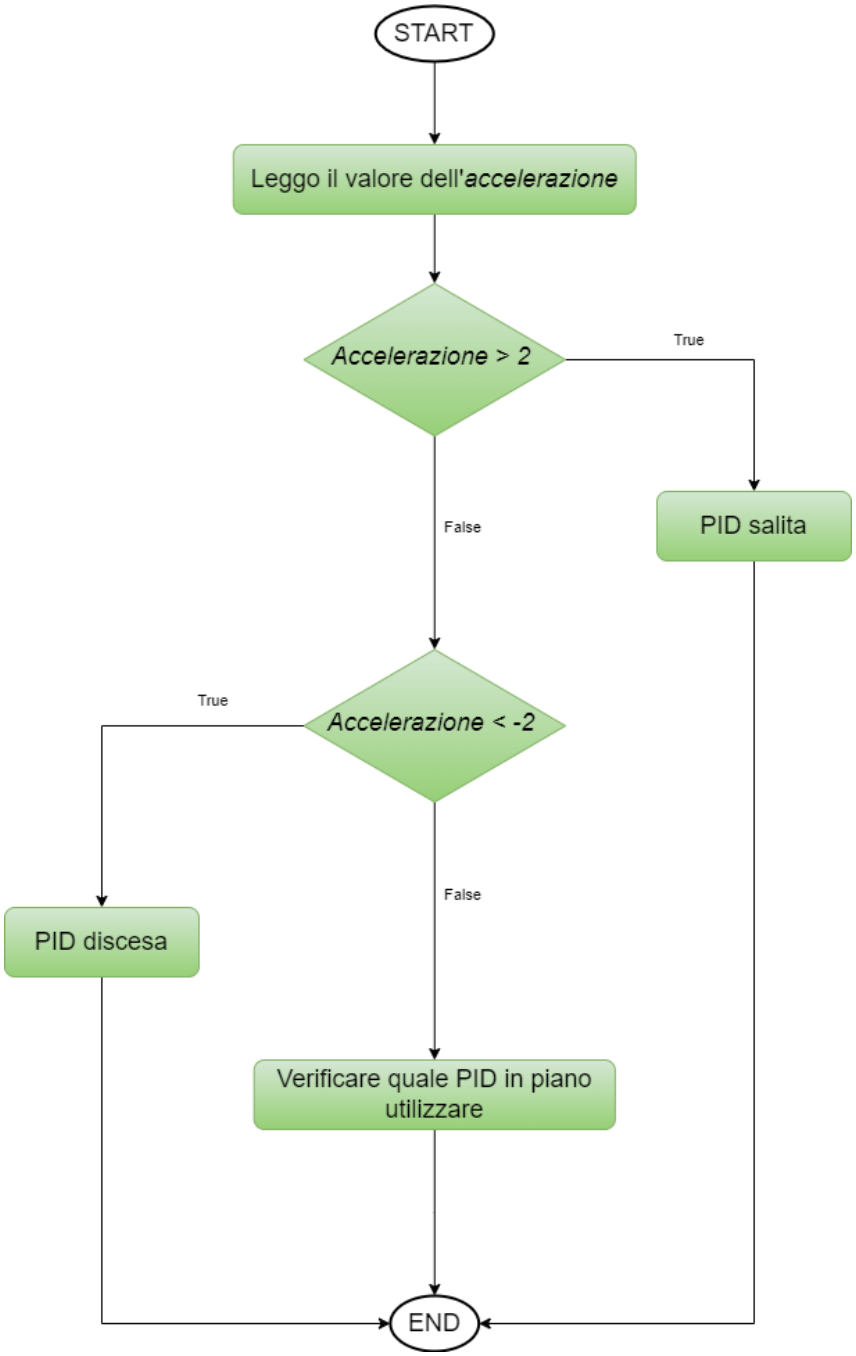


Figura 5.6.: Diagramma di flusso sull'accelerazione

Infine, vengono riportate le righe di Codice 5.1 che hanno permesso di effettuare lo switch tra i due PID. Non è, invece, presente la verifica in piano, poiché è successiva a quanto sotto riportato.

```
1         bno055_vector_t a = bno055_getVectorGravity();
2         float ax = a.x;
3
4         if(ax > 2){
```

```
5             u_trazione = PID_controller(&pid_traction_ASC,
    vehicleState.motor_speed_RPM, vehicleState.motor_speed_ref_RPM);
6         }
7         else if (ax < -2){
8             u_trazione = PID_controller(&pid_traction_DESC,
    vehicleState.motor_speed_RPM, vehicleState.motor_speed_ref_RPM);
9         }
10
```

Listing 5.1: Codice per banco di PID in salita con l'accelerazione

E' bene evidenziare che, nel risultato finale, questa soluzione è stata scartata in quanto non completamente affidabile. La scelta di non includerla non è dovuta al fatto di non aver restituito risultati soddisfacenti, bensì perché poteva non garantire appieno la stabilità. Infatti, può verificarsi che l'accelerazione sul solo asse x possa essere sbagliata a causa di disturbi esterni: ciò potrebbe portare la macchina ad effettuare uno scatto e quindi ad avere un'accelerazione sull'asse longitudinale errata. Inoltre, una situazione del genere porterebbe ad una scelta errata del PID da utilizzare, oltre che ad una risposta instabile.

5.4.2. Soluzione 2: Angolo di Pitch

La seconda soluzione presa in considerazione per rilevare la presenza di una rampa è stata quella di valutare il pitch, ovvero l'angolo di inclinazione longitudinale di un veicolo o di un oggetto in generale, rispetto all'orizzonte. In termini automobilistici e aeronautici, il pitch rappresenta l'inclinazione del veicolo lungo l'asse trasversale (da sinistra a destra) che fa variare l'altezza della sua estremità anteriore rispetto a quella posteriore.

Il pitch viene calcolato utilizzando sensori di inclinazione come accelerometri e giroscopi o, come in questo lavoro di tesi, tramite il sensore IMU. L'accelerometro utilizza sia la componente dell'accelerazione lungo l'asse x (traversale) che z (verticale): note entrambe le componenti, l'angolo di pitch (θ) può essere calcolato mediante la seguente formula:

$$\theta = \arctan\left(\frac{a_x}{a_z}\right) \tag{5.2}$$

in cui a_x e a_z sono le accelerazioni rispettivamente lungo l'asse x e z.

Detto ciò, si può facilmente notare come l'angolo di pitch sia un indicatore più affidabile rispetto alla semplice accelerazione lungo l'asse x, in quanto utilizza come parametro anche l'accelerazione lungo l'asse z, rendendo la misura dell'angolo più robusta e meno soggetta ad errori dovuti a possibili disturbi esterni.

