Università degli studi di Genova

Scuola Politecnica

Dipartimento di Ingegneria Navale, Elettrica, Elettronica e delle Telecomunicazioni (DITEN)

TESI DI LAUREA MAGISTRALE
IN
INGEGNERIA ELETTRONICA

Progettazione e realizzazione di uno strumento per la generazione di dataset sintetici per la guida autonoma

Relatori:

Prof. Francesco Bellotti
Prof. Riccardo Berta

Candidato:

Taverna Bruno Marco

Dicembre 2021

Abstract

The development of automatic driving systems requires operation and verification in various driving scenarios. A driving scenario is characterized by the different maneuvers of multiple traffic entities and is described by means of formats that are being standardized at an industrial level, such as for example Open Scenario. To carry out the recognition of scenarios (in real-time or off-line), it is necessary to use Machine Learning models trained with adequate datasets, but to date there is no public availability of datasets labeled with automotive driving scenarios.
This thesis is part of the development of a synthetic dataset generation system, based on the Carla simulator for autonomous driving, containing video clips labeled with driving scenarios. In particular, the thesis developed two new classes of scenarios (Traffic and Overtake) and created a new traffic generation model, which was also applied to the five pre-existing scenario types. The model is based on an algorithm that allows to generate the vehicles in the 3D environment on the whole road network (and not in the few predefined spawn-points in Carla's maps) and exploiting the context information (for example, the number and the direction of the lanes, the presence of other objects). This fine-grained generation technique has made it possible to define various parameters whose values (or, more frequently, ranges of values) can be specified by the user in the generation script of the dataset generator.
This allowed both to generate many more samples in the dataset, and to reduce conflict situations by 70%, which previously led to having to manually discard 50% of the video clips produced by the system.
An experimental test has shown that the overall system is capable of generating about 800 video clips (lasting 4 seconds) per day, with a variance that depends on the complexity of the specified scenario (in particular the number of vehicles involved), which is a considerable speed up wrt manual specification. At this time frame, however, it is necessary to add the time necessary for manual inspection in order to discard videos that do not meet the nominal specifications (about 15%). As an example, a dataset of 4,200 4-second videos and one of 3,000 2-second videos was created (in particular to allow a faster response from the system for only those types of scenarios that are actually recognizable in this shorter timeframe), for a total of 7.200 videos, recorded even with different weather conditions. The dataset has been successfully employed for the training and verification of two types of convolutional neural networks, which have achieved high recognition rates of the synthetic scenarios produced.

Sommario

Lo sviluppo di sistemi di guida automatica richiede il funzionamento e la verifica della loro operatività in vari scenari di guida. Uno scenario di guida è caratterizzato dalle diverse manovre di molteplici entità di traffico e viene descritto per mezzo di formati che si stanno standardizzando a livello industriale, quale per esempio Open Scenario. Per effettuare il riconoscimento degli scenari (in real-time o off-line), è necessario l'utilizzo di modelli di Machine Learning addestrati con dataset adeguati, ma ad oggi non sono disponibili pubblicamente dataset etichettati con scenari di guida automobilistica.
La presente tesi rientra nello sviluppo di un sistema di generazione di dataset sintetici, basato sul simulatore Carla per la guida autonoma, contenenti videoclip etichettati con gli scenari di guida. In particolare, la tesi ha sviluppato due nuove classi di scenari (Traffic e Overtake) e ha realizzato un nuovo modello di generazione del traffico, che è stato applicato anche ai cinque scenari pre-esistenti. Il modello si basa su un algoritmo che permette di generare i veicoli nell'ambiente 3D su tutto il reticolo stradale (e non nei pochi pochi spawn-point predefiniti nelle mappe di Carla) e sfruttando l'informazione del contesto (ad esempio, il numero e la direzione delle corsie, la presenza di altri oggetti). Questa generazione a grana fine ha reso possibile la definizione di vari parametri i cui valori (o, più frequentemente, range di valori) possono essere specificati dall'utente nello script di configurazione del generatore di dataset.
Questo ha permesso sia di generare molti più campioni nel dataset, sia di diminuire del 70% le situazioni di conflitto, che portavano a dover scartare manualmente il 50% dei video-clip prodotti dal sistema.
Una verifica sperimentale ha dimostrato che il sistema complessivo è in grado di generare circa 800 videoclip (della durata di 4 secondi) al giorno, con una varianza che dipende dalla complessità degli stessi (in particolare il numero di veicoli coinvolti), che è una velocizzazione significativa rispetto alla specifica manuale. A questo tempo bisogna però aggiungere quello necessario per l'ispezione manuale onde scartare video che non rispettano le specifiche nominali (circa 15%). A titolo di esempio, si è realizzato un dataset di 4.200 video da 4 secondi e uno di 3.000 video da 2 secondi (in particolare per permettere una risposta più rapida da parte del sistema per soli quei tipi di scenario che siano effettivamente riconoscibili in questo tempo più breve), per un totale di 7.200 video, registrati anche con differenti condizioni atmosferiche. Il dataset è stato utilizzato con successo per l'addestramento e la verifica di due tipi di reti neurali convoluzionali, che hanno raggiunto alti tassi di riconoscimento degli scenari sintetici prodotti.

