UNIVERSITÁ POLITECNICA DELLE MARCHE

FACOLTÁ DI INGEGNERIA

CORSO DI LAUREA TRIENNALE IN
INGEGNERIA INFORMATICA E DELL'AUTOMAZIONE

Studio e implementazione su sistemi embedded di soluzioni per la guida e la navigazione autonoma di autoveicoli elettrici in scala

Study and implementation on embedded systems of autonomous driving and navigation solutions for scale electric vehicles

Relatore:
Prof. Andrea Bonci

Tesi di Laurea di:
Agnese Bruglia

Anno Accademico 2022/2023

"Success is not final, failure is not fatal:
it is the courage to continue that counts."

"Il successo non è definitivo, il fallimento non è fatale:
ciò che conta è il coraggio di andare avanti."

Winston Churchill

Ringraziamenti

Vorrei ringraziare per primi i miei genitori e tutta la mia famiglia, per avermi sempre motivata a dare il meglio e per avermi sostenuta, economicamente e moralmente, con pazienza appoggiando ogni mia decisione e non facendomi mai mancare nulla.

Ringrazio mia sorella, Sabina, per essere stata la mia prima sostenitrice durante tutto questo percorso accademico e un'amica con cui confrontarmi nei momenti di difficoltà, certa che da lei avrei sempre ricevuto un abbraccio e un incoraggiamento.

Ringrazio il mio ragazzo, Enrico, per avermi trasmesso la sua immensa forza e coraggio e per essermi sempre stato accanto mettendo da parte le sue preoccupazioni e spronandomi a diventare la versione migliore di me stessa.

Ringrazio i miei amici e amiche, quelli che ci sono dai tempi del liceo e quelli che invece da poco, perhè in loro ho sempre trovato una risata nei momenti felici e un porto sicuro dove rifugiarmi in momenti più difficili.

Ringrazio anche il mio relatore, il Professor Andrea Bonci, per avermi permesso di approfondire questo argomento e tutti i professori, universitari e del liceo, che mi hanno guidata durante il mio percorso di studi e il cui sostegno è stato fondamentale per il raggiungimento di questo traguardo.

Infine, ringrazio mia nonna, Grazia, che non solo ha costituito, per me come per molti altri, un'ispirazione in quanto donna che è riuscita a realizzarsi nell'ambito delle scienze, ma anche per essere stata una nonna presente e premurosa, avendomi cresciuta con amore e avendomi fatta diventare la persona che sono oggi.

Ancona, Dicembre 2023

Sommario

Questa tesi si focalizza sulla creazione di un sistema di guida autonoma per un veicolo elettrico in scala 1/10 progettato specificamente per partecipare alla Bosch Future Mobility Challenge (BFMC), una competizione internazionale organizzata dal Centro Ingegneristico della Bosch di Cluj Napoca, Romania.

L'obiettivo principale del progetto è consentire al veicolo di navigare autonomamente in un ambiente che simula un percorso in miniatura. Ciò significa che il veicolo deve essere in grado di seguire una traiettoria predefinita, mantenendo la sua posizione all'interno delle linee di una corsia e affrontando le curve in modo appropriato. Inoltre, deve essere in grado di rispondere ai segnali stradali, parcheggiare sia parallelamente che perpendicolarmente e interagire con altri oggetti lungo il percorso, come altri veicoli e pedoni, evitandoli o interagendovi consonamente.

Per raggiungere questo obiettivo, il sistema utilizza, tra i vari sensori, una telecamera per l'elaborazione delle immagini, che gli consente di individuare la traiettoria da seguire ed eventuali oggetti lungo il percorso, due controllori PID, di cui il primo è responsabile del mantenimento di una velocità costante del veicolo, mentre il secondo regola l'angolo di sterzata per ottenere una guida il più fluida possibile, e un sensore ad ultrasuoni, che misura la distanza dagli oggetti circostanti.

In sintesi, il progetto nel suo intero si propone di sviluppare un sistema di guida autonoma completo e sofisticato che possa competere con successo nella Bosch Future Mobility Challenge. In particolare, il lavoro svolto in questa tesi non si focalizza su elaborazione di immagini e algoritmi di generazione della traiettoria, parcheggio o altre manovre, ma sull'aspetto del progetto che si occupa di acquisizione e attuazione dei comandi e controllo dei motori.

Indice

Elenco delle figure

1

Introduzione alla Guida Autonoma

Indice

1.1 Autonomous Vehicles

Il concetto di *guida autonoma* si riferisce a veicoli o sistemi di veicoli in grado di muoversi e navigare senza l'intervento umano diretto. Questi veicoli autonomi (da ora abbreviati in "VA") sono progettati per percepire l'ambiente circostante, elaborare le informazioni attraverso sensori come telecamere, lidar (Light Detection and Ranging), radar (Radio Detection and Ranging), gps e altri dispositivi e prendere decisioni in tempo reale basate su queste informazioni.

Il concetto è associato inoltre all'intelligenza artificiale e all'apprendimento automatico: i VA infatti utilizzano algoritmi complessi e modelli di machine learning per interpretare i dati acquisiti dai sensori e quindi identificare oggetti, veicoli, pedoni e ostacoli sulla strada, nonché per pianificare il percorso e prendere decisioni su come accelerare, frenare, girare o fermarsi.

Il mercato di veicoli autonomi, come si vede dalla figura 1.1, associata alla statistica presentata sul sito sul sito *"statista.com"* che si trova al link "https://www.statista.com/statistics/1230664/projected-number-autonomous-cars-worldwide" che è una stima del numero di VA che ci saranno globalmente ogni anno fino al 2030, crescerà sempre più: il numero di persone che sceglieranno di possedere un'auto di questo tipo è destinato ad aumentare esponenzialmente.

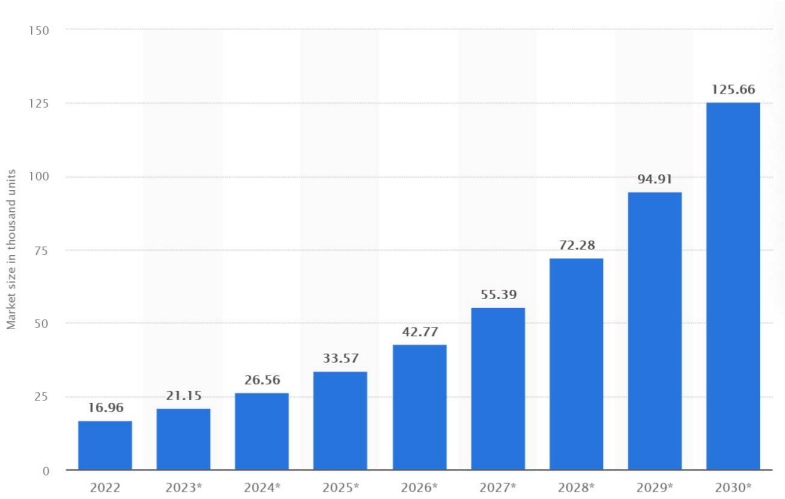

Figura 1.1: Numero di VA a livello globale

1.2 Livelli di automazione

I VA sono classificati in 6 livelli di automazione definiti dalla Society of Automotive Engineers (SAE). Questi vanno, come si nota dall'immagine 1.2, dal livello 0 (nessuna automazione) al livello 5 (automazione completa), in cui ogni livello eredita le funzionalità di quello precedente, e indicano il grado di automazione del veicolo e il coinvolgimento del conducente. Alcune automobili moderne sul mercato hanno funzionalità di guida assistita che rientrano in categorie di automazione parziali, come il parcheggio automatico o l'assistenza al mantenimento della corsia. Tuttavia, l'obiettivo finale della guida autonoma sarebbe quello di sviluppare veicoli che possano gestire completamente tutte le attività di guida senza l'intervento umano, migliorando la sicurezza stradale e offrendo maggiore comodità agli utenti.

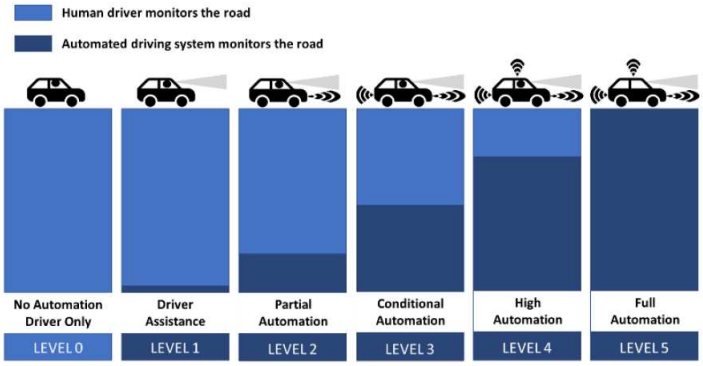

Figura 1.2: Livelli di Automazione

Tra le ragioni che hanno spinto gli sviluppatori dei VA verso la loro progettazione e il loro costante miglioramento, si distingue la riduzione degli incidenti stradali grazie all'implementazione di sistemi di guida sempre più sofisticati, che incrementano la sicurezza stradale. L'immagine 1.3, relativa alla statistica presentata sul sito *"statista.com"* che si trova al link "https://www.statista.com/statistics/1238242/impact-of-vehicle-automation-on-collision-rates/", è una stima di quelli che potrebbero essere i fattori di riduzione degli incidenti stradali entro l'anno 2030 relativi ai diversi livelli di automazione; come si può vedere c'è una notevole differenza tra i livelli di automazione minore e quelli ad automazione maggiore, che garantiscono cioè maggiore sicurezza.

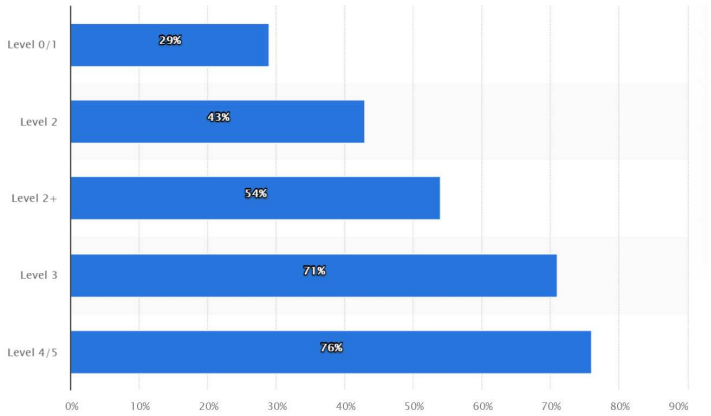

Figura 1.3: Fattori di riduzione di incidenti stradali

1.2.1 Livello 0

Il Livello 0, anche noto come *"Nessuna Automazione"*, rappresenta il livello più basso nella scala di automazione dei veicoli. In questo livello il veicolo non dispone di alcuna tecnologia di automazione: sterzo, accelerazione, frenata, parcheggio e qualsiasi altra manovra sono affidati totalmente al conducente, che ha quindi l'intera responsabilità della guida, e non vi è alcun tipo di assistenza da parte dell'auto.
Il Livello 0 perciò è caratterizzato da:

- *Nessuna automazione*: il veicolo non dispone di alcuna funzione di assistenza alla guida: non ci sono sistemi di controllo automatico della velocità, assistenza al mantenimento della corsia, sistemi anti-collisione o altre tecnologie di assistenza;
- *Responsabilità totale del conducente*: il conducente è completamente responsabile della guida;
- *Nessun riconoscimento di ambiente*: il veicolo non è equipaggiato con sensori o telecamere per riconoscere l'ambiente circostante; non può quindi percepire altri veicoli, pedoni, segnali stradali o qualsiasi altra informazione visiva o sensoriale;
- *Assenza di automazione di base*: anche le caratteristiche di sicurezza di base, come i freni anti-bloccaggio (ABS) o i sistemi di controllo della trazione, sono considerate standard di sicurezza e non rientrano nella categoria di automazione in questo livello.

Questo livello rappresenta quindi la situazione tipica della guida prima dell'introduzione delle moderne tecnologie di assistenza alla guida e dell'automazione nei veicoli.

1.2.2 Livello 1

Il Livello 1 è anche noto come *"Guida Assistita"* e rappresenta il primo livello di automazione dei veicoli. Nonostante alcune funzioni di assistenza alla guida siano automatizzate, il conducente rimane comple-

tamente responsabile e deve essere pronto a intervenire in qualsiasi momento.
Le funzionalità distintive per questo livello perciò sono:

- *Adaptive Cruise Control (ACC)*: il veicolo può mantenere una velocità costante impostata dal conducente, ma può anche adattarla in base al traffico circostante: se il veicolo davanti rallenta o si ferma, il sistema riduce automaticamente la velocità per mantenere una distanza di sicurezza;
- *Assistenza al parcheggio*: il veicolo può assistere il conducente nel processo di parcheggio e, attraverso dei sensori di prossimità, avvertirlo se si avvicina troppo a un altro veicolo o a un qualsiasi altro oggetto;
- *Lane Departure Warning System (LDWS)*: è un sistema che individua le linee della corsia sulla strada e notifica il conducente con un segnale di allerta nel caso in cui ci sia un involontario cambio di carreggiata da parte del guidatore.

Tutte queste funzioni operano solo in situazioni specifiche e richiedono l'attenzione costante del conducente: infatti anche se alcune attività di guida sono automatizzate, il conducente è responsabile per la sicurezza e deve rimanere vigile mentre guida.
Affinché queste proprietá elencate funzionino correttamente, il veicolo ha bisogno di utilizzare diversi tipi di sensori, come sensori a ultrasuoni, radar ad ampio raggio (LRR) ed eventuali telecamere. In particolare il sensore a ultrasuoni utilizza il tempo di propagazione dell'onda del suono per localizzare oggetti molto vicini al veicolo, un LRR utilizza la differenza di tempo tra l'emissione di onde elettromagnetiche da un'antenna e la riflessione di queste su corpi che si trovano intorno al veicolo per capirne la posizione e infine la telecamera serve al guidatore per monitorare ciò che si trova vicino all'auto.

1.2.3 Livello 2

Il Livello 2 è anche noto come *"Automazione Parziale"*: il veicolo può gestire contemporaneamente la direzione, cioè la sterzata, e la velocità in specifiche situazioni.
Le funzionalità chiave del Livello 2 perciò sono:

- *Lane Keeping Assist (LKA)*: può essere visto come il complementare del LDWS: il sistema può mantenere il veicolo nella corsia di guida, correggendo automaticamente la direzione se il conducente si allontana dalla carreggiata senza segnalare; questo significa che il veicolo è in grado di effettuare sterzate automatiche per seguire la curva della strada e quindi girare in modo autonomo durante la guida su autostrade o in altre situazioni di traffico controllato;
- *Park Assist (PA)*: il veicolo può assistere il conducente nel processo di parcheggio, ciò può includere l'assistenza al parcheggio parallelo o perpendicolare, dove il veicolo può manovrare automaticamente per parcheggiare;

- *Riconoscimento degli ostacoli*: il veicolo può riconoscere ostacoli come altri veicoli, pedoni e oggetti sulla strada e reagire adeguatamente per evitarli o ridurre l'impatto.

Nonostante il Livello 2 offra un grado di automazione maggiore rispetto al livello precedente, il conducente deve rimanere attivo e vigile durante la guida e deve essere pronto a intervenire in qualsiasi momento. Ciò é noto come "conducente di backup" e sottolinea l'importanza della vigilanza umana anche in veicoli di Livello 2. In questa fase, il veicolo dovrebbe essere equipaggiato con diversi tipi di sensori: sensori a ultrasuoni, LRR, videocamere e Short-Range Radar (SRR).

