# UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE ED ELETTRICA E MATEMATICA APPLICATA

Corso di Laurea Magistrale in Ingegneria Informatica



Report Finale

## Report Finale Autonomous Veichle Driving


Docenti:
**Prof. Mario Vento**
**Prof. Antonio Greco**

Studente:
**Giuseppe Marotta - 0622702302**
**Luca Memoli - 0622702309**
**Antonio Russomando - 0622702407**
**Raffaele Solimeno - 0622702308**


Report finale del corso di:
**Autonomous Veichle Driving**

ANNO ACCADEMICO 2024/2025

# Introduzione

Il presente report documenta il processo di progettazione, sviluppo e valutazione di un sistema di guida autonoma, realizzato nell'ambito del corso di Autonomous Vehicle Driving. Sfruttando le potenzialità offerte dal simulatore CARLA, l'obiettivo primario è stato quello di creare un agente capace di navigare autonomamente in scenari realistici. Tale capacità include la gestione di manovre come i cambi di corsia, l'attraversamento di incroci, il pieno rispetto della segnaletica stradale e l'interazione con gli altri agenti.

Il cuore del nostro sistema è rappresentato da una versione evoluta del modulo `BehaviorAgent`, fornito nativamente da CARLA. Questo agente è stato oggetto di un'attenta configurazione e di mirate modifiche al fine di ottimizzare le sue prestazioni, ponendo particolare enfasi sui criteri di valutazione definiti: il completamento efficace del percorso assegnato, la minimizzazione delle penalità dovute a infrazioni al codice della strada.

Un aspetto cruciale dell'implementazione ha riguardato la capacità dell'agente di gestire eventi critici e situazioni impreviste. Sono state sviluppate logiche specifiche per il rispetto dei semafori rossi, per l'efficace evitamento di ostacoli (siano essi veicoli fermi o in movimento, o pedoni distratti), per il mantenimento costante di una distanza di sicurezza adeguata e per la prevenzione del fenomeno del *tailgating*.

L'architettura software del nostro agente è ispirata ai principi di modularità promossi da Autoware. Questa scelta ha portato a una suddivisione dei componenti logici – percezione dell'ambiente, pianificazione del percorso a breve e lungo termine, meccanismi di controllo del veicolo e gestione comportamentale – con il duplice vantaggio di incrementare la chiarezza strutturale del codice, facilitare la scalabilità futura del sistema e semplificare le operazioni di manutenzione e debugging.

Nel corso del progetto, le performance dell'agente sono state sottoposte a test intensivi e ad analisi dettagliate attraverso una varietà di scenari simulati. L'obiettivo costante di queste valutazioni è stato il progressivo miglioramento del punteggio di guida globale, metrica fondamentale che sintetizza l'efficacia e la sicurezza del comportamento autonomo del veicolo. Questo report si propone di illustrare le sfide affrontate, le soluzioni implementate e i risultati conseguiti in questo percorso di sviluppo.

# Capitolo 1

# Analisi della baseline

Prima di intraprendere lo sviluppo di funzionalità avanzate e personalizzazioni, è stato fondamentale condurre un'analisi approfondita del comportamento dell'agente di base, il `BehaviorAgent`. Questo implementa una serie di comportamenti essenziali per la navigazione, ma, come vedremo, con margini significativi di miglioramento in termini di fluidità, sicurezza percepita e gestione di scenari complessi.
L'agente `BehaviorAgent` opera principalmente attraverso una logica cablata e una serie di gestori di comportamento specifici. La sua configurazione iniziale permette di scegliere tra diversi profili comportamentali (es. `cautious`, `normal`, `aggressive`) che influenzano parametri come la velocità massima, la distanza di sicurezza e la rapidità di reazione. La percezione dell'ambiente si affida in larga misura alle informazioni sullo stato del mondo fornite direttamente dal simulatore.

## 1.1 Criticità della baseline

Dalle prime fasi di test e dall'analisi del codice del `BehaviorAgent` standard, sono emerse diverse limitazioni che ne compromettevano la normale fluidità di guida.

