

Autonomous Vehicle Driving Project
Gruppo 16

Vittorio Fina - 0622701315 - v.fina@studenti.unisa.it
Giovanni Puzo - 0622701109 - g.puzo@studenti.unisa.it
Salvatore Ventre - 0622701343 - s.ventre@studenti.unisa.it

ANNO ACCADEMICO 2020/2021

Contents

1 Executive Sumary	3
1.1 Introduzione	3
1.2 Descrizione del problema	3
1.3 ODD - Operational Design Domain	3
2 Progettazione ed implementazione	5
2.1 Detector di semafori	5
2.1.1 Sensori	5
2.1.2 Stato del semaforo	7
2.2 Gestione degli ostacoli	10
2.2.1 Veicoli	10
2.2.2 Pedoni	11
2.3 Local Planner	12
2.4 Behavioural Planner	12
3 Analisi sperimentale del sistema	14
3.1 Analisi quantitative dei risultati	14
3.2 Analisi qualitative dei risultati	17
3.3 Video Demo	20

1 Executive Sumary

1.1 Introduzione

Il presente elaborato riguarda la descrizione della progettazione e dell'implementazione di un software di guida autonoma da eseguire sul simulatore *CARLA*¹. In questo capitolo saranno elencati i problemi affrontati e le assunzioni fatte in fase di progettazione.

1.2 Descrizione del problema

Il problema affrontato riguarda la realizzazione di un software per rendere un'automobile capace di guidare autonomamente in *CARLA* all'interno della mappa **Town01**, contenuta in */Game/Maps/Town01*.

Il punto di partenza del software é la baseline fornita² composta da un *Motion Planner* ed un *controller* di base, in grado di effettuare una pianificazione di un percorso dati due punti, dichiarati come `PLAYER_START_INDEX` e `DESTINATION_INDEX` [`main.py` linea 44 e 45] e facendo in modo che l'auto possa percorrere la traiettoria richiesta. I moduli aggiunti, richiesti dalle specifiche del progetto, affrontano i seguenti problemi:

- *Evitare le collisioni* con tutti i pedoni e i veicoli, utilizzando i dati *perfetti* dei sensori di *CARLA*.
 - Come da specifica, quindi, le informazioni come posizione, orientamento e velocit degli ostacoli vengono prelevate tramite i metodi messi a disposizione dal simulatore.
- Considerare la *presenza di semafori* all'interno della scena, fermandosi al semaforo rosso e passando o ripartendo solo quando il semaforo é verde. La presenza del semaforo dovr essere rilevata mediante un detector, che fornisce anche lo stato del semaforo, codificato come `[Stop, Go]`.
 - Rilevare lo stato del semaforo é quindi affidato ad un detector³ di semafori che rileva la posizione dello stesso ed il suo stato, distinguendolo tra semaforo rosso e verde. Il semaforo giallo, non considerato dal detector, é stato preso in considerazione in altro modo.

1.3 ODD - Operational Design Domain

In questa sezione vengono definite le circostanze operative trattate dal software progettato e le ipotesi effettuate durante la realizzazione. Come descritto nella Sezione 1.2, il software é stato pensato e realizzato per girare esclusivamente sul simulatore *CARLA*, nella mappa **Town01**, presente in Figura 1. Di seguito vengono quindi riportate le ipotesi fatte per la realizzazione del progetto.

- **Condizioni metereologiche:** si é ritenuto opportuno tenere in considerazione esclusivamente una condizione serena, identificabile in Carla come `CLEARNOON`, [`main.py` linea 80]. Ci ´e dovuto ai vincoli dati dal detector.

¹ Documentazione disponibile al link: <https://carla.org/>

² Scaricabile a questo link: <https://drive.google.com/file/d/1E44ipRKW-qoyArkcaB9NmWzLsuyvRSuN/view?usp=sharing>

³ Disponibile al link: <https://github.com/affinis-lab/traffic-light-detection-module>



Figure 1: Mappa Town01 presente in Carla

- **Velocità massima:** considerata ad un massimo di 6 m/s . La scelta è stata presa tenendo in considerazione il fatto che l'algoritmo di detection dei semafori fornito non è molto accurato, soprattutto al verificarsi di particolari condizioni, come ombra, luce intensa del sole, vegetazione circostante, ecc. Ragion per cui, si è scelta una velocità minore che garantisse un miglior lavoro di rilevamento dei colori del semaforo durante la corsa del veicolo.
- **Modalità grafica del simulatore:** il software è stato progettato a livello EPIC, perché come descritto nella Sezione 2.1, è stato necessario per far performare al meglio il detector.
- **Veicoli in movimento:** in questa categoria vengono considerati i veicoli in moto, sia sulla corsia di percorrenza dell'automobile controllata, sia su altre corsie che procedono in direzione opposta rispetto alla direzione dell'automobile controllata.
- **Veicoli statici:** non sono stati presi in considerazione veicoli non in movimento, cioè parcheggiati lungo la carreggiata.
- **Pedoni in movimento:** in questa categoria vengono considerati i pedoni in movimento che possono intersecare la traiettoria del veicolo controllato. Non vengono gestiti ed evitati pedoni che *collidono con l'automobile ferma* o che attraversano la strada ad una *distanza troppo stretta* dall'auto.
- **Manovre di sorpasso:** non vengono tenute in considerazione manovre di sorpasso ed, in generale, manovre di cambio corsia.
- **Manovre alternative:** in questo caso manovre in *retromarcia* non sono contemplate.
- **Segnaletica luminosa verticale:** è stato considerata l'assenza di segnali di stop e la presenza ad ogni intersezione di semafori ad una distanza di circa *5metri* dall'incrocio, sempre a *destra* della corsia di percorrenza.