INDICE

ELENCO FIGURE

INTRODUZIONE 1

1.1 Ambito

Uno dei principali settori che vengono quasi quotidianamente trattati negli ultimi anni è il settore dell'A*utomotive.* Attualmente [1] i ricercatori prevedono che entro il 2025 vedremo sulla strada circa 8 milioni di veicoli autonomi o semi-autonomi. Però prima che ciò avvenga, le auto a guida autonoma dovranno prima avanzare attraverso 6 livelli di avanzamento della tecnologia di assistenza alla guida. La Society of Automotive Engineers (SAE) ha definito 6 livelli di automazione della guida che vanno dal livello 0 (completamente manuale) al livello 5 (completamente autonomo). Questi livelli sono stati adottati dal Dipartimento dei trasporti degli Stati Uniti.

- *Livello 0 (No Driving Automation)*: La maggior parte dei veicoli attualmente in circolazione sono di livello 0. Il "compito di guida dinamica" è svolto prevalentemente dall'essere umano, nonostante ci possano essere sistemi automatici per aiutare il conducente, come potrebbe essere il sistema di frenata di assistita, che comunque non viene qualificata come manovra di automazione.

- *Livello 1 (Driver Assistance)*: Questo è il livello più basso di automazione. Il veicolo è dotato di un unico sistema automatizzato per l'assistenza alla guida, come lo sterzo o l'accelerazione (cruise control). Il cruise control adattivo, in cui il veicolo può essere tenuto a distanza di sicurezza dietro l'auto successiva, si qualifica come Livello 1 perché il guidatore umano monitora gli altri aspetti della guida come lo sterzo e la frenata.

- *Livello 2 (Partial Driving Automation)*: In questo livello, il veicolo può controllare sia lo sterzo che l'accelerazione/decelerazione. Qui l'automazione non è all'altezza della guida autonoma perché un essere umano si siede al posto di guida e può prendere il controllo dell'auto in qualsiasi momento.

- *Livello 3 (Conditional Driving Automation)*: Il passaggio dal livello 2 al livello 3 è molto importante e ingente dal punto di vista tecnologico ma non visibile dal punto di vista umano. Questo perché il livello 3 conferisce la capacità ai veicoli del riconoscimento ambientale dando la possibilità di prendere decisioni in base alle situazioni che si presentano. Nonostante tutto, è ancora richiesto l'intervento umano perché non è detto che il sistema sia in grado di eseguire sicuramente una determinata manovra, quindi il conducente deve rimanere sempre vigile.

- *Livello 4 (High Driving Automation)*: In questo livello i veicoli sono in grado di intervenire in caso di qualche situazione particolare o di guasto al sistema, non

richiedendo quasi mai l'intervento umano anche se ancora possibile. I veicoli di livello 4 possono funzionare in modalità di guida autonoma, ma fino a quando la legislazione e le infrastrutture non evolvono, possono farlo solo all'interno di un'area limitata (di solito un ambiente urbano in cui le velocità massime raggiungono una media di 30 mph) e questo è noto come geofencing

- *Livello 5 (Full Driving Automation)*: I veicoli di livello 5 non richiedono l'intervento umano, eliminando il concetto di guida dinamica. Le auto di livello 5 non avranno nemmeno volanti o pedali di accelerazione/frenata. Saranno liberi dal geofencing, in grado di andare ovunque e fare tutto ciò che può fare un guidatore umano esperto. Le auto completamente autonome sono in fase di test in diverse zone del mondo, ma nessuna è ancora disponibile al pubblico.

