

Università degli Studi di Salerno
DIEM, a. a. 2022-23
Corso: AUTONOMOUS VEHICLE DRIVING
Professore: Antonio Greco

Progettazione ed implementazione di un algoritmo di navigazione in CARLA

Arianna Carratù
Andrea De Gruttola
Matteo Di Renzo
Salvatore Grimaldi

0622701696
0622701880
0622701818
0622701742

a.carratu18@studenti.unisa.it
a.degruttola@studenti.unisa.it
m.dirienzo1@studenti.unisa.it
s.grimaldi29@studenti.unisa.it



Indice

1	Introduzione	3
2	Guida autonoma e simulazione.....	5
2.1	CARLA.....	6
3	Definizione dell'Operational Design Domain (ODD)	8
3.1	Scenario 0.....	8
3.2	Scenario 1	10
3.3	Scenario 2	11
3.4	Scenario 3	12
3.5	Scenario 4	13
3.6	Ulteriori considerazioni	14
4	Setup sperimentale	16
4.1	Risorse hardware.....	16
4.2	Setup parametri del server	16
4.3	Metriche di valutazione	18
4.3.1	Infrazioni ed eventi di shutdown.....	19
5	Analisi della baseline.....	21
5.1	Struttura	21
5.2	Funzionalità.....	22
5.2.1	BasicAgent.....	23
5.2.2	BehaviourAgent.....	24
5.2.3	LocalPlanner.....	27
5.2.4	GlobalPlanner	27
5.2.5	VehicleController	27
5.3	Prestazioni e criticità	31
5.3.1	Scenario 0.....	32
5.3.2	Scenario 1	33
5.3.3	Scenario 2	34
5.3.4	Scenario 3	34
5.3.5	Scenario 4	35
6	Soluzione proposta.....	37
6.1	Analisi dei requisiti: difficoltà e priorità	37
6.2	Taratura dei controllori.....	38

6.2.1	Taratura del controllore longitudinale PID	39
6.2.2	Taratura del controllore laterale Stanley	41
6.3	Architettura software (finale).....	43
6.3.1	Basic Agent.....	45
6.3.2	Behavior Agent.....	46
6.3.3	Vehicle Controller.....	56
6.3.4	Definizione e setup dei parametri	56
7	Versioni del software e analisi incrementale dei risultati	59
7.1	Versione con gestione dei sorpassi e dei restringimenti di corsia	59
7.2	Versione con gestione pedoni e biciclette.....	60
7.3	Versione con gestione incroci, semafori e stop	61
8	Analisi qualitativa della simulazione finale	64
9	Limitazioni e vantaggi del simulatore CARLA	65
9.1	Bug riscontrati.....	66
9.1.1	Bug riguardanti i waypoints.....	66
9.1.2	Bug riguardanti i Bounding Box	66
9.2	Dati reali imperfetti: moduli di percezione.....	67
10	Conclusioni e sviluppi futuri	69

1 Introduzione

Nel corso degli ultimi anni la guida autonoma (*Autonomous Vehicle Driving*) si è affermata come uno dei settori più promettenti ed innovativi dell'ingegneria. Il suo obiettivo fondamentale consiste nella realizzazione di veicoli in grado di muoversi autonomamente su strade non pre-adattate allo scopo senza la necessità di un intervento umano diretto. La guida autonoma è destinata a rivoluzionare completamente il traffico stradale, mitigando le attuali esternalità, in particolare incidenti e congestioni. Case automobilistiche, ricercatori e amministrazioni lavorano da tempo sulla guida autonoma e recentemente sono stati compiuti progressi significativi (si pensi, ad esempio, agli ultimi modelli della Tesla). Tuttavia, i dubbi e le sfide da affrontare sono ancora enormi, poiché l'implementazione di un veicolo autonomo comprende non solo complesse tecnologie automobilistiche e informatiche, ma riguarda da vicino anche il comportamento umano, l'etica, i governi e le responsabilità legali.

Il contributo che l'ingegneria informatica può fornire allo sviluppo e alla diffusione di tale tecnologia è tra i più significativi. Basti pensare che un veicolo a guida autonoma adopera tipicamente una combinazione di sensori (come telecamere, LIDAR, radar, GPS, ecc.) e intelligenza artificiale per spostarsi tra diverse destinazioni su strade non pre-adattate. Un sistema di Autonomous Vehicle Driving è complesso e modulare, in quanto costituito da sensoristica propriocettiva ed esterocettiva avanzata e da svariati moduli atti alla percezione, alla navigazione e al controllo.

Tra le sfide più difficili, per le quali non esistono ancora soluzioni standard, vi sono senza dubbio:

- **La realizzazione di moduli percettivi (traffic sign detection, semantic segmentation, object detection, object tracking, ecc.) efficaci ed efficienti in contesti di guida.** La difficoltà più grande sta nella corretta acquisizione delle misure dei sensori e nella loro combinazione ed elaborazione. Le problematiche legate alla percezione del mondo sono molto difficili da risolvere ed avvalorano la tesi secondo cui la guida autonoma possa diffondersi a tutti gli effetti solo quando tutti i veicoli riusciranno a comunicare tra di loro e con la strada.
- **L'implementazione di algoritmi di navigazione che riescano a gestire scenari eterogenei in modo sicuro ed efficiente.** Si consideri che i dati percettivi in ingresso ad un algoritmo di navigazione sono tutt'altro che perfetti: errori dovuti ai sensori e ai moduli percettivi possono inficiarne qualità ed attendibilità. In realtà, anche sotto l'assunzione di dati perfetti, che è quella che consentono di fare alcuni simulatori, non è per nulla scontato realizzare un algoritmo di navigazione efficace ed efficiente.
- **L'individuazione di un giusto compromesso (trade-off) tra prestazioni e complessità computazionale.** I dispositivi di elaborazione e i sensori che assicurano le migliori prestazioni costano molto, solitamente occupano uno spazio considerevole e comportano un consumo energetico molto importante, il che determina un veloce consumo della batteria del veicolo, implicando un significativo impatto negativo sull'usabilità dello stesso. D'altra parte, dispositivi e sensoristica poco avanzati impattano negativamente sulla velocità di risposta del sistema, sulla qualità della percezione e, quindi, sul comportamento del veicolo.
- **L'elevato costo, pericolosità e lentezza dei test su strada.** Fortunatamente, oggi esistono vari simulatori che offrono molteplici vantaggi rispetto ai test su strada, nonostante questi ultimi

rappresentino comunque una tappa finale imprescindibile per la validazione di un sistema di guida autonoma.

Tali problematiche e le loro possibili soluzioni costituiscono alcuni dei punti chiave del corso di *Autonomous Vehicle Driving* tenuto dal professore Antonio Greco presso il *Dipartimento di Ingegneria Informatica dell'Università degli Studi di Salerno*. Nell'anno accademico corrente, l'obiettivo del progetto finale del corso è stato quello di **sviluppare e implementare un algoritmo di navigazione, utilizzando Carla (CAR Learning to Act), un simulatore open-source**. La presente relazione costituisce il resoconto del lavoro progettuale svolto dal gruppo costituito da, in ordine alfabetico, A. Carratù, A. De Gruttola, M. Di Rienzo e S. Grimaldi.

La presente relazione si articola nel modo seguente.

Il **capitolo 2** fornisce una panoramica sull'importanza dei **test in simulazione** nell'ambito della guida autonoma, con particolare riferimento a CARLA, che è il simulatore utilizzato nel progetto.

Il **capitolo 3** è incentrato sulla definizione dell'*Operational Design Domain (ODD)* preso in considerazione, attraverso un'analisi attenta e puntuale degli scenari di CARLA oggetto del progetto, in termini di caratteristiche e criticità dei percorsi.

Il **capitolo 4** è dedicato al **setup sperimentale**, in particolare alla descrizione delle risorse hardware adoperate (CPU, GPU, ecc.), alla spiegazione del setup dei parametri del server locale di CARLA e alla presentazione delle metriche di valutazione dell'algoritmo di navigazione.

Il **capitolo 5** offre una dettagliata analisi della **baseline** messa a disposizione da CARLA. Essa costituisce il punto di partenza del progetto, pertanto è cruciale comprenderne e delinearne l'architettura software, le funzionalità implementate (e la relativa bontà), le prestazioni che assicura alla luce delle metriche di valutazione descritte nel capitolo precedente, e le criticità da risolvere.

Il **capitolo 6** è interamente incentrato sulla **descrizione e analisi della soluzione**, ovvero dell'algoritmo di navigazione progettato e implementato. All'analisi dei requisiti segue la parte di taratura dei controllori longitudinale e laterale. Inoltre, viene fornita una descrizione completa del software finale ottenuto, con particolare riferimento alle classi della baseline cui sono state apportate modifiche, ai nuovi metodi aggiunti e al setup di opportuni parametri di guida, che non sono altro che costanti all'interno del codice.

Il **capitolo 7** si sofferma su una disamina delle **varie versioni del software**, che differiscono tra loro per l'aggiunta in maniera incrementale delle funzionalità implementate. In particolare, vengono descritte una prima versione, capace di gestire sorpassi e restringimenti di corsia, una seconda, che integra la gestione di pedoni e biciclette, e quella finale, che garantisce anche la gestione di incroci, semafori e segnali di stop.

Il **capitolo 8** fornisce un'**analisi qualitativa** dell'algoritmo di navigazione finale. Viene reso disponibile anche il video della simulazione finale effettuata su CARLA.

Il **capitolo 9** si concentra sulle **limitazioni e sui vantaggi di CARLA**: vengono riportati alcuni bug riscontrati riguardanti waypoints collocati o etichettati male e bounding box imprecisi o con informazioni incomplete. Segue una breve digressione sui moduli di percezione, la cui importanza è cruciale quando non si intende sfruttare i dati perfetti di un simulatore, o banalmente quando il test viene condotto su strada, dove non esistono dati pre-elaborati né tantomeno perfetti.

Il **capitolo 10** contiene le **conclusioni** e gli spunti circa possibili **sviluppi futuri**, soprattutto in termini di miglioramenti di alcune delle funzionalità implementate, al fine di aumentare sicurezza, affidabilità e comfort dell'algoritmo di navigazione e di incrementare il punteggio totalizzato in CARLA.

2 Guida autonoma e simulazione

Il test in simulazione è una tappa fondamentale per lo sviluppo ed il miglioramento di un sistema di guida autonoma. È possibile osservare, infatti, come l'utilizzo di un simulatore offra significativi vantaggi:

- **Sicurezza:** un veicolo autonomo comporta rischi elevati se viene testato direttamente su strada. Un simulatore, invece, consente di creare scenari di guida complessi e variegati in modo sicuro e controllato, senza mettere a rischio l'incolumità del test driver, degli occupanti degli altri veicoli e dei pedoni.
- **Bassi costi:** i test su strada sono tipicamente molto costosi in termini di tempo, risorse umane e veicoli impiegati. Si pensi, ad esempio, ai ritardi introdotti dalla necessità di preparare e mettere in sicurezza le strade da percorrere. L'utilizzo di un simulatore ha un impatto benefico sui costi, in quanto consente di svincolarsi da gestioni logistiche e dalla necessità di ingaggiare piloti e disporre di veicoli fisici.
- **Velocità:** un simulatore consente di testare e valutare in tempi brevi un elevato numero di scenari di guida, favorendo così uno sviluppo più rapido degli algoritmi di navigazione.
- **Controllo e varietà degli scenari:** un simulatore permette di ricreare una vasta gamma di scenari di guida (urbani, sub-urbani, rurali, autostradali, ecc.), inclusi quelli estremi e difficilmente riproducibili su strada, garantendo quindi la possibilità di valutare l'affidabilità e l'efficacia degli algoritmi di guida autonoma in condizioni realistiche, eterogenee e sfidanti. Di fatto, la pericolosità dei test su strada fa sì che questi vengano effettuati quasi esclusivamente in scenari semplici e sicuri per veicolo, guidatore e persone circostanti, il che limita enormemente la variabilità delle condizioni testabili. Ad esempio, vengono raramente condotti esperimenti su strada caratterizzati da elevato traffico e/o elevata densità di pedoni. Un simulatore presenta spesso anche la possibilità di selezionare la densità del traffico, l'ora del giorno e le condizioni meteorologiche. Se l'importanza della prima è immediatamente intuibile, occorre enfatizzare anche quella degli altri due fattori: essi influiscono sull'aspetto che la strada ed il mondo circostante assumono, condizionando ovviamente i moduli di percezione. Addirittura, alcuni simulatori permettono di testare l'algoritmo di navigazione su veicoli diversi (es: auto, bus, ecc.)
- **Riproducibilità:** in simulazione è possibile riprodurre ripetutamente gli stessi scenari di guida per valutare diversi approcci, algoritmi e configurazioni. Tale riproducibilità è un fattore chiave per l'identificazione di eventuali debolezze e per la loro correzione prima di passare ai test su strada. In realtà, anche in simulazione, successive ripetizioni di uno stesso esperimento non risultano mai perfettamente uguali tra loro. Infatti, nonostante il simulatore riesca a riprodurre esattamente le stesse condizioni (es: passaggio di altri veicoli, attraversamento di pedoni, ecc.) negli stessi istanti di simulazione, anche una sola azione diversa su pedali e/o sterzo compiuta dall'algoritmo di navigazione tra un'istanza ed un'altra dello stesso esperimento comporta delle differenze, dal momento in cui si verifica, tra di esse.
- **Analisi dettagliata:** un simulatore fornisce dati dettagliati sulle prestazioni, il che permette di analizzare e comparare algoritmi di guida diversi oppure versioni successive dello stesso algoritmo.

Tale caratteristica, unita alla riproducibilità degli scenari, semplifica enormemente l'individuazione di bug, imperfezioni, aspetti migliorabili, così come la verifica della loro correzione.

- **Collaborazione e condivisione:** soprattutto se open-source, un simulatore può essere sfruttato come una vera e propria piattaforma di collaborazione e condivisione per sviluppatori e progettisti impegnati nella guida autonoma, il che non può che accelerare il progresso in tale campo.

In sintesi, un simulatore assicura un ambiente riproducibile, veloce, scalabile, economico e affidabile per testare, valutare e ottimizzare un algoritmo di navigazione per la guida autonoma, prima di affrontare i test su strada veri e propri. La sperimentazione su strada, infatti, è comunque una fase imprescindibile e compierla dopo aver speso diverse ore e aver percorso molti chilometri in simulazione è sicuramente più conveniente rispetto a non sfruttare l'opportunità offerta da quest'ultima.

È doveroso evidenziare che l'utilizzo di un simulatore per la guida autonoma, in realtà, introduce una sfida importante, rappresentata dalla necessità di effettuare **domain transfer**: i risultati ottenuti all'interno di un simulatore potrebbero non trasferirsi in modo diretto sul mondo reale. Ciò deriva dal fatto che in simulazione non si agisce sul mondo reale bensì su una sua rappresentazione semplificata. Per fare un banale esempio, i moduli di detection e tracking non agiscono su oggetti reali bensì su loro rappresentazioni digitali 3D. Esistono varie tecniche di domain transfer: un possibile approccio consiste nel trasformare immagini del mondo reale in immagini del mondo simulato.

2.1 CARLA

CARLA, acronimo di **CAR Learning to Act**, è un simulatore di guida autonoma open-source molto utilizzato in ricerca. Esso è stato creato per funzionare come un'API modulare e flessibile per affrontare una serie di compiti legati alla guida autonoma. Uno dei suoi principali obiettivi è quello di democratizzare la ricerca e lo sviluppo in questo settore, offrendo un sistema facilmente accessibile e personalizzabile dagli utenti. Il simulatore si basa su *Unreal Engine* e adopera lo standard *OpenDRIVE* per definire strade e ambienti urbani.

Esso si compone di **un'architettura client-server scalabile**. Il server si occupa di tutto ciò che riguarda la simulazione stessa: il rendering dei sensori, il calcolo della fisica, gli aggiornamenti sullo stato del mondo e dei suoi attori, ecc. Il lato client consiste in una serie di moduli client che controllano la logica degli attori nella scena e impostano le condizioni del mondo grazie ad una API dedicata (in Python o C++) che fa da intermediario tra server e client e che è in costante miglioramento, con lo scopo di offrire sempre nuove funzionalità.

Ulteriori caratteristiche interessanti di CARLA sono:

- ✓ grafica realistica e dettagliata
- ✓ molteplici mappe e condizioni meteo disponibili
- ✓ Densità del traffico e dei pedoni configurabile
- ✓ veicoli e sensori configurabili
- ✓ scenari altamente personalizzabili
- ✓ integrazione di algoritmi di controllo
- ✓ discreta community a supporto

- ✓ Possibilità di impostare il time-step come fisso o variabile
- ✓ Possibilità di impostare la modalità di comunicazione tra server e client come sincrona o asincrona

CARLA dispone di alcuni moduli per la preimpostazione delle prove in simulazione, in particolare per la gestione del mondo e il comportamento degli attori. In particolare, *ScenarioRunner* è un modulo che consente la definizione e l'esecuzione di scenari di traffico, i quali possono essere definiti tramite un'interfaccia Python o utilizzando lo standard *OpenSCENARIO*. *ScenarioRunner* può anche essere utilizzato per preparare gli agenti di guida autonoma per la valutazione, creando facilmente scenari di traffico complessi e percorsi per gli agenti da navigare. I risultati della valutazione possono essere convalidati e condivisi nella CARLA Leaderboard, una piattaforma open in cui la comunità può confrontare in modo equo i propri progressi, valutando gli agenti in situazioni realistiche di traffico.

È fondamentale evidenziare che **nell'ambito del progetto sono stati utilizzati i dati perfetti messi a disposizione da CARLA** per quanto concerne l'ego-position e la percezione del mondo simulato da parte dell'ego-veicolo. Ciò significa che l'algoritmo di navigazione non si basa su dati provenienti da sensori ed elaborati da moduli di percezione, bensì su dati forniti direttamente dal simulatore, che per ciascun oggetto presente nel mondo simulato garantisce classe, id, posizione, bounding box, velocità, accelerazione, e quant'altro. Questa semplificazione consente di svincolarsi dalle enormi problematiche introdotte da dati imperfetti e di concentrarsi esclusivamente sulla realizzazione dell'algoritmo di navigazione. Naturalmente, nel mondo reale, i dati che arrivano in ingresso ai moduli di navigazione sono tutt'altro che perfetti, sia in termini di percezione, poiché la sensoristica esterocettiva e i moduli atti alla detection e al tracking presentano un certo errore, sia in termini di ego-position, poiché la stima fornita dall'INS (Inertial Navigation System) è sicuramente affetta da errori.

3 Definizione dell'Operational Design Domain (ODD)

Nell'ambito della guida autonoma, l'Operational Design Domain (ODD) si riferisce alle specifiche condizioni in cui un veicolo è in grado di operare in modo sicuro e affidabile. L'ODD, in sostanza, definisce il contesto e le condizioni operative (es: velocità massima consentita, condizioni atmosferiche, ora del giorno, tipologia di strada, elementi dinamici, ecc.) entro le quali il veicolo garantisce le proprie funzionalità di guida autonoma. È fondamentale, soprattutto per ragioni legate alla sicurezza, che l'ODD sia il più accurato e preciso possibile: sviluppatori e produttori devono comprendere le limitazioni del sistema realizzato e devono comunicarle in modo trasparente e chiaro, così da limitare eventuali abusi o utilizzi impropri.

I vincoli sul contesto di guida sono intrinsecamente legati al **livello di guida autonoma**: la Society of Automotive Engineers, in particolare, distingue 5 livelli diversi di guida autonoma, in base al grado di attenzione richiesto al guidatore umano, alle azioni che quest'ultimo deve effettuare sul veicolo e al contesto nel quale il veicolo può svolgere il task di guida autonoma. Il livello che si intende raggiungere nell'ambito del progetto, limitatamente al test in simulazione, è il 4 (anche detto di *autonomia elevata o selettiva*), il quale prevede che il veicolo, solo in contesti di guida specifici, sia totalmente autonomo, con il guidatore che può distrarsi e intervenire quando vuole. Ciò significa che non vi è alcuna necessità di intervento del conducente in determinate circostanze specificate accuratamente nell'ODD.

I contesti di guida presi in considerazione per il progetto sono 5 scenari di simulazione in CARLA, di cui segue una dettagliata descrizione.

3.1 Scenario 0

Lo scenario 0 è ambientato nella Town 12 di Carla ed ha una lunghezza di 4.2 km circa. Le condizioni meteorologiche sono caratterizzate da una nuvolosità discreta; inoltre, il manto stradale è leggermente bagnato e non è in ottime condizioni. Il percorso è interamente a doppio senso (con una sola corsia per senso di marcia), si sviluppa lungo una strada extraurbana, presenta un breve tratto finale in autostrada, ed è composto principalmente da brevi tratti rettilinei alternati a curve ampie e abbastanza veloci. Procedendo lungo la route, si assiste ad un cambiamento delle condizioni di illuminazione a causa del tramonto del sole. Il traffico è moderato.

Il percorso è contraddistinto da una serie di criticità:

- **Intersezione a T bloccata** da un veicolo fermo lungo la propria corsia.
- Presenza di **ciclisti** lungo la propria corsia. Essi, fortunatamente, non occupano il centro della medesima, ma sono leggermente spostati sulla destra, il che ne facilita il sorpasso, poiché non vi è necessità di invadere la corsia opposta.
- **Restringimento della carreggiata** dovuto alla presenza di coni di lavori in corso collocati sulla corsia opposta. Vi sono diversi tratti di questo tipo lunghi anche più di un centinaio di metri. Fortunatamente, in tali tratti i veicoli che transitano lungo la corsia opposta non invadono quella su cui procede l'ego-veicolo o la invadono di poco. Tuttavia, è necessario che quest'ultimo usi accortezza e proceda lentamente, soprattutto se in curva, al fine di evitare collisioni.

- **Attraversamento di pedoni.** È necessario prestare particolare attenzione a questi ultimi, in quanto vi sono casi in cui risultano effettivamente visibili solo a pochi metri di distanza.
- **Veicoli in sosta (sia singoli sia in gruppo)** da superare invadendo la corsia opposta. La presenza di alcuni dei veicoli è associata ad incidenti verificatisi prima del passaggio dell'ego-veicolo. In un caso è presente anche un'auto della polizia.
- Presenza di **segnali di stop**, in corrispondenza dei quali è, naturalmente, necessario fermarsi.
- **Intersezione a T pericolosa** con immissione in autostrada. I veicoli collocati lungo la strada su cui è necessario immettersi procedono a velocità particolarmente sostenuta.
- **Traffic warning e coni di lavori** in corso da superare invadendo la corsia opposta.

Si riportano di seguito delle immagini raccolte in simulazione ritraenti alcuni punti salienti dello scenario 0.





Figura 1. Alcune delle criticità dello scenario 0

3.2 Scenario 1

Lo scenario 1 è ambientato di notte nella Town 12 di Carla ed ha una lunghezza di 3.2 km circa. Le condizioni meteorologiche sono caratterizzate da pioggia e vento inteso. Il manto stradale è bagnato e dissestato in qualche punto. Il percorso è molto simile a quello dello scenario 0. Esso, infatti, è interamente a doppio senso (con una sola corsia per senso di marcia), si sviluppa lungo una strada extra-urbana, presenta lunghi rettilinei alternati a poche curve ampie e veloci. Il traffico è moderato.

Le criticità disseminate lungo il percorso sono praticamente analoghe a quelle già individuate nello scenario precedente, tuttavia, la loro densità è mediamente superiore.

Si riportano di seguito delle immagini raccolte in simulazione ritraenti alcuni punti salienti dello scenario 1.



Figura 2. Alcune delle criticità dello scenario 1

3.3 Scenario 2

Lo scenario 2 è ambientato di giorno nella Town 12 di Carla ed è lungo poco più di 880 m. Le condizioni meteorologiche sono discrete. Il percorso, a differenza dei due precedenti, è ambientato in piena città, pertanto è urbano; inoltre, è interamente a doppio senso (con una sola corsia per senso di marcia), è caratterizzato da rettilinei, un elevato numero di semafori, incroci a T e ad X più o meno pericolosi e svolte strette da affrontare a velocità molto contenuta. La densità di veicoli, in effetti, è elevata: in alcuni tratti si procede a rilento a causa del traffico. Il manto stradale è in ottime condizioni; le corsie sono leggermente più strette rispetto a quelle delle strade percorse negli scenari precedenti.