2 Progettazione ed implementazione

In questa sezione vengono forniti dettagli riguardanti l'implementazione dei moduli sviluppati e/o modificati rispetto alla baseline fornita per lo svolgimento del progetto, considerando le assunzioni riportate nella Sezione 1.3.

2.1 Detector di semafori

Il `traffic-light-detection-module` utilizzato ed integrato è stato inizialmente allenato sul dataset LISA, contenente circa 5800 immagini e successivamente su circa 1800 immagini catturate dal simulatore Carla in modalità `Epic`. L'allenamento del modello è stato effettuato partendo dai pesi pre-allenati di `YOLOv2-COCO`.

2.1.1 Sensori

Ai fini di poter rilevare i semafori in prossimità degli incroci presenti sul percorso, si è scelto di impiegare *due* camere RGB, da poter agganciare al detector presentato in Sezione 2.1, e *due* camere Depth che rilevassero la posizione del semaforo. Le caratteristiche dei quattro sensori sono presenti nel file `cams.json`, riportato in Figura 2, e riepilogate nella Tabella 1.

Tipo	Posizione	Orientamento	Image Size	Field-of-View
RGB	[1.5, 0, 1.3]	[0, 0, 20]	400 · 400	60°
RGB	[2.0, 0, 1.3]	[0, -6.5, 0]	400 · 400	120°
Depth	[1.5, 0, 1.3]	[0, 0, 20]	400 · 400	60°
Depth	[2.0, 0, 1.3]	[0, -6.5, 0]	400 · 400	120°

Table 1: Specifiche dei sensori impiegati



```
{  "cam_front": {    "position": [2.0, 0, 1.3],    "img_size": [400, 400],    "field_of_view": 120,    "orientation": [0, -6.5, 0]  },  "cam_tl": {    "position": [1.5, 0, 1.3],    "img_size": [400, 400],    "field_of_view": 60,    "orientation": [0, 0, 20]  },  "cam_depth_TL": {    "position": [1.5, 0, 1.3],    "img_size": [400, 400],    "field_of_view": 60,    "orientation": [0, 0, 20]  },  "cam_depth_front": {    "position": [2.0, 0, 1.3],    "img_size": [400, 400],    "field_of_view": 120,    "orientation": [0, -6.5, 0]  }}
```

Figure 2: File di configurazione dei sensori

La **prima camera** è stata aggiunta per avere una visuale ravvicinata dell'incrocio ed iniziare a rilevare i semafori ad una distanza accettabile per permettere al veicolo di riconoscerlo ed eventualmente fermarsi nei pressi dell'incrocio. Questi i motivi per cui si è impiegato un *Field-of-View* piuttosto piccolo e si è deciso di orientare tale camera verso destra, poiché come specificato nella Sezione 1.3 i semafori sono esclusivamente presenti alla destra della corsia di percorrenza.

La **seconda camera** è stata aggiunta per poter avere la possibilità di inquadrare l'incrocio ed il semaforo anche ad una distanza ravvicinata, cioè quando la seconda camera non è in grado più di rilevarlo correttamente. In questo caso si è scelto un *Field-of-View* molto più ampio con un raggio d'azione maggiore, ma capace di inquadrare una distanza minore.

La **terza camera** è stata aggiunta per poter effettuare una stima sulla posizione del semaforo rispetto al nostro veicolo ed avere un punto di riferimento dove poter arrestare il veicolo nella massima sicurezza. Si sono scelti *Field-of-View*, posizionamento ed orientamento uguali a quelli della *prima camera* per compiere una stima a partire dalla detection fatta sull'immagine da essa catturata.

La **quarta camera** è stata aggiunta per avere una ridondanza sulla stima della posizione del semaforo in determinate situazioni. Si sono scelti *Field-of-View*, posizionamento ed orientamento uguali a quelli della *seconda camera*.

Per completezza e chiarezza, viene di seguito riportata la posizione delle camere rispetto al veicolo controllato ed in più un'anteprima di visualizzazione dell'esecuzione sul software *CARLA*. È chiaro come la scelta delle camere risulti opportuna ed efficiente, in particolar modo nei pressi dell'incrocio, come si può notare in Figura 3



Figure 3: Anteprima delle camere RGB a bordo dell'auto



Figure 4: Anteprima di una delle camere Depth a bordo dell'auto

Lo scenario ricercato che ha portato al posizionamento dei sensori in questo modo è mostrato in Figura 5. Dalla rappresentazione si evidenzia come la posizione e le caratteristiche delle camere sono state pensate di modo che il semaforo potesse essere ripreso anche dalla *seconda camera* nel momento in cui l'automobile controllata si fosse fermata troppo vicina all'incrocio. Va sottolineato che, come detto in precedenza, i raggi di azione delle due *depth camera* si sovrappongono a quelle delle *camere RGB* in quanto sono stati scelti gli stessi parametri di configurazione.