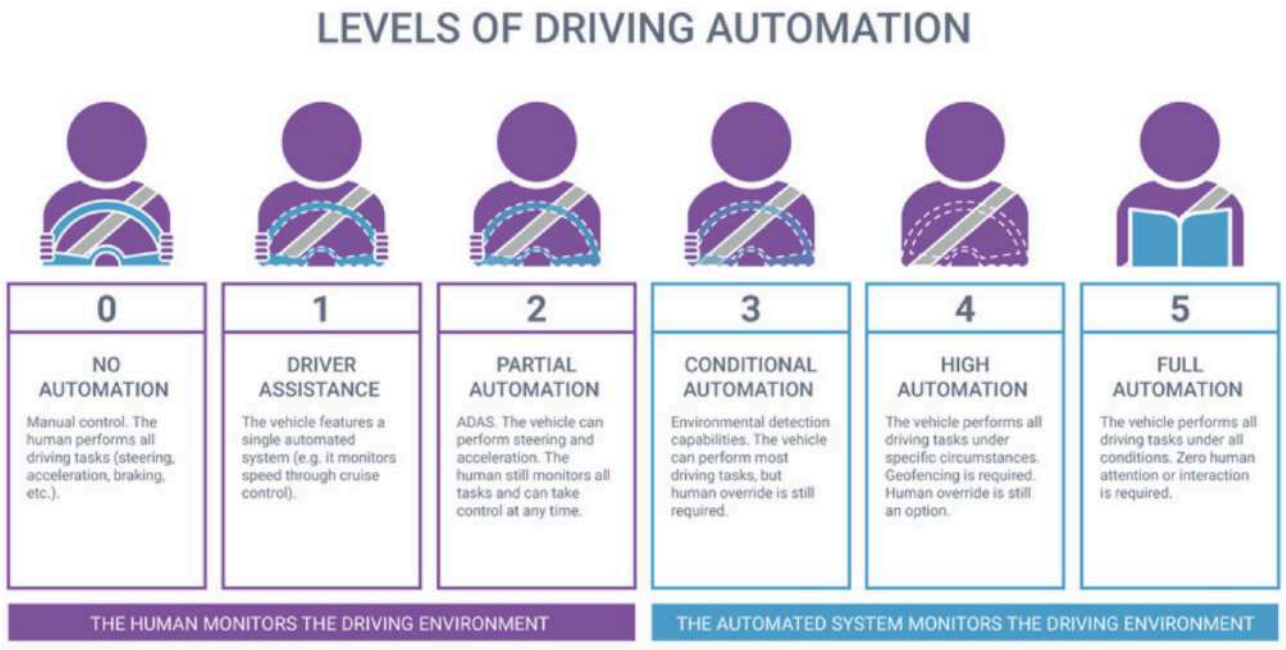


Figura 1: Livelli di automazioni definiti da SAE

La Figura [1] mostra graficamente e spiega brevemente quali sono i sei livelli appena descritti.
Il futuro dei veicoli autonomi è promettente ed entusiasmante, ma la produzione è ancora a pochi anni di distanza da qualcosa di superiore al Livello 2. Questo, non a causa della capacità tecnologica, ma a causa della sicurezza o della sua mancanza [2]. I veicoli connessi (come le auto autonome) sono ricchi di funzionalità di sicurezza fisica, dalle cinture di sicurezza, agli airbag, ma non così ricchi di funzionalità di sicurezza digitale. Da questo punto le auto non sono ancora pronte ad entrare nelle nostre vite quotidiane ma la strada che si sta seguendo è quella giusta.

Non abbiamo però ancora dato una definizione formale di cosa sia un veicolo autonomo. Un veicolo autonomo è un veicolo che è in grado di muoversi autonomamente grazie all'utilizzo di sensori, telecamere e tutto ciò che gli permette di percepire l'ambiente circostante e d comportarsi di conseguenza a seconda della situazione, il tutto senza intervento umano. I risultati che in questo campo si stanno ottenendo sono ogni giorni sempre maggiori, soprattutto grazie all'aumento della potenza di calcolo, agli aggiornamenti nel campo del machine learning, artificial intelligence e alla disponibilità di sensori sempre più precisi e performanti.

Attualmente la tecnologia di guida autonoma può essere suddivisa in più parti, tra cui localizzazione e mappatura [3] [4] [5], pianificazione del movimento [6], [7], decisione comportamentale [8], [9], comprensione della scena [10], e riconoscimento di scenari ecc.

In un ambiente reale, il numero di situazioni e le loro combinazioni che un veicolo può affrontare sono pressoché infinite, e questo fa nascere la necessità di creare dataset attraverso un meccanismo di apprendimento. I possibili modi che permettono la creazione di questi dataset sono attraverso il raccoglimento di dati sul campo e quindi nel mondo reale (in strada o su pista chiusa) oppure tramite modulazioni e simulazioni ottenendo così dati sintetici. La maggior parte dei dataset, vengono creati tramite la raccolta di dati reali e questo perché sono facilmente reperibili e soprattutto sono reali. Questi dataset sono formati da immagini e video, ma uno dei problemi che si riscontrano con questi dati è la scarsa complessità dovuta ad una difficile cattura di situazioni più complesse e pericolose (Corner case), che tratteremo nei capitoli successivi, nonostante un numero di sensori sempre più utilizzati come possono essere i RADAR o i LIDAR. Un'altro dei problemi, per quanto riguarda i dati reali, sono la fase di pre-processing dei dati stessi in quanto c'è poi tutta una parte di annotazione e preparazione che le reti neurali successivamente richiedono per l'addestramento. Per chi si avvicina a questo tipo di approccio di acquisizione dei dati come noi o piccole start-up è un problema a causa degli alti costi da sostenere, quindi quello che viene fatto è adottare il metodo di acquisizione di dati sintetici che vengono ottenuti come detto prima attraverso simulazioni. Ovviamente il vantaggio di poter raccogliere dati di questo genere è dal punto di vista temporale, perché si possono ottenere grandi quantità di dati in poco tempo, poco costosi, per nulla pericolosi e quindi questo ci permette di creare anche scenari più complessi, parametrizzabili e controllati dall'utente.

1.2 Obbiettivo

Dopo questa introduzione dell'ambito in cui andremo a lavorare, l'obbiettivo della tesi è quello di usufruire di un metodo di lavoro per la produzione parametrizzabile di scenari automobilistici in un simulatore di guida, con lo scopo di creare dataset sintetici etichettati che possano essere più vari possibili. Il tutto partendo da una lista di 5 scenari prefigurati precedentemente e poco parametrizzabili, con l'intento di rendere meno deterministici quelli già realizzati aggiungendo parametri e modificando quelli già presenti, oltre all'aggiunta di nuovi scenari. Successivamente, verificarne il riconoscimento di questi scenari attraverso l'utilizzo di reti neurali e confrontare i risultati con quelli precedentemente ottenuti. Per realizzare queste funzionalità sono stati utilizzati strumenti opensource quali Carla simulator, OpenScenario e Pyoscx che vedremo più avanti.

Nonostante sia molto difficile classificare grandi quantità di dataset, è possibile tuttavia crearli, in generale, basandosi sull'acquisizione di informazioni tramite l'utilizzo di sensori quali per esempio Camera, Lidar e Radar:

- *Camera*: Molti di questi forniscono immagini o sequenze della fotocamera, come il set di dati Cityscapes[11], il set di dati BDD100k [12], il set di dati openDD [13] o il set di dati Mapillary Vistas [14], per citarne alcuni. Tutti questi sono set di dati su larga scala per la percezione visiva e forniscono etichette adeguate per le attività correlate come la segmentazione semantica e di istanza e il rilevamento di oggetti. Inoltre, esistono alcuni set di dati specifici per il rilevamento di oggetti sconosciuti da immagini, come il set di dati RoadAnomaly [15], il set di dati Lost&Found [16] e il set di dati Fishyscapes [17]. Inoltre, esistono set di dati con un compito sottostante specifico come il rilevamento di VRU (Eurocity Persons) [18].