Il percorso è contraddistinto da alcune criticità:

- La maggior parte degli **incroci**, il cui numero è senz'altro elevato, è regolata da **semafori**. In casi di questo tipo, è possibile assumere che quando il semaforo è verde, l'ego-veicolo possa approcciare e completare l'intersezione senza pericolo.
- Il **traffico intenso** rende, in uno specifico punto, l'attraversamento di un'intersezione a T, preceduta da un segnale di stop e non regolata da semafori, particolarmente complesso.
- Nonostante le **curve** da affrontare siano poche, esse sono particolarmente **strette**, pertanto occorre usare prudenza e procedere lentamente, al fine di evitare collisioni sia con marciapiedi sia con altri veicoli. La maggior parte delle curve sono associate a svolte da effettuare in corrispondenza di **incroci a T e ad X**, il che ne aumenta ulteriormente il coefficiente di difficoltà.

Si riportano di seguito delle immagini raccolte in simulazione ritraenti alcuni punti salienti dello scenario 2.

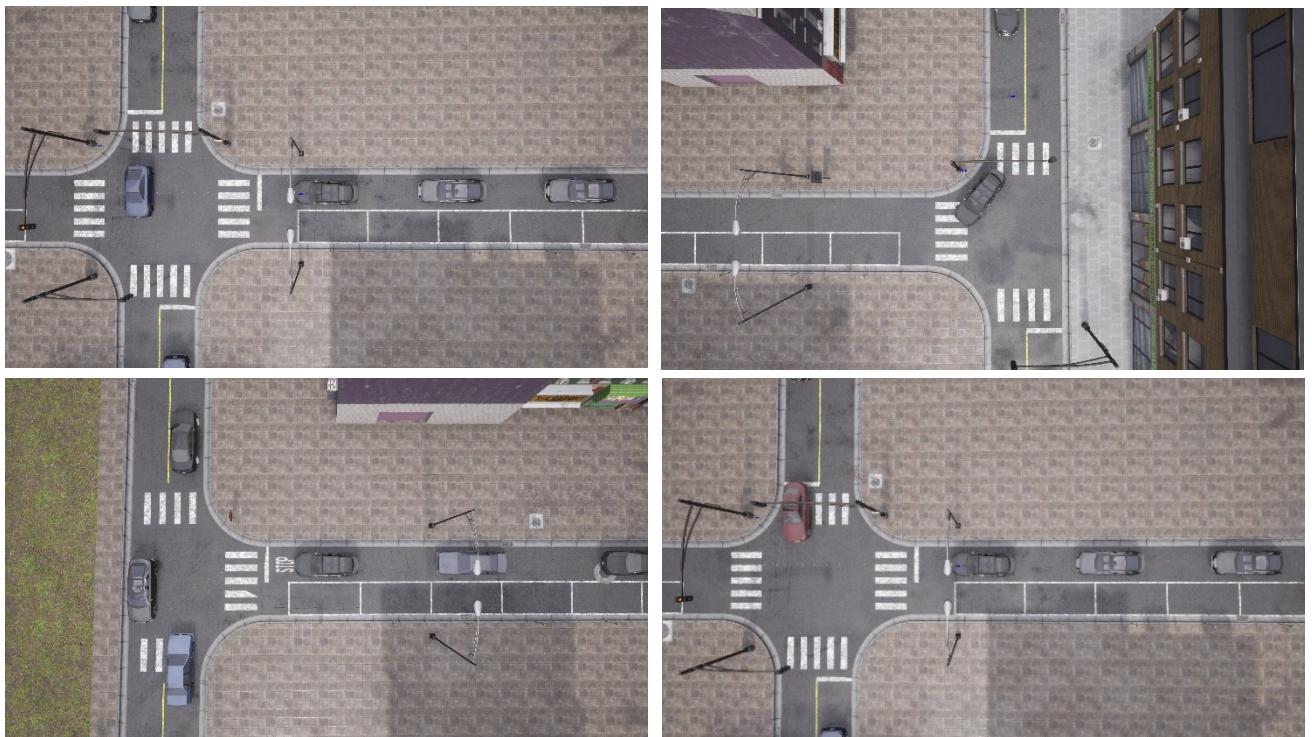




Figura 3. Alcune delle criticità dello scenario 2

3.4 Scenario 3

Lo scenario 3 è ambientato di giorno nella Town 12 di Carla ed è lungo poco più di 1.2 km. Le condizioni meteorologiche sono ottimali. Il percorso è urbano e interamente a doppio senso. In particolare, la sua porzione iniziale e la sua porzione finale hanno una sola corsia per senso di marcia, a differenza del tratto centrale, che gode di due corsie per senso di marcia. Il manto stradale è in ottime condizioni ed il traffico è molto intenso. Vi sono rettilinei, incroci sia a T sia ad X, semafori, svolte strette in cui è necessario mantenere una velocità contenuta. Lo scenario 3, pertanto, presenta numerosi punti in comune con lo scenario 2; rispetto a quest'ultimo, tuttavia, è caratterizzato da ulteriori criticità:

- Incrocio a T regolato da semafori con immissione su strada a doppio senso con 2 corsie per senso di marcia, con successivo cambiamento della propria corsia.
- Incrocio a T regolato da semafori in cui confluiscono 3 strade a doppio senso, una con singola corsia per senso di marcia, e 2 con doppia corsia per senso di marcia.
- Incrocio a X regolato da semaforo in cui confluiscono solo strade a doppio senso con 2 corsie per senso di marcia.
- **Cambio di corsia** (lungo lo stesso senso di marcia) a sinistra.
- Incrocio a X regolato da semaforo in cui confluiscono 2 strade a doppio senso con singola corsia per senso di marcia e 2 strade a doppio senso con 2 corsie per senso di marcia.

Si riportano di seguito delle immagini raccolte in simulazione ritraenti alcuni punti salienti dello scenario 3.



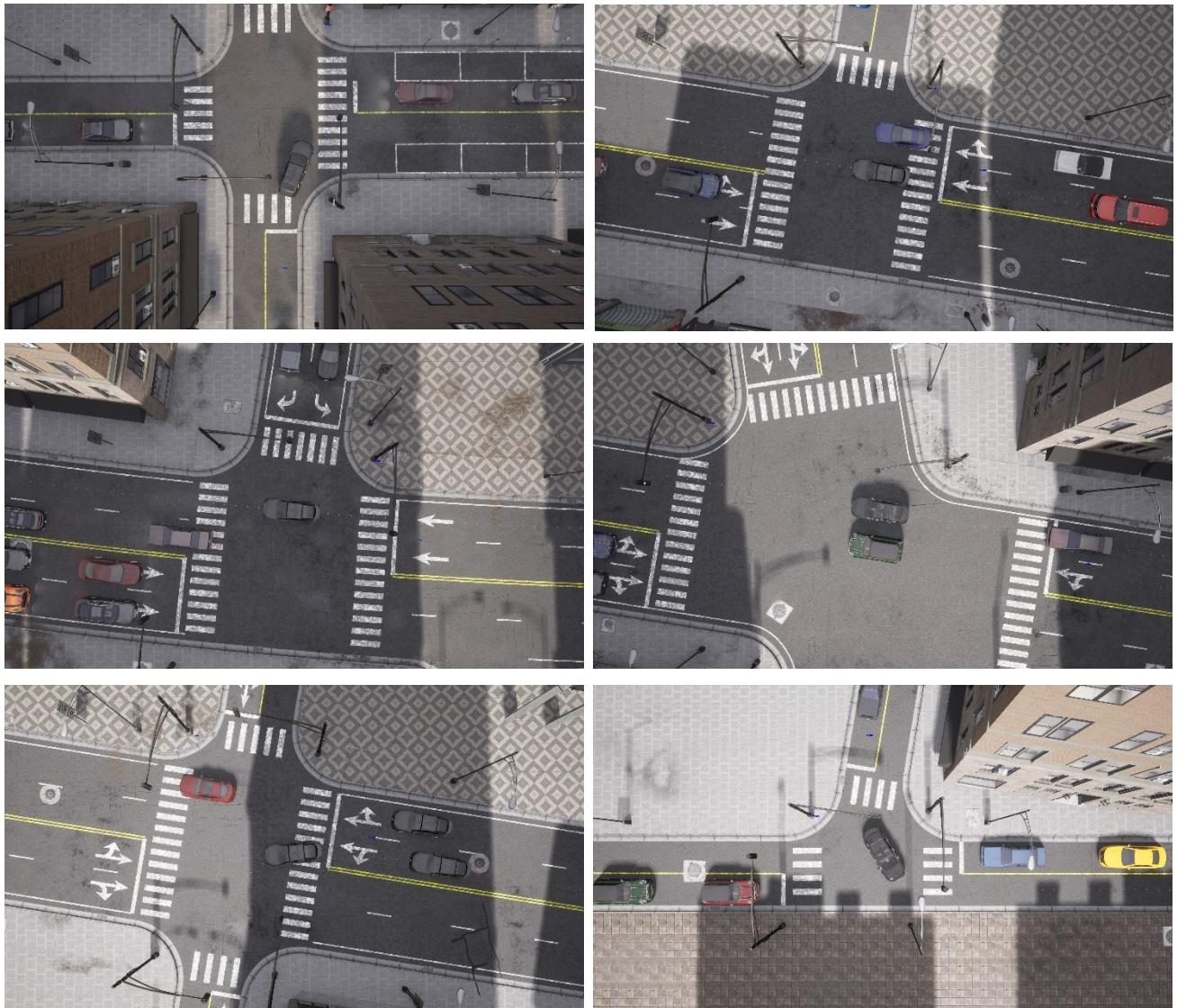


Figura 4. Alcune delle criticità dello scenario 3

3.5 Scenario 4

Lo scenario 4 è ambientato lungo un percorso rurale diurno nella Town 12 di Carla ed è lungo circa 900 m. Le condizioni meteorologiche sono discrete. Il percorso è interamente a doppio senso (con una sola corsia per senso di marcia) ed è contraddistinto da rettilinei e alcuni incroci sia a T sia a X non regolati da semafori. Il manto stradale è in buone condizioni, mentre il traffico è abbastanza intenso.

Le criticità più rilevanti sono le seguenti:

- **Veicolo della polizia fermo sulla propria corsia, da superare invadendo la corsia opposta.**
- **Un pedone che compare all'improvviso.** In particolare, si tratta di una bambina che attraversa la strada dopo essere passata dietro un ostacolo che la rende invisibile fino a pochissimi metri di distanza.
- Un ulteriore pedone che attraversa all'improvviso la strada in un rettilineo.

- Incroci a T e ad X non regolati da semafori, preceduti da segnali di stop, e alquanto trafficati. In particolare, subito dopo un incrocio a T in cui l'ego-veicolo deve svoltare a sinistra, compare un ciclista a bordo strada che inizia ad attraversare quest'ultima: è necessario fermarsi prontamente per evitare di investirlo. Inoltre, immediatamente dopo l'attraversamento dell'ultimo incrocio del percorso, che è a X, compare all'improvviso un pedone che attraversa la strada: anche in questo caso è necessario usare prudenza.

Si riportano di seguito delle immagini raccolte in simulazione ritraenti alcuni punti salienti dello scenario 4.



Figura 5. Alcune delle criticità dello scenario 4

3.6 Ulteriori considerazioni

Gli scenari sopra descritti sono alquanto eterogenei: 0 e 1 sono ambientati su strade extra-urbane veloci, 2 e 3 sono ambientati in un centro cittadino, 4 è ambientato in un contesto rurale. Le problematiche da

affrontare sono a loro volta molteplici ed eterogenee, in quanto comprendono: pedoni (anche disattenti), ciclisti, restringimenti improvvisi, veicoli e oggetti statici da superare invadendo la corsia opposta, lavori in corso, segnali di stop e semafori da rispettare, traffico più o meno intenso, curve ampie e veloci ed altre lente e strette, incroci di vario tipo, ecc.

In merito a questi ultimi, è importante evidenziare che differiscono tra loro rispetto a:

- presenza/assenza di semafori regolatori
- velocità degli altri veicoli
- volume di traffico
- larghezza delle strade che vi confluiscano e numero di corsie
- direzione della svolta da intraprendere
- presenza/assenza di ulteriori elementi di disturbo, come pedoni o ciclisti

Inoltre, è importante specificare che la velocità massima raggiungibile dell'ego-veicolo è di 50 km/h, indipendentemente dal tipo di strada percorsa.

4 Setup sperimentale

Questo capitolo fornisce una descrizione dettagliata delle **risorse hardware** adoperate, del setup dei **parametri del server** CARLA e delle **metriche** utilizzate per valutare le performance dell'algoritmo di navigazione.

4.1 Risorse hardware

Per eseguire i test in simulazione è stato usato un **server locale**. La macchina utilizzata ha una CPU AMD Ryzen 5 5600x, una GPU AMD Radeon RX 6700 XT e 32Gb di RAM DDR4 3200Mhz. Essendo i container di CARLA, disponibili dai repo ufficiali, predisposti per l'uso di CUDA e di GPU passthrough per GPU NVIDIA con nvidia-toolkit (e quindi incompatibili per l'uso con GPU AMD), CARLA è stato installato manualmente seguendo la documentazione [[link](#)].

Per installare ed eseguire *scenario manager* e *carla-leaderboard* è stato necessario utilizzare una distribuzione **Linux** con supporto a ROS. In questo modo è stato possibile soddisfare le dipendenze dei pacchetti ROS del gruppo ros-bridge che CARLA usa per la comunicazione dei sensori virtuali nell'ambiente di simulazione [[link](#)]. La distribuzione scelta è stata Ubuntu 20.04, con supporto a ROS Noetic.

Sia la componente server che quella client dell'architettura distribuita di CARLA sono state mandate in esecuzione sulla stessa macchina, al fine di ovviare all'overhead del protocollo TCP/IP e ai limiti della connessione che sarebbe stata utilizzata per far comunicare due macchine distinte. La comunicazione avrebbe potuto costituire un collo di bottiglia importante poiché nella modalità di simulazione sincrona di CARLA, che abbiamo utilizzato, il server aspetta il "tick" del client per procedere al rendering del prossimo step. Aggiungere i ritardi comunicativi a questo meccanismo può causare rallentamenti significativi. Gli script utilizzati per lanciare i test di simulazione impostano alcune variabili d'ambiente per stabilire i percorsi della root dell'installazione di CARLA, della leaderboard e dello scenario manager, i file xml con la definizione delle route da lanciare e diversi file di configurazione. L'unico programma che viene lanciato direttamente dallo script di avvio è *leaderboard_evaluator.py*.

4.2 Setup parametri del server

Nel file *leaderboard_evaluator.py* vengono settati dei parametri fondamentali per l'esecuzione del server di CARLA, che agiscono sia sulle **performance** che sulla **fedeltà della simulazione** e, per tale motivo, vanno tenuti in considerazione per la ripetibilità delle prove effettuate su macchine diverse. I parametri che abbiamo impostato sono principalmente quattro: **(i) sincronia**, **(ii) time-step**, **(iii) sincronia del traffic manager** e **(iv) fisica ibrida**. Come già accennato in precedenza, la sincronia determina il comportamento del server rispetto al client. Nel caso di una configurazione sincrona, il server, dopo aver completato i calcoli necessari per l'avanzamento di uno step temporale della simulazione, aspetta il client prima di procedere ad effettuare i calcoli per lo step successivo e far avanzare il tempo simulato. In modalità asincrona, invece, il server procede al massimo della velocità consentita dall'hardware per far avanzare la simulazione, senza fermarsi ad aspettare il client tra uno step e l'altro. Questo comporta che, se il client non riesce a stare al passo, i suoi controlli non vengono applicati al veicolo in ogni step del tempo simulato e ci sono quindi dei momenti in cui la simulazione avanza ma non si hanno ancora i nuovi input di controllo dal client.

La modalità di simulazione che abbiamo scelto è quella sincrona, la stessa utilizzata dal server messo a disposizione dal Dipartimento. Nel nostro caso, questo ha comportato una maggiore semplicità di gestione degli input del mondo simulato e dei conseguenti controlli da applicare, poiché ci ha permesso di assumere che tali input (grazie anche al parametro che sarà trattato a breve) fossero costanti e ognuno di essi fosse recepito dal client. Senza questa premessa, la gestione dei controlli sarebbe stata ancora più complessa da attuare a causa dell'imprevedibilità data dall'asincronia tra client e server, che avrebbe comportato il rischio di non applicare dei controlli critici in alcuni istanti temporali. In scenari futuri, usare la modalità asincrona potrebbe essere vantaggioso per creare un contesto simulativo più complesso ma più simile alla realtà, che è per sua natura asincrona. Dovendo usare la modalità sincrona, secondo la documentazione di CARLA occorre settare di conseguenza anche il parametro relativo alla sincronia del traffic manager, che altrimenti non procederebbe in maniera coerente con il resto della simulazione.

Un altro parametro fondamentale per la gestione della simulazione, lato server, è il **time-step**. Si può avere un time-step variabile o un time-step fisso. Nel primo caso il tempo della simulazione che passa tra due step consecutivi coincide con il tempo che il server impiega per computare gli stessi. C'è quindi una corrispondenza tra il tempo della simulazione e il tempo reale, la quale incide negativamente sulla ripetibilità delle prove: a seconda delle caratteristiche dell'hardware usato o di altri fattori che incidono sulle performance di calcolo della macchina in un dato momento, a un certo intervallo del tempo della simulazione corrisponderà un numero diverso di istanti temporali (time-step) prodotti. **Avendo invece un time-step fissato, come nel nostro caso, si impone una relazione tra il tempo della simulazione e il tempo di esecuzione:** usando un valore di 0.05 s per il time-step, ad esempio, sarà necessario produrre 20 frame per avanzare di un secondo nel tempo della simulazione. Questo introduce una corrispondenza tra il numero di frame prodotti e l'avanzamento del tempo della simulazione. Se il server produce un numero di frame al secondo maggiore dell'inverso del valore assegnato al parametro time-step, allora la simulazione procede più rapidamente (questo può essere utile per ridurre il tempo reale di esecuzione delle prove, se l'hardware lo consente). Chiaramente, nel caso opposto la simulazione procede al rallentatore, per cui occorre scegliere opportunamente il parametro del time-step per non portare all'estremo i tempi di esecuzione. Il time-step dovrebbe essere impostato anche in relazione al tempo impiegato dall'intera catena di elaborazione del software di guida autonoma; in particolare, dovrebbe essere sempre maggiore di quest'ultimo.

Con queste premesse, abbiamo verificato che il tempo di esecuzione del nostro algoritmo di guida autonoma finale, in esecuzione sul Client di CARLA, fosse effettivamente minore del time-step di simulazione (**nel nostro caso pari a 0.05s**). Ciò comporta che il nostro algoritmo posto in situazioni di guida reali sarebbe in grado di elaborare gli input che riceve e generare gli output di controllo al tasso di aggiornamento indicato. Chiaramente, gli input che arrivano al nostro algoritmo sono i dati perfetti del mondo nel caso della simulazione, mentre nel caso di uno scenario reale sarebbero i dati elaborati dai moduli di percezione a partire dai sensori. Nelle prove su strada, quindi, **l'elaborazione dei dati dei sensori andrebbe a costituire un ulteriore onere computazionale da gestire opportunamente.**

L'ultimo parametro che è stato considerato e gestito è quello relativo alla **modalità fisica ibrida**. Come indicato nella documentazione di CARLA [[link](#)], quando la modalità fisica ibrida è attiva, la fisica dei veicoli viene calcolata soltanto per quei veicoli che si trovano all'interno di un certo raggio (specificato dal parametro *radius*) di distanza dall'ego-veicolo. I veicoli che si trovano al di fuori di tale raggio vengono

teletrasportati ad ogni step, facendo dei calcoli semplificati sulla loro accelerazione lineare per stabilire l'entità della traslazione. Queste semplificazioni sono abbastanza efficaci da far sì che l'update della posizione dei veicoli distanti appaia comunque realistico e la transizione da quei veicoli per cui la fisica viene calcolata e quelli che vengono teletrasportati risulti visivamente fluida. La fisica ibrida è attivata in CARLA di default, con il parametro `radius` settato a 70. Attivare tale modalità permette di avere un guadagno prestazionale in alcuni casi significativo, a causa della riduzione del numero di calcoli relativi alla fisica dei veicoli. Tuttavia, questo rende la simulazione meno precisa e realistica, e potrebbe dare problemi nel momento in cui è essenziale avere dei dati precisi per valori come velocità e accelerazione (ad esempio quando bisogna calcolare i tempi necessari per effettuare un sorpasso guardando le velocità dei veicoli distanti). Per tali motivi si è scelto di impostare il parametro della fisica ibrida a false.

4.3 Metriche di valutazione

La valutazione delle performance dell'algoritmo di navigazione si fonda sulle metriche di valutazione di CARLA Autonomous Driving Leaderboard [\[link\]](#), una *open platform* il cui obiettivo principale è valutare la competenza di guida di agenti di veicoli autonomi, in scenari di traffico realistici, in modo equo e riproducibile, semplificando il confronto tra diversi approcci.

La competenza di guida di un agente può essere caratterizzata da più metriche. La *Leaderboard* di CARLA ha, pertanto, selezionato una serie di metriche che aiutano a comprendere i diversi aspetti della guida. Un agente viene tipicamente messo alla prova in molteplici scenari (*route*). Per ciascuno di essi vengono calcolate separatamente le medesime metriche, che sono le seguenti:

- **Route completion (score_route):** percentuale della lunghezza della *route* che viene completata da un agente.
- **Infraction penalty (score_penalty):** la *Leaderboard* tiene traccia di diversi tipi di infrazioni. Tale metrica aggrega tutte le infrazioni commesse dall'agente adoperando una *serie geometrica*. L'agente inizia con un punteggio ideale uguale a 1.0, che viene ridotto ad ogni infrazione commessa (vedi 4.3.1).

$$\prod_j^{ped,\dots,stop} (p_i^j)^{\# \text{infrazioni}_j}$$

- **Driving score (score_composed):** metrica principale della *Leaderboard*, definita come il prodotto tra il completamento del percorso (*route completion*) e la penalità dovuta alle infrazioni commesse (*infractions penalty*). Formalmente, indicando con R_i la percentuale di completamento e con P_i la penalità per le infrazioni dell'i-ima route, esso è dato da:

$$R_i * P_i$$

Al termine di tutte le route pianificate, viene generata anche una metrica globale per ciascuna delle tre precedenti, in termini di media aritmetica dei risultati ottenuti sulle singole route. Tali metriche sono denominate **global route completion**, **global infraction penalty** e **global driving score**. Tra queste, il

global driving score (media di tutti i driving score) è la metrica principale su cui la *Leaderboard* classifica i partecipanti e costituisce, pertanto, la metrica più rilevante per la valutazione del lavoro svolto. Per ciascuna metrica globale viene calcolata anche la deviazione standard, che, in quanto indice di dispersione intorno alla media, risulta utile per identificare la presenza di inconsistenze nelle prestazioni dell'agente. Ad esempio, se l'agente in alcuni scenari ottiene un punteggio alto e in altri ottiene un punteggio basso, la media potrebbe non riflettere accuratamente le prestazioni complessive. Tuttavia, un'alta deviazione standard indicherebbe la presenza di questa inconsistenza per la specifica metrica.