Nella Figura 5.5 è riportato un grafico rappresentate l'accelerazione lungo l'asse x. Il grafico comprende tutta la rampa e, in particolare, si ha che la:

- **Regione 1**: la macchina non ha ancora intrapreso la rampa, e si trova in piano;

- **Regione 2**: la macchina si trova nella salita, dopo essersi stabilizzata, quindi è completamente inclinata;
- **Regione 3**: la macchina si trova in piano, però in cima alla rampa;
- **Regione 4**: la macchina si trova nella discesa, dopo essersi stabilizzata, quindi è completamente inclinata;
- **Regione 5**: la macchina ha terminato la rampa ed è tornata in piano.

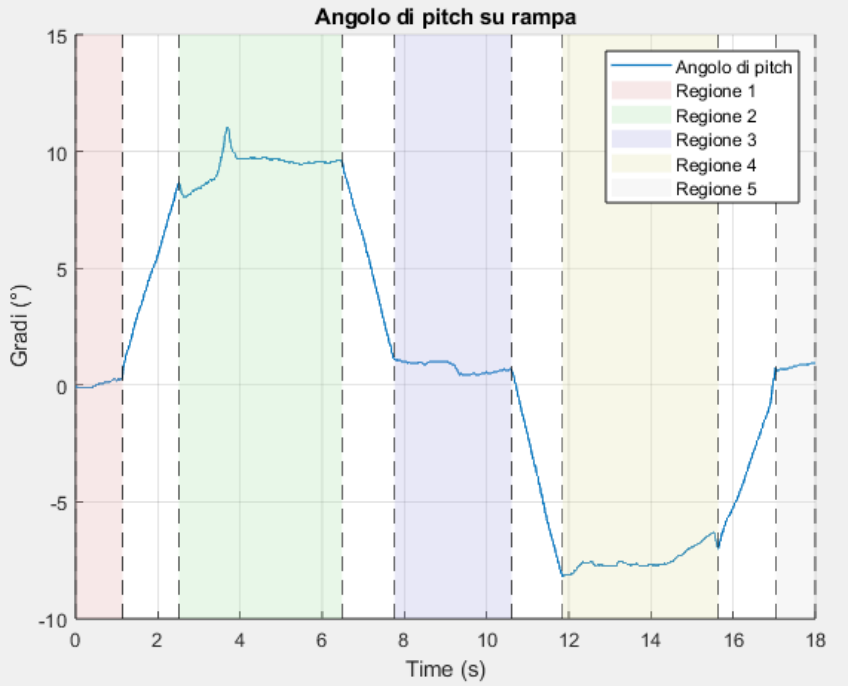


Figura 5.7.: Angolo di pitch lungo la rampa

Da un'analisi del grafico si nota come l'angolo di pitch abbia una finestra maggiore di variazione, il che permette a disturbi esterni di avere meno rilevanza.

Di seguito in Figura 5.8 viene riportato il diagramma di flusso per la scelta del PID da usare mediante il pitch.

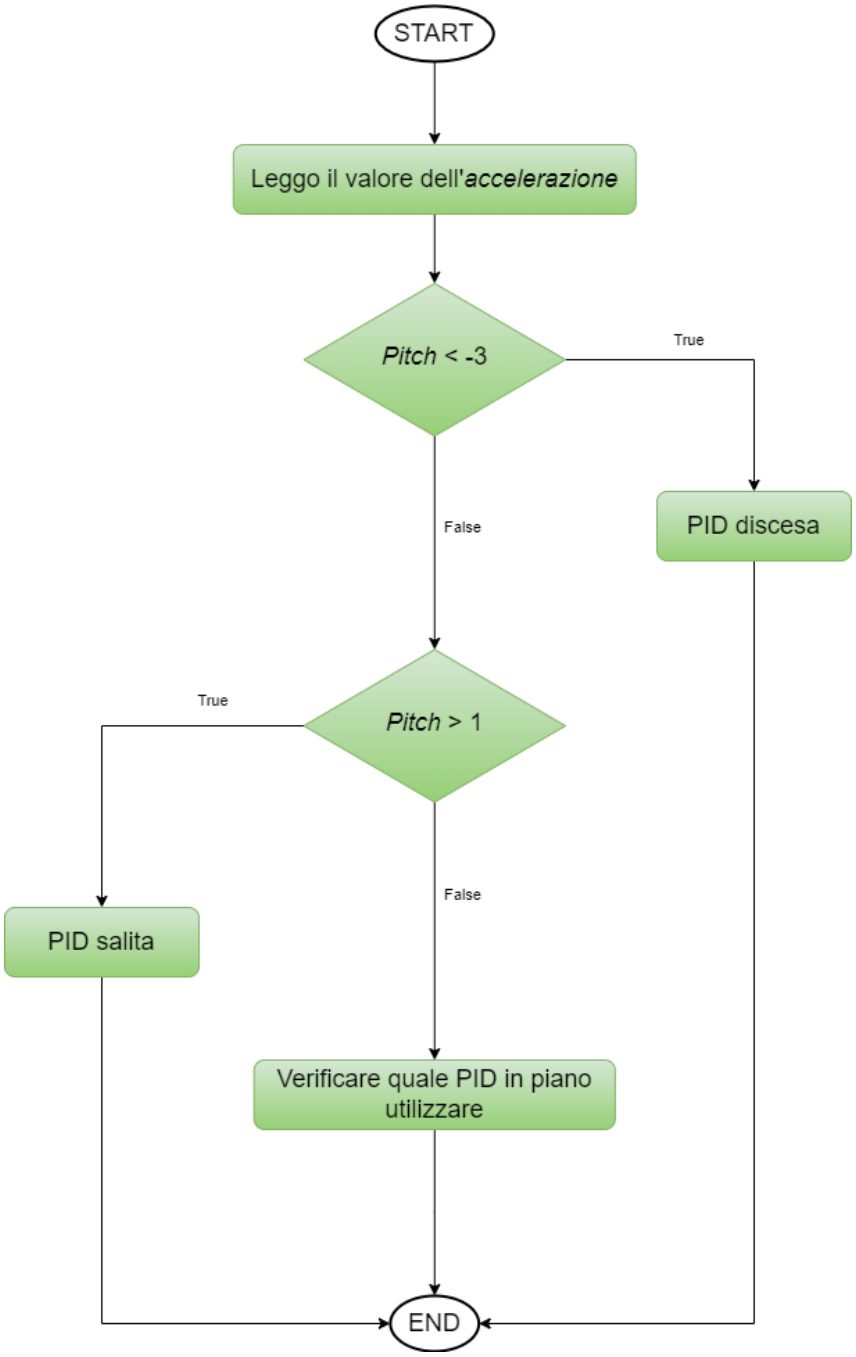


Figura 5.8.: Diagramma di flusso sull'angolo di pitch

Nel Codice 5.2 viene riportata l'implementazione per la condizione di switch tra un PID e l'altro in pendenza mediante l'angolo di pitch.

```
1         bno055_vector_t p = bno055_getVectorEuler();
2         float pitch = p.y;
3
4         if(pitch < 3){
5             u_trazione = PID_controller(&pid_traction_DESC,
    vehicleState.motor_speed_RPM, vehicleState.motor_speed_ref_RPM);
```

```
6           }
7           else if (pitch > 1){
8               u_trazione = PID_controller(&pid_traction_ASC,
    vehicleState.motor_speed_RPM, vehicleState.motor_speed_ref_RPM);
9           }
10
```

Listing 5.2: Codice per banco di PID in salita con il pitch

Quella riportata è stata utilizzata come soluzione finale, in quanto ritenuta sufficientemente robusta per il caso studio.