- *Lidar*: Accanto a quei dataset puramente basati su telecamere, esiste ormai anche una quantità crescente di dataset multimodali. Uno dei primi di questi set di dati è il set di dati KITTI [19], che fornisce sia i dati della fotocamera che quelli LiDAR. La combinazione di dati LiDAR e fotocamera appare in diversi set di dati, come Apolloscape [20], A2D2 [21], Waymo Open [22], PandaSet [23] e KAIST [24]. Molti di essi forniscono riquadri di delimitazione 3D come etichette [22], [21], [19], alcuni forniscono inoltre etichettatura di segmentazione semantica della nuvola di punti [22], [21]. Il dataset canadese delle condizioni di guida avverse [25] fornisce specificamente immagini e dati LiDAR per le condizioni meteorologiche invernali.

- *Radar*: I dati RADAR sono rari nei set di dati su larga scala per la percezione nella guida automatizzata. A nostra conoscenza, esistono solo alcuni di questi set di dati. nuScenes [26], Astyx HiRes2019 [27], Oxford RobotCar [28] e RADIATE [29] forniscono dati multimodali da RADAR, fotocamera e LiDAR e, per quanto ne sappiamo, tutti questi set di dati forniscono etichette di riquadri di delimitazione per i loro dati RADAR, ad eccezione dell'Oxford RobotCar dove non sono disponibili etichette di verità al suolo.

Inoltre è molto importante utilizzare un metodo che possa permette il riconoscimento degli scenari automobilistici.
In funzione al modo in cui si affronta questo problema[30] si possono distinguere due grandi famiglie di approcci alla classificazione in ambito machine learning (ML):

- Metodi convenzionali o standard, in cui si adotta la logica booleana di vero o falso per cui ogni pixel appartiene o non appartiene in modo rigido ad una classe: al pixel viene associata una etichetta che indica la classe a cui esso è più "simile" secondo una determinata regola di decisione (classificazione hard).

- Metodi non convenzionali, in cui si adotta la logica multivalore, come avviene per le reti neurali artificiali, per cui è ammessa per i pixel la possibilità di appartenenza a più classi: al pixel vengono associate una serie di etichette che indicano il grado di somiglianza o possibilità di appartenenza ad ogni classe (classificazione soft).

Tre sono gli elementi fondamentali che costituiscono il processo di classificazione:

- *Addestramento (training)*: Vengono fornite una serie di informazioni, ottenute tramite l'utilizzo di sensori, per mettere in grado il sistema di riconoscere gli elementi di una classe da quelli di un'altra. Questo si realizza in genere attraverso la scelta di un insieme di pixel campione per ognuna delle classi di interesse, per costruire le regole di decisione nello spazio delle caratteristiche.

- *Assegnamento (allocation)*:Viene assegnata in modo automatico a tutti gli altri pixel dell'immagine l'etichetta relativa ad una delle classi tematiche di interesse.

- *Validazione (validation*): in cui viene verificata la capacità di generalizzazione del classificatore attraverso l'analisi degli errori della mappa tematica, ottenuta come risultato del processo di classificazione automatica. Questo risultato permette di identificare la qualità del prodotto sulla classificazione automatica.

Sulla base di questi tre elementi, è possibile fare una ulteriore suddivisione dei metodi di classificazione in due grandi categorie:

- Classificazione *Supervised*: In questo caso si cerca di costruire un modello partendo da dei dati di addestramento etichettati, con i quali cerchiamo di fare previsioni su dati non disponibili o futuri.

- Classificazione *Unsupervised*: Al contrario della classificazione supervised, i dati inizialmente non sono etichettati ma lo diventano successivamente attraverso tecniche quali il *Clustering*. Con queste tecniche siamo in grado di osservare la struttura dei dati e di estrapolare informazioni.

Fra le diverse metodologie che possono essere utilizzate, quelle al giorno d'oggi più considerate e sulle quali si sta facendo più affidamento sono le reti neurali (NN) nell'ambito del deep learning (DL), in particolare le Deep Neural Network(DNN) che vengono addestrate su dati provenienti dai sensori come per esempio i Radar. In [31] Beglerovic ha proposto metodi di Deep Learning (DL) end-to-end per la classificazione della guida di scenari relativi a sistemi LKA (Lane-Keep-Assist Systems) utilizzando dati provenienti da diversi sensori concentrandosi principalmente sulle reti neurali convoluzionali (CNN). Sono stati proposti due modelli di classificazione: Il primo è un modello online per la classificazione degli scenari durante la guida. Il secondo è un modello offline per la post-elaborazione, che fornisce una maggiore precisione.