4.3.1 Infrazioni ed eventi di shutdown

La CARLA *Leaderboard* prevede delle metriche individuali per le infrazioni considerate. Ognuna di esse ha, infatti, un coefficiente di penalità che viene applicato ogni volta che si verifica. All'aumentare della penalità, dato che questa è un numero compreso tra 0 e 1, diminuisce la gravità dell'infrazione. Le infrazioni previste, ordinate per gravità decrescente e con i relativi coefficienti, sono le seguenti:

- Collisioni con i pedoni: 0.50
- Collisioni con altri veicoli: 0.60
- Collisioni con elementi statici: 0.65
- Non fermarsi al semaforo rosso: 0.70
- Non fermarsi allo stop: 0.80

Alcuni scenari presentano anche comportamenti che possono bloccare l'ego-veicolo indefinitamente. Questi scenari hanno un time-out di quattro minuti dopo i quali l'ego-veicolo viene rilasciato per continuare il suo percorso. Tuttavia, in caso di superamento di tale limite temporale viene generato uno *Shutdown Event* ed applicata una penalità:

- Time-out dello scenario: 0.70

Inoltre, l'agente dovrebbe mantenere una velocità minima in linea con il traffico circostante. La velocità dell'agente viene, dunque, confrontata con la velocità dei veicoli vicini. Il mancato mantenimento di una velocità adeguata comporta una penalità. La penalità applicata dipende dall'entità della differenza di velocità, fino al seguente valore:

- Mancato mantenimento della velocità minima: 0.70

L'agente deve anche dare precedenza ai veicoli di emergenza provenienti da dietro. La mancata precedenza ad un veicolo di emergenza incorre in una penalità:

- Mancata precedenza al veicolo di emergenza: 0.70

Oltre a queste, c'è un'altra infrazione che non ha coefficiente di penalità ma influenza il calcolo del completamento del percorso (R_i):

- Guida off-road: se un agente guida fuori strada (è sufficiente, in realtà, anche solo una sua parte), quella percentuale del percorso non sarà considerata nel calcolo del completamento del percorso (*route completion*).

In aggiunta, si possono verificare alcuni eventi (*shutdown events*) che interrompono la simulazione, impedendo all'agente di continuare. In questi casi, la *route* corrente viene interrotta e la simulazione passa a quella successiva, attivandola normalmente:

- **Deviazione del percorso:** se un agente devia più di 30 metri dal percorso assegnato.
- **Agente bloccato:** se un agente non intraprende alcuna azione per 180 secondi di simulazione.
- **Time-out di simulazione:** se nessuna comunicazione Client-Server può essere stabilita in 60 secondi.
- **Time-out del percorso:** se la simulazione di un percorso richiede troppo tempo per finire.

Ogni volta che uno qualsiasi di questi eventi accade, vengono registrati diversi dettagli, che sono visualizzati come un elenco nelle singole metriche del percorso.

Il *global infractions* comprime i dati del singolo percorso in un unico valore ed è dato come numero di eventi per Km.

Si noti, infine, che non vi è alcuna penalità associata all'eccesso di velocità da parte dell'ego-veicolo.

5 Analisi della baseline

L'analisi della **baseline** fornita da CARLA si articola nella presentazione della sua organizzazione in termini di moduli software e delle relazioni tra questi, nella descrizione delle funzionalità presenti, che costituiscono il punto di partenza dell'algoritmo di navigazione da realizzare, nella disamina delle sue prestazioni e, infine, nell'individuazione delle sue criticità, che si proveranno a risolvere nell'ambito del progetto.

5.1 Struttura

Al fine di chiarire la struttura software della baseline proposta da CARLA *Leaderboard*, nella figura seguente viene mostrato il diagramma UML della sua architettura:

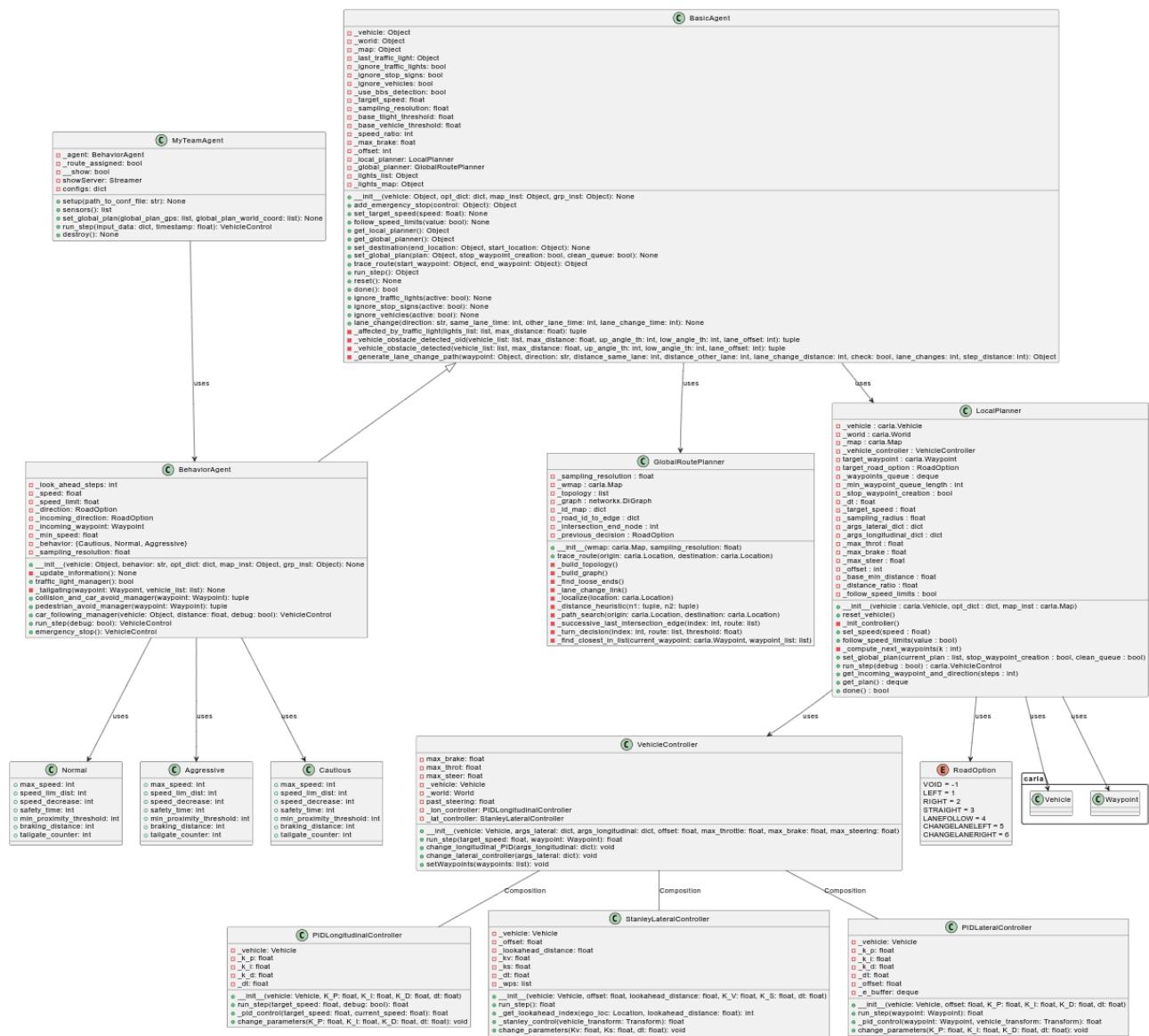


Figura 6. Diagramma UML dell'architettura software della baseline

Le principali classi della baseline e le relazioni tra esse sono illustrate nel seguito:

1. **MyTeamAgent**: rappresenta un agente di guida autonoma, che è un'entità che può percepire l'ambiente in cui si trova e agire su di esso. Si avvale di un *BehaviorAgent* per implementare la logica di guida da attuare.
2. **BasicAgent**: rappresenta un agente di guida molto semplice che può interagire con il mondo simulato di CARLA per ottenere informazioni sull'ambiente (ad es. la posizione dell'ego-veicolo) e, sulla base di ciò e del suo obiettivo, impedisce comandi di controllo (sterzata, accelerazione e frenata) al veicolo. Utilizza *GlobalRoutePlanner*, da cui riceve il "piano" globale da seguire, e *LocalPlanner* per la definizione del "piano" locale.
3. **BehaviorAgent**: estende *BasicAgent*, introducendo comportamenti di guida più avanzati, avvalendosi delle classi *Cautious*, *Normal* e *Aggressive*.
4. **RoadOption**: è un'enumerazione utilizzata per rappresentare le possibili direzioni di guida, come: girare a sinistra, a destra, andare dritto, seguire la corsia, cambiare corsia a sinistra o a destra.
5. **VehicleController**: ha il compito di controllare l'ego-veicolo. Si compone di sottoclassi specifiche per il controllo longitudinale e laterale del veicolo: *PIDLongitudinalController*, *StanleyLateralController* e *PIDLateralController*, le quali implementano rispettivamente un controllore PID (Proportional-Integral-Derivative) longitudinale, un controllore Stanley laterale, un controllore PID laterale.
6. **LocalPlanner**: si occupa della generazione di un percorso locale per il veicolo, sfruttando un oggetto *VehicleController* per guidare il veicolo verso un waypoint obiettivo. Adopera anche l'enumerazione *RoadOption* per definire il comportamento del veicolo sul percorso.
7. **Cautious, Normal, Aggressive**: definiscono differenti stili di guida che possono essere assunti da un *BehaviorAgent*.
8. **GlobalRoutePlanner**: genera un piano di percorso globale per l'agente, sfruttando l'API di CARLA per ottenere informazioni sulla mappa della simulazione.

Di tale architettura si descrivono di seguito nel dettaglio le classi *BasicAgent* e *BehaviorAgent*, in quanto maggiormente rilevanti rispetto agli scopi del progetto. Viene, invece, fornita una descrizione meno dettagliata per le classi *LocalPlanner*, *GlobalRoutePlanner* e *VehicleController*.

5.2 Funzionalità

Si riportano di seguito le funzionalità già implementate dalla baseline fornita, in particolare, analizzando la logica con cui queste sono state realizzate.

5.2.1 BasicAgent

BasicAgent rappresenta la classe base dalla quale partire per costruire un agente autonomo. Essa utilizza il modulo *GlobalPlanner* per seguire un percorso specifico a livello globale e il modulo *LocalPlanner* per definire il percorso da effettuare a livello locale. Il modulo in esame definisce anche un algoritmo di navigazione basilare, che si occupa di evitare la possibile collisione con altri veicoli e di gestire la presenza o meno di semafori rossi. *BasicAgent*, alla presenza di questi eventi, effettua, infatti, semplicemente una frenata di emergenza. Tale classe presenta, dunque, una serie di attributi e metodi che consentono di svolgere queste semplici azioni:

- **affected_by_traffic_light:** tale metodo tiene conto dei semafori che agiscono sul veicolo. I semafori che vengono identificati come averti effetto sull'agente sono quelli rivolti verso di esso (ossia veicolo e semaforo non sono in direzioni opposte), sulla sua stessa strada e che sono entro una certa distanza (di default 5 metri). A partire dai waypoints associati ai semafori, esso ottiene il punto in cui l'ego-veicolo dovrebbe fermarsi. Se esiste un semaforo davanti al quale ci si è fermati al passo precedente, allora valuta se è ancora rosso. Se non è più rosso, si passa a valutare eventuali altri semafori. Restituisce l'attore (semaforo) che influenza il veicolo e il valore booleano True nel caso in cui la luce del semaforo sia di colore rosso.
- **vehicle_obstacle_detected:** questo metodo controlla la presenza di veicoli di fronte all'agente che ne bloccano il passaggio. Il controllo viene effettuato in maniera diversa a seconda che l'agente o il target si trovino o meno all'interno di un incrocio. Questa differenziazione viene fatta a causa della natura degli incroci, all'interno dei quali le informazioni fornite dai waypoints non sono sufficientemente affidabili.

In particolare, per quanto riguarda il comportamento al di fuori degli incroci, vengono utilizzate le informazioni sulle *lane_id* e sulla *road_id* per filtrare i possibili veicoli coi quali si possono avere collisioni. Una volta filtrati solamente i veicoli che si trovano sulla stessa strada dell'agente e sulla stessa corsia (tenendo conto di un offset in modo da avere maggior margine), per essi si calcola la distanza tra la posizione frontale dell'agente e la posizione posteriore del veicolo. Se questa distanza è minore di una certa soglia data come parametro, allora viene restituito True.

Il comportamento all'interno degli incroci, invece, non fa uso delle informazioni di lane e road, bensì si avvale delle informazioni delle posizioni dei veicoli rispetto a quella dell'agente: viene proiettato il bounding box dell'ego-veicolo lungo la traiettoria che deve percorrere per una distanza *max_distance* fornita come parametro. Si considerano poi tutti i veicoli entro la *max_distance* e si crea per ognuno di essi un poligono, utilizzando le informazioni del bounding box. Se uno di questi poligoni interseca la proiezione del bounding box dell'agente lungo la sua traiettoria, allora vengono restituiti il valore booleano True, il veicolo target e la sua distanza dall'agente; nel caso in cui non ci sia nessuno sulla traiettoria dell'agente viene, invece, restituito False.

- **generate_lane_change_path:** metodo che genera un path utilizzabile per cambiare corsia, nel caso in cui il cambio di corsia sia possibile. Quest'ultimo è considerato possibile se la corsia adiacente è di tipo *driving* e il suo senso di marcia è il medesimo della corsia attuale dell'ego-veicolo oppure si tratta di una corsia a doppio senso. Esso prende in ingresso il waypoint da cui deve partire il percorso per il cambio di corsia, la direzione verso cui spostarsi, le distanze da percorrere rispettivamente sulla corsia iniziale, su quella di cambio e durante lo spostamento tra una corsia e

l'altra, il numero di cambi di corsia e la distanza a cui campionare i nuovi waypoints del path, detta *step_distance*.

Per ognuna delle tre distanze in ingresso, si generano i waypoint necessari per percorrerle completamente. La prima parte del percorso viene generata a partire dal waypoint in ingresso, prendendo un waypoint ogni *step_distance* metri, fino a che non si è percorsa la prima distanza sulla corsia iniziale. Successivamente, si considera la distanza da percorrere durante il cambio corsia. A partire dall'ultimo waypoint generato fino a quel punto, si genera un nuovo waypoint alla distanza indicata da percorrere nel cambio di corsia. Se sono stati indicati più cambi di corsia, questa fase viene eseguita più volte, finché non viene raggiunta la distanza specificata. Infine, vengono generati i waypoint sulla corsia di destinazione, per una distanza complessiva pari a quella indicata. Se il cambio di corsia non è possibile, viene restituito un path vuoto.

- **lane_change:** richiama il metodo *generate_lane_change_path* per creare un percorso che rappresenta un cambio di corsia. A differenza di quest'ultimo, in ingresso riceve il tempo (e non le distanze) per cui l'agente deve permanere in ciascuno step del cambio corsia. Calcola, quindi, ciascuna distanza necessaria a *generate_lane_change_path* moltiplicando il tempo in ingresso per la velocità a cui si sta spostando il veicolo. Si noti che tale funzione assume che la velocità dell'auto rimanga costante durante tutto il cambio di corsia; se la velocità dell'auto cambiasse, le distanze calcolate potrebbero non corrispondere al tempo desiderato per il cambio di corsia.

5.2.2 BehaviourAgent

La classe *BehaviourAgent* implementa un agente che percorre gli scenari per raggiungere una destinazione obiettivo, calcolando il percorso migliore possibile. Essa specializza *BasicAgent*, migliorando in particolar modo l'algoritmo di navigazione, che si serve in maniera più completa dei metodi di *awareness* (in particolare *vehicle_obstacle_detected*) presenti nella classe ereditata. Essa definisce, infatti, dei *manager* che vengono integrati nell'algoritmo di navigazione dell'agente e che dovrebbero permettergli di evitare le collisioni con gli altri veicoli, mantenendo una distanza di sicurezza dal veicolo davanti, di fermarsi al semaforo rosso, e di evitare le collisioni con i pedoni. Inoltre, supporta tre diversi stili di guida: normale, cauto e aggressivo.

I principali metodi di tale classe sono:

- **update_information:** è responsabile dell'aggiornamento delle informazioni relative all'ego veicolo sulla base dello stato attuale del mondo circostante. Esso aggiorna la velocità corrente, il limite di velocità, la *road option* (di default uguale a "segui la corsia"), il numero di step avanti a cui guardare, nonché il prossimo waypoint e la direzione.
- **traffic_light_manager:** determina unicamente se l'auto è influenzata da un semaforo rosso tramite il metodo *affected_by_traffic_light*. Il comportamento dell'auto quando è influenzata da un semaforo (ad esempio, fermarsi ad aspettare), in realtà, non viene affatto gestito.
- **tailgating:** serve a gestire il cambio di corsia dell'agente in caso di veicoli che sopraggiungono a velocità maggiore rispetto ad esso. Tale metodo, nel caso in cui esista un veicolo dietro l'agente

che viaggia più velocemente di quest'ultimo, controlla che il cambio di corsia sia consentito (si hanno più corsie per senso di marcia e non ci sono ostacoli nella corsia di destinazione).

- **pedestrian_avoid_manager:** adottato per evitare collisioni con i pedoni nelle vicinanze del veicolo (entro 10 metri), fa uso del metodo `vehicle_obstacle_detected`, che può essere in realtà utilizzato su qualsiasi tipo di attore di CARLA, per identificare se ci sono possibili collisioni con esso. All'interno dell'algoritmo di navigazione, se dei pedoni vengono identificati come in rischio di collisione con l'agente e, la loro distanza è minore della distanza di frenata prevista, allora si effettua una frenata di emergenza.
- **collision_and_car_avoid_manager:** metodo richiamato all'interno del ciclo di controllo principale per prevenire collisioni con altri veicoli, sulla base del waypoint corrente dell'ego-veicolo e dei veicoli nelle vicinanze. Considera i veicoli entro 45 metri e verifica la presenza di ostacoli avvalendosi della funzione di BasicAgent `vehicle_obstacle_detected`, con parametri diversi a seconda che l'agente debba cambiare corsia verso sinistra o verso destra oppure debba andare dritto. Controlla anche il tailgating se il veicolo sta seguendo la sua corsia, non si trova in un incrocio, viaggia a una velocità superiore a 10 km/h e il contatore del tailgating è a zero. Tale funzione restituisce il medesimo output di `vehicle_obstacle_detected` (il veicolo ostacolo e la distanza da esso, oltre a un booleano).
- **car_following_manager:** manager incaricato di seguire (se presente) il veicolo di fronte l'ego-veicolo, adattando la velocità di conseguenza. Viene chiamato all'interno del ciclo di controllo se la funzione `collision_and_car_avoid_manager` restituisce un valore diverso da False. In tal caso, il `car_following_manager` regola la velocità dell'agente, in relazione al veicolo che lo precede e della sua velocità. In particolare, il metodo calcola il TTC (*Time To Collision*) definito come: $TTC = \frac{d}{v_{ego} - v_{tgt}}$, dove v_{ego} e v_{tgt} sono rispettivamente la velocità dell'agente e quella del veicolo target, mentre d è la distanza tra essi.

Se il TTC è minore del `safety_time` definito dallo stile di guida, il veicolo rallenta, cercando di mantenere una velocità target che è il minimo tra la velocità del veicolo da seguire diminuita di un certo valore, la velocità massima dell'agente (entrambi dipendenti dallo stile di guida) e il limite di velocità.

Se il TTC è compreso tra il `safety_time` e il suo doppio, il veicolo cerca di seguire la velocità del veicolo davanti, mantenendo una velocità target che è il minimo tra la velocità attuale del veicolo da seguire (se superiore a una velocità minima), la velocità massima permessa dal comportamento del veicolo e il limite di velocità meno una data quantità `speed_lim_dist` per garantire una distanza di sicurezza.

In tutti gli altri casi (cioè quando il veicolo non è a rischio di collisione), la velocità target dell'agente è uguale al minimo tra la velocità massima dipendente dallo stile di guida e la differenza tra il limite di velocità della strada e `speed_lim_dist`.

- **emergency_stop**: è responsabile della creazione di un comando di controllo del veicolo per eseguire un arresto di emergenza quando necessario, impostando l'acceleratore a 0, il freno al valore massimo di frenata e il freno a mano a False.
- **run_step**: definisce il flusso principale di controllo e dunque l'algoritmo di navigazione adottato dall'agente, includendo l'aggiornamento delle informazioni sul mondo e la gestione di vari comportamenti del veicolo ad ogni step di simulazione, per consentirgli di rispondere dinamicamente al mondo che cambia. Di seguito è riportata una rappresentazione grafica del suo flusso di istruzioni:

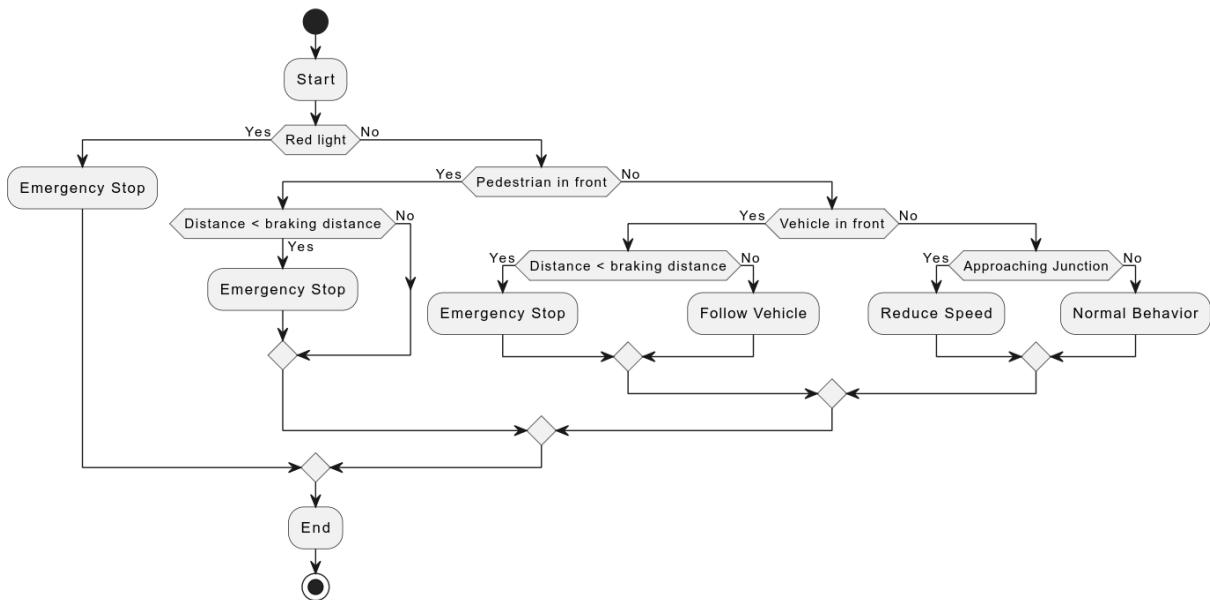


Figura 7. Flusso logico del run step della baseline

La logica di navigazione è divisa in diverse parti:

- **Aggiornamento**: vengono aggiornate le informazioni relative all'ego-veicolo rispetto al mondo.
- **Tailgating**: controlla se il veicolo è attualmente nel mezzo di un "tailgating" e, in caso affermativo, decrementa il contatore ad esso associato.
- **Controllo dei semafori rossi**: se il veicolo arriva dinanzi a un semaforo rosso, esegue un arresto di emergenza.
- **Evitamento dei pedoni**: se c'è un pedone in prossimità dell'ego-veicolo e la distanza tra loro è inferiore a una certa distanza di frenata, l'ego-veicolo effettua una frenata di emergenza.
- **Following dei veicoli**: se c'è un veicolo davanti all'agente, ad una distanza inferiore a quella di frenata, l'ego-veicolo esegue una frenata di emergenza, altrimenti segue l'altro veicolo mantenendo una distanza di sicurezza.
- **Incrocio**: se il veicolo sta per entrare in un incrocio e deve svoltare a sinistra o a destra, riduce la sua velocità.
- Se nessuna delle condizioni precedenti è soddisfatta, il veicolo continua normalmente, cercando di mantenere una velocità target che è la minima tra la velocità massima dello stile di guida e il limite di velocità meno una quantità per garantire una certa distanza di sicurezza.

5.2.3 LocalPlanner

Questa classe rappresenta il Local Planner dell'agente ed implementa il comportamento base di seguire una traiettoria di waypoint generata in tempo reale. I waypoint rappresentano punti di passaggio o destinazioni lungo il percorso che il veicolo deve seguire. Per fare ciò, si avvale dei controllori forniti dalla classe *VehicleController*.

Quando sono disponibili percorsi multipli (ad esempio, agli incroci), il pianificatore locale fa una scelta casuale, a meno che non sia già stato specificato un piano globale.