5.5. Taratura dei PID

Dopo questa panoramica generale, si passa all'analisi della taratura scelta sia per il PID in salita che per quello in discesa.

In Figura 5.9 viene riportata la risposta del controllore alla variazione della pendenza, mantenendo la codifica in colori per far riferimento alla posizione sulla rampa come descritto nel Capitolo 5.4.1.

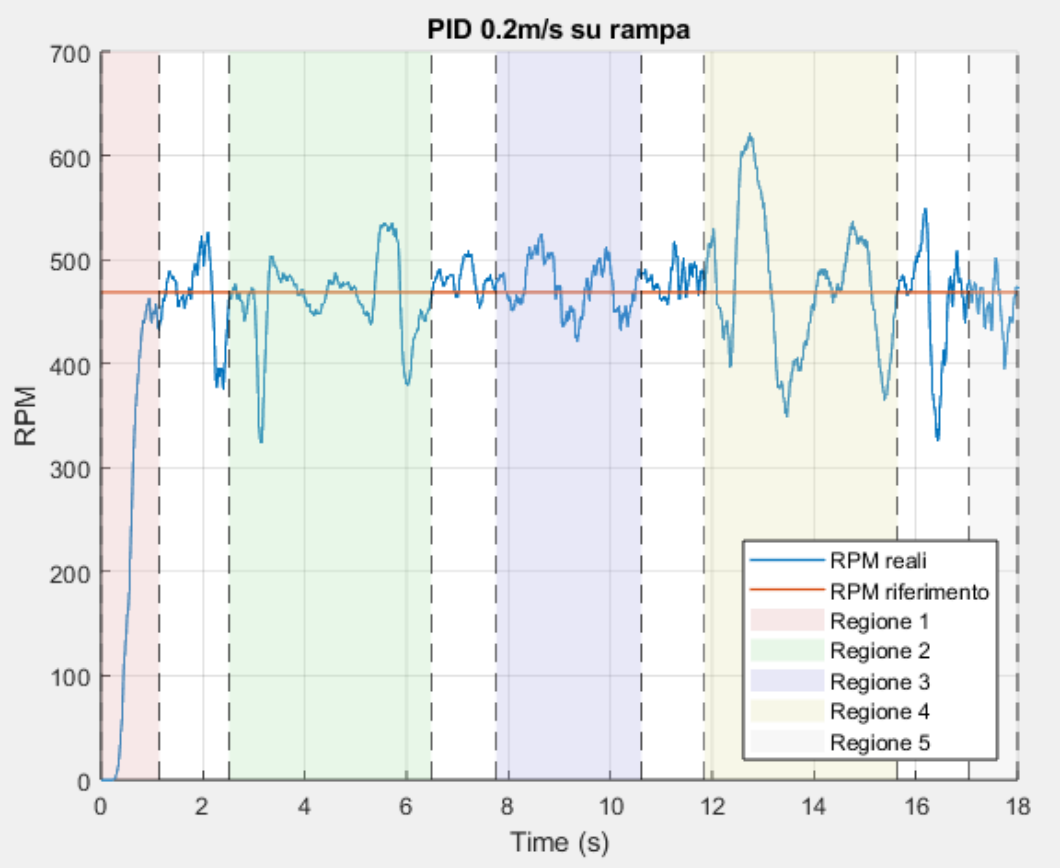


Figura 5.9.: Risposta del PID sulla rampa

Si nota che sono presenti dei picchi che si discostano di un valore considerevole in alcuni punti. Uno di questi è presente dopo 3 secondi, quando la macchina si trova completamente in salita: in questo caso gli RPM si abbassano notevolmente in quanto il controllore deve ancora reagire al maggiore sforzo di controllo necessario. Un altro picco, invece, si nota al secondo 13, dove, una volta iniziata la discesa, gli RPM aumentano prima che il PID esegua un'azione di frenata riportando la velocità sotto controllo.

5.5.1. Taratura PID salita

Il PID che controlla il movimento lungo una salita è stato tarato con i seguenti parametri:

Kp:	0.00006
Ki:	0.0009
Kd:	0

Tabella 5.1.: Taratura PID in salita

5.5.2. Taratura PID Discesa

Il PID che regola il movimento lungo una discesa è stato tarato con i seguenti parametri:

Kp:	0.00008
Ki:	0.002
Kd:	0

Tabella 5.2.: Taratura PID in discesa

Come si può notare, nella discesa, il parametro integrale Ki è di un ordine di grandezza superiore rispetto a quello della salita: infatti, la discesa può portare il motore in oscillazione, e questo parametro serve proprio per ridurle.

5.6. Pendenza senza PID

In conclusione del capitolo, si illustra un altro possibile modo di come affrontare una strada che presenta un'inclinazione non ignorabile.

Osservando i parametri del motore, si nota in Figura 5.10 che il primo gestisce il throttle-matching in maniera automatica.

No.	Setting item	Option 1	Option 2	Option 3	Option 4	Option 5	Option 6	Option 7	Option 8	Option 9
1	RPM/Throttle Matching	*Enabled*	Disabled							
2	LiPo Cells	*Auto*	2S	3S						
3	Cutoff Voltage	Disabled	Low	*Medium*	High					
4	ESC Thermal Protection	*105℃/221℉*	125℃/257℉							
5	Motor Rotation	*CCW*	CW							
6	BEC Voltage	*6.0V*	7.4V							
7	Drag Brake Force	Disabled	Level 1	Level 2	Level 3	*Level 4*	Level 5	Level 6	Level 7	Level 8
8	Drag Brake Rate	Level 1	Level 2	Level 3	Level 4	*Level 5*	Level 6	Level 7	Level 8	Level 9
9	Max. Reverse Force	25%	50%	75%	*100%*					

Figura 5.10.: Parametri del motore programmabili

Se questa impostazione è attivata attraverso il controllo a circuito chiuso della velocità il motore realizzerà la funzione di controllo automatico della velocità di crociera, ovvero, quando la resistenza del veicolo cambia, l'ESC regolerà automaticamente la coppia di uscita. Così facendo si potrebbe evitare di creare e tarare altri due PID per il controllo in pendenza.

Questo procedimento semplifica molto le cose permettendo di risparmiare il tempo della taratura e gestione di altri PID nel banco, però non garantisce un controllo totale sulla velocità.

Alla luce di ciò si è deciso di non optare per questo metodo disabilitando suddetta funzione in quanto potrebbe portare a comportamenti anomali della trazione.