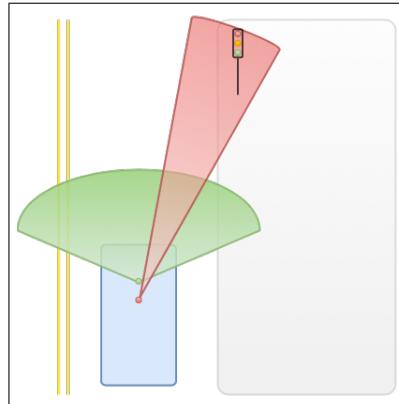


Figure 5: Rappresentazione dello scenario in Carla

2.1.2 Stato del semaforo

Rilevare lo stato del semaforo è necessario per intervenire sul comportamento dell'auto fermandosi agli incroci, nel caso di semaforo rosso, oppure proseguire, nel caso di semaforo verde. In Carla gli stati del semaforo sono tre, riportati con i colori [rosso, giallo, verde]. Tuttavia, il detector descritto nella Sezione 2.1, come riportato nella documentazione relativa, non è capace di classificare il *colore giallo*.

Questi vincoli hanno richiesto interventi stringenti per la codifica degli stati e la confidenza di classificazione degli stessi. A tal proposito si è scelto di codificare gli stati del semaforo, tenendo conto di questi vincoli, in questo modo:

- **Verde:** se il detector rileva un semaforo verde, lo stato verrà codificato come `GO`.
- **Rosso:** se il detector rileva un semaforo rosso, lo stato verrà codificato come `STOP`.
- **Assente:** se il detector non rileva alcun semaforo, lo stato in questo caso verrà codificato come `NO_TL`, cioè '*no traffic-light*'.

Intervenire sulla confidenza di classificazione del detector ha richiesto varie prove sperimentali, dettando una soglia sia per la classificazione di un semaforo verde che per la classificazione di un semaforo rosso. Dal momento che le performance del detector non sono continue, ma variabili, si è deciso quindi di porre un filtro su più fasi alle detection. Una possibile conferma a quanto detto è presente in Figura 3, dove è visibile una detection di un semaforo rosso, ma con un'accuracy del 60% sulla camera di destra. Per diminuire quanto più possibile l'induzione in tutte le detection, il detector fornito scarta in partenza le bounding boxes rilevate con un'accuratezza minore di 0.2 [`traffic_light_detection_module/config.json` linea 12]; a questo punto, tra tutte le boxes rilevate, viene estratta quella con accuracy più alta [`traffic_light_detection_module/postprocessing.py` linea 69-70].

Per stabilizzare l'output del detector e modificare effettivamente lo stato del semaforo si è dunque pensato di stabilizzare l'output per un certo numero di frame, così da essere sicuri che lo stato predetto sia quello effettivo. A tal proposito, dopo varie prove sperimentali, si è scelto di cercare di stabilizzare il numero di rilevazioni prima di cambiare stato del semaforo secondo i seguenti criteri:

- **Semaforo Rosso:** la detection del semaforo rosso deve essere stabile per 10 *frame*.
- **Semaforo Verde o Assente:** la detection in questo caso deve essere stabile per 20 *frame*.
- **Soglia di indecisione:** se per 60 *frame* la detection non è stabile, lo stato del semaforo è codificato come `NO_TL`.

È evidente che impostare un numero di frame minore per un certo stato vuol dire un cambiamento di stato più veloce. L'idea dietro questi valori è che, in prossimità di un incrocio con semaforo rosso, si possa gestire con più reattività il cambiamento di stato per avere una maggiore affidabilità nel fermarsi effettivamente all'incrocio. La ragione per cui i frame per gli stati `GO` e `NO_TL` sono raddoppiati rispetto allo stato di `STOP` si basa sul fatto che la decisione deve essere presa in sicurezza, poiché si è preferito decidere di fermarsi più rapidamente che proseguire o ripartire. La *soglia di indecisione* si è rivelata necessaria per fronteggiare le performance non costanti del detector. In tal caso si affronta una situazione per cui lo stato non è stabile per sufficienti frame ed il detector non rileva lo stato in maniera continua e precisa, cambiando la sua predizione troppo spesso. È ragionevole pensare di impostare una soglia maggiore e codificare tale indecisione come stato `NO_TL` per permettere al veicolo di non essere bloccato da questi eventi. In Tabella 2 si nota come in due frame successivi e con lo stesso stato del semaforo possa cambiare la classificazione del detector.

Classificazione Corretta	Classificazione Errata
	

Table 2: Confidenza delle classificazioni del detector

Per validare l'applicazione di queste soglie sono stati percorsi vari scenari ed è stato analizzato il comportamento del detector sulle immagini in input: si è dunque notato che riesce a classificare con una confidenza maggiore il semaforo rosso ed è più indeciso sul semaforo verde. Questa ipotesi ha portato ad analizzare le bounding boxes predette seguendo i valori presentati nella Tabella 3.

Camera	Treshold	Riferimento al codice
Camera_Front	0.20	<code>traffic_light_detector.py linea 26</code>
Camera_TL	0.35	<code>traffic_light_detector.py linea 27</code>