1.2.4 Livello 3

Il Livello 3 è anche noto come *"Automazione Condizionale"*. In questa fase, il veicolo può gestire la maggior parte delle attività di guida in modo autonomo, inclusi il controllo della velocità, la sterzata e il mantenimento della corsia e consentire al guidatore di togliere le mani dal volante.
Il Livello 3 è perciò caratterizzato dalle seguenti funzionalità:

- *Automazione in determinate condizioni e assunzione temporanea della guida*: il veicolo può guidare autonomamente in specifiche situazioni, come autostrade o strade a corsia singola, senza richiedere l'interazione continua del conducente, che può quindi temporaneamente togliere le mani dal volante e consentire al veicolo di operare in modo indipendente, ma deve essere pronto a riprenderne il controllo;

- *Riconoscimento dell'ambiente*: il sistema è in grado di riconoscere l'ambiente circostante, compresi veicoli, pedoni, segnali stradali, semafori e altri oggetti sulla strada;

- *Automated Emergency Braking (AEB)*: è un sistema che aiuta a prevenire incidenti stradali avvertendo il conducente dell'impatto imminente, aiutandolo a fermare l'auto in maniera ottimizzata oppure a fermarla autonomamente se la situazione è più critica;

- *Driver Monitoring (DM)*: è una funzionalità che monitora l'attenzione del conducente sulla strada durante la guida utilizzando sensori a infrarossi o videocamere: se questo non presta abbastanza attenzione e si presenta una situazione di pericolo, il sistema lo avverte con un segnale, che può variare dal luminoso al rumoroso, e se questo non reagisce, il sistema prende il controllo del veicolo portandolo in uno stato di sicurezza;

- *Assistenza nel cambio corsia*: il veicolo è in grado, previa attivazione del conducente, di cambiare corsia autonomamente; alcuni sistemi possono anche riconoscere quando un conducente sta cercando di cambiare corsia e assistere a questo processo;

- *Traffic Jam Assist (TJA)*: è una funzionalità ADAS (Advanced Driver-Assistance System) che aiuta il guidatore a navigare il veicolo attraverso il traffico mantenendo una distanza di sicurezza dai veicoli davanti e reagendo al suo circondario.

Nonostante l'automazione avanzata in questo livello, il conducente deve essere pronto a intervenire in situazioni inaspettate o quando il sistema richiede assistenza, entro un certo periodo di tempo chiamato "periodo di avviso": la transizione dal sistema autonomo al controllo manuale richiede solitamente alcuni secondi.
Per riuscire ad ottenere tutte le funzionalità sopra elencate, i sensori che dovrebbero essere impiegati sono: videocamera a lunga distanza, DR (Dead Reckoning), utilizzato per superare i limiti del gps in caso di perdita del segnale, e sensori lidar, utilizzati per permettere al sistema di creare una mappa tridimensionale dell'ambiente circostante, consentendogli di essere consapevole di ciò che lo circonda.
Bisogna notare però che, a causa delle sfide legali, etiche e di sicurezza discusse alla sezione 1.3, il Livello 3 di guida autonoma presenta ancora diverse questioni aperte e può quindi variare in termini di implementazione da un produttore all'altro.

1.2.5 Livello 4

Il Livello 4 è anche noto come *"Automazione Alta"*: il veicolo è completamente autonomo e può gestire tutte le attività di guida senza richiedere l'intervento del conducente.
La grande differenza tra il Livello 3 e il Livello 4 è che, in una situazione di emergenza, nel primo sarebbe il guidatore a dover intervenire e prendere il controllo del veicolo, mentre nel secondo potrebbe essere il veicolo stesso ad intervenire.
Questo livello gode di determinate proprietà e caratteristiche, tra cui:

- *Guida autonoma senza intervento umano*: il sistema può guidare autonomamente senza richiedere l'intervento del conducente: in determinate situazioni è in grado di gestire tutte le fasi della guida, compresi accelerazione, sterzata, cambio corsia, frenata, parcheggio e altre manovre;

- *Automazione in condizioni specifiche*: il veicolo riesce a operare in modo completamente autonomo in un certo dominio di condizioni specifiche, come autostrade, zone urbane limitate o aree industriali; fuori da queste potrebbe richiedere l'intervento umano o funzionare solo a un livello inferiore di automazione;

- *Riconoscimento dell'ambiente avanzato*: il veicolo è dotato di sensori sofisticati come telecamere, LIDAR, RADAR e sensori ad ultrasuoni, che gli consentono di riconoscere e reagire in tempo reale a una vasta gamma di situazioni stradali, come ad esempio altri veicoli, pedoni, ciclisti, segnali stradali, semafori e altro ancora;

- *Risposta a situazioni complesse*: il sistema è in grado di affrontare situazioni stradali complesse o impreviste, come lavori in corso, incroci affollati, situazioni di emergenza e condizioni meteorologiche avverse, grazie ad algoritmi di intelligenza artificiale e di apprendimento automatico;
- *Assenza di supervisione umana*: durante l'operazione autonoma non è richiesta la supervisione continua da parte del conducente e i passeggeri possono dedicarsi ad altre attivitá durante il viaggio senza doversi preoccupare della guida.

Tuttavia é importante osservare che, nonostante l'alta automazione, il veicolo potrebbe non essere in grado di gestire tutte le situazioni stradali, perciò anche se questo può operare in modo autonomo nella maggior parte delle situazioni, potrebbe comunque richiedere il coinvolgimento umano in circostanze di emergenza o situazioni stradali difficili.

1.2.6 Livello 5

Il Livello 5 è noto come *"Automazione Completa"* e rappresenta il massimo livello di automazione dei veicoli: questi sono completamente autonomi e possono gestire tutte le attività di guida in qualsiasi situazione o ambiente senza richiedere alcun intervento umano.
Le caratteristiche e le funzionalità distintive del Livello 5 sono:

- *Automatic Pilot (AP)*: rappresenta la capacità del veicolo di autoguidarsi, ovvero di effettuare tutte le manovre e di gestire la guida in sicurezza e in modo completamente autonomo, senza bisogno di intervento umano;
- *Nessuna limitazione ambientale*: il sistema può operare in qualsiasi ambiente stradale: autostrade, strade urbane, strade di campagna, senza limitazioni, in più non è influenzato da eventuali condizioni meteorologiche o stradali avverse;
- *Assenza totale di supervisione umana*: non è richiesta la presenza umana all'interno del veicolo durante il viaggio; i passeggeri quindi possono dedicarsi completamente ad altre attività durante il percorso;
- *Riconoscimento e risposta a tutte le situazioni stradali*: il veicolo riesce a riconoscere e reagire in tempo reale a qualsiasi situazione stradale, compresi altri veicoli, pedoni, ciclisti, animali, segnali stradali, semafori, lavori in corso e molto altro; può inoltre affrontare situazioni di emergenza e prendere decisioni complesse per garantire la sicurezza dei passeggeri e degli altri oggetti o persone sulla strada;
- *Comunicazione con altri veicoli e infrastrutture*: il sistema comunica con altri veicoli e con l'infrastruttura stradale, come ad esempio semafori intelligenti e segnali stradali digitali per ottimizzare il flusso del traffico e migliorare la sicurezza stradale.

Si può quindi notare che la differenza principale tra il Livello 4 e il Livello 5 è che in quest'ultimo il veicolo è in grado di rispondere autonomamente a qualsiasi tipo di situazione stradale e di operare in maniera indipendente su ogni tipo di strada senza avere un dominio di funzionamento ristretto; addirittura potrebbe verificarsi il caso che nei veicoli che rientrano nel Livello 5 non ci sia nemmeno la possibilità per un passeggero di prendere il controllo, cioè che a bordo non ci siano né volante né pedali.
Sebbene la realizzazione di veicoli di Livello 5 rappresenti l'obiettivo finale della guida autonoma, al momento questa tecnologia è ancora in fase di sviluppo e testing. Come anche per il Livello 4, ci sono ancora diversi ostacoli da superare, tra cui aspetti legati alla sicurezza, alle normative, alla tecnologia e all'etica, prima che i veicoli completamente autonomi diventino una realtà comune sulle strade di tutto il mondo.

1.3 Ostacoli alla Realizzazione

Allora perché, se le automobili a guida completamente autonoma sembrano poter rappresentare una notevole innovazione nel settore ingegneristico e non solo, non sono ancora diventate una realtà?
La risposta è che questa tecnologia è tanto innovativa quanto controversa: da una parte avrebbe il potenziale di migliorare la vita di milioni di guidatori portando vantaggi non indifferenti, dall'altra purtroppo deve superare ancora diverse barriere.
I benefici che potrebbe recare l'utilizzo di questi veicoli sono molteplici; tra essi si riportano:

- Maggiore sicurezza: in quanto più efficienti di un guidatore umano, porterebbero a una drastica riduzione del numero di incidenti stradali;
- Spese ridotte: si ridurrebbe il consumo dell'auto e aumenterebbe il tempo di conservazione dei componenti;
- Maggiore produttività: il tempo che si dedicherebbe alla guida potrebbe essere impiegato per svolgere attività più importanti;
- Maggiore comfort: gli interni dell'auto sarebbero più spaziosi e confortevoli;
- Avanzamento tecnologico: i VA rappresenterebbero un grande passo avanti nel settore delle conquiste tecnologiche.

Nonostante i vantaggi, si presentano diversi ostacoli alla realizzazione di questi veicoli:

- Legge: ci sono continui dibattiti circa chi incolpare in caso di fallimento del sistema o incidenti, a chi imputare cioè le responsabilità legali;
- Infrastrutture: ad esempio, una strada con le linee delle corsie tracciate male o sbiadite potrebbe rappresentare un problema per il corretto funzionamento del veicolo;

- Pericoli: a un maggiore utilizzo di funzioni automatiche e controllate da un computer corrisponde un maggiore grado di vulnerabilità del sistema nel suo complesso;
- Lavoro: rappresenterebbero una minaccia per gli impiegati del settore dei trasporti, che potrebbero essere sostituiti dai VA e che potrebbero quindi perdere il lavoro;
- Prezzo: ci sarebbe una enorme barriera nei prezzi: inizialmente sarebbero molto elevati, anche se probabilmente in seguito alla popolarizzazione del prodotto potrebbero scendere;
- Etica.

Quello dell'etica è il problema fondamentale che si pone davanti agli ingegneri che si occupano di automazione dei veicoli e nasce dal fatto che il decision-making è affidato a un agente non umano. Principalmente ci si riferisce a situazioni estreme in cui il veicolo è obbligato a fare quella che è considerata una *scelta morale*, come ad esempio una scelta che risulterebbe in diverse sfumature di "vite salvate vs. vite sacrificate": ci si riconduce così al cosiddetto *"Trolley Problem"*.

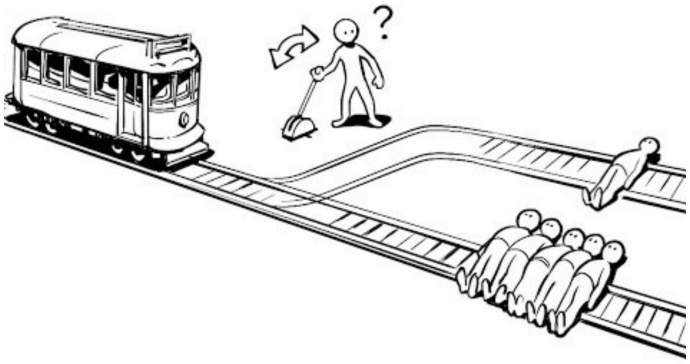

Figura 1.4: Dilemma del Tram

Il *Trolley Problem*, o *Dilemma del Tram*, è un esperimento mentale di filosofia etica formulato nel 1967 da Philippa Ruth Foot che propone un dilemma etico: un autista di un tram conduce un veicolo che può solo cambiare rotaia, ma non frenare. Ci sono due possibili binari; sul binario principale si trovano cinque individui immobilizzati, mentre sul secondo binario c'è una sola persona, anche lei incapace di muoversi. Mentre il tram si sta rapidamente avvicinando a loro, la persona che si trova vicino alla leva del deviatoio é costretta a prendere una decisione difficile: lasciare che il tram continui il suo percorso, causando la morte delle cinque persone, oppure azionare lo scambio e far deviare il tram, risultando nella morte di una sola persona.
Similmente alla situazione descritta dal Dilemma del Tram, il VA potrebbe trovarsi in una condizione in cui deve scegliere tra sacrificare chi si trova a bordo e salvare gli altri o viceversa.

Al giorno d'oggi questo tema dell'etica del decision-making è molto dibattuto e costituisce l'ostacolo principale alla realizzazione di VA di Livello 4 e 5.

1.4 Bosch e BFMC

Bosch combina la sua esperienza in settori come la meccanica, l'elettronica, lo sviluppo hardware e software e la telematica ottenendo come risultato di questa integrazione di competenze la creazione di tecnologie pronte per la produzione e soluzioni complete di sistema che permettono già oggi una guida altamente automatizzata (Livello 4) in modo affidabile e sicuro.
È un produttore leader di sensori e algoritmi per la guida autonoma: l'azienda sviluppa una vasta gamma di sensori avanzati, tra cui telecamere, radar, lidar e ultrasuoni, che permettono ai VA di percepire l'ambiente circostante: questi sensori forniscono dati cruciali sulle condizioni della strada, come la presenza di veicoli, pedoni e ostacoli.
Oltre ai sensori, Bosch sviluppa anche sofisticati algoritmi basati sull'intelligenza artificiale e sull'apprendimento automatico, algoritmi che elaborano i dati provenienti dai sensori per riconoscere pattern, oggetti e situazioni sulla strada. L'elaborazione dati è essenziale per consentire ai VA di prendere decisioni in tempo reale, come fermarsi per evitare una collisione o cambiare corsia in modo sicuro.
Proprio perchè Bosch è un così grande sostenitore della guida autonoma, da qualche anno sta sponsorizzando la Bosch Future Mobility Challenge, una competizione internazionale incentrata sullo sviluppo di algoritmi di guida autonoma per veicoli elettrici in scala 1:10 con Livello 5 di automazione. Il lavoro svolto lungo questa tesi si focalizza quindi sulla realizzazione di un veicolo elettrico in scala 1:10 con Livello 5 di automazione, ovvero completamente autonomo.
Il veicolo perciò è munito di vari sensori, ovvero una camera per l'elaborazione immagini e riconoscimento delle linee della corsia, un encoder per effettuare il controllo di velocità, che deve essere mantenuta costante se non in presenza di una rampa, un IMU per il controllo della sterzata, che deve essere il più fluida possibile, e un sonar, utilizzato per evitare impatti con gli oggetti lungo il percorso; grazie a tali sensori è in grado di navigare autonomamente in un ambiente simulato seguendo una traiettoria prestabilita, sterzando correttamente e interagendo con eventuali oggetti lungo il percorso, come segnaletica stradale di varia natura e pedoni.

1.5 Utilizzo nella Tesi

Il lavoro svolto lungo questa tesi punta perciò alla realizzazione di un veicolo elettrico autonomo in scala 1:10, e all'implementazione su *sistemi embedded* di soluzioni per la sua navigazione attraverso un percorso prestabilito; si tratta quindi di una situazione diversa rispetto a quella presentata per veicoli a grandezza reale.

Per prima cosa, un *sistema embedded* è un tipo di sistema informatico specializzato integrato all'interno di un dispositivo o di una macchina. A differenza dei computer tradizionali, questi sistemi sono progettati per eseguire *specifiche funzioni* o compiti dedicati; sono *incorporati* fisicamente nel dispositivo in cui operano e sono *ottimizzati* per ottenere alte prestazioni nelle attività specifiche per cui sono stati creati. Sono utilizzati in una vasta gamma di applicazioni, come elettrodomestici, veicoli, dispositivi medici e apparecchiature industriali, svolgendo ruoli fondamentali per il funzionamento efficiente di numerosi dispositivi e apparecchiature.