Capitolo 6.

Sterzo

In questo ultimo capitolo verrà discusso il problema del movimento laterale del veicolo, che è stato effettuato tramite un servo motore, del quale è stato parlato nel Capitolo 2.6.

Per quanto riguarda il suo funzionamento, è stato utilizzato un approccio analogo al controllo del motore in trazione, utilizzando un segnale PWM, come descritto in precedenza nel Capitolo 3.2.

Una differenza sostanziale risiede però nel tipo di controllo. Infatti, invece di un controllo in velocità, come nel caso del motore brushless, il servomotore viene controllato in posizione: si fa, quindi, riferimento all'angolo compreso tra la posizione di riposo e quella di massimo discostamento. Nel contesto applicativo è stato utilizzato un servomotore con angolo di rotazione compreso tra 0° e 180°. La relazione tra il duty cycle e l'angolo di rotazione è di tipo lineare ed è quella che segue:

$$\theta = \delta_\theta \cdot 180^\circ \tag{6.1}$$

Il controllo richiede una portante di frequenza da 50Hz e quindi un periodo di 20ms. Inoltre, l'angolo di rotazione minimo si ottiene con un impulso di 0.5ms mentre quello massimo con uno di 2.5ms.

La relazione che permette di assegnare correttamente il valore di duty cycle al servomotore risulta, quindi, essere la seguente:

$$\delta = \frac{0.5 + \left(\frac{\theta}{180^\circ}\right) \cdot 2}{20} \qquad \theta \in (0^\circ, 180^\circ) \tag{6.2}$$

6.1. Attuazione del servo motore

Dopo aver spiegato concettualmente il funzionamento del servomotore, si passa ad analizzare come effettivamente si implementa utilizzando l'ambiente di sviluppo CubeIDE, partendo dalla definizione di un Timer per la generazione del PWM nel file *.ioc*.

Per fare ciò si è scelto di utilizzare il TIM1 e, dopo aver selezionato la sua funzione come generatore di PWM, si possono vedere i valori di Prescaler e Counter period

selezionati per ottenere una frequenza di 50Hz del segnale PWM, come riportato in Figura 6.1:

$$frequenza = \frac{Timer_{clock}}{(Prescaler)(Counterperiod)} = \frac{84.000.000}{1680 \cdot 1000} = 50Hz \qquad (6.3)$$

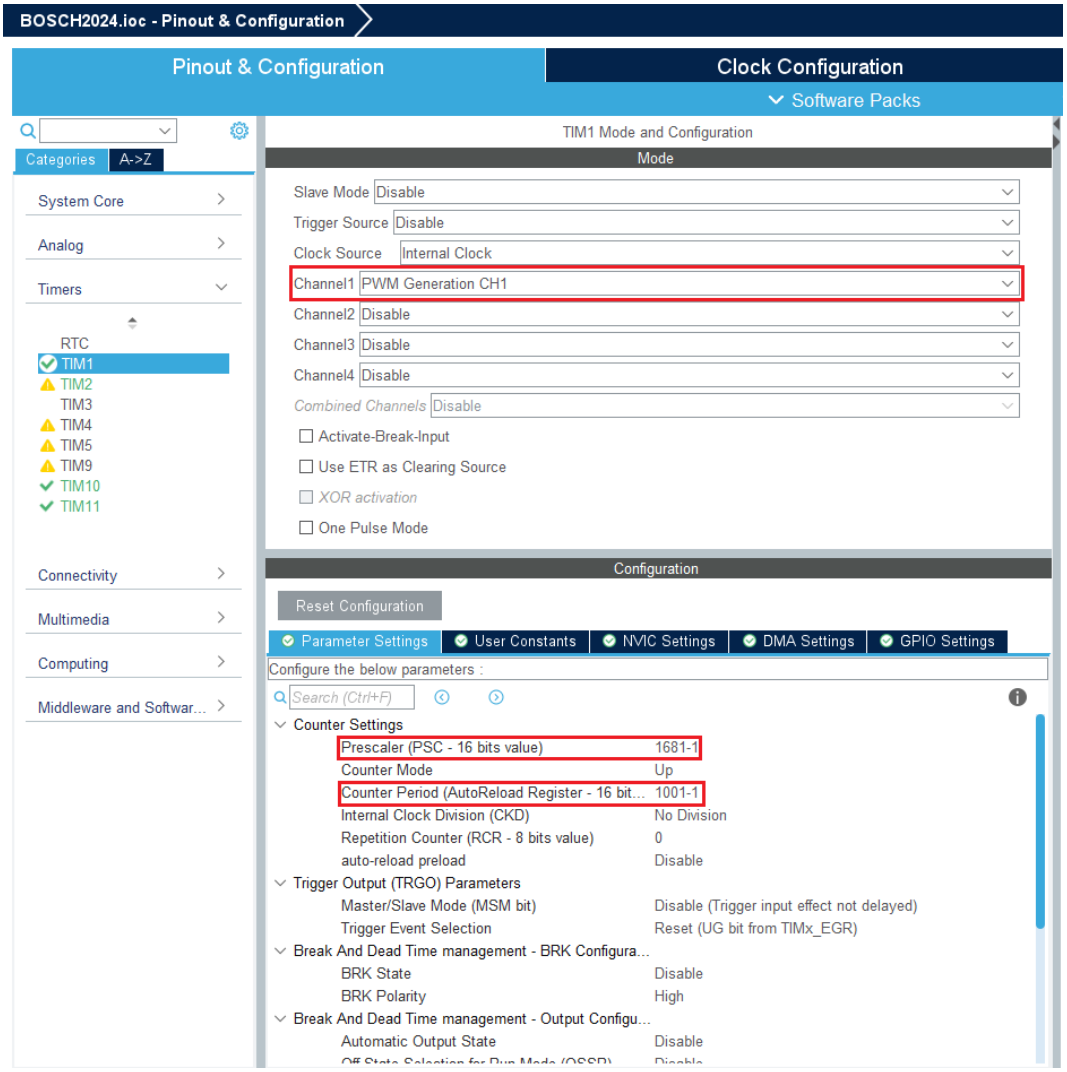


Figura 6.1.: .ioc servo motore

Come di consueto, il codice si divide in due librerie, che sono:

1. Header file;
2. Source file;

6.1.1. Header file servo motore

In questa libreria sono definiti dei valori costanti che dipendono dal tipo di servomotore utilizzato, ma anche dalla meccanica del veicolo. Questi sono gli angoli

di massima sterzata e quello neutro, che approfonditi nel Capitolo 6.2.

```
1         #define DRITTO 47
2         #define MAX_ANGOLO 46
3         #define MIN_ANGOLO -46
4         #define PI 3.141592654
5
6         void Set_Angle(float);
7
```