- **Gestione inadeguata della distanza laterale da altri agenti:** il comportamento dell'ego vehicle presenta carenze nella gestione della prossimità a utenti "vulnerabili" come ciclisti e motociclisti. Sebbene il `collision_and_car_avoid_manager` possa rilevare questi agenti come "veicoli", la logica di evitamento e di car-following è primariamente focalizzata sulla distanza longitudinale. Manca un meccanismo esplicito per garantire un'adeguata distanza di sicurezza laterale durante il transito o in fase di affiancamento. Questo porta a manovre che generano una collisione.
- **Mancato rilevamento e reazione a ostacoli e segnaletica temporanea:**
  - **Ostacoli statici**: l'architettura della baseline non include moduli dedicati al rilevamento di ostacoli statici generici, come barriere stradali (ad esempio, transenne), coni di deviazione, o la segnaletica verticale indicante lavori in corso (che non siano semafori).
  - **Ostacoli dinamici**: la baseline non gestisce in modo efficiente i veicoli (automobili, motocicli e bici) e pedoni; tali agenti, infatti, vengono rilevati attraverso i metodi
    `collision_and_car_avoid_manager` e `pedestrian_avoid_manager`, ma nelle

simulazioni non vengono gestiti correttamente e spesso l'ego vehicle collide con tali attori.

- **Gestione dei semafori e della segnaletica stradale**: durante le simulazioni della baseline, l'ego vehicle non rispetta la segnaletica stradale verticale (segnali di stop, semafori ecc..) ed orizzontale (strisce pedonali).
- **Gestione limitata degli incroci**: l'agente non possiede una logica basata sulle norme del codice stradale per affrontare gli incroci. Tende a fare affidamento sul pianificatore per seguire la traiettoria, ma tale comportamento può produrre uno stallo o agire in modo "insicuro" in presenza di altri veicoli.

Queste criticità evidenziano come, sebbene il `BehaviorAgent` costituisca un buon punto di partenza per la navigazione di base, sia necessario migliorarne l'intelligenza, la fluidità e la sicurezza. Questo ha guidato le scelte implementative per i miglioramenti descritti nei capitoli successivi.

# Capitolo 2

# Modularizzazione della baseline

Una delle direttrici fondamentali del nostro lavoro è stata la ristrutturazione del codice dell'agente `BehaviorAgent` originale verso un'architettura più modulare. La baseline, pur essendo funzionale, presentava una struttura monolitica in cui le diverse logiche di percezione, decisione e controllo erano spesso intrecciate. Questo approccio rendeva difficoltoso l'isolamento dei problemi e l'introduzione di nuove funzionalità. Per guidare il processo di modularizzazione, abbiamo tratto ispirazione dall'architettura ad alto livello di Autoware, che abbiamo adattato al contesto del nostro progetto e alle funzionalità del `BehaviorAgent`. Sebbene non si tratti di una re-implementazione completa in funzione di Autoware, ne abbiamo adottato i principi guida per organizzare il nostro codice.