È quindi ovvio che le due situazioni siano molto differenti tra loro. I VA a dimensione reale rappresentano veicoli che operano e navigano in condizioni di traffico del mondo reale: sono sperimentati direttamente sulle strade pubbliche o in aree designate per i test, affrontando situazioni di traffico, pedoni, segnali stradali e altre variabili reali. Questi veicoli devono gestire situazioni complesse, come incroci affollati, condizioni meteorologiche variabili e comportamenti imprevedibili degli altri utenti della strada, ponendo una particolare enfasi sulla sicurezza degli occupanti e degli altri utenti della strada.

D'altra parte, i VA in scala sono modelli ridotti utilizzati per la ricerca e lo sviluppo. Vengono testati in ambienti controllati come piste di prova o simulazioni computerizzate e sono utilizzati per testare specifici componenti o algoritmi, come sensori, telecamere, lidar o algoritmi di controllo, in modo sicuro e senza i rischi associati alla guida su strade pubbliche.

Se da un lato i VA in scala consentono di testare diversi livelli di automazione e componenti in maniera sicura e priva di rischi, sono anche molto più accessibili dal punto di vista economico rispetto ai VA a grandezza originale.

Per i fini della tesi è stato quindi sviluppato un ambiente di simulazione consono per le dimensioni del veicolo utilizzato comprendente strada a doppia corsia, incroci multipli, segnaletica stradale, come attraversamenti pedonali, linee di stop a terra e segnali stradali di vario genere, e pedoni per permettere un corretto testing del sistema. Inoltre il veicolo era equipaggiato con diversi tipi di sensori che gli garantivano di navigare efficacemente lungo il percorso: un encoder, di cui si parla alla sezione 2.1.4, una videocamera, 2.1.10, un IMU, 2.1.11 e un sensore sonar, 2.1.12.

2

Il Sistema

Indice

2.1 Componenti Hardware

Ai fini della Bosch Future Mobility Challenge, è stato fornito a tutti i partecipanti un kit hardware di partenza dagli organizzatori della competizione.
Come si può facilmente notare dall'immagine 2.1, il kit è composto da diverse componenti:

- NucleoF401RE (Controller Board);
- Raspberry Pi 4 Model b (Brain Board);
- VNH5012 H-bridge Motor Driver;
- AMT103 Encoder;
- Motore DC;
- DC/DC Converters;
- Servo motore;
- Batteria;
- Chassis;
- Camera;
- IMU;
- Sensore a ultrasuoni (aggiunto rispetto al kit di partenza e quindi non presente nell'immagine).

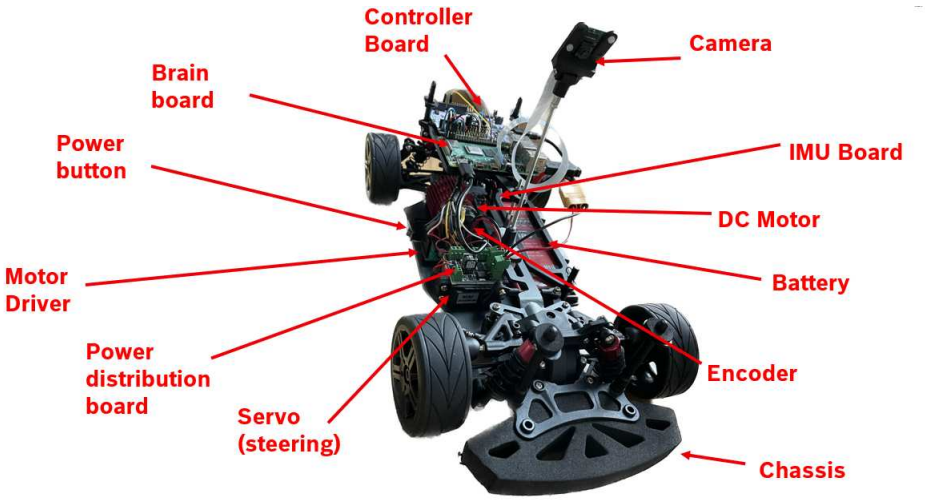

Figura 2.1: Sistema completo

2.1.1 NucleoF401RE

La scheda Nucleo F401RE è un microcontrollore prodotto da STMicroelectronics appartenente alla famiglia STM32 serie F4. Tutte le schede della serie si basano sul processore Arm ® Cortex ®-M4 a 32 bit, che offre prestazioni fino a 84 MHz, include una Floating Point Unit (FPU) per operazioni in virgola mobile e permette di ottenere valori di consumo energetico straordinariamente bassi nelle modalità di esecuzione e arresto.
La scheda offre diverse interfacce di comunicazione, tra cui *GPIO (General Purpose Input/Output)*, *I2C (Inter-Integrated Circuit)*, *SPI (Serial Peripheral Interface)* e *UART (Universal Asynchronous Receiver/-Transmitter)*, di cui si parla al capitolo 5.1, per collegare facilmente sensori, attuatori e altri dispositivi esterni. Dispone di 512 KB di memoria flash e 96 KB di RAM, offrendo ampio spazio per l'esecuzione del codice e l'archiviazione dei dati.
Integrati nella scheda si hanno inoltre:

- 3x USARTs che funzionano a $10,5[Mbit/s]$, per permettere la comunicazione seriale;
- 16x I/O GPIOs;
- 10x TIMERs, da 16 e 32 bit, con frequenza fino a $84[MHz]$;
- 1x USB 2.0 OTG.

Come si puó vedere dalla figura 2.2, la scheda è provvista di due pulsanti, uno blu e uno nero: il primo è programmabile, mentre il secondo serve per eseguire il reset del microcontrollore.
Sono presenti anche tre LED, di cui il LED1 indica la connessione USB, il LED2 è programmabile e il LED3 è acceso quando la scheda è alimentata.

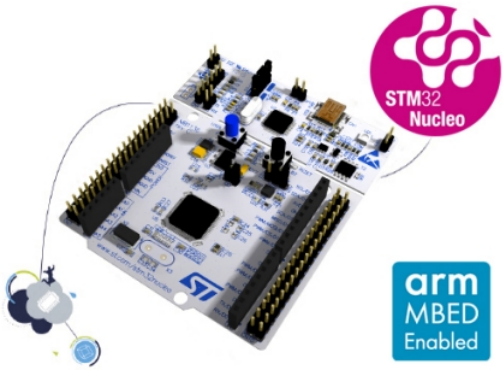

Figura 2.2: Nucelo F401RE

Infine, il microcontrollore può essere alimentato sia tramite USB sia tramite un'alimentazione esterna ed è dotato di un debugger e un programmatore integrato ST-Link/V2-1, che consente di caricare il codice

direttamente sulla scheda. È inoltre compatibile con l'ambiente di sviluppo STM32CubeIDE, un ambiente integrato per la programmazione e il debug del codice, che però non è stato utilizzato per il lavoro svolto in questa tesi.
Per quanto riguarda invece la piedinatura della scheda, che è mostrata a figura 2.3, questa ha complessivamente 64 pin divisi in due set:

1. *Arduino Uno*, costituito da connettori femmina:
    - *CN6* e *CN8* sono dedicati a ingressi e uscite analogici;
    - *CN5* e *CN9* sono dedicati a ingressi e uscite digitali.
2. *St Moprho*, costituito dai connettori maschi *CN7* e *CN10*, che estendono ulteriormente le funzionalità della scheda.

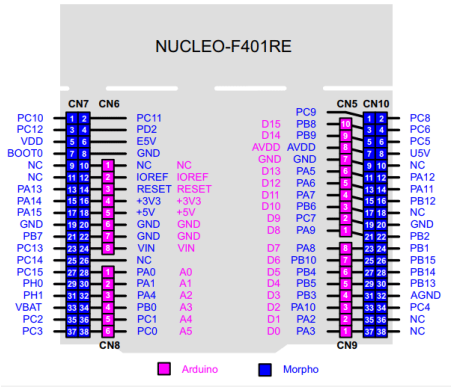

Figura 2.3: Nucleo F401RE Pinout

Nel contesto del lavoro svolto, questa scheda è stata utilizzata come controllore dei motori e dei sensori collegati, che è in grado di governare in maniera corretta grazie all'interpretazione dei comandi ricevuti dalla Raspberry Pi 4 tramite comunicazione seriale, di cui si parla alla sezione 5.1.

Si può trovare lo user manual della scheda al seguente link: https://www.st.com/resource/en/user_manual/um1724-stm32-nucleo64-boards-mb1136-stmicroelectronics.pdf.

2.1.2 Raspberry Pi 4

Si tratta di un mini computer, in particolare di un SoC (System on Chip), un circuito integrato che in un solo chip contiene un intero sistema.
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, 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 e utilizza schede microSD per il sistema operativo e l'archiviazione dati.
Ha anche un header GPIO a 40 pin per connettere diversi tipi di dispositivi elettronici e l'alimentazione avviene tramite un connettore USB-C.

Figura 2.4: Raspberry Pi 4

Grazie all'hardware performante, un sistema operativo leggero (basato su Kernel Linux) e l'utilizzo del linguaggio di alto livello Python è perfetta per l'elaborazione di immagini e video: un compito che richiede un elevato costo computazionale.
Questo infatti è il compito che la Raspberry Pi 4 svolge nel progetto esaminato: l'elaborazione di immagini attraverso una apposita camera (2.1.10), oltre che il decision-making in funzione delle immagini elaborate e dei dati raccolti dall'IMU (2.1.11) e la trasmissione di comandi adeguati al microcontrollore tramite comunicazione seriale.

È possibile trovarne la documentazione al seguente link: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf.

2.1.3 VNH5012 H-bridge Motor Driver

Il VNH5012 è un driver di ponte H utilizzato per controllare motori DC, cioè motori a corrente continua, o motori passo-passo in entrambe le direzioni.
Un "ponte H" è un microcontrollore che deve il nome alla forma del suo circuito elettrico, formato da transistor disposti a forma di H e da un carico posizionato in mezzo a questi.
Questo driver consente di controllare la direzione di rotazione del motore cambiando la polarità della tensione inviata. È in grado di gestire correnti relativamente elevate ed è controllato tramite segnali digitali, compresi i segnali PWM, che permettono di regolare la velocità del motore.

Infine è dotato di protezioni integrate contro sovracorrente, surriscaldamento e inversione di polarità per proteggere sia il driver che il motore da danni.
All'interno del progetto presentato viene utilizzato per applicare una quantità regolabile di potenza a un carico per azionare il motore DC (2.1.5) in entrambe le direzioni, permettendo quindi al veicolo di muoversi avanti o indietro.
Se ne può trovare il datasheet al seguente link: https://www.pololu.com/file/0J504/vnh5019a-e.pdf.

Figura 2.5: H-bridge

2.1.4 AMT103 Encoder

L'encoder in generale è un dispositivo di controllo elettromeccanico che viene impiegato per ottenere un feedback: genera un segnale ad onda quadra che viene interpretato da un controllore.
In particolare, l'encoder AMT103 è di tipo *rotativo incrementale a doppio canale*, ovvero 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 abbastanza semplice: il sensore è costituito da un disco rotante scanalato attaccato all'albero che ruota simultaneamente all'albero, come si vede dall'immagine 2.6.

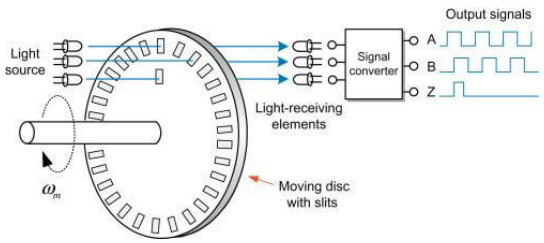

Figura 2.6: Principio di funzionamento dell'encoder

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. Quando il disco ruota, gli slot lasciano passare momentaneamente la luce, così da permetterle raggiungere il ricevitore, che emette un segnale logico alto. Quando il disco ruota ulteriormente, il raggio di luce è ostruito e il ricevitore non riceve il raggio; 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'unità di tempo), mentre con la lettura anche del segnale B si riesce a rilevare il senso di rotazione. È disponibile anche un ulteriore segnale chiamato canale Z, che fornisce una posizione assoluta di riferimento, ma che è inutilizzato.
Come si può vedere dalla figura 2.7 sul dispositivo ci sono 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 2.7: ATM103 Encoder

L'AMT103 fornisce quindi una risoluzione elevata consentendo una misurazione precisa dell'angolo di rotazione.
Grazie a questo encoder, all'interno del progetto 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.

Se ne riporta il datasheet al seguente link: https://www.cuidevices.com/product/resource/pdf/amt10.pdf.

2.1.5 Motore DC

Il motore DC utilizzato per la trazione è il Reely 531009 Spare part Electric Motor.
I motori DC sono dispositivi che trasformano l'energia elettrica in energia meccanica, sono alimentati da una fonte di alimentazione a corrente continua e sono controllati variando la tensione o la corrente applicata ai loro terminali.

Nel contesto del lavoro svolto in questa tesi, il suo scopo è quello di far muovere il veicolo in avanti o all'indietro ad una velocità che viene mantenuta costante grazie all'utilizzo di un controllore *PID*, sezione 3.2, e grazie ai dati forniti dall'encoder.
Come anche il Servo Motore, presentato alla sezione 2.1.7, il suo funzionamento viene controllato attraverso la tecnica *PWM*, tecnica di modulazione del segnale discussa alla sezione 3.1.

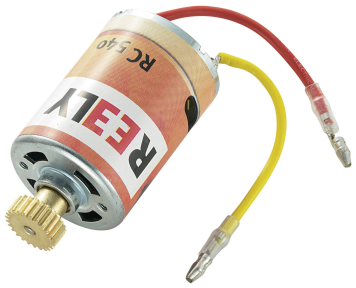

Figura 2.8: Motore DC

Si può trovare il datasheet al seguente link: https://asset.conrad.com/media10/add/160267/c1/-/en/000236699DS01/datasheet-236699-reely-531009-spare-part-electric-motor.pdf.

2.1.6 Convertitori DC/DC

Il convertitore DC/DC utilizzato è l'OKR-T/10-W12-C. Questo è un dispositivo elettronico che converte una tensione continua in un'altra tensione continua. In particolare l'OKR-T/10-W12-C è di tipo "non isolato", cioè non c'è alcun isolamento elettrochimico tra ingresso e uscita.
Questo convertitore serve a generare i 5V per alimentare le varie componenti del sistema ed è caratterizzato da:

- Corrente massima in uscita: 10 A;
- Tensione in ignresso: 4,5 - 14 V;
- Tensione in uscita: 0.591 - 6 V;
- Dimensioni: 16.5 x 10.4 x 7.62 mm.

Se ne trova il datasheet al seguente link: https://www.murata.com/products/productdata/8807038484510/okr-t10-w12.pdf?1675222286000.

Figura 2.9: DC/DC Converter

2.1.7 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 2.10: Servo motore

Controllato dal microcontrollore tramite tecnica di modulazione *PWM* (sezione 3.1), ha lo scopo all'interno del progetto, grazie a un controllore *PID* (sezione 3.2) e alle informazioni fornite dall'*IMU* (2.1.11), di regolare la sterzata in modo che sia il più fluida possibile.