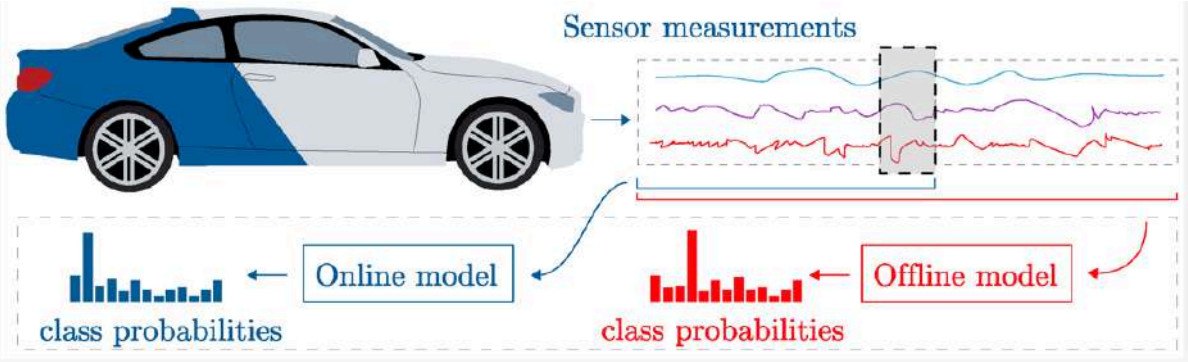


Figura 2: Modelli di classificazione online e offline

Anche le deep recurrent neural network (DRNN) hanno ottenuto un grande successo in una serie di applicazioni nell'ambito della classificazione di immagini e quindi sono nate ricerche come [32],[33] dove vengono creati enormi dataset di immagini etichettate pronte all'uso in ambiente di classificazione e rilevamento di veicoli come nella Figura 3 o da[34] dove l'accento viene posto sulle region proposal networks(RPN) per ipotizzare in maniera più accurata la posizione di un oggetto.

Figura 3: Esempi di cluster ottenuti nell'articolo[14]

Nonostante i recenti sviluppi, mancano comunque grandi dataset specializzati in scenari automobilistici. Il motivo è la difficoltà nel riuscire a suddividerli in cluster che contengano i singoli scenari, proprio perché è un lavoro molto grande da fare a mano e non esistono ancora algoritmi automatici che siano in grado di classificarli.

STATO DELL'ARTE 2

In questo capitolo, viene descritto lo stato dell'arte per quanto riguarda i vari metodi sulla raccolta dei dati relativi agli scenari automobilistici, soffermandoci in particolare sui corner case e quali sono gli approcci ML utilizzati per riconoscerli.

2.1 Generazione di scenari automobilistici

Negli ultimi anni i veicoli a guida autonoma sono diventati più comuni sulle strade pubbliche, con la promessa di portare sicurezza ed efficienza ai moderni sistemi di trasporto. Aumentare l'affidabilità di questi veicoli su strada richiede un'ampia suite di test software, idealmente eseguiti su simulatori ad alta fedeltà, in cui più veicoli e pedoni interagiscono con il veicolo a guida autonoma. È quindi di fondamentale importanza garantire che il software per la guida autonoma sia valutato rispetto a un'ampia gamma di scenari di guida simulati impegnativi [35]. Nel nostro caso però, l'obbiettivo non è quello di testare un software ma di raccogliere dati per costruire dataset più vari possibili. Il Dipartimento dei trasporti degli Stati Uniti e l'amministrazione della sicurezza del traffico [36] ha esaminato vari lavori svolti da governo e industrie per identificare i diversi modi in cui questo è possibile. Questi metodi sono rappresentati nella Figura [4]:

- *Prova su strada (Open-Road Testing)*
- *Test su pista chiusa (Track Testing)*
- *Modellazione e Simulazione (Modelling and Simulation)*

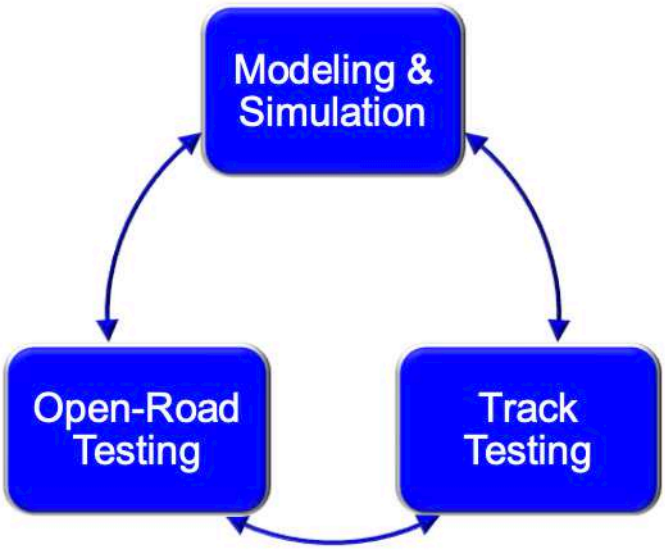


Figura 4: Metodi per l' apprendimento di dati

Queste tre tecniche offrono vari gradi di controllo del test e vari gradi di fedeltà nell'ambiente di test. In molti casi, due o più di queste tecniche possono essere utilizzate in parallelo o in modo iterativo per avere una valutazione più completa della situazione, sfruttando i punti di forza di ciascuno di essi e avere un quadro più completo.

2.1.1 Prova su strada

Questo è il primo dei tre metodi prima citati, la cui raccolta dati si basa su situazioni reali in strade pubbliche consentendo cosi una valutazione completa delle prestazioni dei sistemi ed esponendoli a una varietà estremamente ampia di condizioni del mondo reale. Purtroppo però prove di questo tipo presentano diversi svantaggi dovuti al fatto che non vi è alcun controllo sull'ambiente e problemi per quanto riguardano gli scenari che non possono essere replicabili in luoghi diversi e non possono essere ripetibili su più iterazioni.