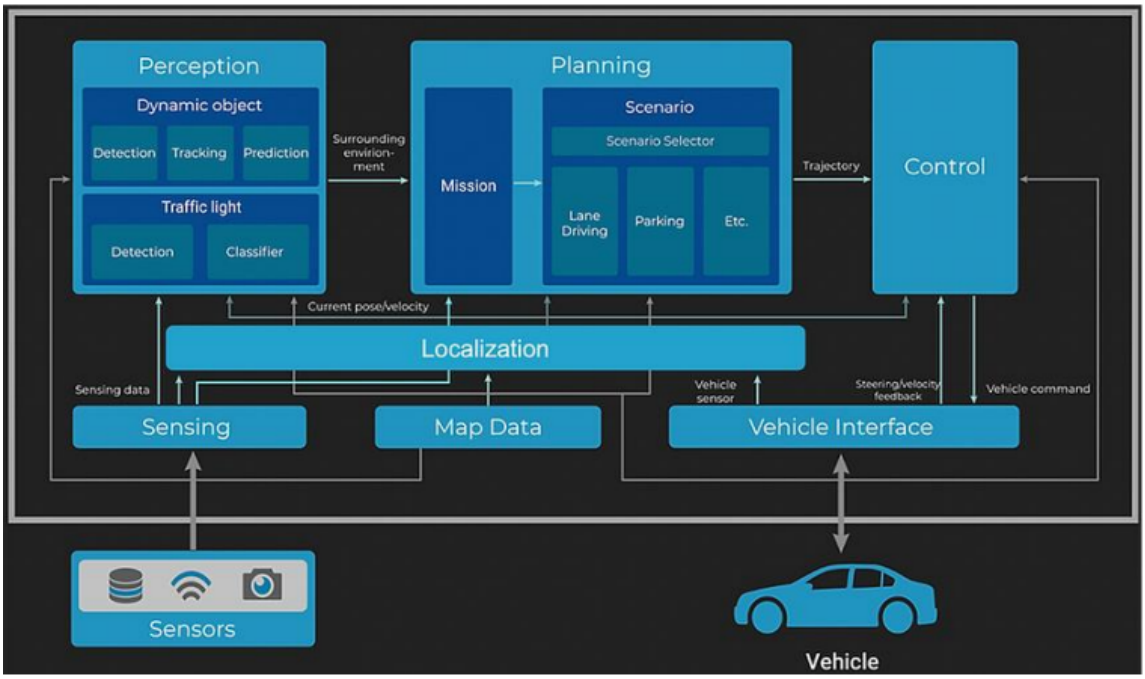


Figura 2.1: Architettura modulare Autoware

## 2.1 Perception

In un sistema completo, questo strato elaborerebbe i dati grezzi dei sensori per costruire una rappresentazione significativa dell'ambiente. Nel nostro caso, dato l'uso del ground truth, questo strato è più "leggero" ma comunque presente. La modularizzazione qui ha permesso di separare nettamente la logica di "cosa c'è intorno" dalla logica di "cosa fare". All'interno del PerceptionModule, è stato sviluppato un componente specializzato denominato StaticObjectPerception, il cui compito è identificare, filtrare e categorizzare i diversi tipi di attori statici presenti nell'ambiente simulato. Questa classe fornisce al BehaviorAgent una visione strutturata degli ostacoli fissi e degli elementi stradali rilevanti nelle vicinanze del veicolo.

```
class StaticObjectPerception:
    """
    Modulo di percezione per oggetti statici (props, semafori, segnali di stop)
    """
    def __init__(self, waypoint: carla.Waypoint, world: carla.World):
        self._waypoint = waypoint
        self._world = world
        self._static_prop_objects: List[carla.Actor] = []
        self._static_traffic_light_objects: List[carla.Actor] = []
        self._stop_sign_objects: List[carla.Actor] = []
        self._all_static_objects : List[carla.Actor] = []
        self.dirts_street_objects : List[carla.Actor] = []
        self.dirtdebris_street_objects : List[carla.Actor] = []
        self.brokentile_street_objects : List[carla.Actor] = []
        self.id_objects_dict : List[str,List[carla.Actor]] =
    [("static.prop",self._static_prop_objects),("traffic_light",self._static_traffic_light_objects


    def _distance_to_waypoint(self, actor: carla.Actor) -> float:
        return actor.get_location().distance(self._waypoint.transform.location)

    def update(self, max_distance: float, waypoint: carla.Waypoint = None):
        if waypoint is not None:
            self._waypoint = waypoint
        self.update_all_information(max_distance)

    def update_all_information(self,max_distance):
        self.clear_information()
        self.get_static_actors()
        self.update_actors_list(max_distance)

    def update_actors_list(self, max_distance):
        for actor in self._all_static_objects:
            for id, objects in self.id_objects_dict:
                if self._distance_to_waypoint(actor) < max_distance and
    "dirtdebris" not in actor.type_id and id in actor.type_id:
                    objects.append(actor)

    def get(self,name: str, distance = 30):
        for id, objects in self.id_objects_dict:
            if name in id:
                return objects

    def get_static_actors(self):
        self._all_static_objects = list(self._world.get_actors())
```

```
    def clear_information(self):
        for id, objects in self.id_objects_dict:
            objects.clear()
```

Listato 2.1: StaticObjectPerception class

Lo StaticObjectPerception astrae la complessità del rilevamento e filtraggio di basso livello degli oggetti statici. Fornendo liste categorizzate di attori, semplifica il compito del BehaviorAgent, che può così concentrarsi sulla logica decisionale di alto livello. Questo componente contribuisce significativamente alla modularità del sistema di percezione, permettendo di modificare o estendere le capacità di rilevamento degli oggetti statici con un impatto minimo sul resto dell'agente.