2.1.8 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. È una batteria al litio-polimero (LiPo) con una tensione nominale di 7,4 V e una capacità di 5500 mAh. Questo significa che può fornire una corrente di 5500 mA (o 5,5 A) per un'ora prima di esaurirsi completamente. La batteria è composta da due celle collegate in serie per raggiungere la tensione di 7,4 V.
La specifica "20 C" indica la capacità massima di scarica continua; in questo caso, la batteria può essere

scaricata in modo continuo a una corrente massima di 20 volte la sua capacità nominale (5500 mAh), che equivale a 110 A.

Figura 2.11: Batteria

Infine è contenuta in un involucro morbido (Softcase) per una maggiore flessibilità e facilità di installazione e utilizza un connettore XT90, che è un tipo di connettore ampiamente utilizzato nel mondo del modellismo elettrico per garantire un'adeguata connessione elettrica e una trasmissione efficiente di corrente tra la batteria e il dispositivo o il motore che alimenta.
È possibile trovarne il datasheet al seguente link: 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.

2.1.9 Chassis

Uno chassis è una struttura di supporto su cui vengono montati vari componenti di una macchina, costituisce cioè il telaio principale su cui sono installate altre parti, come il motore, le sospensioni, o altre. 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.
Se ne trova il manuale di utilizzo al seguente link: https://asset.conrad.com/media10/add/160267/c1/-/gl/001
406735ML02/manual-1406735-reely-tc-04-onroad-chassis-110-rc-model-car-electric-road-version-4wd-arr.pdf.

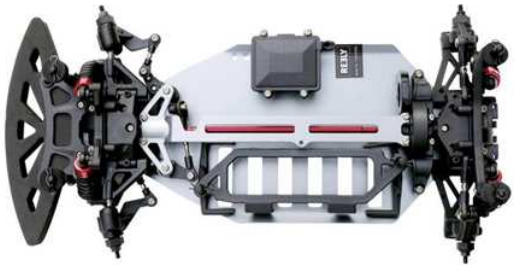

Figura 2.12: Chassis

2.1.10 Camera

La camera utilizzata è la Pi Camera V2.1, sviluppata da Raspberry Pi Foundation per essere utilizzata con le schede Raspberry Pi. Questa telecamera è dotata di un sensore Sony IMX219 da 8 megapixel, il che significa che può catturare immagini ad alta risoluzione con dettagli nitidi; ha anche la capacità di registrare video fino a 1080p a 30 frame al secondo, rendendola ideale per progetti che richiedono la registrazione video di alta qualità. Consente inoltre 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.
La connessione alla scheda Raspberry Pi avviene semplicemente tramite un nastro a 15 pin, che consente una facile integrazione e un'ampia gamma di possibilità di montaggio.

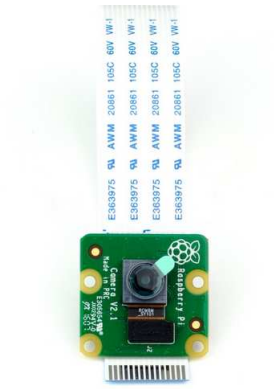

Figura 2.13: Pi Camera

Il suo utilizzo all'interno del lavoro è legato all'elaborazione di immagini: supportata da degli algoritmi di riconoscimento, sia per le linee della corsia sia per altri tipi di oggetto, permette al sistema di riconoscere correttamente il percorso da seguire e gli ostacoli per una navogazione efficiente.

È possibile trovarne la documentazione al seguente link: https://www.raspberrypi.com/documentation/accessories/camera.html .

2.1.11 IMU

Un 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, che 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, permettendo di determinare i cambiamenti nell'orientamento dell'oggetto;
- alcuni IMU sono dotati anche di un magnetometro, che misura il campo magnetico terrestre e fornisce informazioni sull'orientamento rispetto ai punti cardinali.

Tutti i dati dai sensori vengono inviati a un'unità di elaborazione interna all'IMU, che processa i dati grezzi per calcolare quindi orientamento, velocità e posizione dell'oggetto nel tempo. Spesso viene anche utilizzato un filtro di Kalman per migliorarne la precisione, combinando dati misurati e dati previsti. Questo dispositivo comunica con l'utente o dispositivi esterni attraverso diverse interfacce come UART, I2C o SPI.
L'IMU utilizzato nel progetto è il BNO055, prodotto dalla Bosch. Ciò che rende il BNO055 particolarmente potente è la sua capacità di combinare i dati provenienti da tutti e tre i tipi di sensori: accelerometro, giroscopio e magnetometro. 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 nella temperatura e nell'inclinazione, senza richiedere interventi esterni per la calibrazione.
Il suo scopo all'interno del progetto è quello di fornire dati riguardo l'accelerazione del veicolo lungo l'asse y, così da permettere al PID di svolgere un controllo corretto vella velocità di sterzata, mantenendo così fluida la guida.
Se ne trova il datasheet al seguente link: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bno055-ds000.pdf.

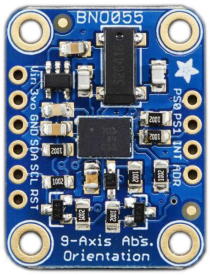

Figura 2.14: IMU

2.1.12 Sensore a ultrasuoni

Il sensore a ultrasuoni utilizzato è l'HC-SR04. Questo, dopo aver emesso un breve impulso sonoro a ultrasuoni, misura il tempo che impiega il segnale a tornare dopo essere riflesso su un oggetto e, conoscendo la velocità del suono nell'aria, calcola la distanza tra se stesso e l'oggetto.
Come si può vedere dall'immagine 2.15, il dispositivo ha 4 pin:

- *VCC*: fornisce l'alimentazione al sensore e di solito è collegato a una sorgente di alimentazione a 5V;
- *Trig (trigger)*: è utilizzato per inviare il segnale di trigger al sensore: quando questo pin riceve un impulso di livello alto, generalmente con durata di 10 microsecondi, il sensore invia una serie di impulsi ad ultrasuoni per misurare la distanza da eventuali oggetti;
- *Echo*: è utilizzato per ricevere il segnale riflesso, di eco: quando il dispositivo riceve un segnale di eco, genera un impulso di durata proporzionale al tempo impiegato dal segnale a tornare;
- *Ground*: è collegato a terra e completa il circuito.

Il funzionamento del sensore può essere quindi suddiviso in tre fasi:

1. *Invio del segnale*: il microcontrollore attiva il sensore inviando un impulso al pin di Trigger;
2. *Ricezione del segnale*: il sensore emette un impulso a ultrasuoni e, una volta che il segnale ritorna, viene generato un impulso sull'Echo pin;
3. *Calcolo della distanza*: utilizzando la durata dell'impulso sull'Echo pin e la velocità nota del suono nell'aria, il microcontrollore calcola la distanza tra il sensore e l'oggetto.

Nonostante gli evidenti vantaggi, cioè le dimensioni ridotte, la facilità d'uso, l'accuratezza e il prezzo accessibile, ci sono però diverse limitazioni. In primo luogo, il sensore è sensibile alle condizioni ambientali, come le variazioni di temperatura e umidità, che possono influenzare la velocità del suono nell'aria e portare a errori nelle misurazioni della distanza.

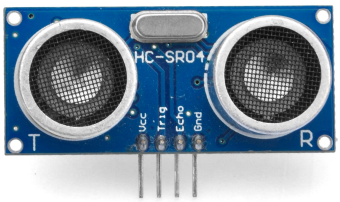

Figura 2.15: Sensore a ultrasuoni HC-SR04

In secondo luogo, il dispositivo ha un angolo di rilevamento limitato, il che significa che è più preciso quando viene utilizzato per misurazioni dirette senza ostacoli laterali: gli oggetti che si trovano al di fuori dell'angolo di rilevamento del sensore potrebbero non essere rilevati correttamente.
Inoltre, il sensore ha una distanza massima di rilevamento, generalmente di alcuni metri: oltre questa distanza, la precisione delle misurazioni diminuisce e potrebbe diventare inaffidabile.
L'HC-SR04 all'interno del progetto è stato utilizzato per implementare la frenata di emergenza nel caso di attraversamento improvviso di pedoni, per realizzare i sorpassi in strade a doppia corsia, per mantenere la distanza di sicurezza da un eventuale veicolo davanti e per realizzare i parcheggi, sia in parallelo che in perpendicolare, senza colpire altri veicoli.
È possibile trovarne il datasheet al seguente link: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwjDmoOGuJaCAxWTgf0HHevCAhEQFnoECBMQAQ&url=https%3A%2F%2Fcdn.sparkfun.com%2Fdatasheets%2FSensors%2FProximity%2FHCSR04.pdf&usg=AOvVaw1iQg0OJ6MFfs9MrkZYGAB4&opi=89978449.

2.2 Integrazione

La figura 2.16 è il diagramma di connessione di tutti i componenti elencati e descritti alla sezione 2.1 che compongono il sistema.

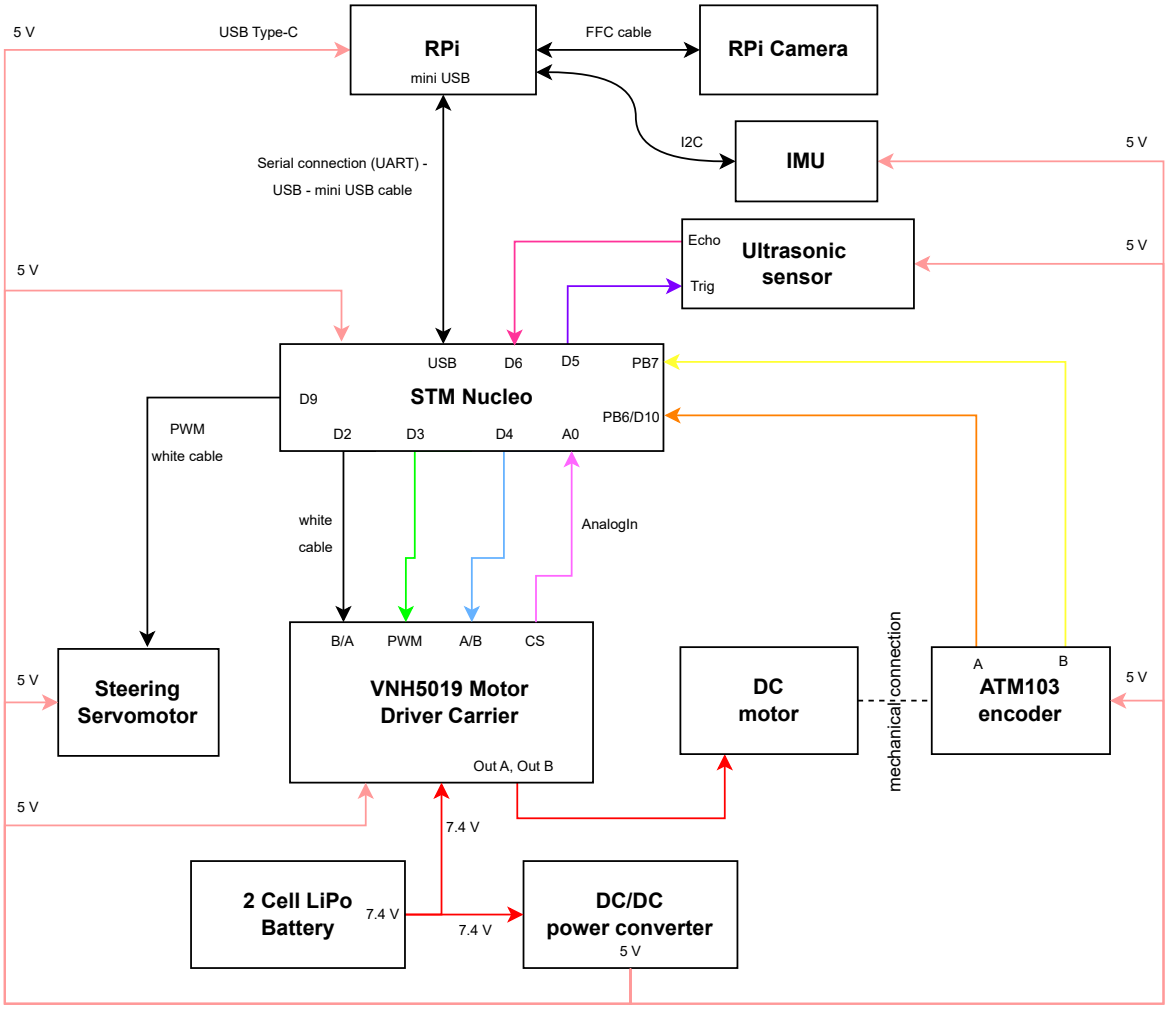

Figura 2.16: Connection Diagram

2.3 Componenti Software: Real-Time OS

Trattandosi di un sistema in cui è cruciale il tema della sicurezza non solo dei passeggeri del veicolo, ma anche delle persone che si trovano all'esterno come discusso alla sezione 1.3, un VA deve essere in grado di reagire in maniera immediata sia in situazioni di emergenza sia in situazioni non altrettanto critiche. Per questo è fondamentale l'utilizzo di Sistemi Operativi Real-Time, *RTOS*.
Un RTOS è un sistema operativo progettato per gestire applicazioni che richiedono risposte istantanee e prevedibili agli eventi; sono quindi ottimizzati per eseguire operazioni in un intervallo di tempo ben definito.
Sono utilizzati in applicazioni critiche come sistemi di controllo industriale, dispositivi medici e, appunto, veicoli autonomi, ovvero in tutte le situazioni in cui la precisione temporale è fondamentale. Gli RTOS assicurano che le attività vengano eseguite in base a priorità prestabilite e gestiscono segnali di interrupt (eventi immediati) in modo efficiente. In sostanza, un RTOS è cruciale in contesti in cui il tempo di risposta rapida è essenziale per il funzionamento sicuro ed efficiente delle applicazioni.
Tutte queste caratteristiche lo distinguono da un Sistema Operativo Desktop (da ora abbreviato in "OS"), come Windows.
Queste due tipologie di sistemi operativi, infatti, differiscono sia nei requisiti sia nell'utilizzo a cui sono destinate. In primo luogo, come già spiegato, gli RTOS sono progettati per garantire risposte prevedibili e immediate agli eventi, soddisfacendo scadenze temporali rigorose e assicurando che le operazioni siano eseguite entro limiti di tempo specifici, mentre gli OS non garantiscono risposte tempestive e prevedibili agli eventi, ma sono ottimizzati per offrire un'esperienza utente interattiva e possono avere variazioni nei tempi di risposta a seconda del carico di lavoro e delle attività in esecuzione.
In secondo luogo, gli RTOS sono generalmente progettati per essere leggeri ed efficienti, focalizzandosi sulla gestione di compiti specifici in modo rapido e deterministico, mentre gli OS sono molto più complessi e pesanti e offrono una vasta gamma di funzionalità, come interfacce grafiche utente, supporto per applicazioni di produttività, connettività di rete avanzata e altro ancora, sono cioè progettati per fornire un'esperienza utente completa e versatile.
Inoltre, gli RTOS sono progettati per gestire le risorse di sistema in modo efficiente, mentre gli OS gestiscono una vasta gamma di risorse, tra cui CPU, memoria, dispositivi di archiviazione, schede grafiche e periferiche di input/output, gestiscono la multimedialità e forniscono un'interazione grafica con l'utente.
Infine, per quanto riguarda la flessibilità, gli RTOS sono spesso progettati per applicazioni specifiche, offrendo perciò una configurazione limitata, mentre gli OS sono altamente personalizzabili e adattabili alle esigenze degli utenti, consentendo l'installazione e l'esecuzione di una vasta gamma di applicazioni software.
I Sistemi Operativi Desktop cioè sono progettati per fornire un ambiente interattivo, flessibile e ricco di funzionalità per gli utenti. Risulta quindi evidente che in situazioni come quelle presentate in un proble-

ma di guida autonoma un OS non sarebbe l'opzione migliore e la scelta cadrebbe inevitabilmente su un RTOS.