La classe permette anche di definire, tra gli altri parametri, la velocità di crociera desiderata del veicolo, la distanza tra i waypoint, la massima sterzata, frenata e accelerazione applicabili al veicolo.

Infine, la classe offre anche funzionalità per visualizzare in debug i waypoint, cambiare la velocità di crociera target, attivare il rispetto dei limiti di velocità, fornire il prossimo waypoint e la sua direzione rispetto al veicolo, ottenere la velocità attuale del veicolo e altre funzioni di utilità.

5.2.4 GlobalPlanner

Tale classe implementa la pianificazione di un percorso ad alto livello che consenta all'ego-veicolo di raggiungere una destinazione: attraverso l'utilizzo di una mappa del mondo e un valore di risoluzione di campionamento, costruisce un grafo rappresentante la topologia della mappa e permette di tracciare un percorso tra due punti (origine e destinazione) utilizzando l'algoritmo A*.

5.2.5 VehicleController

La classe *VehicleController* combina un controllore PID longitudinale ed un controllore laterale di tipo Stanley, rispettivamente implementati dalle classi *PIDLongitudinalController* e *StanleyLateralController*, al fine di eseguire il controllo a basso livello dell'agente, ovvero per seguire la traiettoria stabilita dal Local Planner, sia in termini di velocità sia in termini di path. Si noti che viene messo a disposizione anche un controllore PID laterale, implementato dalla classe *PIDLateralController*, il quale, però, non viene utilizzato. Il metodo più rilevante della classe *VehicleController* prende il nome di *run_step* ed è deputato all'esecuzione di uno step di controllo, che viene realizzato richiamando sia il controllore laterale sia il controllore longitudinale, con l'obiettivo di raggiungere un waypoint target ad una velocità target.

5.2.5.1 PIDLongitudinalController

La classe *PIDLongitudinalController* implementa un PID per il controllo della velocità dell'agente. La legge di controllo realizzata, la cui struttura comprende la somma di tre termini di controllo, è la seguente:

$$\ddot{x}_{des} = K_P(\dot{x}_{ref} - \dot{x}) + K_I \int_0^t (\dot{x}_{ref} - \dot{x}) dt + K_D \frac{d(\dot{x}_{ref} - \dot{x})}{dt}$$

dove:

- \dot{x}_{ref} è la velocità di riferimento per l'ego-veicolo, ovvero la sua velocità desiderata;
- \dot{x} è la velocità effettiva dell'ego-veicolo;

- \ddot{x}_{des} è l'output del controllore e rappresenta l'accelerazione desiderata per compensare il divario tra la velocità di riferimento \dot{x}_{ref} e la velocità effettiva dell'ego-veicolo \dot{x} , le quali rappresentano l'input del controllore medesimo;
- $(\dot{x}_{ref} - \dot{x})$ è l'errore;
- K_P, K_I, K_D sono rispettivamente i guadagni (coefficienti) proporzionale, integrale e derivativo del controllore PID;
- $K_P(\dot{x}_{ref} - \dot{x})$ è il termine proporzionale all'errore, pertanto induce un'azione di controllo direttamente proporzionale all'errore istantaneo, che tende a ridurne il valore. Può causare un effetto di oscillazione intorno al valore desiderato se usato da solo, senza gli altri termini del controllore;
- $K_D \frac{d(\dot{x}_{ref} - \dot{x})}{dt}$ è il termine proporzionale alla variazione temporale (derivata) dell'errore, pertanto introduce una correzione proporzionale alla velocità di cambiamento dell'errore. Tale termine può essere interpretato come una forma di anticipazione del futuro comportamento dell'errore; tuttavia, un coefficiente K_D troppo elevato può amplificare il rumore o provocare una risposta eccessivamente reattiva;
- $K_I \int_0^t (\dot{x}_{ref} - \dot{x}) dt$ è il termine proporzionale all'integrale dell'errore, pertanto consente di tener conto dell'errore accumulato nel tempo, al fine di poterlo compensare. Tale termine, in sostanza, si occupa dell'errore a regime, che potrebbe non essere corretto a dovere dai termini proporzionale e derivativo da soli. Si noti, tuttavia, che una regolazione impropria del coefficiente K_I può portare ad un'eccessiva amplificazione degli errori transitori.

Un'opportuna combinazione dei termini proporzionale, derivativo e integrale consente di ottenere un controllo semplice ed efficiente. Naturalmente, per raggiungere tale scopo, è necessario attuare una procedura di taratura che porti alla scelta dei K_P, K_I, K_D più appropriati, a seconda del processo da controllare e delle specifiche di progetto. Il PID è il regolatore più utilizzato nella pratica industriale grazie alla sua efficacia, standardizzazione, economicità, affidabilità e semplicità di taratura. Quest'ultima può essere addirittura automatizzata per mezzo di semplici esperimenti.

È fondamentale osservare che la legge di controllo che viene effettivamente implementata da *PIDLongitudinalController* non è quella riportata sopra, bensì una sua versione discretizzata:

- Si ricava l'errore err come differenza tra la velocità target e la velocità effettiva;
- Si ricava la derivata dell'errore de come rapporto incrementale degli ultimi due errori calcolati;
- Si ricava il contributo integrale ie come somma degli ultimi 10 errori calcolati moltiplicata per il tempo di campionamento Δt ;
- Si calcola $K_P err + K_D de + K_I ie$ e si limita il risultato entro l'intervallo (-1,1)

Si noti che quando si lavora nel discreto, oltre ai guadagni del PID, è necessario definire anche il suo tempo di campionamento Δt , ovvero l'inverso della frequenza con cui il PID effettua le sue operazioni di calcolo e aggiornamento.

5.2.5.2 StanleyLateralController

La classe *StanleyLateralController* implementa un controllore di Stanley per il controllo dell'angolo di sterzata dell'agente. Il controllore di Stanley è stato ideato da un team di ricercatori presso la Stanford

University nei primi anni 2000 nell'ambito della *DARPA Grand Challenge*, una competizione organizzata dall'agenzia di ricerca avanzata della difesa degli Stati Uniti (DARPA) per promuovere lo sviluppo di veicoli autonomi.

Il controllore di Stanley mira a minimizzare due errori:

- **Heading error:** l'errore di orientamento dell'agente rispetto alla traiettoria prestabilita, espresso come differenza tra la direzione corrente dell'ego-veicolo e la direzione del waypoint obiettivo;
- **Crosstrack error:** il disallineamento laterale rispetto alla traiettoria prestabilita, espresso come distanza tra il centro dell'asse anteriore dell'ego-veicolo ed il punto più vicino del path.

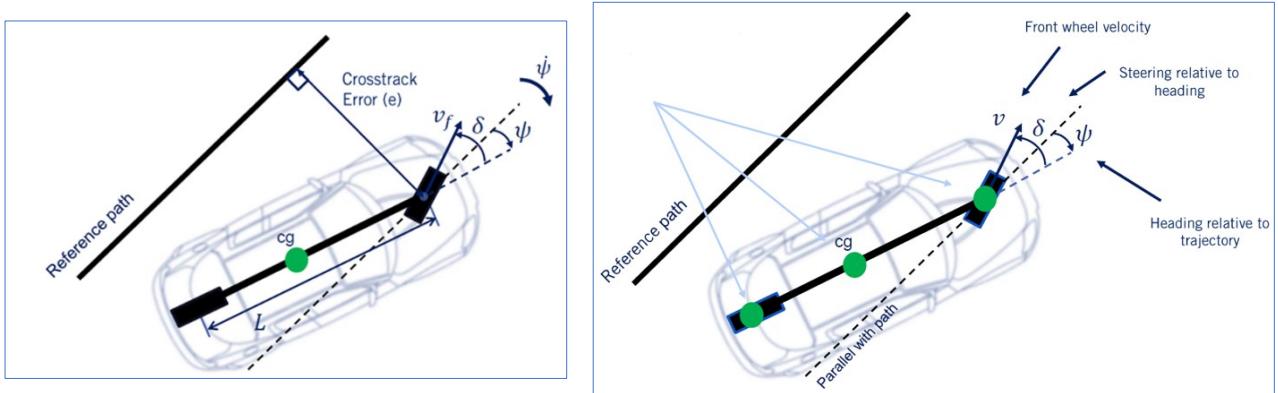


Figura 8. Crosstrack error e Heading error

Pertanto, il controllore di Stanley definisce una legge di sterzata che:

1. Corregge l'errore di orientamento (heading error);
2. Corregge l'errore di crosstrack;
3. Rimane nei limiti fisici della sterzata, i quali dipendono dal modello del particolare veicolo considerato. Di fatto, l'azione (angolo) di sterzata che può essere impartita in un certo istante deve rientrare in un certo range.

La legge di controllo di Stanley è la seguente:

$$\delta(t) = \psi(t) + \tan^{-1} \left(\frac{k_v e(t)}{k_s + v_f(t)} \right), \quad \delta(t) \in [\delta_{min}, \delta_{max}]$$

dove:

- $\psi(t)$ è l'errore di orientamento al tempo t ;
- $\tan^{-1} \left(\frac{k_v e(t)}{k_s + v_f(t)} \right)$ è il termine che porta in conto l'errore di crosstrack, ed è direttamente proporzionale a quest'ultimo e inversamente proporzionale alla velocità del veicolo. Si osservi, infatti, che:
 - $e(t)$ è proprio l'errore di crosstrack;
 - k_v è il coefficiente di guadagno per l'errore di crosstrack. Tale parametro può essere opportunamente regolato per determinare quanto velocemente il veicolo corregge l'errore medesimo;

- $v_f(t)$ è la velocità del veicolo al tempo t. L'inversa proporzionalità dipende dal fatto che a velocità elevata non sono desiderate sterzate brusche, in quanto potrebbero avere esiti estremamente negativi. All'aumentare della velocità e a parità di tutti gli altri parametri, pertanto, aumenta il tempo necessario a correggere l'errore di crosstrack, e la traiettoria percorsa è più dolce;
- k_s è un coefficiente di stabilità numerica, che viene introdotto per evitare che, a velocità $v_f(t)$ nulla o prossima a zero, il termine di crosstrack vada a infinito. Esattamente come k_v , anche k_s va opportunamente regolato: esso non deve essere né troppo piccolo né troppo grande: nel primo caso non riuscirebbe a risolvere il problema di instabilità numerica, nel secondo renderebbe troppo difficile sterzare in modo significativo, soprattutto a velocità sostenute;
- L'arcotangente \tan^{-1} viene introdotta al fine di limitare l'azione di controllo per errori elevati.
- $\delta(t)$ è l'angolo di sterzata risultante ed è dato dalla somma del termine di heading error e del termine che tiene conto dell'errore di crosstrack;
- $[\delta_{min}, \delta_{max}]$ è l'intervallo che stabilisce i limiti fisici di sterzata. L'angolo di sterzata calcolato $\delta(t)$ viene limitato in tale intervallo per assicurarsi che non superi la capacità massima di sterzata del veicolo.

La classe *StanleyLateralController*, che ovviamente lavora nel discreto, funziona nel modo seguente:

- Si calcolano la velocità e la posizione correnti dell'ego-veicolo;
- Si individua il waypoint successivo lungo la traiettoria da seguire, specificando la distanza minima dalla posizione attuale a cui quest'ultimo deve essere collocato;
- Dato un waypoint da raggiungere, occorre anche stabilire l'orientamento desiderato con il quale arrivarci. Per calcolarlo si utilizzano il waypoint individuato al punto precedente e quello immediatamente successivo lungo la traiettoria. Nel caso in cui il waypoint individuato al punto precedente fosse l'ultimo della traiettoria pianificata, allora per il calcolo dell'orientamento desiderato ci si affida ad esso e a quello che lo precede lungo la traiettoria;
- Si effettua il calcolo dell'errore di crosstrack come distanza tra la posizione corrente dell'agente ed il segmento che congiunge i due waypoint considerati al punto precedente;
- Si effettua il calcolo dell'errore di heading come differenza tra l'orientamento corrente dell'agente e l'orientamento desiderato. Naturalmente, i due orientamenti in questione sono due angoli;
- Noti l'errore di heading e l'errore di crosstrack, viene finalmente applicata la legge di controllo di Stanley, il cui output viene infine limitato entro il range (-1,1).

5.2.5.3 PIDLateralController

La classe *PIDLateralController* implementa il controllo laterale con un controllore di tipo PID, ricevendo in input l'offset dal centro della corsia, i guadagni del controllore e l'intervallo di tempo tra le iterazioni di controllo (dt). Dato il waypoint obiettivo, il controllore restituisce un valore di controllo di sterzata tra -1 e 1, dove -1 rappresenta la sterzatura massima a sinistra e 1 rappresenta la sterzatura massima a destra, in funzione dell'errore tra la direzione corrente del veicolo e la direzione verso il waypoint.

5.3 Prestazioni e criticità

Si fornisce di seguito un'analisi delle prestazioni della baseline della Leaderboard di Carla, che si evincono dai dati relativi ai punteggi e alle infrazioni commesse dall'agente nel corso della simulazione preliminare effettuata. Quest'ultima è stata condotta senza apportare alcuna modifica al codice della baseline. È opportuno evidenziare che la baseline adopera come parametri del controllore PID longitudinale:

$$K_p = 1.95; K_i = 0.05; K_d = 0.2; \Delta t = 0.05 s$$

mentre ha come parametri del controllore laterale Stanley:

$$K_v = 1; K_s = 0.01; \Delta t = 0.05 s$$

Le metriche di valutazione adottate sono quelle descritte nel paragrafo 4.3.

Per quanto concerne le medie dei punteggi (*scores_mean*) conseguiti nei 4 scenari, l'agente ha ottenuto:

- score_composed = 22.197683 %
- score_route = 82.428 %
- score_penalty = 0.269948

Per quanto riguarda le deviazioni standard dei punteggi (*scores_std_dev*) conseguiti nei 4 scenari, l'agente ha ottenuto:

- score_composed = 27.653
- score_route = 39.292
- score_penalty = 0.256

Da tali valori si può dedurre che l'agente riesce a completare la maggior parte del percorso nei vari scenari, tuttavia, commette un notevole numero di infrazioni, anche gravi.

In particolare, rispetto alle infrazioni, sembra che l'agente abbia problemi rilevanti con le collisioni con i veicoli e i pedoni, oltre alle collisioni con elementi dell'ambiente. L'agente non ha infranto alcun semaforo rosso, ma ha registrato infrazioni per non essersi fermato agli stop quando necessario e per aver guidato al di sotto della velocità minima. Ci sono anche alcuni casi di time-out di scenario, ma non sembrano esserci deviazioni dal percorso o blocchi del veicolo.

La deviazione standard dei punteggi è piuttosto alta per score_composed e score_route, il che indica che l'agente ha mostrato una prestazione alquanto inconsistente tra i diversi scenari.

Guardando ai record dei singoli scenari, si nota che l'agente ha completato quasi tutti i percorsi, ad eccezione di "RouteScenario_1", dove ha riscontrato un time-out.

Nel complesso, quindi, sembra che l'algoritmo di navigazione dell'agente abbia bisogno di miglioramenti nel prevenire collisioni con veicoli, pedoni e oggetti dell'ambiente, così come nel rispettare le regole di stop e mantenere una velocità adeguata.

[-]global_record { }	
index	-1
route_id	-1
status	Failed
infractions [-]infractions { }	
collisions_layout 0.525	
collisions_pedestrian 0.525	
collisions_vehicle 2.495	
red_light 0	
stop_infraction 0.656	
outside_route_lanes 0.018	
min_speed_infractions 0.788	
yield_emergency_vehicles_infractions 0	
scenario_timeouts 1.313	
route_dev 0	
vehicle_blocked 0	
route_timeout 0.131	
scores_mean [-]scores_mean { }	
score_composed 22.197683	
score_route 82.428	
score_penalty 0.269948	
scores_std_dev [-]scores_std_dev { }	
score_composed 27.653	
score_route 39.292	
score_penalty 0.256	



Figura 9. Risultati simulazione baseline con parametri dei controllori forniti e areogramma delle infrazioni complessive

5.3.1 Scenario 0

Si analizzano, di seguito, nel dettaglio le infrazioni commesse dall'agente nello scenario 0:

- **Collisioni con oggetti statici:** l'agente ha avuto tre collisioni con oggetti statici sulla strada, tra cui un segnale di avvertimento per il traffico, detriti di terra e un cono da cantiere. Questo potrebbe indicare una difficoltà dell'agente nel rilevare e/o evitare ostacoli fissi sul suo percorso.
- **Collisioni con pedoni:** l'agente ha avuto due collisioni con pedoni. Questo è un comportamento estremamente pericoloso che mette a rischio la sicurezza dei pedoni. È importante che l'agente sia in grado di rilevare ed evitare pedoni in tutte le circostanze, essendo questi gli utenti della strada più vulnerabili.
- **Collisioni con veicoli:** l'agente ha avuto ben 14 collisioni con altri veicoli, tra cui 3 biciclette. Questo numero elevato di collisioni indica che l'agente ha difficoltà a mantenere la propria corsia, a seguire una distanza di sicurezza adeguata dagli altri veicoli, o a reagire correttamente alle manovre degli altri veicoli.
- **Infrazione di stop:** l'agente ha ignorato uno stop, ciò indica che l'agente non si è fermato come richiesto dalla segnaletica stradale, il che potrebbe suggerire un problema con il rilevamento o l'interpretazione dei segnali di stop. È, invece, fondamentale che un agente autonomo rispetti tutte

le segnalazioni stradali al fine di garantire la sicurezza della circolazione ed evitare incidenti, in particolar modo agli incroci.

- **Infrazione fuoriuscita dalla corsia:** l'agente è uscito dalla sua corsia per circa 17.7 metri che sebbene rappresenti solo lo 0,42% del percorso totale, indica che l'agente potrebbe avere difficoltà a mantenere la corretta posizione sulla strada.
- **Infrazione della velocità minima:** l'agente ha viaggiato in una occasione ad una velocità media significativamente inferiore rispetto al traffico circostante e ciò potrebbe contribuire a congestioni e rallentamenti del traffico.
- **Time-out dello scenario:** l'agente ha avuto otto time-out dello scenario. Questo suggerisce che l'agente potrebbe avere problemi a completare determinate manovre o azioni in un tempo ragionevole, non essendo in grado di prendere decisioni o di reagire in modo tempestivo, portando a ritardi o inefficienze durante il tragitto.

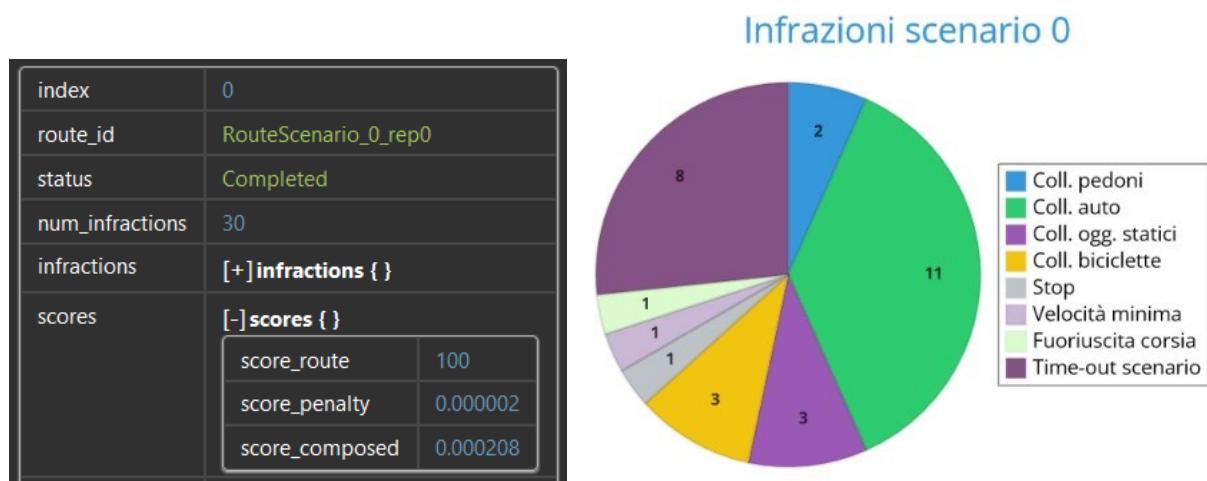


Figura 10. Tabella punteggi e areogramma infrazioni scenario 0

5.3.2 Scenario 1

In tale scenario l'agente ha commesso un totale di 4 infrazioni prima di non riuscire a completare il percorso a causa di un time-out. Si esamina di seguito ciascuna di queste infrazioni:

- **Collisioni con oggetti statici:** l'agente ha avuto una collisione con un oggetto statico del tipo segnale di traffico.
- **Collisioni con veicoli:** l'agente ha avuto una collisione con un altro veicolo.
- **Time-out dello scenario:** l'ego-veicolo è rimasto bloccato fino ad un time-out di quattro minuti dopo i quali è stato rilasciato per continuare il suo percorso.
- **Time-out del percorso:** l'agente non è riuscito a completare il percorso entro il tempo assegnato. Questo indica che l'agente può avere difficoltà a navigare in modo efficiente, rimanendo addirittura bloccato in certe situazioni.

index	1						
route_id	RouteScenario_1_rep0						
status	Failed - Agent timed out						
num_infractions	4						
infractions	[+] infractions { }						
scores	[-] scores { } <table border="1"> <tr> <td>score_route</td> <td>12.14</td> </tr> <tr> <td>score_penalty</td> <td>0.273</td> </tr> <tr> <td>score_composed</td> <td>3.31422</td> </tr> </table>	score_route	12.14	score_penalty	0.273	score_composed	3.31422
score_route	12.14						
score_penalty	0.273						
score_composed	3.31422						



Figura 11. Tabella punteggi e areogramma infrazioni scenario 1

5.3.3 Scenario 2

Nello scenario 2 l'agente ha commesso 3 infrazioni:

- **Collisioni con veicoli:** l'agente è entrato in collisione con un altro veicolo.
- **Infrazione di stop:** l'agente ha commesso un'infrazione a uno stop.
- **Infrazioni sulla velocità minima:** l'agente ha mantenuto una velocità media del 96.88% rispetto a quella del traffico circostante.

index	2						
route_id	RouteScenario_2_rep0						
status	Completed						
num_infractions	3						
infractions	[+] infractions { }						
scores	[-] scores { } <table border="1"> <tr> <td>score_route</td> <td>100</td> </tr> <tr> <td>score_penalty</td> <td>0.475507</td> </tr> <tr> <td>score_composed</td> <td>47.55072</td> </tr> </table>	score_route	100	score_penalty	0.475507	score_composed	47.55072
score_route	100						
score_penalty	0.475507						
score_composed	47.55072						



Figura 12. Tabella punteggi e areogramma infrazioni scenario 2

5.3.4 Scenario 3

Nello scenario 3 l'agente ha completato il percorso con un totale di 4 infrazioni:

- **Collisioni con veicoli:** l'agente è entrato in collisione con un veicolo.
- **Infrazioni sulla velocità minima:** l'agente ha mantenuto una velocità media rispettivamente del 91.93% e del 95.34% rispetto al traffico circostante in due occasioni.

index	3						
route_id	RouteScenario_3_rep0						
status	Completed						
num_infractions	4						
infractions	[+] infractions { }						
scores	[+] scores { }						
	<table border="1"> <tr><td>score_route</td><td>100</td></tr> <tr><td>score_penalty</td><td>0.569219</td></tr> <tr><td>score_composed</td><td>56.921857</td></tr> </table>	score_route	100	score_penalty	0.569219	score_composed	56.921857
score_route	100						
score_penalty	0.569219						
score_composed	56.921857						



Figura 13. Tabella punteggi e areogramma infrazioni scenario 3

5.3.5 Scenario 4

Nello scenario 4 l'agente ha completato il percorso con un totale di 9 infrazioni:

- **Collisioni con pedoni:** l'agente è entrato in collisione due volte con lo stesso pedone.
- **Collisioni con veicoli:** l'agente è entrato in collisione con due veicoli diversi.
- **Infrazioni al segnale di stop:** l'agente ha ignorato tre segnali di stop.
- **Infrazioni sulla velocità minima:** l'agente ha mantenuto una velocità media del 97.5% rispetto al traffico circostante in un'occasione.
- **Time-out dello scenario:** l'agente ha esaurito il tempo per completare lo scenario.

index	4						
route_id	RouteScenario_4_rep0						
status	Completed						
num_infractions	9						
infractions	[+] infractions { }						
scores	[+] scores { }						
	<table border="1"> <tr><td>score_route</td><td>100</td></tr> <tr><td>score_penalty</td><td>0.032014</td></tr> <tr><td>score_composed</td><td>3.201408</td></tr> </table>	score_route	100	score_penalty	0.032014	score_composed	3.201408
score_route	100						
score_penalty	0.032014						
score_composed	3.201408						



Figura 14. Tabella punteggi e areogramma infrazioni scenario 4

Inoltre, durante l'analisi del codice della baseline sono state individuate molteplici criticità che si ritiene impediscono di raggiungere delle prestazioni soddisfacenti:

- L'agente implementato da *BasicAgent* non rispetta i segnali di stop, comportando quindi la relativa penalità.
- All'interno degli incroci i waypoints non sono sufficientemente affidabili.