## 2.2 Planning

Questo è il modulo che rappresenta l'"intelligenza" dell'agente. Qui, le informazioni dallo strato di Percezione vengono utilizzate per prendere decisioni di alto livello sul comportamento da adottare.

### 2.2.1 Behavior Planning

La macchina a stati del `BehaviorAgent` è stata mantenuta come concetto centrale, ma ogni stato è stato incapsulato in un modulo più autonomo.

- Route Planner: Un modulo che, dato un obiettivo, pianifica il percorso globale (sequenza di strade e svolte), tipicamente interagendo con il pianificatore di CARLA.
- Behavior Selector: Il modulo centrale che, in base allo stato attuale, al percorso globale e alle informazioni ambientali, decide quale comportamento specifico attivare.

### 2.2.2 Local Planning

Una volta deciso un comportamento, questo strato è responsabile di generare una traiettoria a breve termine (una sequenza di waypoint locali) e/o un profilo di velocità che il veicolo deve seguire.

- Trajectory Generation: Calcolo dei waypoint locali che definiscono il percorso immediato, tenendo conto della curvatura della strada, degli ostacoli e del comportamento deciso (es. traiettoria per un cambio corsia).
- Speed Profiling: Determinazione della velocità target istantanea, considerando limiti, traffico, curve e la necessità di fermarsi (semafori, ostacoli).

Abbiamo separato la generazione dei waypoint dalla logica di controllo, permettendo di affinare questi due aspetti indipendentemente.

## 2.3 Control

Questo strato traduce la traiettoria e il profilo di velocità desiderati in comandi effettivi per gli attuatori del veicolo.

- Lateral Control: Un controllore di Stanlley che calcola l'angolo di sterzata necessario per seguire la traiettoria locale.
- Longitudinal Control: Un controllore PID che regola acceleratore e freno per raggiungere e mantenere la velocità target.

L'incapsulamento dei controllori PID esistenti in moduli distinti ha facilitato il loro tuning. Per quanto concerne i parametri dei controllori, questi sono stati sottoposti a varie sperimentazioni, ma quelli di default si sono rivelati essere quelli migliori e che inseguivano il riferimento senza sovraelongare.

# Capitolo 3

# Implementazione delle soluzioni proposte

Sfruttando le capacità del nuovo `PerceptionModule`, è stata implementata una nuova logica per il rilevamento e la gestione degli ostacoli statici. Il metodo chiave è il nuovo `static_obstacle_avoid_manager(self, waypoint, static_objs, lane='same')`. Questo metodo, chiamato all'interno di `run_step`, utilizza la lista `static_objs` fornita dal modulo di percezione per identificare gli ostacoli statici dinanzi all'ego vehicle. Tale modulo:

- Analizza gli oggetti statici verificando se si trovano sulla stessa strada e nella corsia dell'agente o, specificamente, nella corsia di sinistra.
- Utilizza la funzione `is_within_distance` con angoli di rilevamento specifici (es. [0, 30] gradi per la corsia corrente, [0, 135] per la corsia sinistra) e una distanza massima (50 metri) per determinare la rilevanza dell'ostacolo.
- Restituisce una lista ordinata di ostacoli rilevati, permettendo all'agente di reagire a quello più vicino.

Questa gestione permette all'agente di avere una consapevolezza più precisa della presenza e della posizione degli ostacoli statici, fondamentale per le manovre successive.