2.3.1 Mbed OS

Sebbene la situazione presentata lungo questa tesi non sia paragonabile come livello di criticità ad una reale, si è comunque scelto di lavorare con un RTOS, in quanto il veicolo deve prendere decisioni in tempo reale e avere un ritardo minimo tra richiesta e azione in casi come reazioni a segnali stradali, frenata per far attraversare un pedone e scelta di che strada prendere ad un incrocio.
L'RTOS utilizzato per il microcontrollore è Mbed OS. Mbed OS è un sistema operativo sviluppato da Arm, una delle principali società di progettazione di semiconduttori nel mondo open source, per dispositivi IoT (Internet of Things) basato su microcontrollori.
Una delle sue principali caratteristiche è che supporta una grande varietà di dispositivi: è compatibile con una vasta gamma di microcontrollori e piattaforme hardware, ciò significa che gli sviluppatori possono utilizzarlo per molti e diversi progetti IoT, indipendentemente dal tipo di dispositivo utilizzato.
Un'altra caratteristica importante è la sicurezza integrata: infatti Mbed OS include protocolli di crittografia e supporto per connessioni sicure, garantendo che i dispositivi siano protetti da minacce esterne. Offre inoltre diverse opzioni di connettività, come Wi-Fi, Bluetooth Low Energy (BLE) e tecnologie cellulari.
Infine, questo RTOS semplifica lo sviluppo fornendo una vasta gamma di librerie, protocolli e strumenti di sviluppo: é possibile utilizzare l'ambiente di sviluppo di Mbed, Mbed Studio, per scrivere, compilare e debuggare il codice, rendendo il processo di sviluppo più efficiente e accessibile.
Nel contesto del lavoro svolto, Mbed OS permette di gestire il multithreading: infatti, grazie all'utilizzo della sua classe 'Timer', vengono eseguiti ogni 1 ms i metodi che implementano le funzionalità principali della macchinina, come il calcolo dei segnali di controllo dei PID (sezioni 3.3 e 3.4) e il calcolo degli rps dell'encoder (sezione 4.3).

3

Trazione e Sterzata

Indice

3.1 Tecnica PWM

La modulazione a larghezza di impulso, o *PWM (Pulse Width Modulation)*, è una tecnica utilizzata per convertire un segnale analogico in un segnale digitale.
Nel processo di PWM, un segnale di riferimento periodico, nella figura 3.1 rappresentato in blu, viene confrontato con un segnale variabile, rappresentato in viola. La durata degli impulsi del segnale di uscita varia a seconda di quanto il segnale di riferimento supera quello variabile: questa variazione permette di rappresentare il segnale analogico in formato digitale.

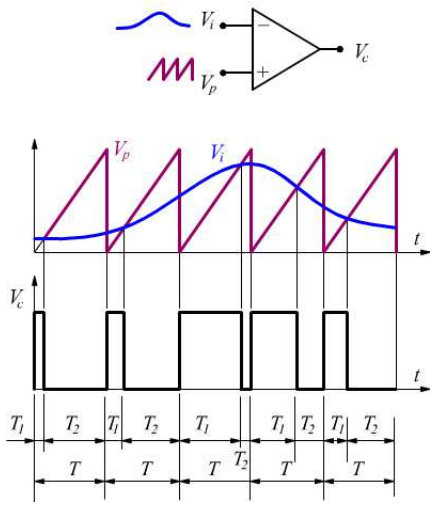

Figura 3.1: PWM

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 è in uno stato alto rispetto al periodo totale: un duty cycle più elevato infatti comporta un segnale di uscita con un'ampiezza maggiore, mentre un duty cycle più basso produce un segnale con un'ampiezza inferiore.
La tecnica PWM offre vantaggi come l'efficienza energetica e la facilità di controllo, rendendola una scelta popolare in molte applicazioni.
All'interno del progetto presentato in questa tesi la tecnica PWM è stata utilizzata per il controllo dei motori sia DC (2.1.5), per mantenere il veicolo alla velocità giusta, che servo (2.1.7), per ottenere una curvatura corretta.

3.2 Controllori PID

I *controllori PID (Proporzionale-Integrativo-Derivativo)* sono il tipo di controllori lineari più utilizzato in ingegneria e automazione per regolare i sistemi dinamici. Il loro impiego consente di controllare in modo soddisfacente un'ampia gamma di processi; per questo sono ampiamente utilizzati in vari settori industriali per mantenere variabili come temperatura, pressione, velocità e posizione ai valori desiderati. I controllori PID utilizzano tre componenti principali per calcolare l'uscita del sistema di controllo:

- *Proporzionale (P)*: questa componente è proporzionale all'errore *e* tra il segnale di riferimento, ovvero il valore desiderato, e il valore misurato del sistema: un valore proporzionale più alto significa una risposta più forte alle variazioni dell'errore attuale;
- *Integrativo (I)*: questa componente è proporzionale all'integrale dell'errore *e*, cioè al suo valor medio, e 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 lo scopo di tentare di 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 nel sistema e di mantenere la variabile controllata il più vicino possibile al valore desiderato.
I parametri del controllore PID *Kp, Ki* e *Kd*, rispettivamente *proporzionale, integrativo* e *derivativo*, devono essere ottimizzati per adattarsi al sistema specifico che si sta controllando 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 come 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 più ampia possibile;
- *Regolatori PID*: hanno il contributo di tutte e tre le componenti, *Kp, Ki e Kd*.

All'interno del progetto sviluppato lungo questa tesi sono stati utilizzati due regolatori PID: uno per il mantenimento di una velocità longitudinale consona, il cui utilizzo è presentato alla sezione 3.3, l'altro per regolare la sterzata del veicolo in modo da renderla il più precisa e fluida possibile, il cui utilizzo è presentato alla sezione 3.4.
Il codice realizzato per la classe dei controllori PID nel progetto si trova al path *"include/signal/sisocontrollers.tpp"* [1].
In particolare se ne riporta il costruttore, che permette di costruire un controllore a partire dai tre parametri proporzionale, integrativo e derivativo e di calcolarne i coefficienti della funzione di trasferimento basandosi sul metodo "Euler Backward". Il costruttore, come si vede a riga 9, introduce anche la "derivative time filter constant", costante che viene utilizzata per implementare il termine derivativo del controller PID. Questa contribuisce alla stabilità del sistema controllato, anticipando e mitigando gli effetti delle variazioni istantanee dell'errore. Rappresenta cioè un filtro derivativo e serve ad attenuare il problema dell'amplificazione del rumore presente nel sistema causato dalla derivazione del segnale di errore.

```
1  /** @brief CPidController class constructor
2    *
3    * Constructs a new discrete pid controller, it requires the pid parameters to calculate
4    * the discrete transferfunction.
5    *
6    * @param f_kp                proportional factor
7    * @param f_ki                integral factor
8    * @param f_kd                derivative factor
9    * @param f_tf                derivative time filter constant
10   * @param f_dt                sampling time
11   */
12 template<class T>
13 CPidController<T>::CPidController( T              f_kp
14                                    ,T             f_ki
15                                    ,T             f_kd
16                                    ,T             f_tf
17                                    ,T             f_dt)
18     :m_pidTf()
19 {
20     // Calculates the coefficients for the discrete transferfunction based on Euler's
21     // backward discretisation method.
22     utils::linalg::CMatrix<T,1,3> l_numPid({ (f_kd+f_tf*f_kp)/f_tf ,
```

[1] Tutto il codice presentato è stato scritto in C++ e può essere trovato alla repository Git-Hub riportata alla sezione 6.2.

```
23          (f_tf*f_dt*f_ki+f_dt*f_kp-2*f_tf*f_kp-2*f_kd)/f_tf ,
24          (f_kd+f_tf*f_kp+f_dt*f_dt*f_ki-f_dt*f_tf*f_ki-f_dt*f_kp)/f_tf});
25      utils::linalg::CMatrix<T,1,3> l_denPid({ 1.0,-(2*f_tf-f_dt)/f_tf,(f_tf-f_dt)/f_tf });
26
27      m_pidTf.setNum(l_numPid.transpose());
28      m_pidTf.setDen(l_denPid.transpose());
29  }
```

All'interno dello stesso codice è possibile trovare la funzione "calculateControl": questa è la responsabile del calcolo del segnale di controllo del regolatore a partire dall'errore calcolato tra il valore di riferimento e il valore attuale.

```
1  template<class T>
2  T CPidController<T>::calculateControl(const T& f_input)
3  {
4      return m_pidTf(f_input);
5  }
```

3.3 Controllo della Trazione

Come anticipato, uno dei due PID realizzati ha il compito di mantenere costante la velocità longitudinale a cui si desidera che il veicolo si muova, di governare cioè il motore DC.
Il calcolo del segnale di controllo del regolatore PID responsabile di questo movimento avviene tramite il calcolo dell'errore tra il valore di riferimento della velocità desiderata e la velocità effettiva del veicolo, entrambe espresse in *Rps, rotazioni al secondo*, ottenuta grazie all'encoder in base alla sua velocità di rotazione.
Il codice realizzato per effettuare questo controllo si trova al path *"src/signal/controllers/motorcontroller.cpp"*. In particolare, la funzione che lo governa è la funzione *"control"*, che viene riportata di seguito.

```
1  int8_t CMotorController::control()
2  {
3      float  l_MesRps = m_encoder.getSpeedRps();
4      bool   l_isAbs = m_encoder.isAbs();
5      float  l_ref;
```

```
6       if(std::abs(l_MesRps) > m_mes_abs_sup) {
7           m_RefRps = 0.0;
8           m_u = 0.0;
9           return -1;
10      }
11      if(std::abs(m_RefRps) < m_ref_abs_inf && std::abs(l_MesRps) < m_mes_abs_inf ){
12          m_u = 0.0;
13          return 1;
14      }
15      if ( l_isAbs ){
16          l_ref = std::abs(m_RefRps);
17      } else {
18          l_ref = m_RefRps;
19      }
20      float l_error=l_ref-l_MesRps;
21      float l_v_control = m_pid.calculateControl(l_error);
22      float l_pwm_control = converter(l_v_control);
23
24      if(m_nrHighPwm>m_maxNrHighPwm && l_MesRps==0){
25          m_pid.clear();
26          m_RefRps = 0.0;
27          m_u = 0.0;
28          m_nrHighPwm = 0;
29          return -2;
30      }
31
32      m_u=l_pwm_control;
33
34      return 1;
35  }
```

La funzione “control” è la funzione principale per il controllo della velocità longitudinale e proprio per questo viene eseguita ogni periodo di tempo prestabilito in modo da garantire un controllo continuo.
In particolare, il sistema acquisisce la velocità di rotazione dell’encoder, misurata in rotazioni per secondo (Rps), e implementa vari controlli. I primi due controlli, alle righe 6 e 11 rispettivamente, sono finalizzati a verificare che la velocità effettiva non superi i limiti prestabiliti, sia superiore che inferiore in modo che, in caso contrario, venga disattivato il controllore. Il terzo controllo, a riga 15, è invece volto a determinare se l’encoder è in grado di fornire una velocità orientata, ossia se può distinguere

se il veicolo si muove in avanti o all'indietro, oppure se la restituisce in valore assoluto. Nel contesto di questa tesi l'encoder è in grado di fornire una velocità orientata.
In seguito viene calcolato l'errore tra il valore di riferimento assegnato alla velocità di rotazione del motore DC e il valore effettivo misurato dall'encoder e viene richiamata la funzione della classe del PID per calcolare il segnale di controllo, che viene convertito poi tramite in un segnale PWM per il controllo del motore.
Viene poi effettuata un'ulteriore verifica per vedere se l'encoder sta misurando correttamente la velocità oppure no e conseguentemente se il segnale di controllo ha un valore troppo elevato.
Infine viene assegnato il valore della variabile di controllo a un'altra variabile "m_u", che verrà utlizzata dalla parte di codice dedicata agli attuatori, 4.1, per l'invio del segnale al driver.

3.4 Controllo della Sterzata

Il secondo PID utilizzato ha la funzione di regolare la sterzata con il fine di renderla il più fluida e precisa possibile. Questo controllo avviene tramite l'utilizzo del segnale prodotto dal PID sulla base dell'errore tra il valore di riferimento della velocità di yaw e la velocità di yaw effettiva calcolata tramite l'IMU, dove la *velocità di yaw* è la velocità angolare proprio rispetto all'angolo di yaw, definito come l'angolo tra la direzione in cui il veicolo è orientato e la direzione in cui il veicolo sta effettivamente andando.
Il codice realizzato per effettuare questo controllo si trova al path *"src/signal/controllers/steercontroller.cpp"*. La struttura del controller utilizzato è molto simile a quella del controller a sezione 3.3 e di seguito ne viene illustrata la funzione più rilevante: "control".

```
1  int8_t SteerController::control()
2  {
3      float l_ref;
4      float l_error = m_yawRef - yaw_rate;
5      float l_v_control = s_pid.calculateControl(l_error);
6      float l_pwm_control = conversion(l_v_control);
7
8      // Verifies that the pwm value is in the limits
9      if(l_pwm_control<m_inf_pwm)
10     {
11         pwm_value = m_inf_pwm;
12     }
13     else
14     {
```

```
15          if (l_pwm_control>m_sup_pwm)
16          {
17              pwm_value = m_sup_pwm;
18          }
19          else
20          {
21              pwm_value = l_pwm_control;
22          }
23       }
24      return 1;
25  };
```

La funzione “control” è la funzione principale della classe “SteerController”, perciò viene eseguita ogni periodo di tempo prestabilito in modo da garantire un controllo continuo, come per quella a sezione 3.3, ed è strutturata anche in maniera simile a quella per la trazione.
Calcola infatti l’errore tra il valore di riferimento della velocità di yaw e il valore effettivo misurato tramite l’IMU, richiama la funzione del PID per calcolare il segnale di controllo e lo converte poi attraverso il convertitore in un segnale PWM.
Infine verifica se il segnale PWM generato si trova all’interno dei limiti consentiti e lo salva all’interno della variabile “pwm_value”, che verrà utlizzata dalla parte di codice dedicata agli attuatori, 4.2, per l’invio del segnale al motore.