Listing 6.1: Header file servo motore

6.1.2. Source file servo motore

Il Codice 6.2, riportato di seguito, mostra la definizione vera e propria del metodo *Set_Angle(float)*: esso prende come variabile di ingresso il float dell'angolo desiderato e genera un segnale PWM adatto.

```
1     void Set_Angle(float angolo){
2       if(angolo < MIN_ANGOLO)
3         angolo = MIN_ANGOLO;
4       else if(angolo > MAX_ANGOLO)
5         angolo = MAX_ANGOLO;
6
7       float angolo_corretto = angolo + DRITTO;
8
9       int min_duty = TIM->ARR / 40; //0.5ms
10
11      float duty = (angolo_corretto/180) * (TIM1->ARR/10); //0ms-2ms
12
13      TIM1->CCR1 = min_duty + (int)duty; //set compare register to
    appropriate value
14    }
15
```

Listing 6.2: Source file servo motore

6.2. Calibrazione meccanica

Per eseguire la calibrazione meccanica del servo motore è necessario verificare l'angolo massimo e minimo di sterzata. Oltre a ciò, un altro angolo fondamentale è quello che permette allo sterzo di direzionare le ruote in una posizione neutrale.

Questi valori vanno trovati ogni volta che il servo viene montato sulla macchina, poiché non è nota la posizione iniziale del servo, cioè quella che possiede prima di diventare solidale al sistema di sterzo della macchina.

La ricerca di questi angoli non segue una procedura rigorosa e standardizzata, ma sono presenti metodi più comodi di altri.

Per modificare l'angolo neutrale è necessario intervenire sul valore della costante *DRITTO* definita nel Codice 6.1 in modo tale che le ruote anteriori assumano nella posizione desiderata.

Trovare l'angolo massimo e minimo, invece, può essere un po' più difficoltoso. Infatti, i movimenti del servo motore non sono completamente fluidi e osservabili per ogni grado di rotazione in quanto affetti da un attrito statico che deve essere superato. Per questo motivo, si è deciso di far incrementare l'angolo del servo motore in modo continuato, aumentando il suo angolo di sterzo di uno ogni secondo. Questo procedimento permette di osservare come varia lo sterzo in tempo reale e quindi poter stabilire qual è l'angolo massimo raggiungibile, ovvero quello in cui le ruote non saranno più in grado di sterzare trovandosi al limite meccanico.

Oltre a ciò, è molto importante tenere traccia in tempo reale anche dell'angolo del servo motore, segnandolo di volta in volta. Infatti, tenere uno storico del valore di questo è fondamentale perché, attraverso prove ripetute, sarà possibile definire l'angolo massimo reale facendo la media tra tutti quelli trovati e ignorando in questo modo eventuali errori dovuti alle condizioni non ideali in cui si lavora (presenza di attrito e resistenze parassite).

Il Codice 6.3 di seguito, riporta solo le righe che descrive il comportamento del timer, necessarie per incrementare la variabile *cnt_sterzo* ogni 10ms, così da avete un aumento corrispondente dell'angolo di sterzo.

```
1         void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
2           if (htim == &htim11) {
3                 cnt_sterzo++;
4           }
5         }
6
```

Listing 6.3: Timer 10ms per il test del servo motore

Nel Codice 6.4, invece, sono riportate le righe di codice che definiscono lo scorrere del tempo utilizzando la variabile contatore precedentemente creata e determinano l'angolo di sterzo da impartire al servomotore.

```
1         if(cnt_sterzo == 100){
2             cnt_sterzo = 0;
3             angolo++;
4             Set_Angle(angolo);
5             printf("angolo: %f\r\n", angolo);
6         }
7
```

Listing 6.4: Test del servo motore

Non è presente la definizione della variabile float *"angolo"*, poiché questa va definita all'interno del main ma fuori dal ciclo principale con valore pari a 0.

Per testare l'angolo minimo di sterzo (quindi dalla parte opposta) sarà sufficiente sostituire la riga 3 del Codice 6.4 con "angolo–;".

6.3. Controllo in catena chiusa per lo sterzo

L'angolo di yaw (ψ) rappresenta la rotazione di un veicolo attorno all'asse verticale e indica la direzione in cui tale veicolo sta puntando rispetto ad una direzione di riferimento, solitamente il nord. È un parametro fondamentale nella dinamica del veicolo, soprattutto per quanto riguarda la stabilità e il controllo. Lo yaw rate ($\dot{\psi}$), invece, è la velocità angolare assunta da questa rotazione ed è misurato in gradi al secondo ($^\circ/s$) o radianti al secondo (rad/s).

Nel caso preso in esame in questo lavoro di tesi, come variabile per effettuare il controllo in catena chiusa sul movimento laterale dei veicolo, verrà usato lo yaw rate. Il valore di riferimento da seguire verrà fornito dalla Raspberry mediante l'uso della telecamera e di un algoritmo che, in base alla strada presente in prossimità della macchina, restituisce il raggio di curvatura. Tramite questo parametro è possibile calcolare lo yaw rate tramite la seguente formula:

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

dove v è la velocità longitudinale del veicolo mentre R è il raggio di curvatura.

L'angolo di yaw rate effettivo a cui è sottoposta la macchina verrà misurato tramite il sensore IMU e confrontato con il valore di riferimento utilizzando un controllore PID per realizzare il controllo a catena chiusa.

In Figura 6.2 è riportato lo schema a blocchi della catena chiusa che regola lo spostamento laterale mediante il procedimento appena descritto.