## 3.1 Implementazione della manovra di sorpasso

Una delle funzionalità più significative introdotte è la capacità dell'agente di eseguire manovre di sorpasso di ostacoli presenti nella propria corsia. Questa logica è orchestrata principalmente dal nuovo metodo `sorpassa(self, waypoint, target, target_length, target_distance, ...)` e si articola come segue:

1. **Attivazione:** La manovra di sorpasso viene considerata quando `static_obstacle_avoid_manager` rileva un ostacolo nella corsia dell'agente (`obstacle_state` è True in `run_step`).
2. **Valutazione della corsia adiacente:** Prima di iniziare il sorpasso, l'agente verifica la presenza di veicoli potenzialmente pericolosi nella corsia di sorpasso utilizzando il metodo ausiliario

_controllo_veicoli_in_corsia_di_sorpasso(self, waypoint, distance). Questo metodo controlla la presenza di altri veicoli nella corsia sinistra entro una certa distanza e in un angolo rilevante.

3. **Generazione del percorso di sorpasso:** Se la corsia di sorpasso è libera, o se si prevede uno spazio sufficiente per la manovra (considerando la velocità del veicolo in arrivo s1 > distanza_sorpasso + 5), viene invocato il metodo _genera_percorso_per_sorpasso(self, distance_for_overtaking, direction='left', two_way=False). Questo metodo costruisce dinamicamente un nuovo piano locale (una sequenza di waypoint e RoadOption) che guida il veicolo:

   - Fuori dalla propria corsia (es. RoadOption.CHANGELANELEFT).
   - Lungo la corsia di sorpasso per una distanza calcolata (distance_for_overtaking, che considera la lunghezza dell'ostacolo e la distanza da esso).
   - Di nuovo nella corsia originale (es. RoadOption.CHANGELANERIGHT).

   Questo nuovo piano viene quindi impostato utilizzando self.set_global_plan(plan), sovrascrivendo temporaneamente il piano globale precedente.

4. **Esecuzione e controllo velocità:** Durante il sorpasso, la velocità dell'agente viene opportunamente gestita (self._local_planner.set_speed(speed * 3.6)).

5. **Condizioni di non sorpasso:** Se la corsia di sorpasso non è sicura e l'ostacolo è troppo vicino, l'agente può decidere di non sorpassare, ricorrendo a una frenata d'emergenza (self.emergency_stop()) o attivando il comportamento di inseguimento veicolo (self.car_following_manager(...)) se l'ostacolo fosse dinamico.

Questa capacità aumenta notevolmente l'autonomia e l'efficienza dell'agente in scenari con ostacoli.

## 3.2 Gestione degli ostacoli statici nella corsia adiacente sinistra

Oltre al sorpasso, l'agente è ora in grado di reagire alla presenza di ostacoli statici rilevati nella corsia immediatamente a sinistra. Se static_obstacle_avoid_manager(..., lane="left") rileva un tale ostacolo (stato_ostacolo_laterale_altra_linea in run_step), l'agente adotta una strategia di "mantenimento a destra". Questo viene implementato modificando direttamente un parametro del controllore laterale del pianificatore locale:

```
self._local_planner._vehicle_controller._lat_controller._offset = 0.85
control = self._local_planner.run_step(debug=debug)
```

Listato 3.1: Spostamento laterale per ostacolo a sinistra

Questo offset induce il veicolo a mantenersi leggermente spostato verso destra all'interno della propria corsia, aumentando la distanza laterale dall'ostacolo nella corsia adiacente.

## 3.3 Maggiore consapevolezza dei pedoni

Un altro importante miglioramento riguarda la gestione dei pedoni.
Nel metodo `pedestrian_avoid_manager`, la distanza di rilevamento dei pedoni è stata significativamente aumentata:

```
lista_pedoni = [w for w in lista_pedoni if dist(w) < 45]
```

Listato 3.2: Modifica distanza rilevamento pedoni

Questo incremento da 10 a 45 metri permette all'agente di identificare i pedoni con maggiore anticipo, consentendo reazioni più dolci e sicure, piuttosto che fare affidamento unicamente sulla frenata d'emergenza a breve distanza.