- In *BehaviourAgent* il metodo *traffic_light_manager* non gestisce il comportamento dell'ego-veicolo in presenza di semaforo rosso. Di conseguenza, non vi è una vera e propria gestione dei semafori nella baseline.
- Il metodo *emergency_stop* di *BehaviorAgent* non apporta alcuna modifica alla sterzata del veicolo. Ciò significa che il veicolo continuerà a seguire la stessa traiettoria durante l'arresto di emergenza. Tuttavia, la modifica può essere importante per evitare di uscire dalla corsia durante l'arresto, in particolare se il veicolo è in curva.

6 Soluzione proposta

Questo capitolo descrive in modo dettagliato la soluzione fornita, ovvero l'algoritmo di navigazione progettato e implementato. Sono riportati un'analisi dei requisiti, con una loro differenziazione in termini di difficoltà attesa e priorità stabilita, la taratura dei controllori longitudinale e laterale, un'accurata descrizione dell'architettura software e dei singoli moduli che la costituiscono, in termini di modifiche e aggiunte apportate alla baseline a disposizione.

6.1 Analisi dei requisiti: difficoltà e priorità

Prima di procedere con la progettazione dei singoli moduli dei planner, abbiamo effettuato l'analisi dei requisiti definendo le principali problematiche da risolvere in ordine di priorità. Abbiamo assegnato le priorità considerando due aspetti principali: il primo è la gravità dell'infrazione assegnata al problema, che incide sul punteggio finale dei singoli percorsi e su quello globale, mentre il secondo è la frequenza con cui tale problema si verifica.

Infatti, non tutte le infrazioni, in CARLA, hanno lo stesso peso; errori come le collisioni con pedoni o con altri veicoli, che in scenari reali porterebbero a gravi conseguenze, comportano una consistente perdita di punti nella valutazione, mentre aspetti secondari della guida, come la gestione della velocità, non sono così penalizzanti. In molti casi, in realtà, gli errori di gestione sono più impattanti in maniera indiretta che per la riduzione del punteggio che comportano.

Ad esempio, la mancata gestione dei sorpassi di oggetti statici dello scenario potrebbe portare a delle collisioni, ma soprattutto bloccare l'ego veicolo per un periodo di tempo tale da arrestare la simulazione e non permettere il completamento del percorso. Questo sarebbe di impedimento per valutare i progressi fatti sulla gestione delle altre problematiche oltre il punto del percorso in cui ci si è bloccati; per tale motivo, le criticità di questo genere vanno necessariamente gestite prima delle altre. A tale categoria appartengono numerose casistiche da gestire.

In primo luogo, vi è sicuramente la necessità di tarare i controllori longitudinale e laterale (rispettivamente PID e Stanley). Infatti, essere in grado di seguire la traiettoria in maniera ottimale è essenziale per il completamento dei percorsi e per poter procedere alla gestione di tutte le altre problematiche. Di conseguenza, il primo requisito da soddisfare di cui ci si è occupati è proprio la gestione dei parametri dei controllori.

Messa da parte questa criticità, le altre due problematiche che, se non gestite correttamente, in vari scenari di guida non permettono il completamento del percorso, sono la gestione dei sorpassi e dei restringimenti di corsia. In entrambe le situazioni la mancata gestione delle stesse comporta dei time-out di simulazione; per i sorpassi di oggetti statici, l'ego veicolo può rimanere bloccato dopo aver colliso con elementi dello scenario, quali segnali, ostacoli o veicoli fermi/parcheggiati. Nel caso del restringimento di corsia, invece, procedere con il veicolo al centro della propria corsia, come in situazioni di traffico ordinario, comporta un'elevata probabilità di scontrarsi con i veicoli della corsia opposta, che a causa del restringimento si avvicinano alla linea di mezzeria e in alcuni casi invadono leggermente la corsia dell'ego veicolo.

Tra le altre problematiche che possono causare time-out di simulazione appare la gestione dei ciclisti. Questi sono presenti nella maggior parte degli scenari e in alcune situazioni tendono a fermarsi e a non proseguire; in tal caso, potrebbero bloccare l'ego veicolo in maniera indefinita a meno che questo non sia in grado di superarli. Il sorpasso dei ciclisti, se non gestito con le giuste tempistiche, comporta la possibilità di addentrarsi in situazioni rischiose e il pericolo di causare nuove collisioni, per cui il tempo richiesto per la risoluzione di tale requisito non è trascurabile; tuttavia, dato il significativo impatto sul punteggio che hanno le collisioni con i ciclisti, la priorità data alla problematica rimane elevata.

Un'altra infrazione molto costosa in termini di penalità, in CARLA, è la collisione con i pedoni (giustificata, tra l'altro, dalla gravità delle conseguenze che comporterebbe in scenari reali). Inoltre, negli scenari analizzati sono presenti attraversamenti di pedoni abbastanza frequenti, e in alcuni casi in maniera improvvisa e inaspettata. Evitare questo tipo di collisioni è quindi fondamentale per raggiungere un buon punteggio finale.

Riguardo alle rimanenti cause di collisione, ci sono incroci e semafori. Se non gestiti correttamente, in entrambi i casi si rischia di collidere con gli altri veicoli che si stanno immettendo nell'incrocio da strade diverse o che sono già presenti quando l'ego veicolo si inserisce. È tuttavia da notare che il numero di collisioni derivanti dalla mancata gestione di queste casistiche è fortuitamente inferiore a quello delle fonti di collisione precedentemente trattate. Per tale motivo, sebbene si tratti comunque di requisiti fondamentali, la priorità temporale assegnata è stata inferiore rispetto a quella data alle altre criticità.

Tra i requisiti di minore importanza che si sono considerati ci sono la gestione degli stop e delle velocità minime. Infatti, non fermarsi per tre secondi davanti a un segnale di stop comporta una leggera penalità, e lo stesso vale nel caso in cui l'ego veicolo proceda a una velocità inferiore di quella dei veicoli ad esso circostanti nel traffico. Il primo requisito è stato soddisfatto, mentre il secondo è stato tralasciato per mancanza di tempo.

Si noti che vanno considerati, tra i requisiti da soddisfare, anche i bug di CARLA che possono portare a comportamenti indesiderati ed eventualmente a infrazioni, e che sono trattati nel capitolo 9. Chiaramente di questi ci si è occupati soltanto dopo aver risolto tutte le altre problematiche principali.

6.2 Taratura dei controllori

Su un veicolo a guida autonoma, come precedentemente accennato, sono necessarie due tipologie di controllo:

- **controllo longitudinale**, che consiste nel controllo della velocità del veicolo,
- **controllo laterale**, che consiste nel controllo della sterzata.

La parte di controllo di un veicolo a guida autonoma ha l'obiettivo di **inseguire una traiettoria** in termini di path e di velocità target. È fondamentale evidenziare che i controllori si limitano ad inseguire una traiettoria, quindi non la modificano o pianificano. Infatti, il modulo che si occupa della pianificazione della traiettoria è il *Local Planner*, il quale si avvale delle preziose informazioni derivanti da *Mission Planner* e *Behavior Planner*. Naturalmente, la scelta dei parametri dei controllori, quindi la loro taratura, ha un impatto rilevante su come la traiettoria venga inseguita e, pertanto, sull'effettiva traiettoria percorsa dal veicolo.

Nel progetto realizzato, il controllo longitudinale è stato implementato con un controllore PID (Proporzionale-Integrale-Derivativo), mentre il controllo laterale è stato implementato con un controllore di Stanley. Si osservi che esistono senz'altro altre possibilità, tuttavia le tipologie di controllori scelte sono quelle ritenute maggiormente stabili allo stato dell'arte per i task di nostro interesse. Di seguito è descritto il procedimento adottato per la taratura dei controllori.

6.2.1 Taratura del controllore longitudinale PID

La taratura del controllore PID è stata fatta adoperando la **procedura euristica di Ziegler-Nichols**, ideata negli anni '40 da John G. Ziegler e Nathaniel B. Nichols. Si è scelto di utilizzare tale metodo poiché è particolarmente veloce, non richiede un modello matematico del processo (che nel nostro caso sarebbe stato praticamente impossibile da ottenere) e assicura in genere ottime prestazioni.

La procedura prevede i seguenti passaggi:

1. Il processo viene fatto controllare da un controllore esclusivamente proporzionale (il guadagno integrale K_i e il guadagno derivativo K_d vengono settati a 0);
2. Il guadagno proporzionale K_p viene gradualmente aumentato, fino a raggiungere un valore critico K_u , in corrispondenza del quale la variabile controllata presenta oscillazioni stabili e sostenute, che non spariscono dopo un transitorio;
3. Si registra il periodo critico T_u delle oscillazioni stabili e sostenute;
4. Si determinano i guadagni integrale K_i , derivativo K_d , e proporzionale K_p per il tipo di controllore desiderato P, PI o PID:

Control Type	K_p	K_i	K_d
P	$0.5 K_u$		
PI	$0.45 K_u$	$1.54 K_u / T_u$	
PID classico	$0.6 K_u$	$1.2 K_u / T_u$	$0.075 K_u T_u$

Nel nostro caso specifico, abbiamo scelto di adoperare un PID, e non un P o un PI, al fine di sfruttare sia il contributo proporzionale all'errore, che consente di portare in conto la differenza tra la velocità attuale del veicolo e quella desiderata, sia il contributo proporzionale all'integrale dell'errore, che consente di portare in conto la storia di quest'ultimo, sia il contributo proporzionale alla derivata dell'errore, che consente di compensare rapidamente l'errore medesimo.

La procedura è stata applicata, fissando a 30 km/h la velocità desiderata, nell'ambito di un percorso semplice ambientato nella Town 12 di Carla, caratterizzato da due lunghi rettilinei e due curve abbastanza ampie. I parametri del controllore di Stanley, adoperato per garantire un controllo efficace della sterzata nel contesto dell'esperimento, sono stati settati a:

$$K_v = 0.5; K_s = 1; \Delta t = 0.05 \text{ s}$$

I parametri scelti, selezionati in seguito ad alcune prove, hanno consentito di percorrere senza alcun problema le due curve del percorso.

Il periodo di campionamento del PID è stato settato a:

$$\Delta t = 0.05 \text{ s}$$

poiché il *fixed time-step* impostato sul server di CARLA era proprio uguale a 0.05 s. Di fatti, scegliere un tempo di campionamento più basso non avrebbe avuto senso, in quanto la frequenza di aggiornamento del PID sarebbe stata più veloce di quella della simulazione; scegliere un tempo di campionamento più alto avrebbe potuto rendere il controllo meno efficace, in quanto avrebbe comportato l'applicazione di un minor numero di azioni di controllo a parità di intervallo temporale considerato. Naturalmente, considerazioni analoghe giustificano l'utilizzo di 0,05 s come tempo di campionamento del controllore di Stanley adottato nel corso della procedura.

La procedura di Ziegler-Nichols ha consentito di individuare, come valore critico e come periodi delle oscillazioni stabili e sostenute:

$$K_u = 2.5; K_s = 1$$

Applicando le formule per il PID sopra riportate, sono stati ottenuti i seguenti valori per i guadagni proporzionale, integrale, derivativo:

$$K_p = 1.5; K_i = 4.2857; K_d = 0.13125$$

Al fine di ridurre la sovraelongazione e le oscillazioni residue, riducendo al contempo il tempo di assestamento, si è scelto di agire su K_d , aumentandolo fino al valore di 0.5. Pertanto, il PID finale è caratterizzato dai seguenti parametri:

$$K_p = 1.5; K_i = 4.2857; K_d = 0.5$$

Si riporta di seguito un grafico rappresentante l'andamento della velocità, lungo il percorso considerato, garantito dal PID finale.

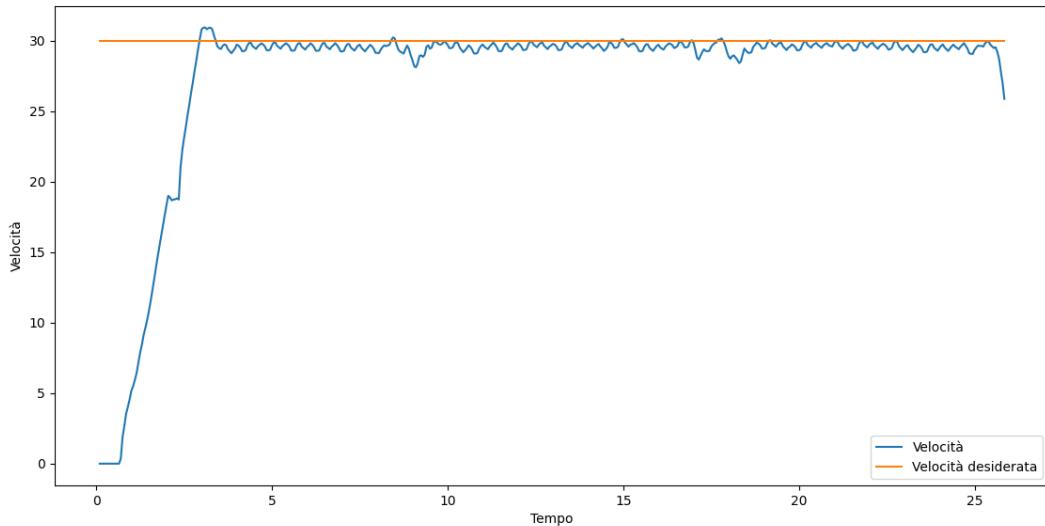


Figura 15. Andamento della velocità controllata dal PID

Il comportamento ottenuto è molto buono. I tre cali di velocità più evidenti, in corrispondenza di 10 s, 20 s e 26 s circa, sono da attribuire alle due curve affrontate e alla conclusione del percorso. Nonostante sembri che nei tratti rettilinei vi siano delle oscillazioni, queste sono comunque molto contenute: la velocità non scende mai sotto i 29 km/h. Le lievi oscillazioni sono dovute anche al fatto che il controllo viene applicato

ad un rate estremamente elevato (ogni 0.05 s). La sovraelongazione è praticamente impercettibile, il tempo di salita ed il tempo di assestamento sono molto contenuti, pertanto accettabili. Si evidensi che una risposta al gradino ideale sarebbe caratterizzata proprio da errore a regime nullo o molto piccolo, tempo di salita e di assestamento piccoli, e sovraelongazione nulla oppure molto piccola.

Successivamente, il PID ottenuto è stato testato anche su tutti gli scenari descritti nel capitolo 3. I risultati sono stati ottimali in qualsiasi contesto: nei rettilinei, nelle curve, con velocità target superiore e inferiore a 30 km/h. Il veicolo si è sempre dimostrato estremamente celere nel raggiungere la velocità target e preciso nel mantenerla.

Si osservi che, invece di utilizzare il metodo Ziegler-Nichols, si sarebbe potuta effettuare anche una taratura "manuale". In tal caso, alla luce dei diversi effetti, talvolta contrastanti, dei guadagni proporzionale, integrale, integrativo sui parametri di qualità tipici di una riposta a ciclo chiuso con riferimento a gradino (tempo di salita, sovraelongazione, tempo di assestamento ed errore a regime), si sarebbe dovuto procedere nel modo seguente:

1. Taratura di K_p atta all'ottenimento di un tempo di salita basso;
2. Taratura di K_i al fine di eliminare o ridurre il più possibile l'errore a regime;
3. Taratura di K_d al fine di correggere il transitorio, ovvero con l'obiettivo di minimizzare tempo di assestamento e sovraelongazione.

Naturalmente, nella maggior parte dei casi una taratura *from scratch* non è banale. Spesso, invece di procedere per step a comportamenti stagni, si può ottenere un buon risultato andando a modificare insieme, facendo vari tentativi, K_p e K_i , trovando un buon compromesso in termini di tempo di salita e sovraelongazione, per poi andare a correggere il transitorio individuando un K_d adatto.

Date la semplicità, la velocità e l'affidabilità della metodologia di Ziegler-Nichols, si è deciso di adoperare quest'ultima invece di una taratura manuale.

È rilevante evidenziare, infine, che date le ottime risposte fornite dal controllore PID ottenuto in tutte le condizioni di guida (in rettilineo, in curva, al variare della velocità desiderata) e in tutti gli scenari testati, non è stato necessario introdurre dei guadagni dinamici.

6.2.2 Taratura del controllore laterale Stanley

La taratura del controllore Stanley è stata fatta testando diverse combinazioni di k_s e k_v , fino ad individuare una configurazione che si comportasse in modo ottimo in tutti gli scenari e per tutte le velocità target di interesse.

La prima configurazione testata è stata:

$$k_v = 1 ; k_s = 0.5$$

Essa garantiva un buon controllo laterale solo per velocità target inferiori o uguali a 30 km/h circa e su strade caratterizzate da rettilinei e curve larghe. Infatti, nell'affrontare le curve più strette, soprattutto in corrispondenza di svolte a destra o a sinistra agli incroci, l'agente rischiava di colpire i veicoli in transito sulle corsie adiacenti. Inoltre, quando l'agente cercava di superare ostacoli statici collocati sulla sua corsia, invadendo temporaneamente una corsia adiacente per poi rientrare sulla propria corsia originaria, il

controllore così tarato non riusciva ad eliminare l'errore di crosstrack abbastanza velocemente, determinando l'uscita di strada dell'agente.

Per via di tali considerazioni, questa configurazione è stata scartata.

Per risolvere i problemi riscontrati si è pensato di aumentare sensibilmente k_v , e leggermente k_s , così da effettuare una correzione più repentina dell'errore di crosstrack. In particolare, la configurazione provata è stata:

$$k_v = 5 ; k_s = 1$$

Tale configurazione risolveva il problema nel caso degli incroci e, in generale, delle curve strette, per velocità inferiori o uguali a 30 km/h. Tuttavia, non si trattava ancora della combinazione ideale dei guadagni, in quanto per velocità superiori a 30 km/h (es: 50 km/h) le curve larghe dei primi due scenari venivano affrontate male: il controllore faceva fatica a tenere basso il crosstrack error e, di conseguenza, ne risultava una pericolosa invasione della corsia adiacente.

Alla luce di ciò, si è deciso di testare un'ulteriore configurazione, caratterizzata da un k_v analogo alla prima e da un k_s più basso, così da rendere la sterzata più agevole anche a velocità più elevate. In particolare, la configurazione testata è stata:

$$k_v = 1 ; k_s = 0.2$$

Tale combinazione di coefficienti riduceva effettivamente la criticità delle curve larghe a velocità più elevata di 30 km/h, tuttavia, reintroduceva la criticità riguardante i sorpassi che aveva già affetto la primissima configurazione.

Sulla base di tali considerazioni si è deciso di aumentare in modo sensibile k_v , e di ridurre in modo altrettanto sensibile k_s , al fine di ottenere una configurazione che fosse in grado di abbassare l'errore di crosstrack più velocemente. In particolare, l'ultima configurazione testata è stata la seguente:

$$k_v = 3 ; k_s = 0.05$$

Questa rappresenta anche la configurazione finale del controllore laterale. Essa, di fatto, ha consentito di risolvere completamente, almeno fino ad una velocità target massima di 50 km/h, il problema delle curve larghe dei primi due scenari; tuttavia, ha comportato, seppur in maniera più lieve, la reintroduzione della criticità delle svolte in corrispondenza degli incroci, determinando nuovamente il rischio di collidere con veicoli in transito sulle corsie adiacenti. Per porre rimedio a tale problematica, anche alla luce della natura rischiosa delle intersezioni, si è deciso di impostare 20 km/h come velocità massima di attraversamento di un incrocio. In tal modo, il controllore così tarato riesce perfettamente a correggere l'errore di crosstrack anche per curve molto strette. La configurazione è stata ampiamente testata in tutti gli scenari e su ciascuno di essi ha performato egregiamente, pertanto, non è stato necessario introdurre coefficienti dinamici neanche per il controllore di Stanley.

Infine, è opportuno evidenziare che il tempo di campionamento scelto per il controllore è:

$$\Delta t = 0.05s$$

Le ragioni di tale scelta sono analoghe a quelle già presentate nel paragrafo 6.2.1. Inoltre, non ci sarebbe stato alcun motivo per cui impostare tempi di campionamento differenti per il controllore longitudinale e per il controllore laterale.

6.3 Architettura software (finale)

In seguito allo sviluppo delle soluzioni ai problemi discussi in precedenza, si è giunti ad un'architettura del software risultante che risulta essere sostanzialmente la stessa della baseline, con la rilevante differenza di implementare una quantità di funzionalità notevolmente maggiore. Da questo punto di vista la baseline fornita si è rivelata essere una buona base, alquanto solida, dalla quale partire per aggiungere funzionalità via via sempre più complesse.