Table 3: Soglie impostate al detector

L'obiettivo che ci si è posti per determinare lo stato del semaforo è quello di recuperare la classificazione del semaforo più vicino all'automobile e stabilizzare l'output per un certo numero di frame, basandosi sui dati provenienti dalle due camere. L'aggiornamento dello stato è stato regolato in maniera euristica: ad ogni frame, se il detector rileva sull'immagine catturata da **CAMERA_TL** un certo stato del semaforo con una confidenza maggiore di 0.35, allora sarà adottato quello come stato corrente del semaforo; altrimenti si va a correggere tale predizione guardando alla detection fatta sull'immagine di **CAMERA_FRONT**: se essa ha una confidenza maggiore di 0.20, che per quanto detto precedentemente significa che il detector ha rilevato il semaforo sulla seconda camera, allora sarà adottato lo stato associato a tale detection, in caso contrario si deciderà per lo stato **NO_TL** in quanto le detections non hanno raggiunto una confidenza tale da poter scegliere uno stato [`traffic_light_detector.py linee 145-154`]. Una volta stabilito lo stato del semaforo nel frame corrente, si va a confrontarlo con quello del frame precedente per determinare se è avvenuto un cambiamento di stato o meno e aggiornare i relativi contatori in modo che, una volta arrivati al numero di frame minimo per dichiarare uno stato del semaforo stabile, si possa settare un attributo che sarà usato dal behavioural planner per prendere le relative decisioni di path planning [`traffic_light_detector.py linee 164-184`].

Come già accennato nella Sezione 1.2, il detector non è capace di rilevare il semaforo giallo, che può interpretare come rosso o verde. Questo stato è tuttavia presente in Carla ed è stato sperimentalmente provato che il detector tende a classificare questo stato *molto più frequentemente come rosso* che come verde, quindi ci si aspetta che il veicolo si possa fermare all'incrocio. È rilevante notare che, essendo stato il detector addestrato in modalità grafica EPIC, per un più corretto funzionamento del software e del detector stesso, è consigliabile **avviare la simulazione in modalità EPIC**. È stato notato infatti, che in modalità **LOW** le classificazioni sono

molto più discontinue e imprecise, oltre al fatto che è molto frequente la classificazione del sole come segnale di GO.

2.2 Gestione degli ostacoli

La gestione degli ostacoli ha richiesto delle analisi preliminari, utili a capire in che modo coordinare le informazioni provenienti da *CARLA* sia per i pedoni che per i veicoli. Infatti, tutte le informazioni sono state estrapolate direttamente dal simulatore: orientamento, posizione e velocità sono state ricavate utilizzando i dati *perfetti* dei sensori di *CARLA* e quindi affetti il meno possibile da errori. Per poter essere utilizzati per il nostro scopo, ovvero evitare il più possibile ogni tipo di collisione, sono stati trasformati nella terna di riferimento del nostro veicolo e, successivamente, sono state calcolate e gestite le possibili intersezioni con la traiettoria corrente del veicolo. Nelle successive sezioni verrà affrontato tale argomento nel dettaglio.

2.2.1 Veicoli

La gestione dei veicoli è stata realizzata esaminando tutti i veicoli generati dal simulatore *CARLA* e presenti all'interno della mappa: per ognuno vengono calcolate la posizione e l'orientamento, opportunamente trasformate nella terna di riferimento del nostro veicolo ed, infine, viene effettuata una selezione tra veicoli che precedono la nostra auto (Scenario 1) e veicoli che viaggiano nella corsia opposta o che rappresentano un ostacolo da superare durante il tragitto dell'auto (Scenario 2). Tutto questo meccanismo ha come scopo quello di garantire che non avvenga nessun tipo di collisione con qualsiasi tipo di veicolo. (*Per veicolo viene intesa un'autovettura, un ciclomotore o motocicletta e ciclista su bici, come riportato in Sezione 1.3*).

- **Scenario 1: Veicoli che precedono la nostra auto.** I veicoli che appartengono a questa categoria sono tutti quelli che si trovano davanti al nostro veicolo, nella stessa corsia e hanno un orientamento simile, in particolare è stata considerata una differenza massima di orientamento - *Yaw angle* - pari a *30 gradi*; si mantiene un riferimento al veicolo che è più vicino rispetto all'asse del movimento del nostro veicolo, oltre a rispettare le condizioni precedentemente descritte [`main.py` linee 910-914]. In tale situazione la velocità del nostro veicolo viene settata uguale al veicolo che ci precede ed utilizzata durante la creazione del profilo di velocità [`velocity_planner.py` linea 131].
- **Scenario 2: Tutti i restanti veicoli.** A questa categoria appartengono tutti i veicoli che non fanno parte dello scenario precedente e che si trovano ad una distanza, denominata *LOOKAHEAD_VEHICLE*, inferiore ai *10 metri*. Per tutti questi veicoli, viene effettuato un calcolo sulla loro traiettoria futura e, se essa interseca quella del nostro veicolo, quest'ultimo si arresta per evitare collisioni. In caso contrario, i veicoli verranno gestiti come un ostacolo da evitare durante la selezione del *best path*, all'interno del *Local Planner* [`main.py` linee 917-927]. In particolare, per valutare la possibile intersezione tra le traiettorie, lavorando nel vehicle frame, il nostro veicolo viene proiettato di 10 metri in avanti (sull'asse x) mentre, per il veicolo che stiamo analizzando, si considera come punto iniziale della traiettoria la sua posizione attuale traslata indietro di 5 metri sull'asse x e come punto finale la posizione attuale proiettata di *speed* metri in avanti (sempre sull'asse x), dove *speed* è la velocità che esso ha nel frame corrente [`utils.py` linee 179-194]; si è scelto di traslare il punto iniziale di *-5 metri* per una maggiore sicurezza nella valutazione di una possibile collisione.