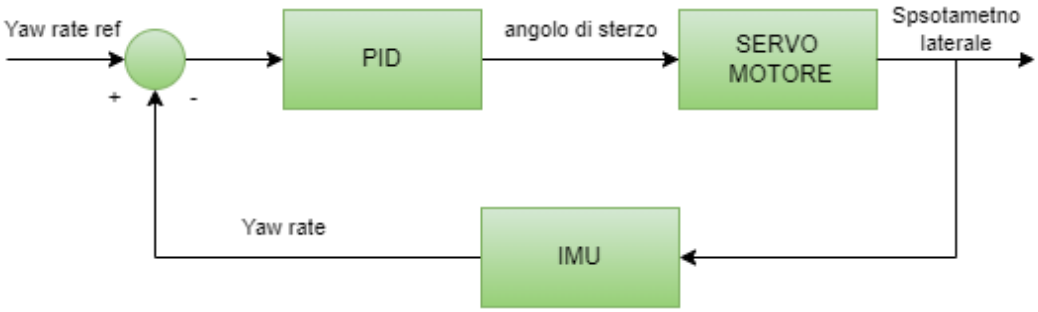


Figura 6.2.: Catena chiusa IMU

Di seguito, nel Codice 6.5, è riportata l'implementazione della chiamata del PID al fine di ottenere uno sforzo di controllo adatto per il servo motore.

```
1         //Get yawrate from IMU
2         bno055_vector_t v = bno055_getVectorGyroscope();
3         vehicleState.yaw_rate_deg_sec = v.z;
4         vehicleState.yaw_rate_rad_sec = (vehicleState.yaw_rate_deg_sec
      * M_PI) / 180;
5         last_read = dataRX.curvature_radius_ref_m;
```

```
6
7           // Decido quale PID usare in base al raggio di curvatura
8           if (dataRX.curvature_radius_ref_m >=
    MAX_CURVATURE_RADIUS_FOR_STRAIGHT) {
9               vehicleState.yaw_rate_ref_rad_sec = 0;
10              u_sterzo = PID_controller(&pid_steering, vehicleState.
    yaw_rate_rad_sec, vehicleState.yaw_rate_ref_rad_sec);
11              servo_motor(-u_sterzo); //Minus because yawrate and
    steering are opposite
12          } else {
13              vehicleState.linear_speed_m_s = vehicleState.
    motor_speed_RPM * RPM_2_m_s;
14              if (dataRX.curvature_radius_ref_m == 0)
15                  vehicleState.yaw_rate_ref_rad_sec = 0;
16              else
17                  vehicleState.yaw_rate_ref_rad_sec = vehicleState.
    linear_speed_m_s / dataRX.curvature_radius_ref_m;
18
19              // Salvo i valori assoluti dello yaw_rate
20              float yaw_rate_ref_rad_sec_abs = vehicleState.
    yaw_rate_ref_rad_sec;
21              float yaw_rate_rad_sec_abs = vehicleState.yaw_rate_rad_sec
    ;
22              if (vehicleState.yaw_rate_ref_rad_sec < 0)
23                  yaw_rate_ref_rad_sec_abs = -vehicleState.
    yaw_rate_ref_rad_sec;
24              if (vehicleState.yaw_rate_rad_sec < 0)
25                  yaw_rate_rad_sec_abs = -vehicleState.yaw_rate_rad_sec;
26
27              u_sterzo = PID_controller(&pid_steering,
    yaw_rate_rad_sec_abs, yaw_rate_ref_rad_sec_abs);
28
29              //Minus because yawrate and steering are opposite
30              if (dataRX.curvature_radius_ref_m >= 0 && u_sterzo > 0)
31                  u_sterzo *= -1.0;
32              if (dataRX.curvature_radius_ref_m < 0 && u_sterzo < 0)
33                  u_sterzo *= -1.0;
34
35              servo_motor(u_sterzo);
36              //servo_motor(0);
37          }
38
```

Listing 6.5: Implementazioe PID sterzo

6.4. Taratura PID dello sterzo

E' bene evidenziare che anche il PID dello sterzo ha ricevuto una taratura accurata per prevenire comportamenti anomali e garantire l'azione desiderata. In particolare, sono stati scelti i seguenti parametri:

Kp:	20
Ki:	250
Kd:	0
Kb:	50

Tabella 6.1.: Taratura PID in discesa

Osservando i valori di Kp e Ki, si nota che il parametro integrale è sensibilmente più alto di quello proporzionale al fine di garantire il comportamento desiderato. Infatti, un Ki così elevato fa si che il comportamento del servomotore a regime sarà estremamente stabile ed è tale da smorzare istantaneamente le oscillazioni.

Tale andamento del PID è quello ideale in quanto un "colpo di frusta" sullo sterzo potrebbe causare perdita di trazione della macchina e conseguente sbandamento. Questa volta è stato inserito anche il parametro Kb, utilizzato nell'algoritmo di anti-windup mediante il metodo della back calculation.

In Figura 6.3 è stato riportato il comportamento del PID al fine di seguire un riferimento sinusoidale in yaw rate.

Si osserva come, a causa dell'alto parametro integrale, è presente un ritardo non trascurabile prima di raggiungere il riferimento, e la misurazione dell'IMU è più rumorosa rispetto a quelle viste fino ad ora dell'encoder in quanto non è stato inserito alcun filtro a media mobile,

E' stato deciso di non utilizzare tale filtro vista la necessità di una risposta rapida del servo motore per poter reagire prontamente in presenza di ostacoli.

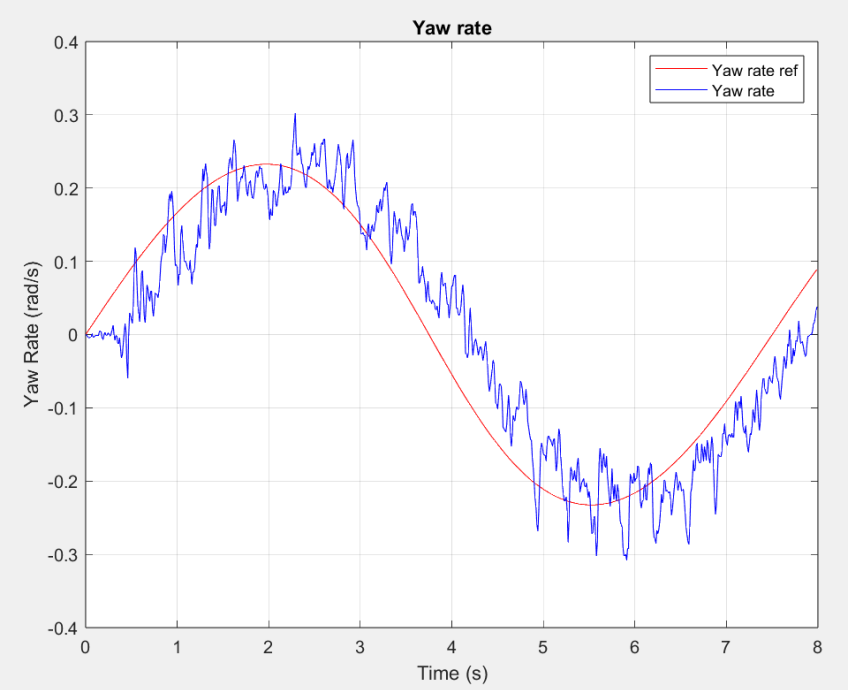


Figura 6.3.: Yaw rate sinusoidale

6.5. Anti-windup: Back-calculation

Un problema comune nei controlli basati su PID (Proporzionale-Integrale-Derivativo) è il fenomeno del windup dell'integratore. Ciò si verifica quando l'uscita del controllore supera i limiti fisici dell'attuatore, causando una saturazione che porta a prestazioni degradate, oltre che all'instabilità del sistema. Per mitigare questo problema, vengono impiegati algoritmi di anti-windup.

Il principio dell'anti-windup è quello di limitare l'azione integrativa durante i periodi di saturazione dell'attuatore, mantenendo la stabilità del sistema di controllo.

Un metodo efficace per la sua implementazione è il metodo della **back calculation** [5]. Questo approccio corregge l'accumulo dell'azione integrativa retrocalcolando l'errore derivante dalla differenza tra l'uscita effettiva del controllore (limitata) e l'uscita desiderata (non limitata). In altre parole, la back calculation reintroduce un segnale corretto nel termine integrale del PID, riducendo così l'effetto del windup.