## 3.4 Evoluzione del comportamento in run_step

Il metodo `run_step` rappresenta il cuore del ciclo decisionale dell'agente autonomo, eseguito ad ogni intervallo di simulazione per determinare il comportamento del veicolo. La sua operatività può essere scomposta nelle seguenti fasi principali:

1. **Aggiornamento dello stato e percezione ambientale:**
   - Inizialmente, vengono aggiornate le informazioni di base del veicolo (velocità, limiti, direzione corrente) tramite il metodo `_update_information()`.
   - Successivamente, il modulo di percezione (`self.percezione`) viene aggiornato con la posizione attuale del veicolo, invocando `self.percezione.update_perception_infor`
     Da questo modulo vengono estratti gli attori statici rilevanti (quelli di tipo `"static.prop"`) già prefiltrati, pronti per essere utilizzati dai moduli decisionali specifici.
2. **Adattamento comportamentale dinamico:**
   - Viene introdotta una logica di adattamento della velocità massima in base alle condizioni meteorologiche: la velocità massima consentita per l'agente (`self._behavior.max_speed`) viene ridotta proporzionalmente all'intensità dei depositi di precipitazione (`self._world.get_weather().precipitation_deposits`), promuovendo una guida più sicura in condizioni avverse.
   - Viene gestito un contatore per comportamenti temporizzati, come il `self._behavior.tailga`
3. **Acquisizione delle info contestuali:**
   - L'agente raccoglie informazioni dettagliate sull'ambiente circostante interrogando una serie di moduli "manager" specializzati. Questi forniscono dati su:
     - Segnali di stop: presenza e distanza (`self.stop_sign_manager()`).
     - Pedoni: Rilevamento, distanza e indicazione se si trovano nella stessa corsia del veicolo (`self.pedestrian_avoid_manager()`).
     - Ciclisti/Motociclisti (Biker): Rilevamento, distanza e indicazione se si trovano nella stessa corsia (`self.biker_avoid_manager()`).

  - Ostacoli Statici Frontali: Identificazione nella corsia di marcia corrente, utilizzando i dati forniti dal modulo di percezione (`self.static_obstacle_avoid_manage lane='same', ...`).
  - Veicoli frontali: Rilevamento di altri veicoli che precedono l'agente (`self.collision_and`
  - Veicoli Laterali/Bordo Strada: Identificazione di veicoli fermi o presenti lateralmente (`self.side_obstacle_avoid_manager()`).
  - Ostacoli statici nella corsia sinistra: rilevamento nella corsia adiacente di sinistra (`self.static_obstacle_avoid_manager(..., lane='left', ...)`).
- Vengono inoltre aggiornate informazioni relative alla presenza e alla navigazione all'interno degli incroci (attributi `_in_incrocio`, `_lista_waypoint_incroci`).

4. **Processo decisionale gerarchico:** Il nucleo del metodo è una struttura decisionale gerarchica, realizzata tramite una cascata di istruzioni `if/elif/else`, che stabilisce la priorità delle azioni da intraprendere:

   - *Semafori:* Priorità massima; in caso di semaforo rosso, viene comandata una frenata d'emergenza.
   - *Segnali di Stop:* Se attivo e vicino, l'agente si ferma o adotta un comportamento di car-following adattato; altrimenti, rallenta progressivamente.
   - *Interazione con Pedoni e Ciclisti (Biker):*
     - Pedoni: Frenata d'emergenza se il pedone è troppo vicino e in traiettoria; altrimenti, rallentamento significativo.
     - Ciclisti/Motociclisti: Se il "biker" si muove parallelamente l'agente attua uno scarto laterale. Se invece sta attraversando o la sua traiettoria è conflittuale, il comportamento è simile a quello per i pedoni.
   - *Veicoli frontali:* Attivazione del car-following; frenata d'emergenza se la distanza è critica.
   - *Sorpasso veicoli laterali fermi:* Se rilevato un veicolo fermo lateralmente, l'agente avvia una manovra di sorpasso.
   - *Comportamento agli Incroci:* Logica complessa per la navigazione, includendo rilevamento di altri veicoli, adattamento della velocità, e stima del "commitment" all'interno dell'incrocio. Frenata d'emergenza se l'incrocio non è sicuro.
   - *Sorpasso ostacoli statici frontali:* Se un ostacolo statico blocca la corsia, l'agente tenta una manovra di sorpasso.
   - *Gestione Ostacoli Statici nella Corsia Sinistra:* L'agente si sposta leggermente a destra all'interno della propria corsia.
   - *Comportamento di default (Normale):* In assenza delle condizioni precedenti, l'agente procede normalmente, mantenendo la velocità target (adattata a meteo e limiti) e resettando eventuali offset laterali (tramite una nuova funzione `move_to_center()`).