4

Motori e Sensori

Indice

4.1 Motor Driver e Motore DC

Il motore DC, come visto alla sezione 3.3, è controllato tramite la tecnica PWM. Rispetto però a quello al paragrafo 3.3, il codice presentato in questa sezione, che si trova al path *"src/hardware/drivers/dcmotor.cpp"*, si trova a un livello più basso: questo infatti permette al microcontrollore di interfacciarsi con il Motor Driver (sezione 2.1.3), che a sua volta è collegato al motore DC, e di occuparsi dell'invio del segnale PWM al driver per il controllo della trazione.
In particolare, di seguito sono riportate le funzioni più importanti della classe "CMotorDriverVnh".
Per primo il costruttore della classe: questo prende come parametri i tre pin del microcontrollore a cui è collegato il driver, di cui il primo è il pin di output per il segnale PWM e gli altri due sono i pin di output, A e B, che servono per specificare la direzione in cui il veicolo deve procedere, in particolare ad A corrisponde la direzione avanti e a B indietro. Prende inoltre come parametri il limite massimo di velocità sia in avanti sia all'indietro a cui è consentito al veicolo di muoversi.

```
1  CMotorDriverVnh::CMotorDriverVnh(PinName f_pwmPin, PinName f_inaPin,
2          PinName f_inbPin, float f_pwm_inf_limit, float f_pwm_sup_limit)
3      :m_pwm(f_pwmPin)
4      ,m_ina(f_inaPin)
5      ,m_inb(f_inbPin)
6      ,m_pwm_inf_limit(f_pwm_inf_limit)
7      ,m_pwm_sup_limit(f_pwm_sup_limit)
8  {
9      m_pwm.period_us(200);
10 }
```

Come seconda la funzione più importante della classe "CMotorDriverVnh": "SetSpeed", che serve per il vero e proprio controllo del motore. Viene per prima cosa effettuata una verifica sul valore di PWM generato: a un valore negativo corrisponde che il veicolo si muova all'indietro, perciò al pin A viene assegnato valore 0 e al pin B 1, e a un valore positivo corrisponde che il veicolo si muova in avanti, perciò al pin A viene assegnato valore 1 e al pin B 0.
Infine viene scritto sul pin di PWM il valore assoluto del segnale di controllo: questo permetterà al Motor Driver di controllare la rotazione del motore DC.

```
1  void CMotorDriverVnh::setSpeed(float f_pwm)
2  {
3      if (f_pwm < 0) //backward
```

```
4       {
5           m_ina = 0;
6           m_inb = 1;
7       }
8       else
9       {
10          m_ina = 1;
11          m_inb = 0;
12      }
13      m_pwm = std::abs(f_pwm);
14  }
```

È anche è stato implementato un metodo per il caso specifico in cui il veicolo si debba fermare completamente, come ad esempio di fronte a un segnale di stop o a un pedone. Questo metodo assegna a entrambi i pin, A e B, valore zero e scrive "0.0", che corrisponde a velocità nulla, anche sul pin di PWM.

```
1  void CMotorDriverVnh::brake()
2  {
3       m_ina = 0;
4       m_inb = 0;
5       m_pwm.write(0.0);
6  }
```

4.2 Servo Motore

Anche il Servo Motore è controllato tramite la tecnica PWM, come visto alla sezione 3.4. Il codice presentato in questo paragrafo però, che si trova al path *"src/hardware/drivers/steeringmotor.cpp"*, a differenza di quello a sezione 3.4 si trova a un livello più basso: permette infatti al microcontrollore di interfacciarsi direttamente con il motore e di occuparsi dell'invio del segnale PWM per controllarlo.
Di seguito vengono analizate le funzioni più importanti della classe *"CSteeringMotor"*.
La prima che si incontra è il costruttore della classe: questo prende come parametri il pin di output a cui è collegato il motore, che è per il segnale PWM, e i limiti di sterzata negativa e positiva rispetto alla posizione neutra, che corrisponde a quella con le ruote del veicolo dritte, espressi in gradi, rispettivamente $+30°$ e $-30°$. Inoltre assegna come posizione iniziale dello sterzo quella neutra, scrivendo sul pin di output "0.07525", che è il valore di PWM corrispondente a questa configurazione.

```
1  CSteeringMotor::CSteeringMotor(PinName f_pwm, float f_inf_limit, float f_sup_limit)
2      :m_pwm(f_pwm)
3      ,m_inf_limit(f_inf_limit)
4      ,m_sup_limit(f_sup_limit)
5  {
6      m_pwm.period_ms(20);
7      // Set position to zero
8      m_pwm.write(0.07525);
9  }
```

La funzione che serve per il vero e proprio controllo del motore però e la funzione “setAngle” riportata di seguito. Per prima cosa viene effettuata una verifica sul valore di angolo di sterzata generato per controllare che sia compreso all’interno del range di valori consentito, ovvero tra i $+30°$ e i $-30°$; viene poi scritto sul pin di output il valore corrispondente all’angolo convertito in segnale PWM.

```
1  void CSteeringMotor::setAngle(float f_angle)
2  {
3      if(f_angle < m_inf_limit)
4      {
5          f_angle = m_inf_limit;
6      }
7      if(f_angle > m_sup_limit)
8      {
9          f_angle = m_sup_limit;
10     }
11     m_pwm.write(conversion(f_angle));
12 }
```

4.3 Encoder

L’encoder è il sensore che nel progetto viene utilizzato come strumento per il controllo della velocità a cui si muove il veicolo. In particolare, grazie a un timer contatore, è possibile calcolare la velocità dell’encoder espressa in *Rps, rotazioni al secondo*, in base al numero di impulsi contati in un periodo di tempo, così da permettere al controller a sezione 3.3 di controllare la velocità del veicolo.
Il codice realizzato per l’encoder si trova al path *“src/hardware/encoders/quadratureencoder.cpp”* e di

seguito ne vengono presentate le funzioni principali.
La prima è il costruttore della classe “CQuadratureEncoder”, che prende come parametri la durata del task del dispositivo espressa in secondi, il puntatore all’oggetto timer che viene utilizzato e la risoluzione a cui l’encoder deve lavorare. Quello utilizzato all’interno del progetto però, e perciò riportato di seguito, è il secondo costruttore che si incontra nel codice: quello della classe “CQuadratureEncoderWithFilter”. Questo prende gli stessi parametri del primo aggiungendo il riferimento a un filtro che viene utilizzato con lo scopo di filtrare il rumore associato all’encoder.

```
1 CQuadratureEncoderWithFilter::CQuadratureEncoderWithFilter(
2         float                           f_period_sec
3         ,hardware::encoders::IQuadratureCounter_TIMX*        f_quadraturecounter
4         ,uint16_t                       f_resolution
5         ,signal::filter::IFilter<float>&       f_filter)
6         :CQuadratureEncoder(f_period_sec,f_quadraturecounter,f_resolution)
7         ,m_filter(f_filter)
8 {
9 }
```

Si può quindi dedurre che ci sono due possibili classi di implementazione dell’encoder, una con il filtro per il rumore e una senza.
La funzione principale per l’utilizzo del sensore è la funzione “run”, di cui ne esiste una per ogni classe. In particolare la funzione “run” della classe “CQuadratureEncoder” viene eseguita ogni periodo stabilito nel costruttore in modo da ottenere un controllo continuo grazie alla funzione “startTimer”, che lega appunto l’esecuzione della prima al periodo specificato.

```
1 void CQuadratureEncoder::startTimer(){
2     m_timer.attach(mbed::callback(this,&CQuadratureEncoder::_run), m_taskperiod_s);
3 }
```

La “run” permette quindi di prendere e memorizzare il numero di impulsi contati dall’encoder nell’utlimo periodo e di resettare il contatore.

```
1 void CQuadratureEncoder::_run(){
2     m_encoderCnt = m_quadraturecounter->getCount();
3     if (m_encoderStpsEnable)
4     {
```

```
5           m_encoderStps += m_encoderCnt;
6       }
7       m_quadraturecounter->reset();
8   }
```

La stessa funzione, ma per la classe “CQuadratureEncoderWithFilter”, richiama invece la prima ‘run” e applica il filtro sul valore ottenuto.

```
1  void CQuadratureEncoderWithFilter::_run(){
2      CQuadratureEncoder::_run();
3      float temp = m_encoderCnt;
4      m_encoderCntFiltered = static_cast<int16_t>(m_filter(temp));
5  }
```

L’ultima funzione che si riporta, fondamentale per la gestione delle informazioni fornite dall’encoder, è “getSpeedRps”. Questa permette, a partire dal numero di impulsi ottenuti grazie alle funzioni ‘run”, di ottenere l’effettiva velocità dell’encoder espressa in Rps che verrà utilizzata dal controller di sezione 3.3 per effettuare il controllo della trazione.

```
1  float CQuadratureEncoder::getSpeedRps(){
2      return static_cast<float>(m_encoderCnt)/ m_resolution / m_taskperiod_s;
3  }
```

4.4 Sonar

Il sensore a ultrasuoni viene utilizzato all’interno del progetto come strumento per la localizzazione di oggetti lungo il percorso affrontato dal veicolo. Il suo funzionamento viene spiegato a sezione 2.1.12.
In particolare viene effettuato un controllo continuo per cui, in base alla distanza misurata di un oggetto dal veicolo, questo decrementa la sua velocità fino a fermarsi. Diversamente, in caso ad esempio di un pedone che attraversa la strada in maniera improvvisa, viene implementata la frenata di emergenza e il veicolo si ferma del tutto subito.
Il codice realizzato per questo sensore si trova al path *“src/hardware/sensors/sonar.cpp”* e di seguito ne vengono illustrate le funzioni principali.
La prima funzione è costruttore della classe “sonar”, che prende come parametri il riferimento all’oggetto per la comunicazione seriale, i due pin rispettivamente di *trig* ed *echo*, l’inetrvallo di tempo per la ripe-

tizione della generazione dell'impulso espresso in secondi e il tempo dopo il quale il sensore smette di attendere una risposta dall'oggetto o dalla superficie riflettente, il "time_out", anche questo espresso in secondi. Il costruttore abbina il segnale logico alto nel pin di echo all'esecuzione della funzione *"starT"*, il segnale logico basso sul pin di echo all'esecuzione della funzione *"updateDist"* e abilita l'*IRQ (Interrupt Request)* dello stesso pin in modo che possa generare un interrupt.

```
sonar::sonar(RawSerial& f_serialPort, PinName trigPinPinName echoPin,
        float update_Speed, float time_out)
    :triggerP(trigPin)
    ,echoP(echoPin)
    ,m_serialPort(f_serialPort)
{
    updateSpeed = update_Speed;
    timeout = time_out;
    echoP.rise(this,&sonar::starT);
    echoP.fall(this,&sonar::updateDist);
    echoP.enable_irq();
}
```

Il metodo che fa partire il contatore che permette sia di ripetere l'esecuzione delle funzioni sia di ottenere i valori di tempo è "startTimer".

```
void sonar::startTimer()
{
    timer.start(); // start the timer
}
```

La funzione che, però, all'accensione del veicolo fa effettivamente partire la misurazione è "startUpdates", che richiama la funzione "trigger".

```
void sonar::startUpdates(void)
{
    trigger();
}
```

La funzione “trigger” genera un impulso sul pin di trig della durata di 10 microsecondi, impulso che, riflesso, permette di misurare la distanza degli oggetti. Inoltre abbina alla funzione un valore di timeout, ovvero il periodo di tempo dopo il quale il sensore smette di attendere una risposta dalla superficie riflettente: questo timeout è una misura di sicurezza che impedisce al sonar di rimanere in uno stato di attesa indefinita nel caso in cui non riceva alcun segnale riflesso.

```
1 void sonar::trigger(void)
2 {
3     tout.detach();
4     triggerP = 1;
5     wait_us(10);
6     triggerP = 0;
7     tout.attach(this,&sonar::trigger, timeout);
8 }
```

In seguito si incontra la funzione “starT”, richiamata ogni volta che sul pin di echo si trova un segnale logico alto. Questa è legata all’inizio del processo di calcolo della distanza di un oggetto dal veicolo: inizialmente controlla se il contatore ha raggiunto il suo valore massimo, ovvero 600 secondi, e se è così lo resetta; successivamente salva nella variabile “start” il valore in microsecondi del momento in cui sul pin di echo torna il segnale generato dal sensore dopo essere stato riflesso da una superficie, che corrisponde appunto a un segnale logico alto.

```
1 void sonar::starT()
2 {
3     if(timer.read()>600)
4     {
5         timer.reset ();
6     }
7     start = timer.read_us();
8 }
```

La funzione “updateDist”, invece, viene eseguita ogni volta che sul pin di echo il segnale logico passa da alto a basso: viene salvato nella variabile “end” il valore in microsecondi della durata del segnale logico alto, che corrisponde al tempo impiegato dall’impulso a tornare indietro dopo essere stato riflesso sull’oggetto, e viene calcolata la distanza sulla base delle variabili “start” ed “end” e della velocità del suono nell’aria. Infine si specifica che l’esecuzione della funzione “trigger” deve avvenire ogni periodo

di tempo indicato nel costruttore.

```
1 void sonar::updateDist()
2 {
3     end = timer.read_us();
4     done = 1;
5     distance = (end - start)*0.03431/2;
6     tout.detach();
7     tout.attach(this,&sonar::trigger, updateSpeed);
8 }
```

Infine, è possibile mettere in pausa la misurazione richiamando la funzione “pauseUpdates”, che ferma la ripetizione della chiamata alla funzione “trigger”.

```
1 void sonar::pauseUpdates(void)
2 {
3     tout.detach();
4     echoP.rise(NULL);
5     echoP.fall(NULL);
6 }
```

5

Attuazione dei Comandi

Indice

5.1 Comunicazione Seriale

La *Comunicazione Seriale* prende il suo nome dalla modalità di trasmissione delle informazioni, in cui il dispositivo mittente invia i dati al dispositivo destinatario un bit alla volta, cioè in modo sequenziale.
Esistono diverse modalità di comunicazione seriale, ognuna caratterizzata da un tipo di sincronizzazione: *sincrona*, *asincrona* e *isocrona*.
In questa tesi è stata adottata la comunicazione asincrona, che offre una maggiore flessibilità e precisione nella trasmissione delle informazioni, adattandosi alle variazioni nei tempi di trasmissione e contribuendo al successo del progetto.

5.1.1 Comunicazione Asincorna

Nel contesto della comunicazione seriale asincrona non c'è alcuna sincronizzazione tra il mittente e il ricevitore; invece i dati vengono inviati in pacchetti separati, ciascuno con un proprio segnale di sincronizzazione. Ogni pacchetto inviato contiene un *bit di start*, che indica all'apparecchio ricevente l'inizio della trasmissione, i dati da inviare, generalmente otto bit, e potrebbe anche essere incluso un *bit di parità*, utilizzato per rilevare eventuali errori nella trasmissione; infine ci sono i *bit di stop*, che segnalano al ricevitore la fine del pacchetto.
Tuttavia, quando si inviano grandi quantità di dati, la comunicazione asincrona può risultare meno efficiente, poiché vengono trasmessi più volte bit che non contengono informazioni utili come i bit di start e di stop. Questo può ridurre l'efficienza della trasmissione, specialmente quando è importante ottimizzare la velocità e l'efficacia della comunicazione dei dati.
Nella comunicazione asincrona un bit è definito da un intervallo di tempo determinato dalla velocità di trasmissione (*Baud Rate*): ad esempio, a 115200 baud (bit al secondo o bps), un singolo bit dura 1/115200 secondi, circa 8.68 microsecondi (µs). In un contesto asincrono, a partire dall'inizio della comunicazione, ogni 8.68µs si avrà un nuovo bit nel pacchetto; pertanto, campionando a intervalli di 8.68µs, il ricevitore può distinguere chiaramente un bit da un altro nel flusso di dati in arrivo. Questo processo di campionamento regolare consente una corretta interpretazione dei dati trasmessi.

5.1.2 Protocollo UART

Per la comunicazione asincrona esistono vari standard utilizzati, tra cui il *protocollo UART* (*Universal Asynchronous Receiver/Transmitter*), utilizzato in questa tesi. I pacchetti di dati trasmessi con questo protocollo sono generalmente composti da un *bit di start (Start Bit)* che indica l'inizio della trasmissione, *otto bit di dati (Word Length)* che rappresentano l'informazione trasmessa, un *bit di parità (Parity Bit)* utilizzato talvolta per la verifica degli errori nella trasmissione, e uno o due *bit di stop (Stop Bits)* che segnalano la fine del pacchetto.

La comunicazione avviene attraverso due canali dedicati: un canale di *trasmissione (TX)* e un canale di *ricezione (RX)* collegati tra i dispositivi coinvolti come mostrato in figura 5.1, cioè il canale di trasmissione di un dispositivo è collegato al canale di ricezione dell'altro e viceversa. Questo collegamento incrociato tra i canali consente ai dispositivi di comunicare in modo bidirezionale, trasmettendo e ricevendo dati in modo asincrono.