2.2.2 Pedoni

Analogamente a quanto avviene per i veicoli, vengono esaminati tutti i pedoni presenti nella mappa di *CARLA* e viene effettuato un controllo in base alla loro distanza dal nostro veicolo: questa distanza prende il nome di **LOOKAHEAD_PEDESTRIAN** ed è settata a *10 metri*. Tutti i pedoni che hanno una distanza inferiore ad essa vengono selezionati e, successivamente, vengono calcolate le loro future traiettorie per verificare se intersecano o meno la traiettoria corrente del nostro veicolo. Sulla base del risultato di tale controllo, possiamo distinguere due scenari: pedoni che intersecano la traiettoria della nostra auto e pedoni che non hanno intersezione con la nostra auto.

La scelta di utilizzare una distanza di *lookahead* pari a 10 metri è dettata dal fatto che la velocità massima alla quale può viaggiare il nostro veicolo in *CARLA* è stata impostata a 6 m/s . Essendo settata di default una decelerazione pari a $-2,5 \text{ m/s}^2$, lo spazio necessario per arrestare il veicolo, in condizioni ideali, è di 7,2 metri. Per questo motivo, la scelta di 10 metri, garantisce uno spazio di frenata adeguato, considerando anche che la funzione di proiezione futura delle traiettorie `check_obstacle_future_intersection()`, calcola la proiezione dell'ostacolo basandosi su una velocità di 10 m/s per metterci in condizioni di maggiore sicurezza; come punto iniziale della traiettorie si è fatta una scelta analoga al caso dei veicoli riportato nella Sezione 2.2.1.

- **Scenario 1: Pedoni con intersezione.** Nel caso in cui la traiettoria di un pedone intersechi quella del nostro veicolo [`main.py` linee 939-947], quest'ultimo effettua una frenata di emergenza per evitare di collidere con il pedone. In questo scenario, fanno parte tutti quei pedoni che si apprestano ad attraversare la strada o che sono già all'interno della carreggiata. Il veicolo riprenderà la sua manovra quando il pedone avrà liberato la traiettoria di manovra del veicolo.
- **Scenario 2: Pedoni senza intersezione.** I pedoni che appartengono a questa categoria sono tutti quelli che non hanno una traiettoria che in proiezione futura possa intersecare quella del veicolo da noi controllato (ad esempio i pedoni che camminano parallelamente al nostro veicolo), sempre all'interno della distanza **LOOKAHEAD_PEDESTRIAN**. In questo scenario, i pedoni vengono considerati come ostacoli da essere evitati durante la selezione del *best path*, all'interno del *Local Planner* [`main.py` linea 950].

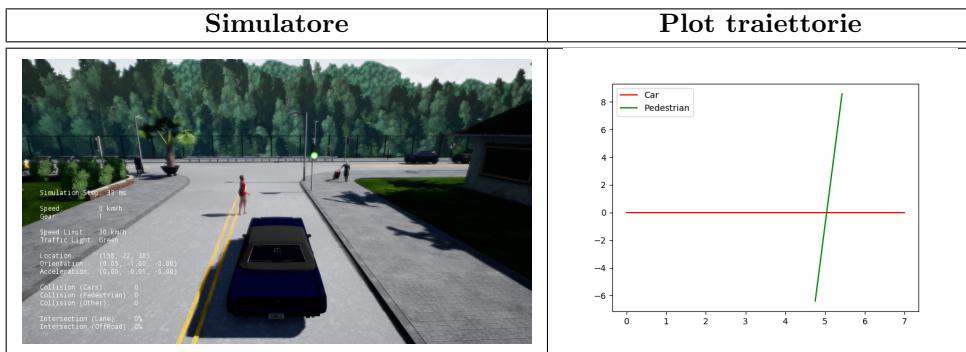


Table 4: Intersezione delle traiettorie durante uno scenario in Carla.

In chiusura, per chiarire meglio quanto spiegato in precedenza, la Tabella 4 mostra un esempio grafico di valutazione della possibile collisione con un pedone.

2.3 Local Planner

Rispetto al codice di base fornito, si è scelto di modificare il *Local Planner* aggiungendo la gestione delle frenate di emergenza dovute ad ostacoli di cui si è menzionato nella sezione 2.2. La modifica ha riguardato l'inserimento di una frenata con un profilo di velocità finale nulla, permettendo di arrestare il nostro veicolo nel minor tempo possibile [`velocity_planner.py` linea 136]. E' stato modificato il `send_control_command()` in modo tale da inviare un comando di massima frenata - `cmd_brake = 1.0` - nei casi di emergenza descritti nella sezione 2.2 [`main.py` linea 1147].

2.4 Behavioural Planner

Data la baseline fornita e per lo scopo del progetto, è stato necessario intervenire in una correzione del Behavioural Planner, apportando modifiche alla macchina a stati finiti già fornita di modo che il comportamento potesse essere adattato al compito da svolgere. In Figura 6 è possibile vedere l'automa a stati finiti impiegato nel sistema progettato per il comportamento atteso dell'automobile controllata.