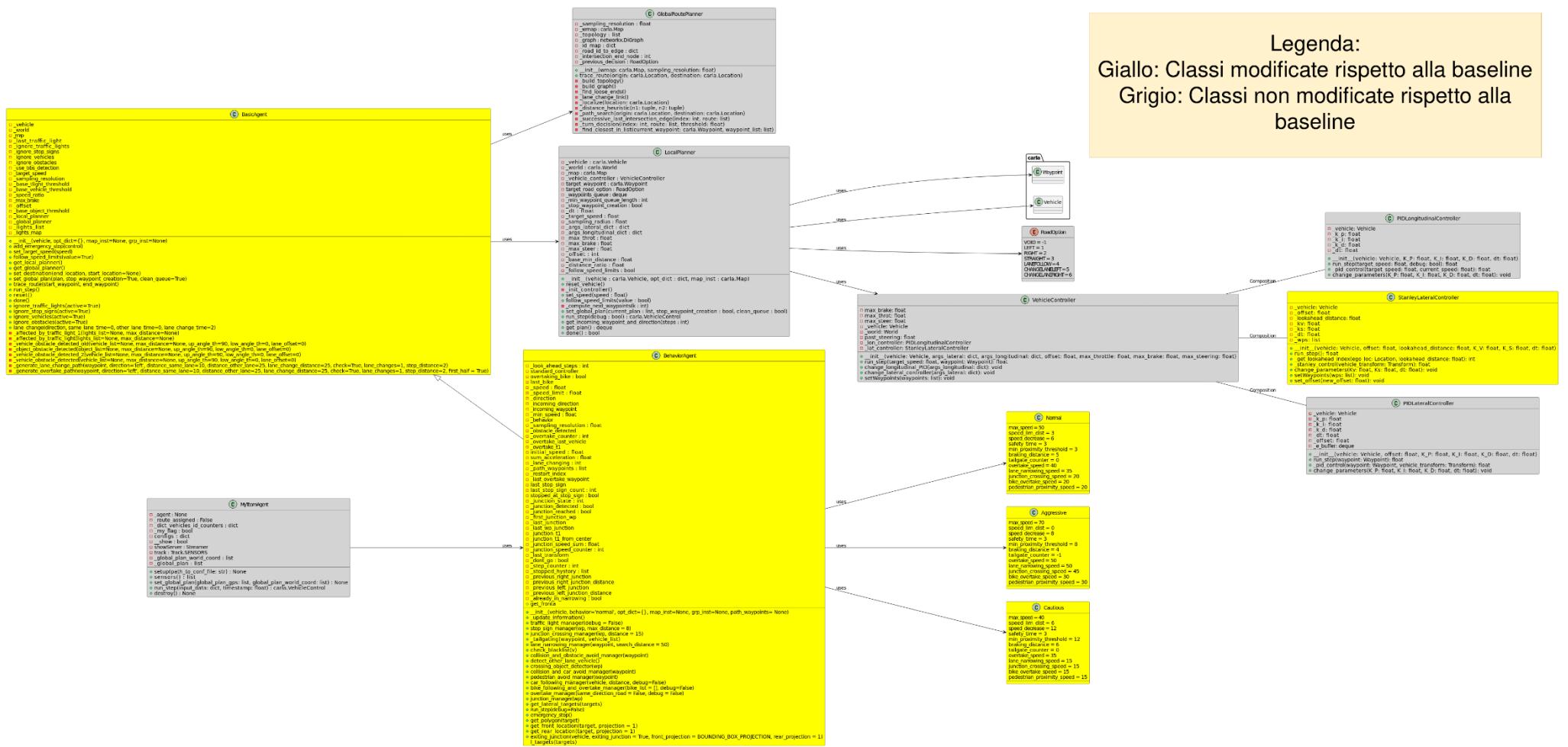


Figura 16. Diagramma UML dell'architettura software finale

6.3.1 Basic Agent

Una delle modifiche principali che sono state apportate al *BasicAgent* riguarda il modulo percettivo degli ostacoli, veicoli e ciclisti, rendendolo più flessibile e robusto a diverse situazioni, che coinvolgano un numero plurimo di pericoli. In particolare, le funzioni implementate sono:

- **vehicle_obstacle_detected_2:** metodo realizzato a partire dall' originale *vehicle_obstacle_detected*, già presente nella prima versione della baseline fornita, che è stato modificato per permettere l'acquisizione delle informazioni di più veicoli. Avere la consapevolezza di essere in rischio di collisione con più veicoli è necessario per la gestione dei sorpassi e, nel nostro caso specifico, per i sorpassi di biciclette, di cui si discuterà nel dettaglio successivamente. Il metodo presenta la medesima implementazione dell'originale *vehicle_obstacle_detected*, con la differenza che se un veicolo si trova ad essere in rischio di collisione con l'agente, viene aggiunto ad una lista di veicoli, la quale viene restituita per intero, invece di restituire solo il veicolo più vicino.
- **object_obstacle_detected:** questo metodo è stata aggiunto al *BasicAgent* a partire da una versione iniziale di *vehicle_obstacle_detected*, modificata per garantire la detection di un insieme di ostacoli, piuttosto che di un singolo ostacolo. L'insieme degli ostacoli che intralciano il cammino dell'agente è necessario al modulo di gestione dei sorpassi, per calcolare lo spazio da percorrere sulla corsia a sinistra prima di rientrare, dopo aver superato l'ostacolo, nella corsia a destra. La detection viene fatta principalmente considerando il bounding box degli ostacoli presenti sul cammino. Il motivo di questa scelta è che alcuni dati presenti in CARLA non sono sempre corrispondenti a ciò che accade a schermo (di questi problemi si discuterà in maniera più approfondita in seguito), per cui si è resa necessaria una detection più fine e più a basso livello. La detection tramite bounding box funziona allo stesso modo in cui viene implementata nella baseline: si proietta il bounding box dell'agente fino ad una distanza *max_distance* passata come parametro, se tale bounding box interseca il bounding box di un oggetto nei paraggi, allora questo viene considerato un ostacolo. Tutto ciò è valido se l'oggetto che interseca il bounding box non è un veicolo. Un veicolo, infatti, è considerato un ostacolo solo quando la sua velocità è nulla, oppure la sua *lane_id* è pari alla *lane_id* sulla quale si trova l'agente +1. Questi casi specifici, infatti, vanno a modellare il problema dei veicoli fermi a bordo strada. Questi veicoli vengono considerati ostacoli a tutti gli effetti e vengono gestiti in quanto tali, tramite la meccanica del sorpasso, che sarà descritta successivamente.
- **generate_overtake_path:** metodo utilizzato per generare la prima o la seconda metà di un path per il sorpasso di un ostacolo statico (oggetto statico o veicolo fermo a bordo strada), nel caso in cui la corsia lungo cui si voglia effettuare il sorpasso abbia un senso di marcia opposto a quello che si sta percorrendo. Esso prende in ingresso il waypoint dal quale inizia il path, la direzione verso cui fare il sorpasso, la distanza da percorrere sulla corsia corrente, durante il cambio corsia e dopo aver cambiato corsia. La logica utilizzata per generare il path è la stessa presente nel metodo *generate_lane_change_path* del *BasicAgent* della baseline, con la differenza che nel caso in cui si stia generando la seconda metà del path, i waypoint da prendere sulla lane del cambio corsia sono

presi nella direzione opposta a quella di percorrenza, in quanto il senso di marcia sulla corsia di sorpasso è inverso rispetto a quello della corsia adiacente.

6.3.2 Behavior Agent

La classe `BehaviorAgent` contiene la maggior parte dei cambiamenti implementati durante il progetto e racchiude la logica dell'algoritmo di navigazione. Essa è composta dai seguenti metodi:

- **`collision_and_obstacle_avoid_manager`**: manager ad alto livello che si occupa di gestire le chiamate alla funzione `object_obstacle_detected`, sia per gli oggetti statici che per i veicoli. A seconda che si stia effettuando o meno un cambio di corsia passa alla funzione dei parametri diversi, restituisce True o False in caso vengano rilevati o meno degli ostacoli e, in caso affermativo, restituisce la lista degli ostacoli e le loro distanze dall'agente.
- **`collision_and_car_avoid_manager`**: manager ad alto livello che si occupa di gestire le chiamate alla funzione `vehicle_obstacle_detected_2` che, a seconda se si stia cambiando corsia o meno, riceve parametri diversi.
- **`crossing_object_detector`**: metodo che si occupa di gestire gli attraversamenti di corsia da parte di veicoli (principalmente biciclette), che potrebbero tagliare la strada all'agente. Il metodo filtra tutti i veicoli nel raggio di una decina di metri e calcola l'angolo compreso tra il versore direzionale frontale dell'ego-veicolo e il versore direzionale frontale di ogni target filtrato. Se questo angolo è compreso tra 80 e 100 gradi, allora significa che il veicolo è orientato in maniera perpendicolare all'agente, per cui potrebbe stare per attraversare. In questa condizione, il manager restituisce True, altrimenti restituisce False.
- **`lane_narrowing_manager`**: metodo responsabile di identificare i restringimenti di corsia presenti lungo la strada. Questi restringimenti vengono segnalati mediante dei coni posti sulla corsia opposta, che è quella che subisce il restringimento stesso. La funzione controlla se nei dintorni dell'agente sono presenti dei coni stradali e, quando il numero di coni supera un certo valore, viene restituito True, altrimenti False.
- **`bike_following_and_overtake_manager`**: manager utilizzato per la gestione dei sorpassi delle biciclette incontrate lungo il percorso. Lo scopo di questo metodo è quello di settare un flag, attributo di `BehaviourAgent`, chiamato `overtaking_bike`, il quale ha valore True se il veicolo sta eseguendo il sorpasso di una o più biciclette in successione, altrimenti ha valore False. Prende come parametro una lista contenente gli attori che rappresentano i ciclisti da dover sorpassare. Se il flag non è settato nel momento in cui viene chiamato il manager, allora l'agente rallenta, viene settato il flag, si memorizza l'ultima bicicletta presente nella lista in una variabile di istanza e si setta un offset laterale al controllore di Stanley. Alternativamente, se il flag è già settato nel momento in cui viene chiamato il manager, si controlla la posizione dell'ultima bicicletta memorizzata precedentemente. Se tale bicicletta si trova alle spalle dell'agente, allora il sorpasso può considerarsi concluso, per cui vengono resettati sia il flag sia l'ultima bicicletta memorizzata e viene azzerato l'offset laterale. Per capire se la bicicletta si trova dietro il veicolo, si calcola il vettore

direzione *ego-target*, che ha come origine il centro dell'*ego-veicolo* e, come destinazione il centro dell'ultima bicicletta memorizzata. Si calcola dunque l'angolo compreso tra il versore frontale dell'*ego veicolo* e il vettore *ego-target*. Se questo angolo è maggiore di 90° (più un certo margine), allora si considera la bicicletta come superata e, dunque, il veicolo può tornare a seguire la sua traiettoria originaria.

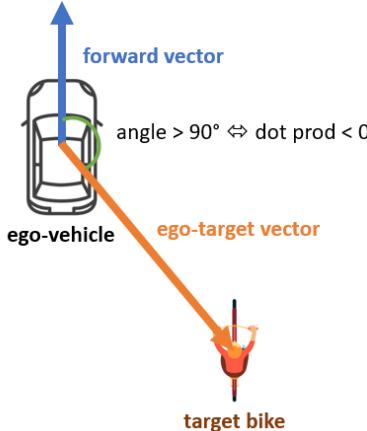


Figura 17. Schema vettori per rilevare una bici dietro l'*ego-veicolo*

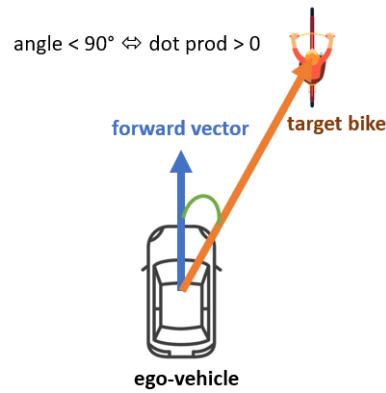


Figura 18. Schema vettori per rilevare una bici davanti l'*ego-veicolo*

- **detect_other_lane_vehicle:** metodo utilizzato per identificare i veicoli che si muovono verso l'*ego-veicolo*, nella corsia accanto con senso di marcia opposto alla sua e, che non hanno ancora superato la posizione dell'*agente*. Esso viene adottato principalmente durante il sorpasso di oggetti statici, del quale si discuterà nel dettaglio in seguito. Il metodo, inizialmente, filtra tutti i veicoli nel raggio di una certa distanza e che si trovano su una corsia diversa rispetto quella in cui si trova l'*ego-veicolo*. Per ognuno di questi veicoli *target* viene calcolato il vettore *ego-target* che congiunge il centro dell'*ego-veicolo* al centro del *target*. Si calcola il prodotto scalare tra il *forward vector* dell'*ego-veicolo* ed il vettore che congiunge l'*ego* al *target*. Se il prodotto scalare è maggiore di 0, allora significa che il *target* si trova davanti l'*agente* e viene, dunque, aggiunto ad una lista, la quale viene restituita, ordinata per distanza, una volta controllati tutti i veicoli in esame.

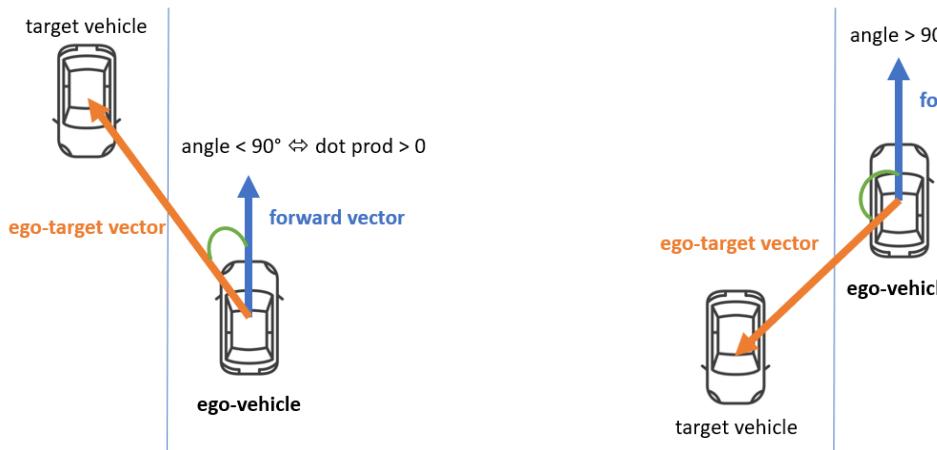


Figura 19. Schema vettori per rilevare un'auto davanti (fig. a sinistra) o dietro (fig. a destra) l'*agente* nella corsia accanto.

- **overtake_manager**: metodo utilizzato per la gestione del sorpasso degli oggetti statici. Questo è uno dei metodi più complessi che sono stati implementati. È, infatti, realizzato con una struttura a stati e necessita di essere chiamato più volte per implementare la corretta esecuzione di un sorpasso. Di seguito è riportato uno schema che ne illustra la logica:

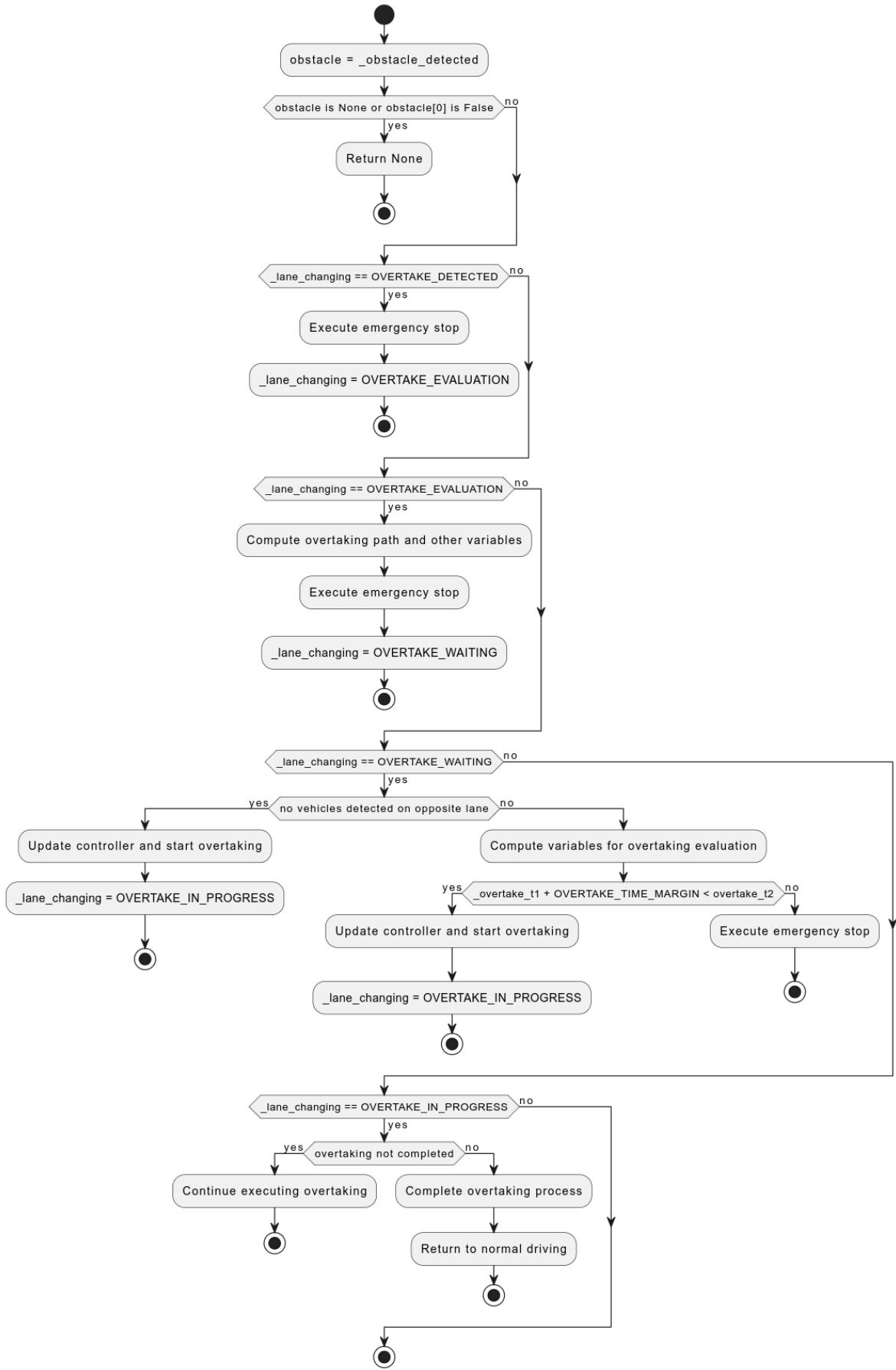


Figura 20. Schema di flusso della gestione dei sorpassi in overtaking_manager

L'overtake manager si compone di 5 stati:

1. OVERTAKE_NOT_DETECTED: stato nel quale non è stato rilevato alcun ostacolo statico sul cammino dell'agente.
2. OVERTAKE_DETECTED: stato nel quale è stato rilevato un ostacolo. Esso viene settato in seguito alla rilevazione di un ostacolo mediante la funzione *collision_and_obstacle_avoid_manager*. Quando il sistema si trova in questo stato inizia a frenare e vi permane finché il veicolo non è completamente fermo.
3. OVERTAKE_EVALUATION: una volta che il veicolo è fermo, inizia la fase di valutazione del percorso da effettuare per eseguire il sorpasso. Viene generato il path considerando come lunghezza quella complessiva degli ostacoli (tale scelta progettuale comporta la necessità di avere le informazioni di tutti gli ostacoli sulla strada), sommata ad una quantità costante che rappresenta del margine (per avere maggiore robustezza). I waypoint del path vengono generati usando il metodo *generate_overtake_path*, chiamato due volte: una per generare la prima metà del path, un'altra per la seconda metà, necessarie al sorpasso. Successivamente si ricongiunge il path originario al path calcolato per il sorpasso, prendendo il waypoint del path originario più vicino all'ultimo waypoint del path del sorpasso. L'indice di questo waypoint viene memorizzato per riprendere il percorso da quel punto una volta effettuato il sorpasso. Infine, si calcola il tempo necessario ad effettuare il sorpasso, calcolato come: $t_1 = \frac{d}{v_{overtake}}$, dove
 - d è la lunghezza del path di overtaking,
 - $v_{overtake}$ è la velocità con la quale l'agente effettua il sorpasso.

A tale tempo si aggiunge una costante che rappresenta un margine; tale scelta è dovuta al fatto che il moto non è interamente rettilineo e che la velocità non è sempre uniforme. Il tempo calcolato viene memorizzato in una variabile di istanza e lo stato cambia in OVERTAKE_WAITING.

4. OVERTAKE_WAITING: in questo stato il veicolo è fermo in attesa che si verifichino le condizioni necessarie per effettuare il sorpasso in maniera sicura. Vengono controllati i veicoli che arrivano dalla corsia accanto, con senso di marcia opposto a quello dell'agente: se non vengono rilevati veicoli, si assume che la strada sia libera e che il sorpasso possa venir effettuato in perfetta sicurezza. Se, invece, viene rilevato un veicolo, viene inizializzato un counter. Finché il counter, che viene incrementato ad ogni ciclo di esecuzione, non raggiunge un determinato valore, per il veicolo che sta sopraggiungendo si calcola la somma di tutti i valori di accelerazione che assume. Quando il counter raggiunge la soglia, si calcola il tempo che il veicolo impiegherà ad arrivare in prossimità dell'ultimo waypoint del path del sorpasso, usando la legge oraria del moto rettilineo uniformemente accelerato (si è scelto di adottare tale modello per semplicità, considerando la media delle accelerazioni del veicolo target per poter approssimare il moto ad uniformemente accelerato):

$$t_2 = -\frac{v_0 + \sqrt{v_0^2 + 2as}}{a}$$

dove:

- v_0 rappresenta l'ultimo valore di velocità assunto dal veicolo target,
- a è la media delle accelerazioni assunte dal target,
- s è la distanza del target dall'ultimo waypoint del path del sorpasso.

A questo punto, se si verifica la condizione $t1 < t2$, allora si effettua il sorpasso, passando allo stato OVERTAKE_IN_PROGRESS. In caso contrario, il veicolo rimane fermo ed effettua lo stesso procedimento per ciascuno dei veicoli che giungono dalla corsia opposta.

5. **OVERTAKE_IN_PROGRESS:** in questo stato l'agente segue i waypoints sul suo percorso e quando i waypoints del path del sorpasso terminano, si utilizza l'indice memorizzato in precedenza per riprendere il path originario. Si resettano, dunque, tutte le variabili utilizzate per il sorpasso e si torna nello stato OVERTAKE_NOT_DETECTED.
- **junction_crossing_manager:** manager che si occupa dell'identificazione degli incroci e che è in grado di determinare se l'incrocio è stato superato o meno, restituendo True quando l'incrocio è stato identificato ma ancora non superato e False in tutti gli altri casi. Si avvale di due flag, entrambi variabili di istanza di *BehaviourAgent*: *junction_detected* e *junction_reached*, che rispettivamente indicano quando l'agente ha rilevato un incrocio e quando lo ha raggiunto. Non appena l'incrocio viene superato, entrambi i flag sono resettati a False.
 - **stop_sign_manager:** manager per la gestione dei segnali di stop. Il comportamento atteso di fronte ad un segnale di stop è quello di fermarsi per almeno 3 secondi. Il metodo fa uso di un flag ed un counter. Il flag, chiamato *stopped_at_stop_sign*, viene usato per indicare che il veicolo è fermo ad un segnale di stop. Il counter viene usato per tener conto del tempo trascorso davanti al segnale. Se il flag non è settato, allora il manager cerca ad una distanza ricevuta in ingresso dei segnali di stop. Se vengono trovati dei segnali di stop nei paraggi, per ognuno di essi si controlla che non sia l'ultimo segnale di stop al quale l'ego-veicolo si è fermato e, in tal caso, si calcola il prodotto scalare tra la direzione del waypoint del segnale e la direzione dell'agente. Se il prodotto scalare è minore di 0 (il segnale non è orientato verso la nostra direzione), allora si setta il flag e si restituisce True, altrimenti si restituisce False.
 - **junction_manager:** metodo incaricato di gestire tutta la logica degli incroci incontrati dall'agente. È un manager molto complesso che si occupa di gestire molteplici tipologie di incroci, a partire da quelli regolati tramite regole di precedenza, passando per quelli con segnali di stop, fino agli incroci gestiti da semafori. Anche questo metodo è organizzato in stati, in particolare è composto complessivamente da 7 stati. Indipendentemente dallo stato, ogni volta che viene lanciato, il junction manager si occupa di controllare, mediante la funzione *traffic_light_manager*, se nei paraggi ci sono dei semafori che debbano regolare il comportamento dell'ego-veicolo. In tal caso, si aspetta che il semaforo diventi verde, e quando ciò accade, lo stato dell'incrocio viene settato a JUNCTION_IN_PROGRESS, che, come si descriverà meglio in seguito, è lo stato finale nel quale si attraversa l'incrocio. In caso contrario, invece, si passa alla valutazione dello stato dell'incrocio, memorizzato nel flag *junction_state* che può essere:
 1. **JUNCTION_NOT_DETECTED:** tale stato indica che non è ancora stato rilevato alcun incrocio. In questo stato viene chiamata la funzione *junction_crossing_manager*, il cui valore di ritorno se è True, significa che è stato rilevato un incrocio. In tal caso, lo stato dell'incrocio cambia in JUNCTION_DETECTED, altrimenti il ciclo di controllo procede normalmente.