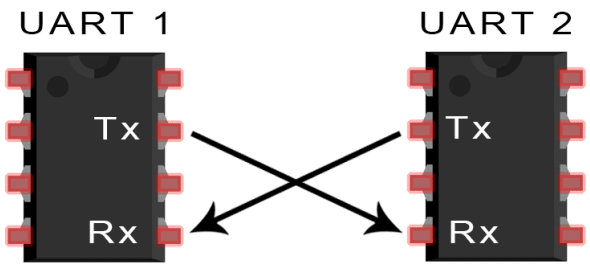

Figura 5.1: Connessione dispositivi protocollo UART

Normalmente, il canale di trasmissione si trova al livello logico alto. Come è mostrato in figura 5.2, quando un dispositivo è pronto a trasmettere abbassa il livello sulla linea di trasmissione, inviando così il primo bit, ossia il bit di start. In seguito, il dispositivo invia i dati partendo dal bit meno significativo e procedendo verso quelli più significativi, per poi concludere con i bit di parità e/o di stop. Questo processo consente al dispositivo di inviare in modo sequenziale e coerente l'intero pacchetto di dati.

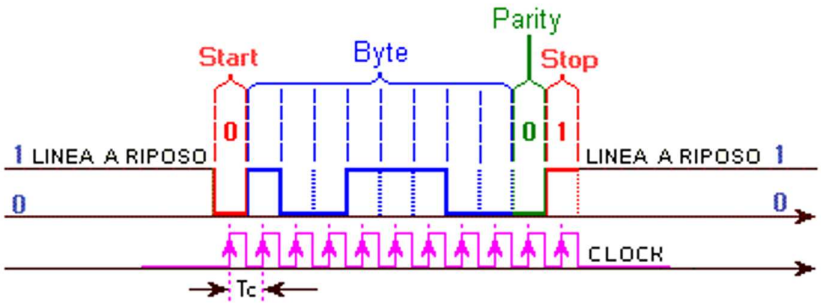

Figura 5.2: UART

Il bit di parità, basandosi sul conteggio del numero di "1" presenti nel pacchetto, che deve sempre essere pari, svolge un ruolo cruciale nel controllo dei dati ricevuti. Ad esempio, se ci sono cinque "1" tra i bit di dati, il bit di parità sarà anch'esso "1".

5.2 Ricezione dei Comandi

Nel contesto del progetto presentato in questa tesi, la Raspberry, che contiene gli algoritmi di decision-making e invia al microcontrollore i comandi da attuare, e il microcontrollore sono collegati tarmite un cavo USB - mini USB e comunicano attraverso comunicazione seriale asincrona utilizzando il protocollo UART. In particolare la Raspberry invia alla STM delle stringhe del tipo:

" #numero : valore ; valore ;; "

Dove "numero" può essere un numero compreso tra 0 e 9, ognuno corrispondente a un comando diverso; possono esserci tra uno e tre valori, a seconda del comando, separati tra loro dal ";" e ci sono i ";;" a chiudere la stringa. Più nel dettaglio, di seguito viene riportata la mappatura che si trova all'interno del file "main.cpp", al path "src/main.cpp", che serve a indirizzare i comandi arrivati dalla Raspberry con numeri da 0 a 9 alla corrispondente funzione di CallBack per la loro attuazione. Di questi 9 comandi, il più usato in questo progetto è il numero 8.

```
1  utils::serial::CSerialMonitor::CSerialSubscriberMap g_serialMonitorSubscribers = {
2      {"1",mbed::callback(&g_robotstatemachine,&brain::CRobotStateMachine::
3          serialCallbackSPEEDcommand)},
4      {"2",mbed::callback(&g_robotstatemachine,&brain::CRobotStateMachine::
5          serialCallbackSTEERcommand)},
6      {"3",mbed::callback(&g_robotstatemachine,&brain::CRobotStateMachine::
7          serialCallbackBRAKEcommand)},
8      {"4",mbed::callback(&g_robotstatemachine,&brain::CRobotStateMachine::
9          serialCallbackNOPIDcommand)},
10     {"5",mbed::callback(&g_encoderPublisher,&periodics::CEncoderPublisher::
11         serialCallbackENCODERPUBcommand)},
12     {"6",mbed::callback(&l_pidController,&signal::controllers::siso::
13         CPidController<double>::serialCallbackTUNEPIDcommand)},
14     {"7",mbed::callback(&g_robotstatemachine,&brain::CRobotStateMachine::
15         serialCallbackMOVEcommand)},
16     {"8",mbed::callback(&g_robotstatemachine,&brain::CRobotStateMachine::
17         serialCallbackCONTROLcommand)},
18     {"9",mbed::callback(&s_pidController,&signal::controllers::siso::
19         CPidController<double>::serialCallbackTUNEPIDcommand)},
20     {"0",mbed::callback(&d_sonar,&hardware::sensors::sonar::
21         serialcallbackDISTANCEcommand)}
22 };
```

5.3 Attuazione dei comandi

Il codice presentato in questa sezione si trova all'interno del file "robotstatemachine.cpp", al path "src/-brain/robotstatemachine.cpp", e costituisce la parte, per così dire, di brain del progetto: contiene infatti le funzioni di CallBack che vengono richiamate alla ricezione dei comandi dalla Raspberry per l'attuazione e una funzione "run", che implementa di fatto il movimento del veicolo.
La funzione "run", che viene eseguita periodicamente grazie a un timer contatore, distingue lo stato di funzionamento del veicolo in quattro possibili stati, e in base a questo ne controlla il comportamento.

- *Stato 1*: è lo stato di *controllo*, in cui cioè vengono controllati la rotazione del motore DC e l'angolo di sterzata. Viene inizialmente effettuata una verifica per vedere se il PID è attivo e, in caso positivo, viene implementato il controllo richiamando le funzioni "control", spiegate alle sezioni 3.3 e 3.4. Se dalla "control" della trazione risulta che l'encoder funziona correttamente ma che sta lavorando a una velocità troppo elevata, viene stampato un messaggio di errore e viene modificato lo stato da 1 a 2, che è lo stato di *Brake*, per fermare la macchina. Se invece risulta che l'encoder non sta funzionando come dovrebbe, viene stampato un altro messaggio di errore e viene di nuovo modificato lo stato da 1 a 2.

```
1  void CRobotStateMachine::_run()
2  {
3      switch(m_state)
4      {
5          case 1:
6              if(m_ispidActivated && m_dcMotorControl!=NULL &&
7                  m_steeringController!= NULL)
8              {
9                  int8_t l_isCorrect = m_dcMotorControl->control();
10                 int8_t s_isCorrect = m_steeringController->control();
11                 if( l_isCorrect == -1 )
12                 {
13                     m_serialPort.printf("@1:Too high speed and the
14                                 encoder working;;\r\n");
15                     m_dcMotor.brake();
16                     m_dcMotorControl->clearSpeed();
17                     m_state = 2;
18                     break;
19                 }
```

```
20                  else if (l_isCorrect == -2 )
21                  {
22                      m_serialPort.printf("@1:Encoder error;;\r\n");
23                      m_dcMotor.brake();
24                      m_dcMotorControl->clearSpeed();
25                      m_state = 2;
26                      break;
27                  }
```

In seguito, grazie al sensore a ultrasuoni, viene effettuato il controllo per verificare che non ci siano oggetti nel circondario del veicolo, controllo di cui si parla alla sezione 4.4: viene ridotta la velocità longitudinale in maniera proporzionale alla diminuzione della distanza rilevata. Viene poi assegnata la sterzata desiderata e fermato il veicolo, modificando lo stato a 4, se la distanza dagli oggetti è minore di 12 cm.

```
1                   float distance = m_sonar.getCurrentDistance();
2                   float proportion = (distance/50);
3                   if(distance > 200)
4                       proportion = 1;
5                   m_dcMotor.setSpeed(m_dcMotorControl->getSpeed()*proportion);
6                   m_steeringControl.setSteer(m_steeringController->getAngle());
7                   if(distance < 12)
8                   {
9                       m_dcMotor.brake();
10                      m_state = 4;
11                  }
12              }
13              break;
```

- *Stato 2*: è lo stato di *brake*, ovvero lo stato in cui al veicolo si assegna velocità longitudinale nulla. Ci si trova in questo stato ad esempio in seguito a errori di malfunzionamento di componenti, nella situazione di attesa dell'attraversamento di pedoni, nel caso in cui la distanza dagli ostacoli sia minore di 12 cm e così via.

```
1           case 2:
2               if( m_dcMotorControl!=NULL){
```

```
3                   m_dcMotorControl->clearSpeed();
4               }
5               break;
```

- *Stato 3*: è il *"Move State"*, ovvero uno stato analogo all'1 ma in cui si conrolla solamente la trazione trascurando la sterzata e perciò non di interesse ai fini di questa tesi.
- *Stato 4*: è lo stato di *frenata di emergenza*. Ci si ricade nel caso in cui dal sensore risulti una distanza da un qualsiasi oggetto minore di 12 cm e se ne esce quando la distanza misurata supera i 12 cm.

```
1           case 4:
2               m_dcMotorControl->clearSpeed();
3               m_steeringControl.setZero();
4               if(m_sonar.getCurrentDistance() > 12)
5               {
6                   m_state = 1;
7               }
8               break;
9       }
10  }
```

Uno specifico stato viene assegnato all'interno delle funzioni di Callback chiamate alla ricezione dei comandi in base ai comandi stessi. In particolare, il comando più utilizzato nel contesto esaminato è il comando 8 che, come si vede alla sezione 5.2, è quello a cui è associata la funzione "serialCallback-CONTROLcommand", esaminata di segutio.
Questa, a partire dalla stringa ricevuta dalla Raspberry, popola tre diversi parametri di tipo float, di cui il primo è per la velocità longitudinale di riferimento, il secondo è per la curvatura, ovvero l'inverso del raggio di curvatura, calcolata dalla Raspberry e il terzo è per la velocità di yaw effettiva misurata dall'IMU.

```
1  void CRobotStateMachine::serialCallbackCONTROLcommand(char const * a, char * b)
2  {
3      float speed, curve, yaw_rate, long_speed;
4      uint32_t l_res = sscanf(a, "%f;%f;%f", &speed, &curve, &yaw_rate);
```

In seguito controlla, per prima cosa, che la comunicazione sia avvenuta in maniera corretta e che siano stati ricevuti tutti i parametri, poi verifica che il segnale di controllo e di riferimento non abbiano valori troppo elevati, nel qual caso stampa un messaggio di errore.

```
    if (3 == l_res)
    {
        if( !m_ispidActivated && !m_dcMotor.inRange(speed)){
        // Check the received control value
            sprintf(b,"The speed command is too high;;");
            return;
        }

        if( m_ispidActivated && !m_dcMotorControl->inRange(CRobotStateMachine::
            Mps2Rps(speed))){ //Check the received reference value
            sprintf(b,"The speed reference is too high;;");
            return;
        }
```

A questo punto, viene settato lo stato del veicolo a 1, cioè lo stato di controllo, e viene verificato se il PID per la trazione è attivo o meno. Se non è attivo, viene direttamente chiamata la funzione che assegna il valore del duty cycle del segnale PWM per il controllo del motore, altrimenti viene settata come velocità di riferimento la velocità longitudinale ricevuta dalla Raspberry, che consente di calcolare l'errore tra questa e la velocità di rotazione dell'encoder per ottenere infine il segnale di controllo.

```
        m.state=1;

        if(!m_ispidActivated)
        {
            m_dcMotor.setSpeed(speed);
        }
        else
        {
            m_dcMotorControl->setRef(CRobotStateMachine::Mps2Rps(speed));
        }
```

Infine, si passa al settaggio della velocità di yaw di riferimento, calcolata come moltiplicazione tra la curvatura e la velocità longitudinale di riferimento, e al settaggio della velocità di yaw effettiva misurata

per il calcolo dell'errore e quindi del segnale di controllo.

```
1           m_steeringController->setRef(curve*long_speed);
2           m_steeringController->setYaw(yaw_rate);
3
4           sprintf(b,"ack;;");
5       }
6       else
7       {
8           sprintf(b,"sintax error;;");
9       }
10
11  }
```

6

Risultati Ottenuti

Indice

6.1 Ambiente Simulato

È stato creato un ambiente di testing fisico con dimensioni simili a quelle fornite dalla Bosch per la competizione per simulare in maniera veritiera la pista di gara.
In particolare sono stati realizzati:

- un percorso a due corsie con diverse curve, un incrocio a tre strade e due parcheggi;
- segnaletica a terra: segnali di stop, attraversamenti pedonali e parheggi;
- diversi segnali stradali;
- un pedone.

Percorso e segnaletica

Come si può vedere dalla figura 6.1, il percorso realizzato è dotato di due corsie, 4 curve con stesso raggio di curvatura e un incrocio a tre strade, la cui funzione è testare la capacità di decision-making del sistema, in particolare riguardo alla scelta di che strada prendere.
Si può anche riconoscere la segnaletica a terra: le linee di stop, ovvero linee perpendicolari a quelle ai lati delle corsie e più spesse posizionate in prossimità dell'incrocio e dell'attraversamento pedonale, e l'attraversamento pedonale.

Figura 6.1: Percorso fisico

Il percorso è inoltre dotato di due parcheggi non presenti nell'immagine in quanto non integrati nel tappeto e perciò spostati da una parte all'altra di questo.

Segnali stradali

Come si può vedere dalla figura 6.2, sono stati realizzati sia diversi tipi di segnali stradali sia, con la stampante 3D, un supporto per questi.

Figura 6.2: Segnali stradali e supporti

Lo scopo era che ad ogni segnale corrispondesse una reazione diversa del veicolo:

- stop: che si fermasse completamente;
- attraversamento pedonale: che rallentasse e controllasse la presenza di pedoni, perciò si fermasse se ne vedeva e procedesse altrimenti;
- parcheggio: che iniziasse la procedura di parcheggio in base alla natura dello stesso, parallelo o perpendicolare;
- rotatoria: che scegliesse che uscita della rotatoria prendere;
- inizio autostrada: che accelerasse nettamente;
- fine autostrada: che rallentasse tornando alla velocità precedente;
- senso vietato: che cambiasse direzione e non entrasse in quella strada;
- senso unico dritto: che, in presenza di incorci, procedesse dritto;
- segnale giallo: che, in presenza di incroci, non guardasse la presenza di altre macchine e quindi non desse loro la precedenza;

Pedone

Al fine di testare la risposta del veicolo sia agli attraversamenti pedonali, garantendo che rallentasse in presenza di pedoni individuati dopo il segnale, sia alle situazioni di frenata di emergenza, sono state simulate queste due situazioni utilizzando una bambola, illustrata nella figura 6.3.

Figura 6.3: Pedone

6.2 Risultati attesi e risultati ottenuti

Si può trovare una copia completa del codice utilizzato alla repository Git-Hub al seguente link: https://github.com/Team-Protom/nucleo.
Gli scopi di questo progetto, in conclusione, erano molteplici, cioè che il veicolo:

- fosse in grado di riconoscere correttamente la traiettoria da seguire;
- fosse in grado di mantenere una velocità longitudinale coerente;
- fosse in grado di sterzare in maniera fluida;
- fosse in grado di riconoscere i diversi segnali stradali e di avere una reazione consona ad ognuno di questi;
- fosse in grado di parcheggiare senza urtare contro gli oggetti circostanti;
- riuscisse a frenare in caso di ostacoli inaspettati sul percorso;
- riuscisse a scegliere una tra le diverse strade possibili in una rotatoria o incrocio;
- desse la precedenza in maniera corretta;
- si fermasse in presenza di un semaforo rosso e ripartisse quando diventava verde;
- riuscisse ad interagire correttamente con altri veicoli sul percorso, cioè ad accodarsi a questi o a superarli.