5. **Esecuzione del Controllo:** Infine, il comando di controllo del veicolo (`carla.VehicleControl`), determinato dalla logica decisionale, viene restituito per essere applicato al veicolo.

## 3.5 Gestione dei veicoli frontali e laterali

L'agente è in grado di rilevare e gestire la presenza di veicoli nelle immediate vicinanze, sia frontalmente nella stessa corsia sia nella corsia adiacente. Nel caso venga rilevato un veicolo frontale (`stato_veicolo` in `run_step`), viene calcolata la distanza reale tenendo conto delle dimensioni di entrambi i veicoli. Se la distanza è inferiore alla soglia di frenata di emergenza (`_behavior.braking_distance`), l'agente esegue una frenata immediata tramite il metodo `emergency_stop()`, per evitare collisioni. In caso contrario, adotta un comportamento di *car-following* regolando la velocità in modo da mantenere una distanza di sicurezza e seguire il veicolo davanti tramite il metodo `car_following_manager()`.

Se viene rilevato un veicolo nella corsia laterale adiacente (`stato_veicolo_laterale`), l'agente valuta la possibilità di effettuare un sorpasso. Calcola la lunghezza complessiva dell'ostacolo laterale e attiva la manovra di sorpasso tramite la funzione `sorpassa()`, che pianifica un percorso adeguato e regola la velocità per completare la manovra in sicurezza.

# Capitolo 4

# Miglioramenti proposti

Il lavoro svolto ha permesso di sviluppare un agente di guida autonoma con capacità significativamente superiori rispetto alla baseline, in particolare per quanto concerne la gestione degli ostacoli statici e l'introduzione di manovre di sorpasso. Tuttavia, come ogni progetto complesso, esistono numerose aree che potrebbero beneficiare di ulteriori affinamenti e sviluppi. In questa sezione, vengono analizzate alcune delle direzioni più promettenti per l'evoluzione futura del sistema.

## 4.1 Perfezionamento della manovra di sorpasso

L'attuale implementazione della manovra di sorpasso, pur funzionale, presenta margini di miglioramento, specialmente in prossimità di elementi stradali complessi come incroci e biforcazioni.
**Pianificazione del percorso di sorpasso:**
Il metodo `_genera_percorso_per_sorpasso` costruisce la traiettoria di sorpasso aggiungendo waypoint sequenziali mediante `next(step_distance)` o `previous(step_distance)`. Questa strategia non include una verifica contestuale per determinare se i waypoint generati ricadano all'interno di un incrocio o lungo una biforcazione stradale. Tale mancanza può condurre alla definizione di percorsi di sorpasso geometricamente validi ma semanticamente scorretti o intrinsecamente pericolosi, qualora la manovra implichi l'attraversamento di un incrocio o l'impegno di una corsia non idonea. Inoltre, l'utilizzo dei metodi `get_left_lane()` e `get_right_lane()` può generare eccezioni qualora le corsie laterali non siano presenti (restituendo `None`), condizione frequente in prossimità di incroci o su strade a corsia unica per senso di marcia. Questa situazione si manifesta criticamente quando un ostacolo è posizionato immediatamente prima di un incrocio e il percorso di sorpasso calcolato interseca l'area dell'incrocio stesso.
Proposte di Miglioramento:

- **Abolizione del sorpasso in prossimità di incroci:** Una strategia prioritaria dovrebbe essere quella di inibire l'inizio di una manovra di sorpasso se l'agente o l'ostacolo si trovano entro una distanza di sicurezza da un incrocio, o se la traiettoria di sorpasso stimata dovesse intersecare l'area dell'incrocio. La manovra potrebbe essere riconsiderata solo dopo aver superato l'incrocio in sicurezza.