Figura 5: Raccolta dati e fase di testing su strada pubblica di un veicolo Waymo

Nonostante la macchina si muova autonomamente, è richiesta la presenza di un autista all'interno del veicolo e questo perché durante la guida vi possono essere situazioni pericolose o incerte in cui il la macchina non sa come comportarsi e quindi c'è bisogno dell'intervento manuale da parte dell'autista per evitare incidenti o situazioni spiacevoli. Come è possibile vedere dalla Figura [5], il veicolo monta sopra di esso una telecamera più altri sensori che non sono visibili e vengono guidati sulle strade pubbliche per raccogliere dati nel mondo reale. Nonostante la grande quantità di dati che si possono raccogliere con questo metodo, è molto difficile catturare dati e quindi situazioni particolari o pericolose come i cosiddetti corna cases, ottenendo così molti scenari ma poco complessi.

2.1.2 Test su pista chiusa

Il test su pista chiusa è il secondo dei tipi di test prima elencati. In questo caso il veicolo non è più situato su una strada aperta al pubblico ma su una strada chiusa. A differenza del primo, questo tipo di test fornisce un alto grado di controllo non replicabile nel mondo reale. I vantaggi di questi tipi di test sono la controllabilità per quanto riguarda le variabili incluse, una maggiore fedeltà se si pensa all'analisi di scenari in cui sono compresi ostacoli o condizioni ambientali realistici, trasferibilità, perché gli scenari possono essere replicati in posizioni diverse e ripetibilità perché si possono eseguire più iterazioni di test con praticamente gli stessi parametri. Ovviamente questo tipo di test presenta anche svantaggi principalmente in termini di tempo e costo perché i test di monitoraggio posso richiedere molto tempo per l'impostazione e l'esecuzione, con naturalmente un aumento dei costi, una variabilità limitata perché non è semplice modificare le condizioni della struttura, la necessità di personale e attrezzature, che in caso di rottura avrebbero anch'esse un costo, (come oggetti che possono rappresentare un ostacolo, misure di sicurezze e dispositivi di misura adeguati alla situazione) e infine la pericolosità dei test perché i veicoli e gli ostacoli reali rendono l'ambiente attorno potenzialmente pericoloso per i partecipanti, dai guidatori agli osservatori dei test.

Figura 6: Test su pista bagnata

Figura 7: Preparazione test con pedone

La Figura[6] e la Figura[7] sono due esempi di test drive dove nella prima si testa la stabilità del veicolo sull'asfalto bagnato mentre nella seconda si sta preparando una situazione in cui davanti al veicolo si trova un pedone che sta attraversando la strada.

2.1.3 Modellazione e simulazione

Il metodo della modellazione e simulazione è l'ultimo dei tre metodi citati precedentemente. Questo metodo si affida a un ambiente virtuale e quindi non si ha bisogno di un veicolo fisico o di un ambiente reale perché tutto viene simulato. In questo caso quindi, tutto ciò che fa parte degli scenari è modellato matematicamente e codificato in modo tale tale da ottenere ambiente e situazioni desiderate. I vantaggi di questo tipo di approccio è sicuramente la controllabilità in quanto la simulazione offre la possibilità di controllare molti aspetti di un test, la prevedibilità perché lo scenario riproduce quello che noi

desideriamo, la ripetibilità perché mantenendo gli stessi parametri di input è possibile replicare gli scenari, la scalabilità in quanto è possibile generare un grande numero di scenari e infine l'efficenza dovuta alla possibilità di accelerare i processi di simulazione ottenendo molti risultati in breve tempo.

Figura 8: Simulazione

2.2 Corner Cases

Un compito essenziale dei modelli ML[37] automobilistici è rilevare e interpretare in modo affidabile situazioni insolite, nuove e potenzialmente pericolose. Il rilevamento di quelle situazioni, che chiamiamo corner cases, è molto importante per sviluppare, applicare e convalidare con successo le funzioni di percezione automobilistica nei veicoli futuri in cui verranno utilizzate più modalità di sensori. Una complicazione per lo sviluppo di rilevatori dei corner cases è la mancanza di definizioni coerenti, termini e descrizioni di tali, specialmente quando si prendono in considerazione vari sensori automobilistici.

Inoltre, come corner cases, bisogna considerare[38] anche i guasti nella guida autonoma causati da situazioni di traffico complesse o imprecisioni del modello che rimangono inevitabili nel prossimo futuro. Mentre molta ricerca è focalizzata su come prevenire tali fallimenti, è stata fatta relativamente poca ricerca sulla loro previsione. Tuttavia, anche il fatto che l'auto alla fine si guasterà è un problema che deve essere affrontato e una previsione precoce dei guasti consentirebbe una maggiore sicurezza per quanto riguarda l'anatomia dei veicoli. Pertanto è fondamentale poter ragionare sull'individuazione di

scenari ed in particolare quelli critici che possono imporre rischi inaccettabili se non considerati.

Il concetto di CCD[39] deve superare varie sfide prima di poter sfruttare tutto il suo potenziale e questi punti possono essere descritti dalle seguenti domande di ricerca (RQ).

- Come descrivere CC su diversi livelli di astrazione?
- Come descrivere automaticamente i dati che contengono CC?
- Come generare o registrare CC dalle descrizioni?
- Come determinare la copertura del CC nei dati di allenamento?