2. **JUNCTION_DETECTED**: in questo stato si rallenta, si raccoglie l'informazione del primo waypoint del junction e, quando si è raggiunta una certa distanza rispetto a tale waypoint, lo stato cambia in **JUNCTION_CHECK_STOP**.
3. **JUNCTION_CHECK_STOP**: in questo stato si controlla la presenza di stop all'incrocio, invocando la funzione *stop_sign_manager*. Nell'eventualità in cui l'incrocio sia regolato da stop, ci si ferma per 3 secondi. Se, invece, non vi è alcuno stop all'incrocio, l'agente si ferma, memorizza in una variabile d'istanza la sua *transform* attuale. In entrambi i casi si passa allo stato **JUNCTION_APPROACHING**.
4. **JUNCTION_APPROACHING**: tale stato serve ad approcciarsi agli incroci che presentano corsie di immissione molto ampie. In questo stato si calcola la distanza dell'agente dal centro dell'incrocio e si seguono i waypoints finché non si raggiunge una determinata distanza minima. A quel punto si passa allo stato **JUNCTION_WAITING**.
5. **JUNCTION_WAITING**: è lo stato in cui l'agente attende le condizioni di sicurezza per attraversare l'incrocio. Innanzitutto, si filtrano i veicoli che si trovano nelle immediate vicinanze dell'incrocio e, a partire dalla *transform* dell'ego-veicolo precedentemente memorizzata, questi si distinguono in veicoli che si trovano alla destra, alla sinistra o di fronte rispetto alla *transform* memorizzata, oppure in veicoli che stanno occupando l'incrocio. Per ogni veicolo target, si scartano quelli posizionati alle spalle della *transform* e si calcola l'angolo compreso tra il vettore destro della precedente *transform* dell'ego-veicolo ed il vettore che congiunge tale *transform* con il target:
 - Se tale angolo è compreso tra 0 e 85 gradi, allora il veicolo si trova a destra.
 - Se tale angolo è compreso tra 85 e 120 gradi, allora il veicolo si trova di fronte.
 - Altrimenti, il veicolo si trova a sinistra.

A tal punto si valuta la possibilità di attraversare l'incrocio. La valutazione viene fatta in maniera diversa a seconda che si debba svoltare a sinistra, svoltare a destra o andare dritti. Il caso più complesso è quello che prevede la svolta a sinistra. Viene definito un flag per ogni direzione da cui possono arrivare veicoli, che viene settato a True solo quando si ritiene che i veicoli provenienti dalla direzione corrispondente non possano intralciare il cammino dell'agente. In particolare, il percorso si considera sicuro quando per tutte le direzioni considerate è valida una delle due condizioni: o il primo veicolo proveniente da una certa direzione è fermo o non arrivano veicoli da quella direzione.

Una volta completata l'analisi dei veicoli che si trovano fuori dall'incrocio, si procede con l'analisi dei veicoli che si trovano all'interno dell'incrocio. A partire da tutti i veicoli che si trovano nei dintorni dell'incrocio (ovvero al suo interno più un certo margine), si distingue in quelli che si trovano a sinistra o a destra, in questo caso rispetto alla posizione corrente del veicolo.

Nel caso in cui si debba svoltare a sinistra, si tiene conto della distanza dal veicolo più vicino che si trova alla sinistra dell'ego-veicolo. Ad ogni ciclo di esecuzione si salva in una variabile di istanza rispettivamente, il veicolo più vicino a destra e quello più vicino a sinistra dell'ego-veicolo e si confronta l'informazione memorizzata in precedenza con quella disponibile nel *run_step* attuale. Se il veicolo che si trova a sinistra dell'ego-veicolo è lo stesso e la sua distanza dall'ego-veicolo è la stessa, allora significa che quel veicolo si è fermato per far passare l'agente.

Attraversare in questo momento l'incrocio sarebbe comunque troppo rischioso, in quanto si potrebbe collidere con un veicolo che si trova nell'incrocio e che proviene da destra. Per questo motivo, la condizione che fa uscire dallo stato di attesa tiene conto del fatto che il veicolo alla sinistra dell'ego-veicolo sia fermo e che il veicolo che precedentemente si trovava sulla destra dell'ego-veicolo si trovi successivamente sulla sua sinistra. Quando queste condizioni si verificano, allora si considera sicuro procedere ad attraversare l'incrocio.

Nel caso in cui, invece, l'incrocio debba essere percorso svoltando a destra o proseguendo diritto, allora la condizione per partire è semplicemente che l'incrocio sia vuoto o che si stia liberando oppure che non stia entrando nessuno.

Se non ci sono le condizioni necessarie a liberare l'incrocio, è comunque possibile cercare di proseguire, e per farlo si utilizza uno stato apposito, *JUNCTION_BIG_STEP*, nel quale si entra quando la distanza con i veicoli che si trovano nell'incrocio è sufficientemente grande, oppure quando il veicolo a sinistra si è fermato per far passare l'agente ed il veicolo che si trova a destra è sufficientemente lontano.

6. **JUNCTION_BIG_STEP:** è lo stato che serve a far proseguire a piccoli passi l'ego-veicolo, anche se l'incrocio non è totalmente sgombro. Questo stato permette ai veicoli che danno la precedenza all'ego-veicolo di fermarsi e lasciarlo passare, quando si accorgono che esso si sta lentamente immettendo nell'incrocio. Quando il sistema si trova in questo stato, si setta una velocità bassa e si prosegue per un numero predefinito di cicli di controllo, senza fare nessun controllo sulle condizioni di libertà dell'incrocio. Se durante questo stato l'ego-veicolo dovesse riuscire a percorrere tutto l'incrocio, il suo stato passerebbe a *JUNCTION_IN_PROGRESS* e l'incrocio si considererebbe superato.
 7. **JUNCTION_IN_PROGRESS:** è lo stato conclusivo dell'incrocio, durante il quale il veicolo percorre, alla velocità definita dal suo stile di guida, l'incrocio stesso. Durante questo stato viene chiamata la funzione *junction_crossing_manager* per determinare quando l'incrocio sia stato superato. Una volta superato l'incrocio, vengono resettati tutti i flag.
- ***run_step*:** è il metodo di esecuzione centrale per un modulo di controllo del veicolo o un automa di guida. Questo metodo esegue una serie di controlli e agisce di conseguenza. Di seguito si riporta un diagramma di flusso per schematizzare l'algoritmo di navigazione che esso implementa avvalendosi di tutti i metodi sopra analizzati e, inoltre, si esaminano le varie fasi di controllo che attua:

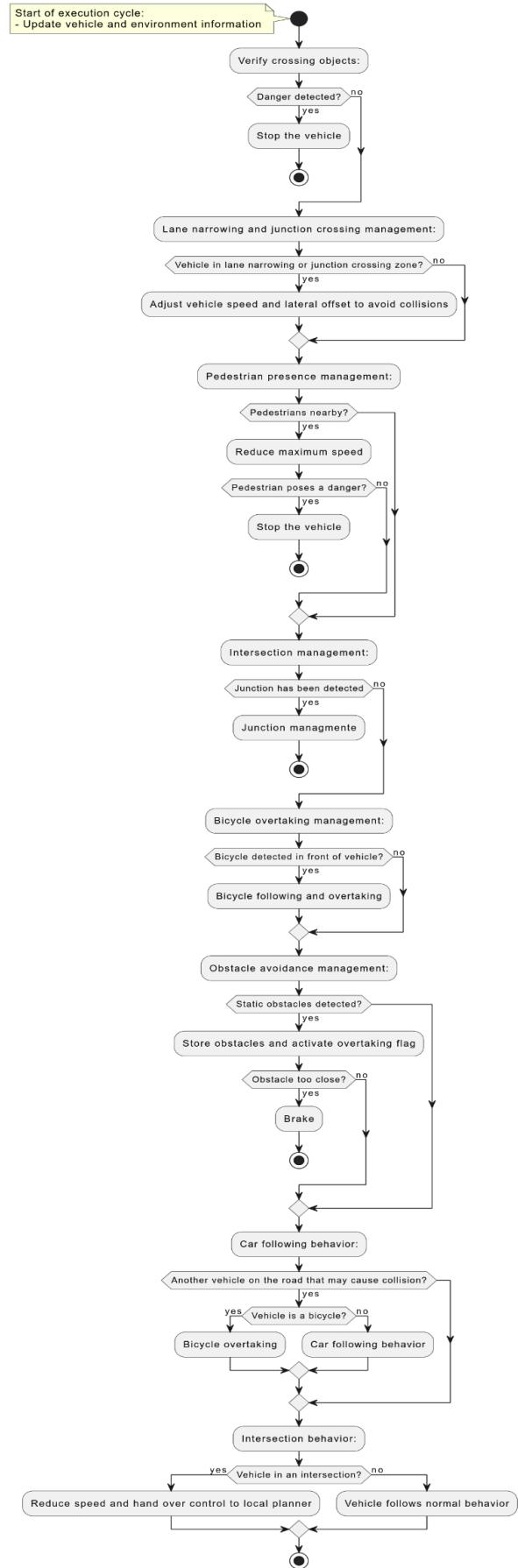


Figura 21. Schema di flusso dell'algoritmo di navigazione.

- **Inizio del ciclo di esecuzione:** viene chiamato il metodo *update_information* per aggiornare le informazioni sul veicolo e sull'ambiente.
- **Verifica degli oggetti in attraversamento:** si verifica se esistono oggetti in movimento (come una bicicletta) che attraversano la strada, invocando la funzione *crossing_object_detector*. Se viene rilevato qualche pericolo, il veicolo si ferma finché il target non ha concluso l'attraversamento. Se nessun pericolo viene rilevato, allora il ciclo di controllo prosegue la sua esecuzione.
- **Gestione del restringimento della corsia:** si controlla se il veicolo si trova in una zona di restringimento della corsia con *lane_narrowing_manager*. In caso di restringimento di corsia, si adatta la velocità dell'ego-veicolo e l'offset per il controllo laterale in modo da evitare le collisioni con i veicoli nella corsia ristretta. Se non viene rilevato alcun restringimento, si prosegue col ciclo di controllo.
- **Gestione della presenza di pedoni:** se sono rilevati pedoni nelle vicinanze, la velocità massima del veicolo viene ridotta. Una volta che il veicolo si è avvicinato a sufficienza al pedone, si ferma finché il pedone non rappresenta più un pericolo.
- **Gestione dell'incrocio:** si invoca *junction_manager* per rilevare eventuali incroci e gestirli nella maniera opportuna.
- **Gestione del sorpasso di bicicletta:** se viene rilevata una bicicletta davanti al veicolo, allora si passa nello stato di sorpasso delle bici, usando la funzione *bike_following_and_overtake_manager*.
- **Gestione dell'evitamento di ostacoli:** se vengono rilevati uno o più ostacoli statici, li si memorizza e si attiva il flag per iniziare il sorpasso, che inizierà al *run_step* successivo. In caso di ostacolo troppo vicino, si effettua una frenata di emergenza.
- **Comportamento di following dell'auto:** se c'è un altro veicolo sulla strada, che potrebbe comportare una collisione (il che è determinato da *collision_and_car_avoid_manager*), e si tratta di una bicicletta, si avvia il manager *bike_following_and_overtake_manager* per sorpassarla, altrimenti si inizia il comportamento di following del veicolo leader gestito da *car_following_manager*.
- **Comportamento all'incrocio quando attraversabile:** se il veicolo si trova in un incrocio e può attraversare (è nello stato JUNCTION_IN_PROGRESS), viene ridotta la velocità e si lascia il controllo al *local planner*.
- **Comportamento normale:** se non si verifica alcuna delle condizioni precedenti, il veicolo adotta il suo comportamento normale, cioè segue la sua velocità massima consentita (minimo tra la massima velocità dipendente dallo stile di guida e limite della strada meno una quantità per garantire una distanza di sicurezza) e si lascia il controllo al *local planner*.

6.3.3 Vehicle Controller

Nella classe *VehicleController* l'unica modifica effettuata rispetto alla versione precedente della baseline è stata l'aggiunta di un offset al controllo effettuato da Stanley, per permettere di deviare di una quantità costante dalla traiettoria predefinita. Questo è utile nelle situazioni in cui si vuole deviare dalla rotta predefinita senza dover andare a modificare i waypoints del path. Un caso di utilizzo è stato nel sorpasso delle biciclette. In questo caso si è scelta tale soluzione in quanto settare un offset per far muovere il veicolo di una certa quantità a sinistra è molto comodo ed è una soluzione molto più immediata rispetto a manipolare i waypoint del path, che in questo caso andrebbero altrimenti gestiti in maniera dinamica, mentre sia l'ego veicolo sia il target sono in movimento. Un altro caso nel quale è stato utile l'utilizzo di un offset laterale è stato per i restringimenti di corsia dovuti alla presenza di coni, nei quali adottare un offset laterale verso destra si è reso necessario per evitare collisioni con i veicoli provenienti dalla corsia con restringimento.

6.3.4 Definizione e setup dei parametri

Per la realizzazione dell'algoritmo di navigazione si è scelto di definire alcuni parametri, rappresentati principalmente da costanti all'interno del codice allo scopo di renderlo facilmente modificabile e leggibile, che permettono il corretto comportamento dell'agente. Tali parametri sono stati tarati in maniera empirica, con numerose prove e aggiustamenti incrementali, per adattarsi nel modo migliore allo stile di guida scelto per il progetto, ovvero lo stile di guida *Normale*.

I parametri utilizzati, il loro valore ed il rispettivo significato sono riportati nella tabella che segue:

Tabella 6-1. Setup dei parametri di guida

Nome Parametro	Valore	Descrizione
ZERO_SPEED_TH	0.01	Velocità sotto la quale un veicolo è considerato fermo.
FIXED_TIME_STEP	0.05	Time step della simulazione.
STOP_SIGN_SECONDS	3	Tempo di attesa di fronte un segnale di stop.
SAMPLING_RESOLUTION	1	Distanza euclidea media tra due waypoints consecutivi nel global path.
STOP_SIGN_TYPE	"206"	ID dei landmarks associati ai cartelli di stop.
LANE_NARROWING_CONE_THRESHOLD	8	Numero di coni minimo da rilevare per considerare la presenza di un restringimento di carreggiata.
LANE_NARROWING_LATERAL_OFFSET	0.9	Offset laterale in metri, da adottare quando il veicolo rileva un restringimento di carreggiata.
MAX_OBSTACLE_PROXIMITY	100	Distanza entro la quale un oggetto dinamico è considerato un potenziale ostacolo.
MAX_VEHICLE_PROXIMITY	30	Distanza entro la quale un veicolo sulla strada è considerato un potenziale pericolo.
MAX_WALKER_PROXIMITY	9	Distanza massima cui l'agente può trovarsi da un

		pedone prima di fermarsi.
WALKER_DISTANCE_PROXIMITY	35	Distanza massima cui l'agente può trovarsi da un pedone prima di rallentare.
MAX_CROSSING_PROXIMITY	9	Distanza massima cui l'agente può trovarsi da un attore che attraversa la strada.
MAX_OTHER_LANE_PROXIMITY	300	Distanza massima di visione dei veicoli che arrivano dalla corsia adiacente.
MEAN_STOP_ACCELERATION	20.4	Decelerazione media calcolata durante una frenata dalla velocità di 50 Km/h.
OBSTACLE_STOP_DISTANCE	14	Distanza a cui l'attore si ferma dagli ostacoli statici da sorpassare.
OVERTAKE_MARGIN	5	Distanza aggiuntiva per il sorpasso degli ostacoli statici. Funge da margine di sicurezza per evitare di rientrare troppo presto da un sorpasso, colpendo gli ostacoli.
OVERTAKE_TIME_MARGIN	3	Tempo aggiuntivo da sommare al tempo necessario ad effettuare il sorpasso. Funge da margine di sicurezza per evitare che i veicoli che arrivino dall'altro lato collidano con l'agente durante il sorpasso.
OVERTAKE_MINUMUM_ANGLE	10	Angolo in gradi, funge da margine di sicurezza quando consideriamo i veicoli che sopraggiungono dalla corsia opposta. Per essere considerati come alle spalle dell'agente l'angolo tra il vettore frontale dell'ego veicolo e il vettore che congiunge l'ego veicolo e il target deve essere maggiore di $90 + \text{OVERTAKE_MINUMUM_ANGLE}$. Questo rende l'agente meno propenso a collisioni durante i sorpassi.
BIKE_OVERTAKE_MINUMUM_ANGLE	40	Stesso discorso del margine precedente, questa volta per i sorpassi delle biciclette.
START BIKE_OVERTAKE_DISTANCE	20	Distanza dalla prima bicicletta, entro cui l'agente inizia la manovra di sorpasso.
BIKE_OVERTAKE_LATERAL_OFFSET	0.9	Offset laterale in metri, da adottare quando il veicolo rileva delle biciclette da superare.
OVERTAKE_COUNTER	20	Contatore da adottare prima di iniziare un sorpasso. Indica per quanti istanti di tempo accumulare i valori di accelerazione dei veicoli che sopraggiungono, prima di calcolarne la media.
OVERTAKE_HORIZON	3	Indica il numero di waypoints dopo l'ultimo waypoint

		del path dei sorpassi da cui riprendere con il path globale.
BOUNDING_BOX_PROJECTION	1.5	Costante moltiplicativa cui proiettare la parte frontale di un veicolo, per considerarlo uscito da un incrocio. Funge da margine per rendersi conto della sua uscita prima che sia davvero avvenuta.

7 Versioni del software e analisi incrementale dei risultati

Al fine di permettere una visione più completa e generale delle modifiche apportate e, in particolare, del loro effetto sulle prestazioni globali, di seguito si riportano le performance fatte registrare dalle diverse versioni del software. Queste ultime differiscono tra loro per l'aggiunta in maniera incrementale delle funzionalità implementate nell'algoritmo di navigazione.

7.1 Versione con gestione dei sorpassi e dei restringimenti di corsia

La prima versione implementata si è concentrata sulla gestione dei sorpassi degli oggetti statici e dei restringimenti di carreggiata. Il motivo fondamentale per cui si è scelto di iniziare con la risoluzione di questi problemi è il fatto che essi erano talmente critici da impedire addirittura il completamento di un percorso, in quanto cause principali dei time-out.

Come atteso, si è ottenuto un leggero miglioramento rispetto alla baseline, arrivando ad un punteggio medio globale di 36.44 %, rispetto al 22.19 % ottenuto da quest'ultima, registrando, quindi, un incremento del 14 % circa rispetto ad essa. Questa versione ha permesso di ottenere uno score route globale di 100, il che significa che tutti i percorsi sono stati completati, decretando, quindi, una deviazione standard di score route uguale a 0. Ciò costituisce un notevole progresso rispetto alla baseline, la quale non era neanche in grado di portare a termine lo scenario 1.

global_record	[-]global_record { }																								
index	-1																								
route_id	-1																								
status	Completed																								
infractions	[-]infractions { } <table border="1"><tbody><tr><td>collisions_layout</td><td>0</td></tr><tr><td>collisions_pedestrian</td><td>0.575</td></tr><tr><td>collisions_vehicle</td><td>1.342</td></tr><tr><td>red_light</td><td>0</td></tr><tr><td>stop_infraction</td><td>0.671</td></tr><tr><td>outside_route_lanes</td><td>0</td></tr><tr><td>min_speed_infractions</td><td>0.671</td></tr><tr><td>yield_emergency_vehicles_infractions</td><td>0</td></tr><tr><td>scenario_timeouts</td><td>0</td></tr><tr><td>route_dev</td><td>0</td></tr><tr><td>vehicle_blocked</td><td>0</td></tr><tr><td>route_timeout</td><td>0</td></tr></tbody></table>	collisions_layout	0	collisions_pedestrian	0.575	collisions_vehicle	1.342	red_light	0	stop_infraction	0.671	outside_route_lanes	0	min_speed_infractions	0.671	yield_emergency_vehicles_infractions	0	scenario_timeouts	0	route_dev	0	vehicle_blocked	0	route_timeout	0
collisions_layout	0																								
collisions_pedestrian	0.575																								
collisions_vehicle	1.342																								
red_light	0																								
stop_infraction	0.671																								
outside_route_lanes	0																								
min_speed_infractions	0.671																								
yield_emergency_vehicles_infractions	0																								
scenario_timeouts	0																								
route_dev	0																								
vehicle_blocked	0																								
route_timeout	0																								
scores_mean	[-]scores_mean { } <table border="1"><tbody><tr><td>score_composed</td><td>36.447744</td></tr><tr><td>score_route</td><td>100</td></tr><tr><td>score_penalty</td><td>0.364477</td></tr></tbody></table>	score_composed	36.447744	score_route	100	score_penalty	0.364477																		
score_composed	36.447744																								
score_route	100																								
score_penalty	0.364477																								
scores_std_dev	[-]scores_std_dev { } <table border="1"><tbody><tr><td>score_composed</td><td>48.041</td></tr><tr><td>score_route</td><td>0</td></tr><tr><td>score_penalty</td><td>0.48</td></tr></tbody></table>	score_composed	48.041	score_route	0	score_penalty	0.48																		
score_composed	48.041																								
score_route	0																								
score_penalty	0.48																								

Figura 22. Tabella risultati complessivi versione 1

Un'analisi delle infrazioni ancora commesse ha rivelato un calo alquanto drastico dei veicoli colpiti in simulazione (da 16 a 2): sorpassi e restringimenti della carreggiata ne erano la principale ragione. Le collisioni con gli oggetti statici sono state azzerate, così come le infrazioni associate ai timeout degli scenari. Tuttavia, si è notato un inaspettato aumento del numero di collisioni con le biciclette e di infrazioni associate ai pedoni. Ciò è sicuramente attribuibile al fatto che l'agente non ha ricevuto il time-out nello scenario 1, per cui ha continuato il suo percorso, incontrando numerose biciclette, il che invece non avveniva nella simulazione della baseline, per via dell'interruzione dello scenario medesimo a causa del raggiungimento del limite temporale.

Chiaramente, senza aver implementato ancora alcun tipo di gestione dei ciclisti in questa prima versione, non è stato possibile impedire che l'ego-veicolo collidesse con essi. Un discorso analogo riguarda gli stop: per via del timeout della simulazione della baseline nello scenario 1, non tutti i segnali di stop presenti venivano incontrati, a differenza di quello che è accaduto con questa versione dell'algoritmo.

Si è notato, inoltre, la presenza di due collisioni con veicoli, le quali sono dovute principalmente alla presenza di incroci non ancora gestiti.

Concludendo, la maggior parte delle penalità fatte registrare da questa prima versione sono imputabili a collisioni con ciclisti e pedoni. Alla luce di tale considerazione, si è deciso, come step successivo del progetto, di risolvere tali problematiche, implementandone un'opportuna gestione.



Figura 23. Areogramma infrazioni complessive versione 1

7.2 Versione con gestione pedoni e biciclette

La seconda versione realizzata si è posta l'obiettivo di ridurre sia le penalità dovute alle collisioni con le biciclette sia quelle dovute alle collisioni con i pedoni, in quanto esse erano le più gravi rimaste, sia in termini di peggioramento del punteggio complessivo sia di pericolo per gli utenti più fragili della strada. Dai risultati ottenuti da questa nuova versione è stato chiaro il raggiungimento dell'obiettivo appena esposto.

Si è ottenuto, infatti, un azzeramento totale delle collisioni con pedoni e ciclisti, il che ha portato il punteggio medio globale a 59.72 %, con un incremento di circa il 37% rispetto alla baseline e di circa il

23% rispetto alla prima versione dell'algoritmo di navigazione. Inoltre, è stato conseguito un abbassamento, rispetto alla versione precedente, della deviazione standard sia di score composed sia di score penalty.