Purtroppo, a causa di un malfunzionamento hardware che non si è fatto in tempo a riconoscere e risolvere, il team che ha partecipato alla BFMC è stato escluso dalle fasi finali della competizione e perciò è stato necessario riconsegnare il kit fornito e la macchinina stessa. Questo ha fatto sì che gli

obiettivi prefissati non siano stati del tutto raggiunti.
In particolare, in seguito a diversi test nell'ambiente fisico reaizzato, è stato possibile fare in modo che il veicolo riconoscesse la traiettoria da seguire, che accelerasse e sterzasse in maniera consona e che frenasse in situazioni di emergenza. Inoltre è stata implementata una reazione ai cartelli stradali di stop, attraversamento pedonale, senso vietato e rotatoria come descritto al paragrafo dedicato, mentre non si è arrivati a perfezionare gli algoritmi di parcheggio.
Si è cercato quindi di completare il progetto utilizzando un hardware leggermente differente da quello fornito per la competizione, utilizzando però il codice per la Raspberry utilizzato per la challenge.

6.3 Veicolo di supporto

È stato quindi utilizzato un secondo veicolo, simile a quello fornito dalla Bosch, per completare la fase di testing del sistema. È stato anche utilizzato un codice per il microcontrollore diverso da quello utilizzato per la competizione e illustrato ai capitoli precedenti: di questo viene riportata di seguito la sezione sulla comunicazione tra le schede.
Come per la BFMC, la Raspberry e il microcontrollore del secondo veicolo utilizzato sono collegati tra loro tramite un cavo USB - mini USB e la comunicazione tra queste due schede avviene tramite comunicazione seriale asincrona con il protocollo UART in modalità *interrupt*. Quella con l'utilizzo di segnali di interrupt è una tecnica che permette di gestire in modo efficiente la trasmissione e la ricezione di dati in un sistema. Gli interrupt in generale sono segnali hardware che indicano al processore di interrompere la normale sequenza di esecuzione del programma per eseguire una particolare routine di servizio e possono essere generati da eventi esterni come la ricezione di un dato o il completamento di una trasmissione.
Durante la ricezione di dati seriali, un interrupt può essere generato quando c'è un nuovo byte disponibile nel buffer di ricezione e un'interruzione simile può essere utilizzata per segnalare la disponibilità del buffer di trasmissione quando è pronto a inviare un nuovo byte.
L'utilizzo di questa tecnica comporta sia risparmio di CPU, in quanto l'uso di interrupt consente alla CPU di svolgere altre attività durante l'attesa di dati seriali, sia un tempo di risposta più veloce: gli interrupt infatti permettono di rispondere immediatamente ai dati in arrivo senza dover controllare continuamente il buffer.
La gestione di questi segnali però richiede di prestare una particolare attenzione alla sincronizzazione dei dati in modo da evitare problemi come la sovrascrizione o l'invio prima che il buffer sia pronto.
In particolare la Raspberry invia alla scheda STM delle stringhe simili a quelle analizzate alla sezione 5.2:

" # 1 : ± velocità_longitudinale ; ± raggio_di_curvatura ;; "

Dove:

- la velocità longitudinale, ovvero la velocità che si vuole che il veicolo mantenga espressa in *m/s*, viene interpretata come float e le sono dedicati tre caratteri, cioè un intero, la virgola e la prima cifra decimale, e si specifica il segno “+” per velocità positive in avanti o “-” per velocità negative all’indietro;
- il raggio di curvatura, ovevro il raggio calcolato dalla camera della curva che il veicolo deve affrontare espresso in *m*, viene interpretato come float e gli vengono dedicati da tre caratteri, cioè un intero, la virgola e la prima cifra decimale, e viene specificato il segno “+” per curvature verso destra e “-” per curvature a sinistra.

Infine ci sono i “;;” che segnalano la fine della stringa.
A differenza della BFMC però viene inviato un unico comando per il controllo del veicolo.

6.3.1 Comunicazione tra le schede

Tutto il codice realizzato per la scheda STM è stato scritto in C e con l’ausilio dell’*STM32CubeIDE*.
Per prima cosa, per poter abilitare la comunicazione tra le due schede STM e Raspberry, è necessario attivare l’UART dall’IDE utilizzato dal file con estensione “.ioc”.
Si inizia cliccando sulla voce “Connectivity” come mostrato in figura 6.4.

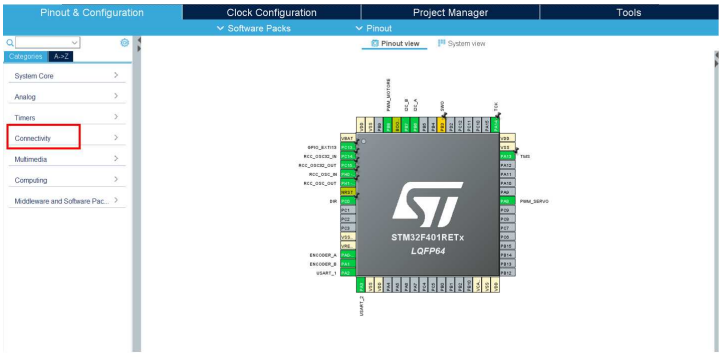

Figura 6.4: Connectivity

In seguito si seleziona la voce “USART2” come mostrato in figura 6.5.

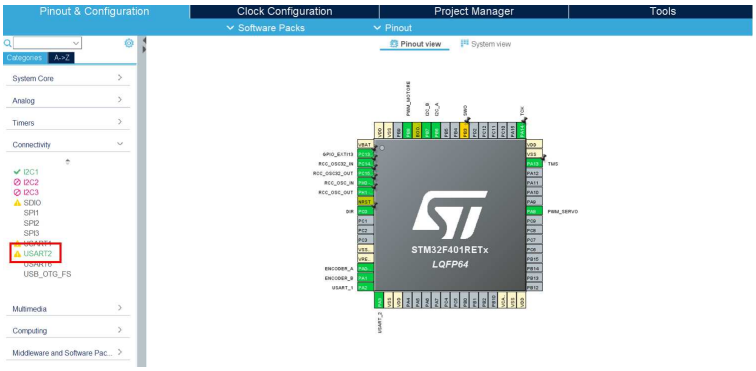

Figura 6.5: USART2

A questo punto, bisogna controllare che la modalità sia “Asynchronous” e che i parametri siano settati come in figura 6.6.

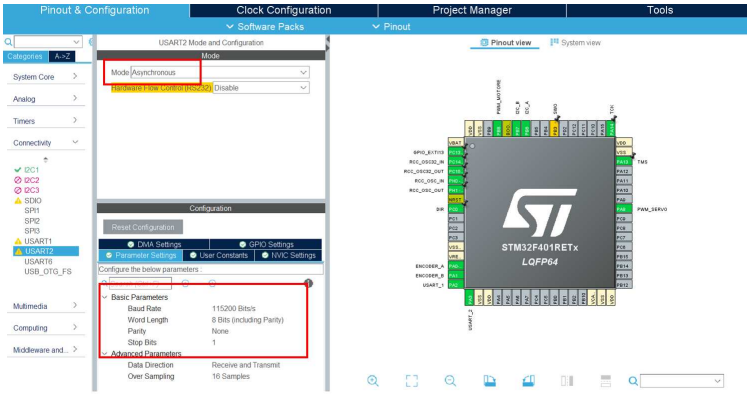

Figura 6.6: Settaggio dei parametri

Infine è necessario attivare il “global interrupt”, come si vede in figura 6.7, perchè la comunicazione seriale funzioni in modalità interrupt.

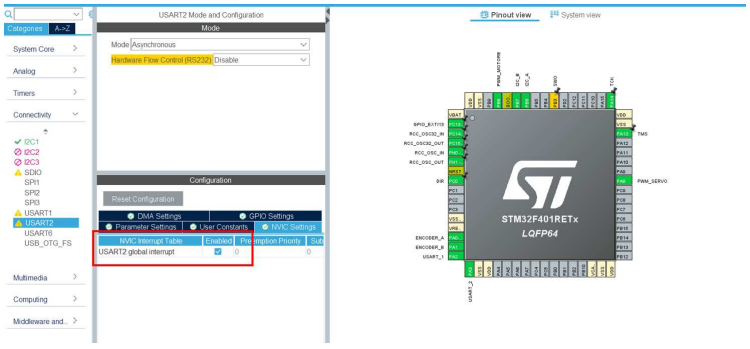

Figura 6.7: Attivazione del golbal interrupt

Più nel dettaglio, di seguito viene riportata la funzione di CallBack, che si trova all’interno del file “main.c”, che viene richiamata al verificarsi del segnale di interrupt in seguito alla ricezione dei dati inviati dalla

Raspberry sulla seriale.

```
1  void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
2  {
3      parametri = ricevi_dati(rx_data);
4
5      HAL_UART_Receive_IT(&huart2, (uint8_t *)&rx_data, sizeof(rx_data));
6  }
```

Questa funzione popola un dato di tipo struct “dato” richiamando la funzione “ricevi_dati”, di cui si parla più avanti, che viene utilizzato per il calcolo del segnale di controllo dei pid per la velocità longitudinale e di sterzata. Come ultima cosa viene riabilitata la ricezione in modalità interrupt.
La funzione “ricevi_dati” viene definita all’interno dell’header file “SerialMonitor.h” riportato di seguito.

```
1  #ifndef INC_SERIALMONITOR_H_
2  #define INC_SERIALMONITOR_H_
3
4
5  #include <main.h>
6  #include "stdio.h"
7
8  struct dato {
9      float speed;
10     float curvature_radius;
11 };
12
13 struct dato ricevi_dati(uint8_t rx_data[]);
14
15
16 #endif /* INC_SERIALMONITOR_H_ */
```

In particolare all’interno di questo file si trova la definizione della struct “dato”, che comprende due parametri, “speed” e “curvature_radius”, entrambi float, di cui nel primo viene salvata la velocità longitudinale da mantenere e nel secondo il raggio di curvatura per il calcolo della sterzata.
L’implementazione della funzione “ricevi_dati”, però, viene fatta all’interno del file “SerialMonitor.c” riportato di seguito.

```
#include <serialMonitor.h>

struct dato ricevi_dati(uint8_t rx_data[])
{
    struct dato parametri;
    parametri.curvature_radius = 0;
    parametri.speed = 1;
    char appoggio[15];
    int i = 0;

    if(rx_data[0]=='#' && rx_data[1]=='1')
    {

        while(rx_data[i] != '\r')
        {
            appoggio[i] = (char)rx_data[i];
            i++;
        }

        printf("@Prova:%s\r\n",appoggio);
        sscanf(appoggio, "#1:%f;%f;;\r", &parametri.speed,&parametri.curvature_radius);
    }
    else
    {
        printf("@Syntax Command Error \r\n");
    }

    return parametri;
}
```

Questa funzione per prima cosa setta i parametri della struct “dato” ai rispettivi valori di partenza, ovvero la velocità longitudinale a 1 m/s e il raggio di curvatura a 0 m, crea una stringa per l’appoggio dei dati ricevuti e setta un contatore a zero.
In seguito controlla che la stringa ricevuta dalla Raspberry inizi con i caratteri “#1”, come verifica della correttezza della ricezione tramite seriale, e copia all’interno della stringa di appoggio i dati contenuti in quella ricevuta. Infine assegna a “parametri.speed” e “parametri.curvature_radius” i valori di velocità e raggio di curvatura inviati dalla Raspberry.

6.4 Test effettuati e possibili miglioramenti

In seguito alla restituzione del primo veicolo alla Bosch, sono stati effettuati dei test con il secondo: questi hanno riguardato, per prima cosa, il corretto funzionamento della comunicazione seriale e, per seconda cosa, la tenuta della strada.
Si è quindi verificato, tramite l'utilizzo di "PuTTY", se, inviando tramite seriale dal PC delle stringhe con formato come illustrato alla sezione 6.3 al microcontrollore, questo riusciva a popolare correttamente la struct "parametri" per poi calcolare i segnali di controllo dei PID. Dal test è risultato che la ricezione avveniva correttamente.
In seconda istanza si è cercato di integrare il codice sviluppato per la Raspberry nel contesto della BFMC.
Lo scopo era lo stesso: realizzare un veicolo che implementasse un livello 5 di automazione, ovvero che riuscisse a navigare autonomamente nel percorso di test realizzato senza aiuto umano, che reagisse in maniera corretta ai segnali stradali realizzati, che mantenesse una velocità longitudinale consona, che riuscisse a sterzare mantenendosi all'interno della corsia e che frenasse in caso di presenza sulla strada di pedoni o altri ostacoli.
L'integrazione del suddetto codicè, però, è risultata difficoltosa a causa di un sistema hardware differente rispetto a quello fornito dalla Bosch per la competizione e a causa del codice stesso.
In particolare, il problema principale era un problema di sincronizzazione dei processi: tutti i diversi processi in esecuzione sulla Raspberry, come il thread di lane detection, il thread di decision making, il thread di sign detection e così via, erano in conflitto tra loro, risultando in un errore di "Broken pipe".
Sono stati perciò effettuati diversi test sul percorso fisico per capire se si potesse risolvere il problema che si era presentato, ma, a causa di un codice molto elaborato e della mancanza di documentazione come il diagramma delle classi, è stato impossibile risolverlo e quindi realizzare effettivamente un veicolo autonomo in scala.
Nonostante ciò, i test per la comunicazione seriale e per il funzionamento adeguato dei motori sono stati positivi, perciò si ritiene che, con l'utilizzo di un codice differente e funzionante per la Raspberry, ciò che è stato sviluppato per la scheda STM sia una base valida da cui partire per realizzare lo scopo di questa tesi.

Bibliografia

[1] P. Bolzern, R. Scattolini, N. Schiavoni, *Fondamenti di Controlli Automatici*, Ed. McGraw-Hill, terza edizione, 2008.

[2] Tobias Holstein, Gordana Dodig-Crnkovic, Patrizio Pelliccione, *Real-world Ethics for Self-Driving Cars*, IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings, 2020.

[3] Hao Zhan, Dan Wan, Zhiwei Huang, *On the Responsible Subjects of Self-Driving Cars. Under the SAE System: An Improvement Scheme*, IEEE International Symposium on Circuits and Systems, 2020.

[4] Betina Carol Zanchin, Rodrigo Adamshuk, Max Mauro Santos, Kathya Silvia Collazos, *On the Instrumentation and Classification of Autonomous Cars*, IEEE International Conference on Systems, Man, and Cybernetics, 2017.

[5] Debbie Hopkins, Tim Schwanen, *Talking about automated vehicles: What do levels of automation do?*, 2021.

[6] Prof. A. Bonci, *07a_IT_PWM TEORIA rev2.pdf*, slides del corso ”Laboratorio di Automazione”, Ingegenria Informatica e dell’Automazione UNIVPM.

[7] Prof. A. Bonci, *10_EN_SW_STM32H7 ENCODER v3.pdf*, slides del corso ”Laboratorio di Automazione”, Ingegenria Informatica e dell’Automazione UNIVPM.

[8] Prof. A. Bonci, *09_EN_SW_STM32H7 SERIAL v2.pdf*, slides del corso ”Laboratorio di Automazione”, Ingegenria Informatica e dell’Automazione UNIVPM.