Il processo di back calculation può essere descritto nei seguenti passaggi:

1. **Identificazione della saturazione**: quando l'uscita del controllore raggiunge i limiti fisici dell'attuatore, si identifica la condizione di saturazione;

2. **Calcolo dell'errore di saturazione**: in questa fase si calcola la differenza tra l'uscita limitata del controllore e l'uscita desiderata. L'errore che si ricava rappresenta quanto l'uscita effettiva si discosta dall'uscita ideale;

3. **Aggiornamento del termine integrale**: utilizzando l'errore di saturazione, si aggiorna il termine integrale per compensare l'accumulo eccessivo. In questo modo si mantiene l'azione integrativa entro limiti controllabili.

Nella Figura 6.4 si può osservare lo schema a blocchi dell'implementazione di questo algoritmo. Per poter esprimere al meglio il funzionamento, il blocco del controllore PID che prima era singolo, è stato separato nelle sue tre componenti, cioè nelle diverse azioni.

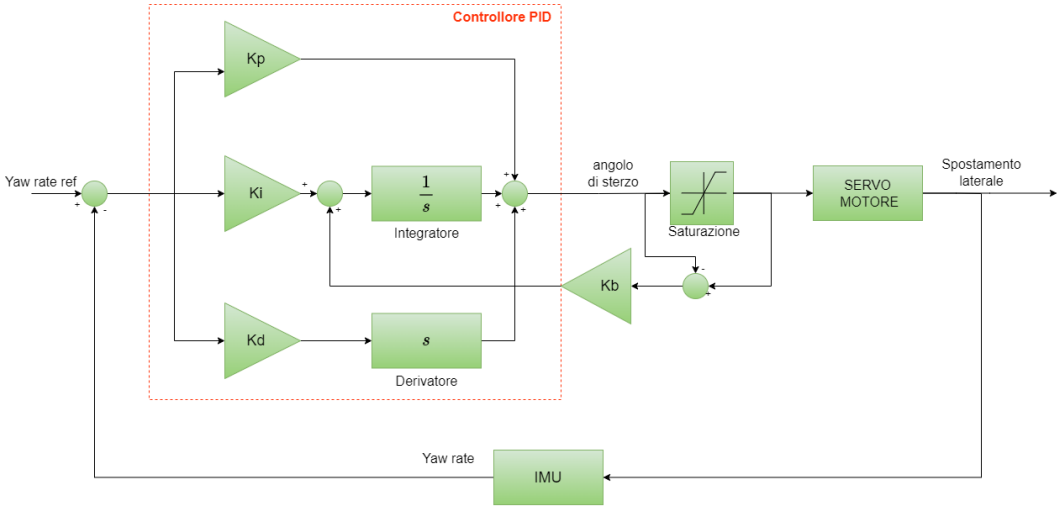


Figura 6.4.: Anti-windup con back calculation

Nel Codice 6.6 di seguito, invece, viene riportata l'implementazione di uno schema a blocchi, che illustra quanto detto in questo capitolo:

```
1     float saturated_u = u;
2
3   if(saturated_u > p->u_max){
4     saturated_u = p->u_max;
5   }
6   else if(saturated_u < p->u_min){
7     saturated_u = p->u_min;
8   }
9
10  float correction = p->Kb * (saturated_u - u) * p->Ki * p->Tc;
11  p->Iterm = newIterm + correction;
12
13  u = saturated_u;
14
```

Listing 6.6: Implementazione anti-windup con back calculation

Quanto riportato appena sopra, è un estratto del Codice 4.2, visto che la spiegazione di questa parte è stata tralasciata al momento, in quanto non utile ai fini della discussione.

Conclusione

Questo lavoro di tesi ha esplorato lo sviluppo di algoritmi per il controllo di direzione e velocità su un microcontrollore, con l'obiettivo di realizzare un sistema di guida autonoma per un veicolo in scala 1:10. Il progetto, sviluppato per partecipare alla Bosch Future Mobility Challenge 2024, ha richiesto un'analisi dettagliata e un approccio sistematico che hanno contribuito al successo dello stesso.

Inizialmente, il lavoro è stato focalizzato sull'hardware del veicolo, integrando componenti fondamentali come il Nucleo FE401RE, la Raspberry PI 4, la Coral TPU e vari sensori tra cui encoder, IMU, LIDAR e telecamere. Questa infrastruttura robusta ha permesso di raccogliere dati ambientali e di controllare il veicolo in tempo reale, creando la base per l'implementazione degli algoritmi di guida autonoma.

Lo studio è proseguito con la preparazione del motore. L'analisi di quest'ultimo e la tecnica di alimentazione tramite PWM sono state essenziali per far si che il motore rispondesse correttamente ai comandi del microcontrollore. Inoltre, la calibrazione dell'ESC e la lettura precisa della velocità tramite encoder, hanno migliorato la reattività e la stabilità del sistema.

Oltre a ciò, il lavoro sulla trazione ha evidenziato l'importanza della mappatura iniziale e della comprensione dell'andamento quadratico del motore. La taratura avanzata con una scheda programmabile e l'implementazione di un controllore PID hanno ottimizzato la risposta del sistema. Le prove di PID, prima su banco e poi nel veicolo, hanno dimostrato l'efficacia delle tarature, permettendo al veicolo di mantenere una traiettoria stabile sia in piano che in pendenza. Questo controllo preciso è stato fondamentale per la navigazione autonoma del veicolo.

In aggiunta, l'attuazione del servo motore per lo sterzo ha richiesto una calibrazione meccanica accurata e un controllo in catena chiusa più complesso. L'implementazione del PID per lo sterzo ha migliorato la guida, rendendola più fluida e precisa, minimizzando gli errori di traiettoria e aumentando la sicurezza del veicolo. L'uso della tecnica anti-windup Back-calculation ha ulteriormente raffinato le prestazioni del sistema di controllo.

Inoltre, l'implementazione degli algoritmi nell'ambiente di sviluppo CubeIDE è stata un passo fondamentale per testare e validare le soluzioni proposte: questo ambiente ha facilitato l'integrazione dei vari componenti e l'esecuzione degli algoritmi in tempo reale. La replicabilità delle tecniche sviluppate è stata dimostrata attraverso una serie di test pratici, che hanno evidenziato la versatilità e l'efficacia dell'approccio adottato.

In conclusione, lo studio fatto ha fornito un contributo significativo alla ricerca e allo sviluppo di tecniche avanzate per il controllo di veicoli autonomi in miniatura. E' stato messo in atto un approccio metodico e ben strutturato, isolando di volta in volta l'argomento di studio: in questo modo è stato possibile raggiungere risultati concreti e replicabili. Le metodologie sviluppate possono essere ulteriormente estese e adattate per affrontare sfide più complesse nella guida autonoma, promuovendo l'innovazione tecnologica e la sicurezza nei trasporti. Questo lavoro, quindi, rappresenta una solida base per future ricerche e applicazioni in questo campo, aprendo la strada a nuove possibilità e miglioramenti continui.