Le infrazioni rimanenti sono attribuibili alla mancata gestione degli incroci, degli stop, e alle collisioni, verificatesi di fatto proprio all'attraversamento di alcune intersezioni. Si è riscontrato, infine, un aumento delle infrazioni legate alla velocità minima, probabilmente dovuto al fatto che la gestione di pedoni e ciclisti ha implicato un abbassamento della velocità dell'agente rispetto a quella dei veicoli circostanti.

global_record	<table border="1"> <thead> <tr> <th colspan="2">[-] global_record { }</th></tr> </thead> <tbody> <tr> <td>index</td><td>-1</td></tr> <tr> <td>route_id</td><td>-1</td></tr> <tr> <td>status</td><td>Completed</td></tr> </tbody> </table>	[-] global_record { }		index	-1	route_id	-1	status	Completed																		
[-] global_record { }																											
index	-1																										
route_id	-1																										
status	Completed																										
infractions	<table border="1"> <thead> <tr> <th colspan="2">[-] infractions { }</th> </tr> </thead> <tbody> <tr> <td>collisions_layout</td><td>0</td></tr> <tr> <td>collisions_pedestrian</td><td>0</td></tr> <tr> <td>collisions_vehicle</td><td>0.288</td></tr> <tr> <td>red_light</td><td>0</td></tr> <tr> <td>stop_infraction</td><td>0.767</td></tr> <tr> <td>outside_route_lanes</td><td>0</td></tr> <tr> <td>min_speed_infractions</td><td>0.958</td></tr> <tr> <td>yield_emergency_vehicles_infractions</td><td>0</td></tr> <tr> <td>scenario_timeouts</td><td>0</td></tr> <tr> <td>route_dev</td><td>0</td></tr> <tr> <td>vehicle_blocked</td><td>0</td></tr> <tr> <td>route_timeout</td><td>0</td></tr> </tbody> </table>	[-] infractions { }		collisions_layout	0	collisions_pedestrian	0	collisions_vehicle	0.288	red_light	0	stop_infraction	0.767	outside_route_lanes	0	min_speed_infractions	0.958	yield_emergency_vehicles_infractions	0	scenario_timeouts	0	route_dev	0	vehicle_blocked	0	route_timeout	0
[-] infractions { }																											
collisions_layout	0																										
collisions_pedestrian	0																										
collisions_vehicle	0.288																										
red_light	0																										
stop_infraction	0.767																										
outside_route_lanes	0																										
min_speed_infractions	0.958																										
yield_emergency_vehicles_infractions	0																										
scenario_timeouts	0																										
route_dev	0																										
vehicle_blocked	0																										
route_timeout	0																										
scores_mean	<table border="1"> <thead> <tr> <th colspan="2">[-] scores_mean { }</th> </tr> </thead> <tbody> <tr> <td>score_composed</td><td>59.724539</td></tr> <tr> <td>score_route</td><td>100</td></tr> <tr> <td>score_penalty</td><td>0.597245</td></tr> </tbody> </table>	[-] scores_mean { }		score_composed	59.724539	score_route	100	score_penalty	0.597245																		
[-] scores_mean { }																											
score_composed	59.724539																										
score_route	100																										
score_penalty	0.597245																										
scores_std_dev	<table border="1"> <thead> <tr> <th colspan="2">[-] scores_std_dev { }</th> </tr> </thead> <tbody> <tr> <td>score_composed</td><td>34.029</td></tr> <tr> <td>score_route</td><td>0</td></tr> <tr> <td>score_penalty</td><td>0.34</td></tr> </tbody> </table>	[-] scores_std_dev { }		score_composed	34.029	score_route	0	score_penalty	0.34																		
[-] scores_std_dev { }																											
score_composed	34.029																										
score_route	0																										
score_penalty	0.34																										

Infrazioni Versione 2

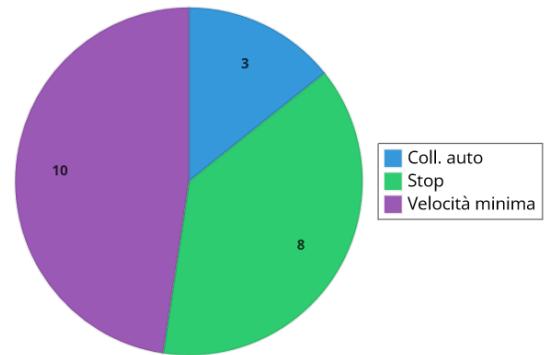


Figura 24. Tabella risultati complessivi e areogramma infrazioni complessive versione 2

7.3 Versione con gestione incroci, semafori e stop

Nella terza e ultima versione dell'algoritmo di navigazione implementato si è provveduto all'eliminazione delle penalità rimanenti nella versione precedente, ovvero quelle riguardanti le collisioni con altri veicoli negli incroci, il non rispetto dei segnali di stop e dei semafori rossi in loro prossimità. La scelta di lasciare la gestione degli incroci per ultima è dovuta sia alla presenza di altre problematiche di maggiore importanza in termini di gravità e, di conseguenza, penalità, sia all'elevata complessità che caratterizza tale gestione, essendo presenti nei vari scenari molteplici tipologie di incroci: con/senza stop, con/senza semafori, a T, a X, larghi/stretti.

Pertanto, con l'aggiunta di un'opportuna gestione degli incroci, si è giunti a risultati più che soddisfacenti sia complessivi sia in termini di singoli scenari, come si evince dalla tabella riportata di seguito:

global_record	[-] global_record { }
	index -1
	route_id -1
	status Completed
infractions	[-] infractions { }
	collisions_layout 0
	collisions_pedestrian 0
	collisions_vehicle 0.096
	red_light 0
	stop_infraction 0
	outside_route_lanes 0
	min_speed_infractions 1.438
	yield_emergency_vehicles_infractions 0
	scenario_timeouts 0
scores_mean	[-] scores_mean { }
	score_composed 76.875161
	score_route 100
	score_penalty 0.768752
scores_std_dev	[-] scores_std_dev { }
	score_composed 24.942
	score_route 0
	score_penalty 0.249

Infrazioni Versione 3

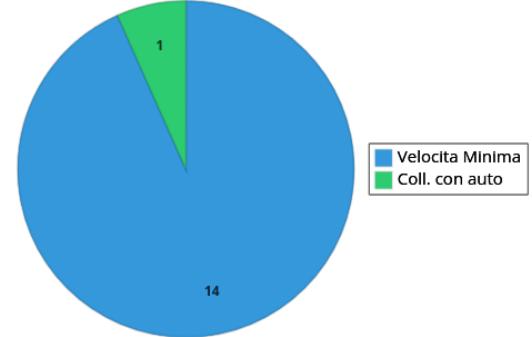


Figura 25. Tabella risultati complessivi e areogramma infrazioni complessive versione 2

Viene delineata di seguito un'analisi più dettagliata dei risultati finali ottenuti, considerando i singoli scenari:

- **Scenario 0:** questo scenario è stato completato con successo con un elevato punteggio di guida (*score_composed*) di **91.04 %**. Ci sono state unicamente due infrazioni per velocità minima, in cui la velocità media era l'84.72 % di quella del traffico circostante.
- **Scenario 1:** tale scenario ha avuto il punteggio di guida più basso, uguale a **48.30 %**. Ci sono state cinque infrazioni in totale. Una di queste è stata una collisione con un altro veicolo. In merito a quest'ultima, dall'analisi del video della simulazione finale, si evince che la collisione è avvenuta in un incrocio a causa dell'altro veicolo che si è fermato troppo vicino all'ego-veicolo mentre questo stava attraversando l'incrocio. Inoltre, ci sono state quattro infrazioni per velocità minima, con la velocità media rispettivamente del 72.56 %, 98.68 %, 79.49 % e ancora 79.49 % rispetto a quella dei veicoli circostanti.
- **Scenario 2:** anche in questo scenario, come nel precedente, il punteggio di guida è stato un po' basso, pari a **51.13 %** a causa di cinque infrazioni per velocità minima, in cui la velocità media era rispettivamente del 92.14 %, 99.86 %, 84.66 %, 13.69 % e ancora 13.69 % rispetto a quella del traffico circostante.
- **Scenario 3:** tale scenario è stato completato con successo con un punteggio di guida molto alto, **95.77 %**. Ci sono state soltanto tre infrazioni per velocità minima, in cui la velocità media era rispettivamente del 97.62 %, 94.06 % e ancora 94.06 % rispetto a quella dei veicoli circostanti.

- **Scenario 4:** questo scenario ha avuto il punteggio di guida più alto, pari a **98.12 %**. C'è stata solo un'infrazione per velocità minima, in cui la velocità media era del 93.73 % rispetto a quella del traffico circostante.

In generale, il problema più frequente è stato il non mantenimento della velocità minima, indicando che l'agente spesso ha viaggiato più lentamente rispetto al traffico circostante. Tuttavia, osservando le singole percentuali si nota che la discrepanza di velocità con gli altri veicoli circostanti non è molto significativa nella maggior parte dei casi. Inoltre, la collisione con un altro veicolo nello scenario 1, che ha avuto un impatto significativo sul punteggio di guida per quello scenario, è dovuta per lo più ad un comportamento poco corretto da parte dell'altro veicolo che alla gestione dell'incrocio ad opera dell'ego-veicolo, il quale, infatti, gestisce perfettamente tutti gli altri incroci, nonostante la loro elevata complessità.

In conclusione, effettuando una comparazione tra queste prestazioni finali e le prestazioni ottenute con la baseline, si è conseguito un incremento significativo del 54.68 % per *score_composed* e, per quanto concerne le altre metriche, *score_route* ha raggiunto il 100% (mentre era pari a 82.4 % nella baseline) e *score_penalty* è passato da circa 0.27% a circa 0.77% (si ricordi che l'ideale è 1). Anche le deviazioni standard delle varie metriche sono diminuite, in particolare per *score_route* si è addirittura azzerata.

8 Analisi qualitativa della simulazione finale

Per la versione finale dell'algoritmo di navigazione si delinea anche un'analisi qualitativa corredata dal link al video [\[Link\]](#) che mostra la simulazione finale in ciascuno dei 5 scenari, i cui punteggi ottenuti sono stati discussi in precedenza.

Innanzitutto, si precisa che lo stile di guida adottato è quello "normale". Dopodiché si osserva che tutti gli ostacoli statici (come, ad esempio, il traffic warning), che l'agente incontra lungo il proprio percorso, vengono evitati con margine opportuno. Inoltre, i ciclisti e i pedoni non vengono in nessuna occasione messi in pericolo di vita né di ferirsi perché l'ego-veicolo non si avvicina mai troppo ad essi: i ciclisti quando incontrati lungo la strada, essendo al bordo, vengono prudentemente sorpassati; mentre l'ego-veicolo si ferma ad una distanza sicura da pedoni e ciclisti che attraversano all'improvviso la strada.

L'ego-veicolo riesce anche a gestire bene i restringimenti delle corsie, senza collidere con i veicoli che transitano in verso opposto, così come sorpassa in maniera adeguata le auto parcheggiate a bordo strada, invadendo la corsia con senso di marcia opposto senza causare incidenti.

In aggiunta, l'agente rispetta correttamente anche tutte le regole previste dal codice della strada: si ferma a tutti gli stop che incontra, ripartendo solo dopo 3 secondi, si ferma anche a tutti i semafori rossi presenti in alcuni incroci, attraversandoli solo quando questi ritornano ad essere di colore verde, riesce a gestire bene tutte le variegate e complesse tipologie di incroci presenti, dando la precedenza ai veicoli a cui spetta e attraversandoli solo nel momento in cui sussistono tutte le condizioni per farlo in sicurezza.

Per quanto concerne gli aspetti riguardanti il comfort di guida, quest'ultimo non è ancora il massimo a causa di frenate un po' brusche. Tuttavia, le curve sono affrontate bene dall'ego-veicolo, anche quelle più strette, e viene mantenuta una buona distanza di sicurezza dal veicolo che precede lungo la propria corsia.



Figura 26. Ego-veicolo che attraversa un restringimento (fig. a sinistra) ed ego-veicolo che sorpassa un oggetto statico (fig. a destra)

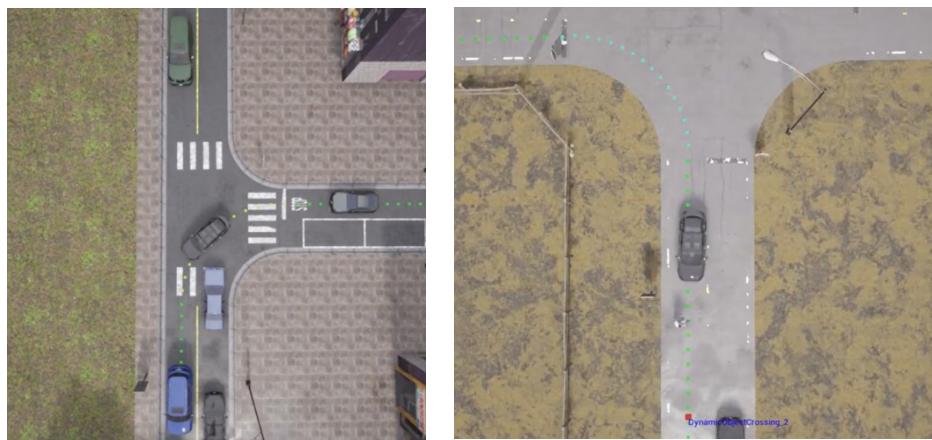


Figura 27. Ego-veicolo che attraversa un incrocio a T (fig. a sinistra) ed ego-veicolo che si ferma per lasciar attraversare un pedone (fig. a destra)

9 Limitazioni e vantaggi del simulatore CARLA

Si illustrano in tale capitolo le limitazioni e i vantaggi che si sono riscontrati nell'utilizzo del simulatore CARLA durante la realizzazione del progetto, in particolare, facendo riferimento sia ai bug incontrati sia ai moduli di percezione necessari all'implementazione di un algoritmo di navigazione, qualora non si abbiano a disposizione i dati perfetti del simulatore.

9.1 Bug riscontrati

CARLA è un software Open Source il cui contributo è libero. Questo comporta che sia gratuito da utilizzare, ma anche che non tutte le funzionalità che presenta siano sempre esenti da difetti. Durante lo sviluppo dell'algoritmo di navigazione, infatti, è capitato di incappare più volte in alcuni bug, soprattutto legati alle informazioni ricavabili dal mondo 3D. Per alcuni di questi bug è stato anche necessario escogitare dei "workaround", in quanto la loro presenza rendeva molto difficile, se non impossibile, completare il percorso previsto dallo scenario. I bug riscontrati e le relative soluzioni trovate sono elencati di seguito:

9.1.1 Bug riguardanti i waypoints

Una parte dei bug incontrati riguarda i waypoints del percorso, in particolare si sono riscontrati problemi sia legati al loro posizionamento sia alla loro etichettatura. Nello scenario 3 si verificano entrambe le tipologie di bug. In particolare, i bug di posizionamento consistono in singoli waypoints leggermente disallineati dalla traiettoria rispetto ad altri. Questi leggeri disallineamenti possono portare il veicolo a sbandare inutilmente durante il percorso, provocando collisioni o uscite fuori strada difficilmente evitabili. Inoltre, sempre nello scenario 3, si nota la presenza di alcuni waypoints classificati come facenti parte di un incrocio, quando, in realtà, si tratta di normali waypoints appartenenti alla corsia di percorrenza. Nello scenario 4, invece, si presenta il problema dei waypoints posizionati in maniera scorretta, all'inizio di un incrocio, che è una posizione alquanto delicata, in quanto a rischio collisione con un cartello di stop presente a fianco, oltre che con i veicoli che transitano attraverso l'incrocio.

Per quanto concerne il bug dell'etichettatura, la soluzione adottata è quella di inserire il primo waypoint etichettato erroneamente come *junction* in una *blacklist*, in modo da ignorare l'effetto.

Per i bug di posizionamento non si sono adottate contromisure, in quanto, pur essendo molto pericolosi in termini di penalità che possono causare, le possibili soluzioni pensate non sono immediate da implementare. Quest'ultime possono essere: aumentare la densità dei waypoints, in modo che quelli posizionati male influenzino meno il cammino del veicolo; inserirli in una blacklist e non considerarli; oppure applicare manualmente un offset al controllo laterale in loro prossimità, in modo tale che il loro effetto sia nullo.

9.1.2 Bug riguardanti i Bounding Box

Alcuni dei problemi incontrati riguardano la non correttezza delle informazioni ottenute dal simulatore, in merito ai bounding box di alcuni degli attori presenti negli scenari. Nonostante la maggior parte degli attori presenti sia caratterizzato dalle corrette informazioni in termini di posizione, rotazione e coordinate del bounding box, ci sono alcuni attori specifici le cui informazioni dei bounding box sono sbagliate o addirittura assenti.

Il problema dei bounding box con informazioni errate si presenta per un insieme molto ristretto di automobili, in particolare nello scenario 0 e nello scenario 1, per due automobili parcheggiate fuori strada. Queste ultime, pur trovandosi fuori strada e, quindi, in condizione di essere facilmente sorpassate senza alcun rischio, presentano dei bounding box leggermente traslati rispetto alla loro mesh 3D nel mondo. Ciò provoca il rilevamento di un falso positivo, quando si individuano i veicoli che intersecano la proiezione dell'ego-veicolo lungo la sua traiettoria. Questo problema è stato aggirato facilmente stilando una *blacklist* di tutti i veicoli con tale problematica. In tal modo, nel momento in cui l'ego-veicolo si approccia ad uno di questi, la soluzione attuata consiste nell'ignorare le informazioni del loro bounding box per rilevare una possibile collisione.

Un ulteriore problema riguarda il bounding box di tutte le biciclette, per le quali le informazioni ad esso relative sono del tutto assenti, collassando i suoi vertici tutti nel medesimo punto. Per fortuna, questo non si è dimostrato essere un grave problema in quanto le informazioni dei bounding box delle biciclette non vengono mai utilizzate dall'algoritmo di navigazione per rilevare le biciclette e sorpassarle correttamente qualora possibile. Al loro posto vengono utilizzate, infatti, solo le informazioni relative alla posizione e all'orientamento delle biciclette, le quali sono correttamente disponibili.

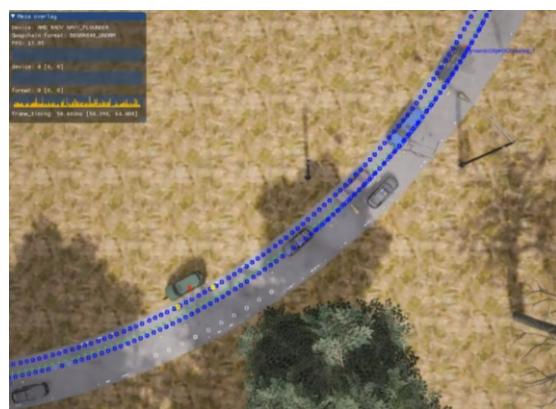


Figura 28 17. Auto fuori strada con bounding box posizionato in modo errato dal simulatore

9.2 Dati reali imperfetti: moduli di percezione

Il mondo reale è incredibilmente vario e dinamico: esso contiene una moltitudine di oggetti, colori, forme, luci e ombre che cambiano costantemente e gli eventi imprevisti sono alquanto comuni. Un simulatore, invece, può rappresentare solo una frazione di questa complessità.

La complessità e l'incertezza del mondo reale richiedono, pertanto, un ampio insieme di moduli per percepire, capire e reagire alle condizioni dell'ambiente di guida in tempo reale.

Nel caso in cui non si abbiano a disposizione i dati perfetti del simulatore, infatti, è necessario avvalersi di moduli di percezione specifici che sono necessari a fornire all'algoritmo di navigazione tutti i dati di cui ha bisogno. Tuttavia, i dati acquisiti dai moduli di percezione non sono affatto perfetti come quelli forniti dal simulatore, che rappresentano esattamente ciò che l'ambiente virtuale contiene, senza errori di misurazione o rumore dovuto ai sensori. I sensori reali, come le telecamere, il lidar e il radar, non sono per niente impeccabili. Essi sono soggetti a rumore, distorsione, errori di calibrazione, limiti di precisione e, inoltre, possono essere influenzati da condizioni esterne, come il clima e la luce.

I moduli di percezione necessari per navigare nel mondo reale, senza i dati perfetti del simulatore, possono essere:

- **moduli di rilevamento:** includono sensori come telecamere, lidar e radar per raccogliere dati dall'ambiente.
- **moduli di localizzazione e mappatura simultanea (SLAM):** utilizzano i dati dei sensori per creare una mappa dell'ambiente circostante e determinare la posizione e l'orientamento del veicolo all'interno di esso.
- **moduli di riconoscimento di oggetti e pedoni:** rilevano gli altri elementi nell'ambiente, tra cui veicoli, pedoni, ciclisti, segnali stradali, semafori e ostacoli vari.
- **moduli di previsione del comportamento:** cercano di prevedere il comportamento futuro degli oggetti rilevati, come la traiettoria di un veicolo o il percorso di un pedone.
- **moduli di fusione dei sensori:** combinano i dati provenienti da diversi sensori per ottenere una comprensione più completa e accurata dell'ambiente circostante. Ad esempio, mentre una telecamera può fornire informazioni dettagliate sul colore e la forma degli oggetti, un sensore lidar può fornire misure di distanza precise.
- **moduli di percezione del meteo:** rilevano e interpretano le condizioni atmosferiche, come pioggia, nebbia, neve o ghiaccio, che possono influenzare la visibilità e la trazione del veicolo.
- **moduli di interpretazione della strada:** identificano caratteristiche stradali come linee di demarcazione, corsie, marciapiedi, barriere, isole di traffico e così via.
- **moduli di percezione dell'illuminazione:** rilevano le condizioni di illuminazione, come l'alba, il tramonto, l'oscurità, l'illuminazione artificiale, ecc., che possono influenzare la visibilità e l'interpretazione delle scene da parte del veicolo.

10 Conclusioni e sviluppi futuri

A causa del tempo limitato disponibile per l'implementazione delle soluzioni proposte, non è stato possibile sviluppare tutte le funzionalità desiderate, fattore che potrebbe aver influenzato il punteggio finale. Una delle caratteristiche di primaria importanza è senza dubbio la gestione della velocità del veicolo, in particolare per monitorare e controllare le infrazioni relative alla velocità minima. Se tale funzionalità fosse stata implementata correttamente, avrebbe potuto annullare completamente le penalità presenti nella simulazione della versione finale. Questo avrebbe potuto, quindi, portare ad ottenere un punteggio massimo in tutti gli scenari in esame.

Un'altra area di miglioramento riguarda la gestione di piccoli bug riscontrati in CARLA. Alcuni di questi bug, come quello relativo ai waypoint disposti in modo errato, potrebbero causare penalità in modo imprevedibile se non gestiti adeguatamente. Si potrebbe incorrere in collisioni accidentali dovute a derapate del veicolo o ad un errato attraversamento di un incrocio. Un approccio possibile per gestire questi bug sarebbe l'incremento della densità dei waypoint, il che renderebbe il comportamento del veicolo meno brusco. Un'altra soluzione potrebbe essere la definizione di un offset tale da compensare lo spostamento laterale dovuto ai waypoint affetti da bug.

In seguito, si potrebbe considerare il miglioramento del modo in cui alcune funzionalità sono state implementate, al fine di semplificare il codice e aumentarne l'efficienza. Ad esempio, la gestione del sorpasso di oggetti statici potrebbe essere realizzata applicando un offset laterale, piuttosto che manipolando i waypoint del percorso globale. Sebbene non siamo sicuri che questa modifica porti solo miglioramenti, poiché non abbiamo avuto l'opportunità di implementarla, certamente renderebbe la fase di EVALUATION della funzione `overtake_manager` più immediata. Ciò, tuttavia, richiederebbe una gestione del calcolo dei tempi differente rispetto all'attuale implementazione.

Alcune implementazioni nel progetto si basano su assunzioni che funzionano bene per gli scenari presentati, ma che non sono necessariamente sempre valide, soprattutto in contesti reali e non simulati. In particolare, l'offset necessario per il sorpasso delle biciclette e per il transito nei restrinimenti di carreggiata è lo stesso per tutti i casi, ed è stato determinato empiricamente. Tuttavia, in scenari differenti, questa soluzione potrebbe non essere sempre adeguata. Pertanto, potrebbe essere necessario introdurre una fase di tuning in cui si calcoli un offset adeguato alla particolare situazione, al fine di prevenire collisioni con ciclisti e veicoli provenienti dalla corsia adiacente.

Potrebbe per ulteriori scenari essere necessario andare a effettuare una gestione degli incroci più precisa, in particolare andando a rilevare se sulle strade che confluiscono nell'incrocio vi sia o meno uno stop. Questa informazione potrebbe essere usata per regolare il comportamento dell'ego-veicolo in merito alla gestione delle precedenze. C'è da evidenziare, tuttavia, che i veicoli di CARLA non rispettano completamente le regole di precedenza, soprattutto in prossimità di incroci, per cui cercare di gestire un'implementazione che tenesse conto perfettamente delle regole stradali avrebbe potuto portare addirittura a peggiorare le performance, vista la poca affidabilità degli altri veicoli.

Infine, anche se non fa parte della valutazione, sarebbe opportuno considerare il comfort di guida, in particolare riguardo alle frenate. Nel contesto del progetto, queste si riducono essenzialmente a frenate d'emergenza, le quali hanno l'obiettivo di arrestare il veicolo il più rapidamente possibile non appena si identifica un potenziale pericolo. Questo, purtroppo, compromette notevolmente il comfort di guida. Pertanto, sarebbe importante implementare frenate più morbide, con decelerazioni che variano dai 2 m/s² ai 5 m/s².