- **Analisi dei waypoint futuri:** qualora si decidesse comunque di permettere sorpassi in contesti stradali complessi (scelta generalmente sconsigliata per gli incroci), sarebbe necessario integrare un'analisi topologica più robusta. Prima di generare i

waypoint di sorpasso, si dovrebbe verificare l'attributo `is_junction` del waypoint corrente e di quelli prospettici. Se un waypoint si trova in un incrocio, si dovrebbero analizzare le possibili corsie e direzioni consentite per selezionare una traiettoria che sia coerente con il mantenimento della corsia desiderata post-sorpasso o che garantisca la massima sicurezza, evitando conflitti con altri flussi di traffico.

## 4.2 Gestione di incroci non semaforizzati e rispetto delle precedenze

La gestione degli incroci non regolati da semafori o da segnali di stop rappresenta una delle sfide più importanti. L'agente attuale potrebbe beneficiare enormemente di una logica decisionale più sofisticata in tali contesti.

- **Implementazione di un modulo decisionale per le precedenze:** Sviluppare un modulo dedicato all'analisi dinamica degli incroci non semaforizzati. Tale modulo dovrebbe:
  - **Rilevare e tracciare veicoli convergenti:** Identificare i veicoli che si avvicinano all'incrocio da altre direzioni, stimandone la velocità e la traiettoria probabile.
  - **Interpretare le regole di precedenza:** Implementare una base di conoscenza delle comuni regole di precedenza. Questo richiede la capacità di "leggere" e interpretare la segnaletica orizzontale e verticale.
- **Gestione specifica delle rotatorie:** Le rotatorie richiedono una logica di precedenza particolare (dare precedenza ai veicoli già impegnati nella rotatoria). Si potrebbe sviluppare un comportamento specifico che:
  - Identifichi correttamente l'ingresso in una rotatoria.
  - Monitori il flusso di veicoli all'interno della rotatoria.
  - Selezioni il momento opportuno per l'immissione sicura.
  - Mantenga la corsia corretta all'interno della rotatoria e selezioni l'uscita appropriata in base al path previsto.

## 4.3 Navigazione su strade a più corsie per senso di marcia

Per migliorare la fluidità e l'efficienza della guida su strade con più corsie per ogni direzione di marcia, si potrebbero introdurre le seguenti evoluzioni:

- **Selezione della corsia:** Implementare una logica che permetta all'agente di scegliere la corsia più appropriata non solo per il sorpasso, ma anche in base alla densità del traffico, alla velocità media delle diverse corsie, alla presenza di veicoli lenti, o in preparazione a svolte o uscite imminenti.

## 4.4 Estensione della manovra di sorpasso per veicoli frontali lenti o fermi

Attualmente, la gestione della presenza di un veicolo frontale si limita alla frenata di emergenza o al mantenimento della distanza di sicurezza tramite il `car_following_manager`. Un miglioramento consiste nell'integrare in questo stesso blocco la possibilità di attivare una manovra di sorpasso quando il veicolo davanti è fermo o procede a bassa velocità ma si trova a una distanza sufficiente per consentire un sorpasso sicuro.

In pratica, dopo aver calcolato la distanza e verificato che non si renda necessario un arresto immediato, si dovrebbe controllare la velocità del veicolo frontale e, qualora questa sia inferiore a una soglia critica, generare un percorso di sorpasso simile a quello previsto per i veicoli laterali, coordinando la regolazione della velocità e la pianificazione della traiettoria per effettuare la manovra in sicurezza, evitando situazioni pericolose come un incrocio.

## 4.5 Applicazione del modulo di Perception

Attualmente, il modulo di percezione viene utilizzato solo per rilevare ostacoli statici. Sarebbe utile estenderlo per rilevare anche ostacoli in movimento (dinamici) e integrarlo in tutte le funzioni del codice. Per garantire una migliore compatibilità con le linee guida di Autoware, sarebbe anche necessario riorganizzare e ottimizzare il codice, migliorando la sua struttura e leggibilità.