Bibliography

[1] Andrea Bonci. *06_EN_SW_STM32H7 TIMERS v1*. Diapositiva PowerPoint. Lezione del corso di Ingegneria informatica e dell'Automazione, Università Politencina delle Marche.

[2] Andrea Bonci. *07_EN_SW_STM32H7 PWM v1*. Diapositiva PowerPoint. Lezione del corso di Ingegneria informatica e dell'Automazione, Università Politencina delle Marche.

[3] Andrea Bonci. *10_EN_SW_STM32H7 ENCODER v3*. Diapositiva PowerPoint. Lezione del corso di Ingegneria informatica e dell'Automazione, Università Politencina delle Marche.

[4] Gianluca Ippoliti. *05_Motori_Brushless*. Diapositiva PowerPoint. Lezione del corso di Ingegneria informatica e dell'Automazione, Università Politencina delle Marche.

[5] Linqi Ye et al. "Anti-Windup Robust Backstepping Control for an Underactuated Reusable Launch Vehicle". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* (Aug. 2020). DOI: 10.1109/TSMC.2020.3020365.

Sitography

[6] *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-electric-road-version-4wdarr.pdf.

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

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

[9] *Coral Accelerator Module datasheet.* URL: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwjM4fbtp4WHAxUlAfsDHeDuAkQQFnoECBQQAQ&url=https%3A%2F%2Fcoral.ai%2Fstatic%2Ffiles%2FCoral-Accelerator-Module-datasheet.pdf&usg=AOvVaw3B1EO-4U_1HfwV3VCMmn2K&opi=89978449.

[10] *Datasheet Raspberry Pi 4 Model B.* URL: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwjarOTfp4WHAxWbTqQEHVebDFgQFnoECBgQAQ&url=https%3A%2F%2Fdatasheets.raspberrypi.com%2Frpi4%2Fraspberry-pi-4-datasheet.pdf&usg=AOvVaw3Y6CnxbHhvf61bbAiQu8Nw&opi=89978449.

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

[12] *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.

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

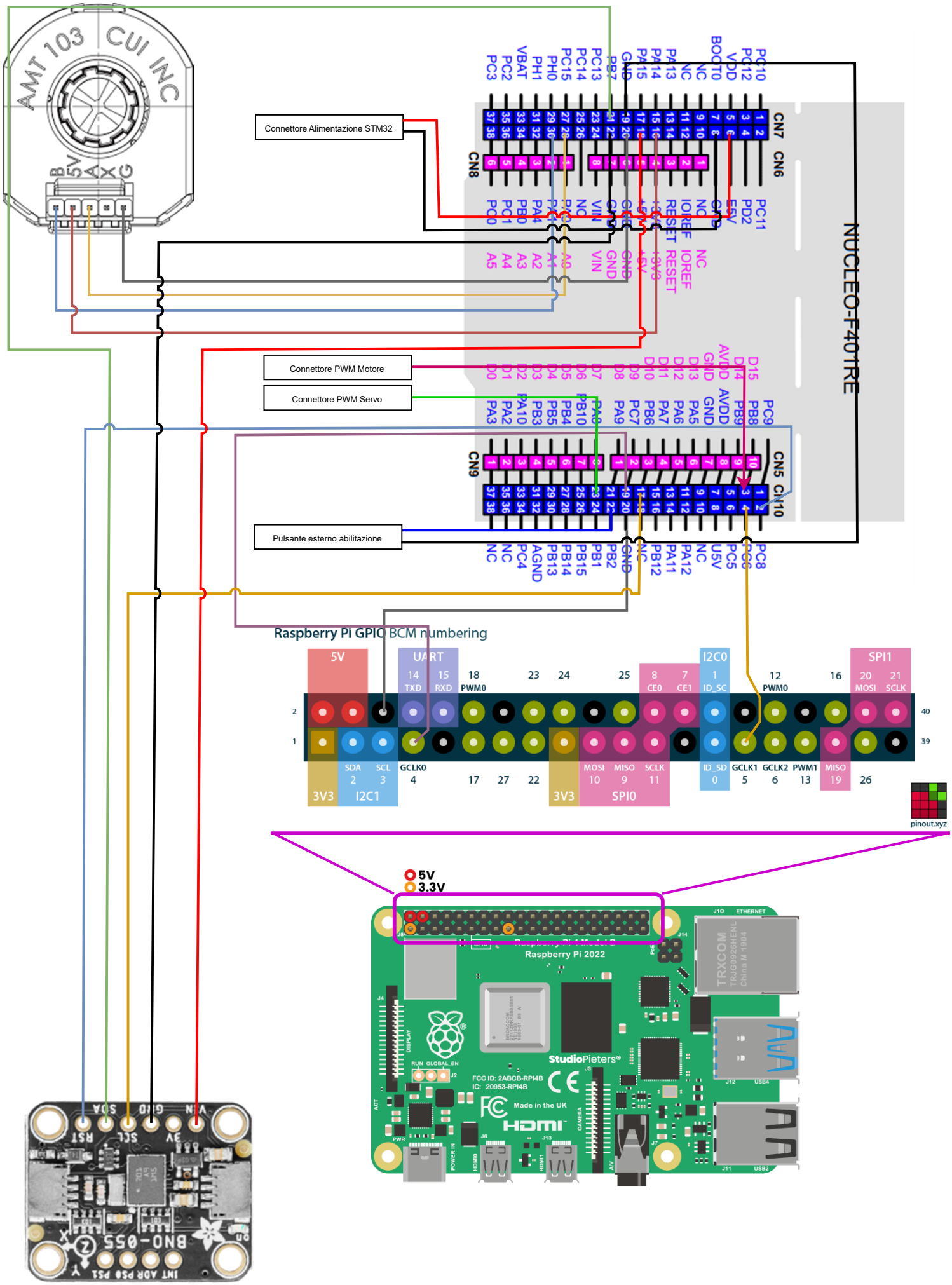
[14] *Raspberry Pi Documentation.* URL: https://www.raspberrypi.com/documentation/accessories/camera.html.

[15] *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.

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

Appendice A.

Schema elettrico delle componenti


AMT 103
CUI INC
B 5V A X G
Connettore Alimentazione STM32
CN7
CN6
CN8
NUCLEO-F401RE
Connettore PWM Motore
Connettore PWM Servo
CN9
CN5
CN10
Pulsante esterno abilitazione
Raspberry Pi GPIO BCM numbering
5V
UART
I2C0
SPI1
TXD
RXD
PWM0
CE0
CE1
ID_SC
MOSI
SCLK
SDA
SCL
GCLK0
ID_SD
GCLK1
GCLK2
PWM1
MISO
3V3
I2C1
SPI0
pinout.xyz
5V
3.3V
Raspberry Pi 2022
StudioPieters®
Made in the UK
HDMI
BNO-055