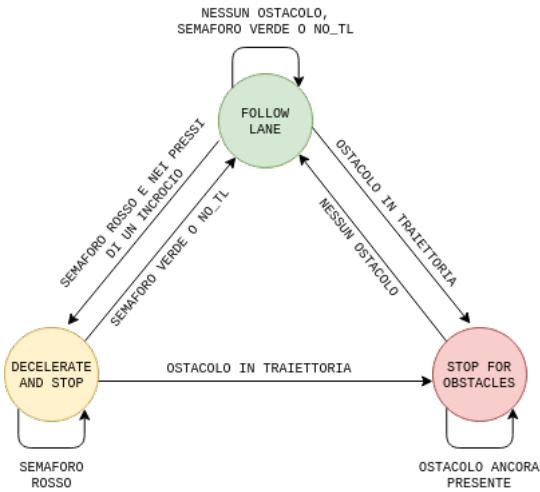


Figure 6: Automa a stati finiti del Behavioural Planner

La gestione dei semafori prevede che il Behavioural Planner conosca lo stato del semaforo e la sua posizione, in modo da poter valutare se e dove far fermare il veicolo. Per quanto riguarda il primo aspetto, dopo aver analizzato le detections fatte al frame corrente come descritto nella Sezione 2.1.2, viene modificato un attributo dello stesso Behavioural Planner indicante l'ultimo stato stabile rilevato dal detector e il Behavioural Planner, dipendentemente da questo, saprà se l'automobile deve fermarsi all'incrocio o può proseguire [`main.py` linea 974]. Per quanto riguarda la posizione del semaforo, viene combinata l'informazione sulla box rilevata dal detector con quella della distanza proveniente dalla depth camera: in particolare, viene recuperato il pixel della box individuata dal detector che si trova a minima distanza dal nostro veicolo e successivamente viene proiettato nel vehicle frame tramite le formule di *Camera Projective Geometry* [`get_t1_pos()` in `behavioural_planner.py` linea 130]. Così facendo, se lo stato del semaforo è `STOP`, viene individuato come goal il waypoint a massimo *4 metri* di distanza dalla posizione stimata del semaforo [`behavioural_planner.py` linea 287].

Naturalmente, viene considerata la box associata all'immagine su cui il detector ha avuto la confidenza necessaria al superamento delle soglie prefissate indicate in Tabella 3 e di conseguenza viene considerata la corrispettiva *depth image* per la stima della distanza dal semaforo [behavioural_planner.py linee 139-144].

Questo processo è eseguito ad ogni iterazione nello stato di FOLLOW_LANE che, in caso di semaforo rosso, aggiorna il goal del Behavioural Planner con quello restituito dalla gestione appena citata del semaforo, passando dunque ad uno stato di DECELERATE_AND_STOP (in cui permarrà finché il semaforo non diventa verde); in caso di semaforo verde o non rilevato, prosegue con un normale aggiornamento del goal con il waypoint più lontano entro però la distanza *lookahead* del BP.

È stato poi aggiunto un terzo stato, STOP_FOR_OBSTACLES, come è possibile vedere in Figura 6, che prevede la gestione di scenari di emergenza con frenate tempestive per evitare situazioni di pericolo nel caso di incrocio di traiettorie con pedoni o veicoli; tale controllo che viene fatto sia nello stato di FOLLOW_LANE che di DECELERATE_AND_STOP. Non appena l'ostacolo non sarà più presente, si ripasserà nello stato di FOLLOW_LANE.

3 Analisi sperimentale del sistema

Le analisi sono state utili per capire quale direzione seguire durante lo sviluppo dell'intero progetto, in quanto la gestione delle collisioni e dei semafori ha richiesto uno studio approfondito al fine di ottenere un risultato adeguato.

3.1 Analisi quantitative dei risultati

Le analisi quantitative sono state realizzate cercando di simulare tutti i possibili scenari nei quali entrassero in gioco semafori, pedoni e veicoli, verificando la corretta gestione di queste ambientazioni. Nel dettaglio i test effettuati sono stati:

- *Test 1: verifica settaggio posizione sensori.* Il primo test è stato effettuato per verificare se la posizione dei sensori fosse adeguata per rilevare lo stato del semaforo e la relativa distanza. In alcuni incroci affrontati il posizionamento e l'orientamento delle camere non permetteva una giusta detection del semaforo, dovuto alla diversa disposizione dei questi lungo i diversi incroci. Nei successivi test è stato perfezionato l'orientamento e il relativo *Field-of-View* di ogni sensore, riassunti in Tabella 1.
 - Indice di partenza: 7
 - Indice di arrivo: 15
 - Numero veicoli: 30
 - Numero pedoni: 30
- *Test 2: verifica della corretta gestione dei semafori.* Questo test è stato effettuato per verificare la gestione dei semafori e la valida implementazione dell'algoritmo basato sul detector. Non sono stati inseriti ne veicoli e ne pedoni. Il test è andato a buon fine in quanto i due incroci sono stati affrontati in maniera corretta, fermandosi alla presenza di un semaforo rosso e ripartendo, successivamente, al verde.
 - Indice di partenza: 16
 - Indice di arrivo: 106
 - Numero veicoli: 0
 - Numero pedoni: 0
- *Test 3: verifica della gestione dei semafori in presenza di veicoli che precedono l'automobile controllata.* In questo test si è partiti dalla configurazione del test precedente, aggiungendo la presenza di veicoli. L'obiettivo è quello di verificare il comportamento del veicolo nei pressi dell'incrocio in presenza di altri veicoli che ci precedono. Il test ha dato esito positivo in quanto il veicolo ha fermato la propria corsa prima di un altro veicolo, il quale era in attesa a causa di un semaforo rosso. Seppur il detector stesse rilevando il segnale di **STOP**, la nostra auto non si ferma vicino all'incrocio, bensì prima del veicolo (fermo) da seguire.
 - Indice di partenza: 16
 - Indice di arrivo: 106
 - Numero veicoli: 200
 - Numero pedoni: 0

- *Test 4: verifica dell'accuratezza dell'algoritmo di detection.* In questo test è stato verificato il funzionamento del detector in presenza della luce del sole (all'interno delle *camere RGB*), per verificare se ne compromettesse il corretto rilevamento. Il sole sembra creare piccole difficoltà al detector, che però riesce nel suo compito.

- Indice di partenza: 7
- Indice di arrivo: 15
- Numero veicoli: 0
- Numero pedoni: 0

oppure

- Indice di partenza: 131
- Indice di arrivo: 51
- Numero veicoli: 0
- Numero pedoni: 0

- *Test 5: verifica dell'algoritmo implementato per la Depth Camera.* Si è reso necessario realizzare un test per verificare il corretto funzionamento delle due *depth camera*: in particolare, è stato utile verificare quale delle due camere entrava in azione quando era richiesto calcolare la posizione del semaforo. È emerso che il lavoro viene *principalmente* svolto dalla *camera RGB* di destra (vedi Sezione 2.1.2), di conseguenza viene considerata la corrispettiva *depth image* di destra per la stima della distanza dal semaforo. Abbiamo comunque deciso di continuare ad usare anche la *Camera Depth* di sinistra per coprire quei pochi casi in cui viene richiesta.

- Indice di partenza: 16
- Indice di arrivo: 106
- Numero veicoli: 30
- Numero pedoni: 30

- *Test 6: 1^a verifica della gestione dei pedoni.* E' stato effettuato un primo test sulla gestione dei pedoni per verificare il corretto funzionamento dell'algoritmo implementato. L'esito è stato negativo in quanto il veicolo non ha effettuato la frenata di emergenza in prossimità del pedone. Approfondendo l'analisi, è emerso che il problema riguardava l'errata stima della distanza LOOKAHEAD_PEDESTRIAN e la difettosa proiezione delle traiettorie da parte della funzione `check_obstacle_future_intersection()`.

- Indice di partenza: 10
- Indice di arrivo: 106
- Numero veicoli: 0
- Numero pedoni: 150

- *Test 7: 2^a verifica della gestione dei pedoni.* Nel secondo test sono state modificate sia la distanza LOOKAHEAD_PEDESTRIAN e sia la funzione di proiezione delle traiettorie. In questo caso il veicolo si è fermato ad una distanza molto piccola dal pedone, mentre in un successivo punto, il veicolo non ha arrestato tempestivamente la sua corsa, investendo il pedone. L'esito non può ritenersi positivo.

- Indice di partenza: 10
 - Indice di arrivo: 106
 - Numero veicoli: 0
 - Numero pedoni: 150
- *Test 8: 3^a verifica della gestione dei pedoni.* Nell'effettuare questo test è stato aggiunto un nuovo parametro nella funzione `check_obstacle_future_intersection()` che tenesse conto della velocità dei pedoni. Settando una velocità di proiezione del pedone molto alta, è stato ottenuto un notevole miglioramento nella complessiva gestione delle collisioni con i pedoni. Applicando questa modifica, infatti, ha visto un riscontro positivo sul test che precedentemente è risultato negativo.
 - Indice di partenza: 10
 - Indice di arrivo: 106
 - Numero veicoli: 0
 - Numero pedoni: 150
 - *Test 9: verifica di uno scenario completo.* Questo test è stato realizzato per verificare il comportamento del veicolo in presenza sia di pedoni che di altri veicoli. Il test ha dato esito positivo, in quanto il veicolo si è fermato dietro un altro veicolo che era in attesa del semaforo verde all'incrocio. Durante l'attesa è stato rilevato un pedone attraversare la strada (tra il nostro veicolo e quello che ci precedeva) e nonostante il semaforo fosse verde, e correttamente rilevato, l'auto è ripartita solo dopo aver atteso il passaggio completo del pedone. Nel successivo incrocio, il veicolo ha arrestato la sua corsa in presenza di semaforo rosso.
 - Indice di partenza: 55
 - Indice di arrivo: 120
 - Numero veicoli: 150
 - Numero pedoni: 150
 - *Test 10: verifica di uno scenario completo critico.* Questo test è stato effettuato con i parametri del test precedente, ma raddoppiando il numero di pedoni e veicoli. Inoltre nel tratto iniziale si ripresenta un problema già descritto in precedenza, ovvero la presenza del sole a disturbare il lavoro del detector. A causa della presenza di tutti questi fattori, questo scenario rappresenta una situazione critica. La presenza del rettilineo iniziale permette al veicolo di raggiungere la sua velocità massima (6 m/s) e di arrestare in maniera tempestiva la sua corsa non appena un pedone invade la corsia di marcia. Dopo aver gestito questa manovra, il veicolo si appresta ad affrontare l'incrocio: la presenza del semaforo rosso viene rilevata e gestita, ma per ragioni dovute al detector, il veicolo riparte in regime di semaforo rosso avendo rilevato erroneamente il verde. L'incrocio viene affrontato dal veicolo in un tempo molto lungo, rallentato dalla forte presenza di pedoni che attraversano la strada, inoltre è possibile notare anche un comportamento irregolare del veicolo che verrà presentato nello *Scenario 3* delle analisi qualitative. Essendosi immesso nell'incrocio con il semaforo rosso, e avendo una forte presenza di pedoni nei pressi di quest'ultimo, il veicolo ferma il normale flusso del traffico. Solo dopo qualche minuto di attesa, il veicolo chiude la sua manovra, supera l'incrocio e prosegue.

- Indice di partenza: 7
- Indice di arrivo: 15
- Numero veicoli: 300
- Numero pedoni: 300

Va sottolineato che, siccome per realizzare i test è opportuno inserire gli indici di partenza e di arrivo, questi non sempre coincidevano con lo scenario immaginato. Durante i tentativi compiuti per scegliere gli indici opportuni, sono stati usati gli scenari relativi caricati per eseguire dei piccoli test ed ottimizzare i tempi.

3.2 Analisi qualitative dei risultati

In questa sezione vengono illustrate le analisi su diversi casi d'uso che dimostrano il funzionamento del sistema in alcuni scenari critici. In particolare:

- *Scenario 1: gestione del semaforo in condizioni limite.* In questo scenario sono presenti due semafori: il primo viene gestito in maniera corretta, mentre il secondo, complice il cambio repentino di stato all'ultimo secondo, crea problemi all'algoritmo di detection e non permette una adeguata gestione dell'incrocio, come mostrato in Figura 7. La causa è da attribuire alla scelta di dover stabilizzare l'uscita dello stato del semaforo da parte dell'algoritmo implementato. Questo comporta rilevare lo stesso stato del semaforo per un determinato numero di frame che, in questo caso, si dimostrano essere pochi per una corretta detection.



Figure 7: Ingresso intersezione con semaforo rosso.

- *Scenario 2: detection del semaforo in condizioni critiche.* In questo scenario si vuole mettere in evidenza la difficoltà del rilevamento del semaforo in particolari condizioni. Infatti, come è visibile dalla Figura 8, la detection del semaforo della camera RGB a destra viene disturbata dalla presenza di oggetti sullo sfondo. Per questo motivo, si è pensato di ovviare al problema inserendo una camera RGB (a sinistra) che abbia un *F-o-V* più ampio e riesca a rilevare il semaforo anche a distanza molto basse.



Figure 8: Rilevamento del semaforo con disturbo.

- *Scenario 3: gestione dei pedoni in condizioni limite.* E' stato notato un comportamento irregolare del veicolo durante la manovra di svolta all'interno di un incrocio. Questo comportamento è dovuto alla presenza del pedone nella direzione frontale del veicolo durante tale manovra (Figura 9). Infatti, il veicolo rileva la presenza del pedone e attua le relativa gestione in base all'implementazione dell'algoritmo di controllo delle collisioni. Tale algoritmo, come descritto nelle sezioni precedenti, effettua un controllo sui pedoni che si trovano all'interno di una certa distanza LOOKAHEAD_PEDESTRIAN (10 m), e di fatti rileva quel pedone come *Pedone con intersezione*. Questo comporta che il veicolo effettui degli "accelera-frena" continua fino a quando non termina la manovra di svolta o il pedone libera la traiettoria futura del veicolo. Si potrebbe pensare di risolvere questo problema con una distanza LOOKAHEAD_PEDESTRIAN minore, ma questo comporterebbe rilevare in ritardo i pedoni durante una corsa in rettilineo a piena velocità, con conseguenze gravi. La scelta di una distanza pari a 10 m sembra il giusto *trade-off* per garantire una adeguata gestione in tutte le circostanze.



Figure 9: Intersezione con gestione del pedone.

- *Scenario 4: detection falsata dagli oggetti ambientali della mappa.* In questo scenario viene riportato ciò che è stato descritto nel *Test 4*. Infatti, come mostrato nella Figura 10, il detector rileva il sole come segnale verde [GO] del semaforo. Questa criticità sembrerebbe riguardare solo la *camera RGB* di sinistra caratterizzata da una *F-o-V* più ampia e posizionata più centralmente, quindi in direzione del sole. Siccome nel nostro caso la detection viene effettuata *principalmente* dalla *camera RGB* di destra (vedi Sezione 2.1.2), il problema risulta essere limitatamente risolto.

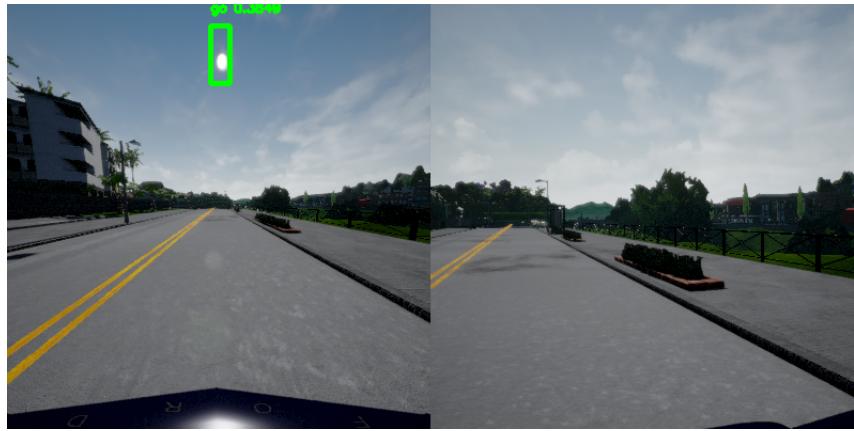


Figure 10: Esempio di detection errata.

3.3 Video Demo

Sono stati realizzati dei video dimostrativi sul comportamento del veicolo in tre situazioni:

- **Video 1:** Gestione collisioni con altri veicoli.

Link: [GestioneCollisioniVeicoli.mp4](#)

- **Video 2:** Gestione collisioni con pedoni.

Link: [GestionePedoni.mp4](#)

- **Video 3:** Stop al semaforo rosso e passaggio con semaforo verde.

Link: [GestioneSemaforo.mp4](#)