Un corner case risulta dalla combinazione di più situazioni normali o parametri che coincidono simultaneamente, rappresentando così un caso o una scena rari o mai considerati. Nella guida automatizzata possono verificarsi anche situazioni completamente nuove che vengono quindi considerate come corner cases, non solo combinazioni di quelle già note. In generale i corner cases[40] sono divisi in più livelli, che aumentano da un livello di astrazione basso a uno alto, aumentando così teoricamente la complessità del rilevamento. Possiamo vederli rappresentati nella tabella della Figura [9]:

Corner Cases	Description	Examples	Literature
Scenario Level Patterns are observed over the course of an image sequence. Recognition requires scene understanding	**Anomalous Scenario** Pattern that was *not observed* during the training process and has *high potential* for *collision*	• *Person suddenly walking onto the street* • *Car accident* • *Car or person breaks traffic rule*	[6], [16], [17] [18], [19]
	Novel Scenario Pattern that was *not observed* during the training process, but *does not increase the potential for collision*	• *Truck appears from a side road (but is going to stop)* • *Accessing the freeway*	[6], [16], [17] [18], [19]
	Risky Scenario Pattern that *was observed* during the training process, but *still contains potential for collision*	• *A car is coming towards me (potentially short time to collision)* • *Overtaking a cyclist*	[6], [17], [18]
Scene Level Non-conformity with expected patterns in a single image	**Collective Anomaly** *Multiple known objects*, but in an *unseen quantity*	• *Demonstration, e.g., critical mass ride* • *Traffic jam*	[20]
	Contextual Anomaly A *known object*, but in an *unusual location*	• *Tree on the street* • *Barrier, e.g., a fence on the street*	[21], [22], [23] [24], [25], [26]
Object Level Instances that have not been seen before	**Single-Point Anomaly (Novelty)** An *unknown object*	• *Bear, tiger, etc.* • *Lost objects* • *Rollator*	[4], [27], [28] [29], [30], [31] [32], [33], [34]
Domain Level World model fails to explain observations	**Domain Shift** A large, constant *shift in appearance*, but *not in semantics*	• *Weather conditions, rain, fog, snow* • *Traffic sign appearance* • *Location (Europe-USA)*	[35], [36], [37] [38], [39], [40] [41]
Pixel Level (Perceived) errors in data	**Local Outlier** *One or few pixels* fall outside of the expected range of measurement	• *Pixel errors (dead pixels)* • *Dirt on the windshield*	[42], [43], [44]
	Global Outlier *All or many pixels* fall outside of the expected range of measurement	• *Lighting conditions* • *Overexposure*	[45], [46]

Corner case detection complexity increases

Figura 9: Livelli di astrazione dei CC

Livello di pixel:

I casi d'angolo a livello di pixel sono causati da errori percepiti nei dati. Questo tipo di livello è il più basso come si vede nella Figura 9 poiché la complessità del rilevamento è relativamente bassa. A livello di pixel, distinguiamo tra due tipi di valori anomali. I valori anomali globali ,che si verificano quando tutti o molti pixel non rientrano nell'intervallo di misurazioni previsto e questo è causato, ad esempio, da condizioni di illuminazione innaturali o da sovraesposizione. Nella Figura [10 (a)], osserviamo una scena di traffico con sovraesposizione come esempio di un valore anomalo globale. Per trattare la sovraesposizione nelle immagini, ad esempio, la correzione della luminosità e della cromia per i pixel nella regione della sovraesposizione può essere combinata per ottenere una migliore rilevazione [17]. Esiste anche un approccio di deep learning per rilevare la sovraesposizione sulle immagini delle telecamere per la guida automatizzata [18]. I valori anomali locali vengono visualizzati se uno o pochi valori di pixel non rientrano nell'intervallo di misurazione previsto. Questo è ad esempio il caso in cui si verificano errori di pixel e ci sono pixel morti. Nella Figura [10 (b)], si possono osservare alcuni pixel morti e per esempio il rilevamento di questi pixel difettosi può essere basato su un metodo edge-adaptive [15] o sulla stima della rotazione del flusso ottico [14]. Una tecnica di deep learning per identificare i pixel difettosi applica una U-Net per estrarre le caratteristiche per un classificatore casuale di foreste [16].

Livello di dominio:

Nel caso di corner cases a livello di dominio, gli spostamenti di dominio sono in genere la causa di questo tipo di corner case in cui si verifica uno spostamento ampio e costante nell'aspetto ma non nella semantica. La Figura [10 (c)] visualizza un esempio di caso d'angolo a livello di dominio. Qui viene mostrata una scena del traffico in condizioni di neve. Gli esempi includono il passaggio a condizioni meteorologiche diverse come neve, pioggia o nebbia.

Livello oggetto:

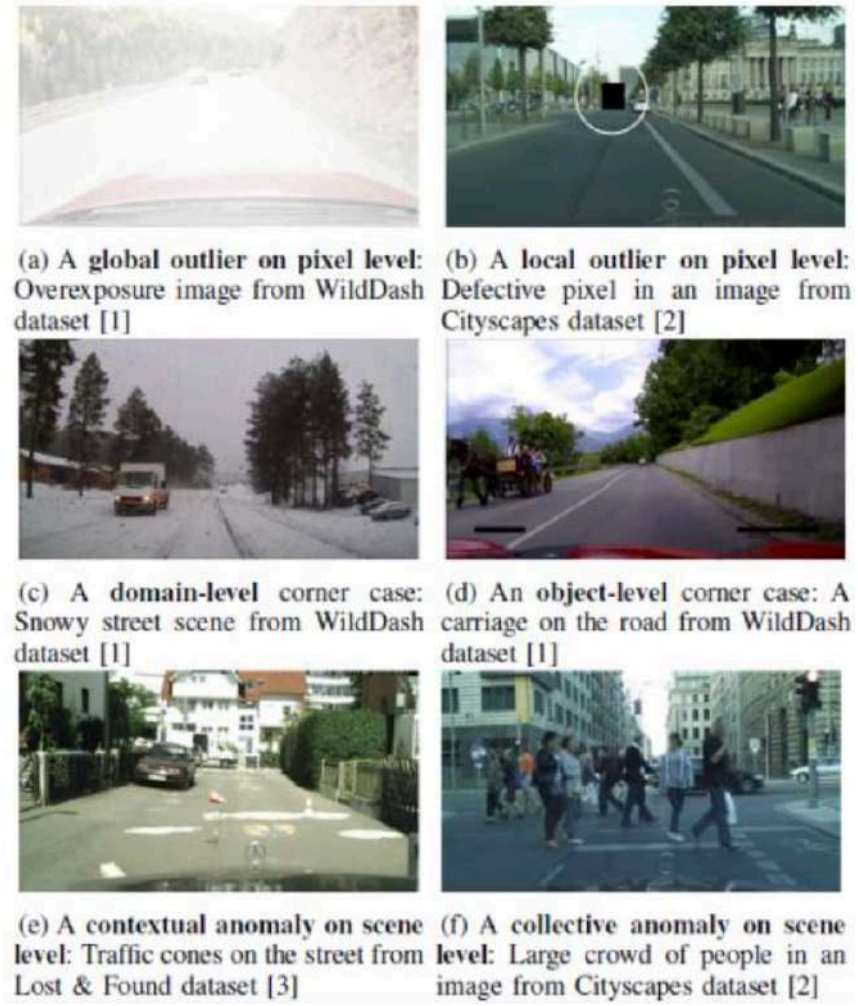
A livello di oggetto, sorgono corner cases, quando le istanze che non sono state viste durante l'addestramento vengono percepite durante l'inferenza. Uno di questi corner case è mostrato a titolo esemplificativo nella Figura [10 (d)]. Qui vediamo una normale scena di strada ma con una carrozza sulla corsia opposta. I corner cases a livello di oggetto includono qualsiasi nuovo oggetto che appare in una scena di strada altrimenti normale, ad esempio un orso o una tigre per strada, qualsiasi altro tipo di ostacolo sconosciuto o altri oggetti sconosciuti come le ombre dei partecipanti al traffico senza che siano effettivamente visibile nell'immagine. Questi sono importanti da rilevare poiché un'ombra sul terreno potrebbe indicare, ad esempio, una persona che arriva dietro l’angolo.

Livello di scena:

I corner cases a livello di scena non sono conformi ai modelli previsti sui singoli fotogrammi dell'immagine. L'aumento teorico della complessità di rilevamento è indotto dalla necessità di comprendere l'intera scena percepita. Distinguiamo tra due tipi di casi limite a seconda del tipo di comprensione richiesta per il rilevamento. Le anomalie contestuali sono oggetti noti, che appaiono in posizioni insolite della scena. Anomalie di questo tipo includono un albero o altre barriere come recinzioni o coni stradali sulla strada. La Figura [10 (e)] mostra un'anomalia contestuale, in cui gli ostacoli sotto forma di coni stradali possono essere individuati sulla strada davanti al veicolo dell'ego. Le anomalie collettive sono oggetti noti, che compaiono in quantità insolita, ad esempio un grande assembramento di persone come una manifestazione o un ingorgo. Nella Figura [10 (f)] è mostrata un'anomalia collettiva, in cui una grande folla di persone attraversa la strada contemporaneamente. Questo può essere visto come una classificazione di una classe, in cui le immagini sono normali o anomale.

Livello di scenario:

I corner cases a livello di scenario denotano l'osservazione di modelli con contesto temporale. Oltre alla comprensione della scena, è necessaria anche la comprensione temporale, che richiede una sequenza di immagini per il rilevamento. Distinguiamo tra tre diversi casi. Scenari rischiosi si verificano quando un modello osservato in forma simile durante l'addestramento appare di nuovo durante l'inferenza e contiene ancora un potenziale di collisione. Può trattarsi di un'auto che si avvicina al veicolo dell'ego, uno scenario che, anche se si verifica frequentemente nei dati, potrebbe comunque avere un esito pericoloso. Un altro scenario rischioso è il sorpasso di un ciclista, poiché dobbiamo anticipare il comportamento imprevedibile del ciclista. Tale scenario è mostrato nella Figura [11 (c)], dove il veicolo dell’ego si avvicina a un ciclista e li supera. Nuovi scenari si verificano quando un modello che non è stato osservato durante l'addestramento appare durante l'inferenza ma non aumenta il potenziale di collisione. Può trattarsi, ad esempio, dell'accesso a un'autostrada se tale scenario non è stato incluso nei dati di allenamento. Un tale scenario non aumenta necessariamente il potenziale di collisione. Nella Figura [11 (b)], è mostrato un nuovo scenario. Osserviamo una situazione che non è stata vista durante l'allenamento. Sembra che un furgone stia guidando sulla strada. Se consideriamo più tempi, però, diventa chiaro che non c'è nessun autista nel furgone e che sta effettivamente parcheggiando in quella posizione. Un altro possibile scenario anomalo potrebbe essere un incidente d'auto che si verifica direttamente accanto o davanti al veicolo dell'ego. Un esempio di scenario anomalo può essere trovato nella Figura [11 (a)]. Qui, una persona cammina inaspettatamente sulla strada da dietro un camper. Questo è uno scenario anomalo, poiché un pedone che si è presentato immediatamente in direzione della strada non è stato visto durante l'addestramento e presenta un alto potenziale di collisione.



(a) A **global outlier on pixel level**: Overexposure image from WildDash dataset [1]

(b) A **local outlier on pixel level**: Defective pixel in an image from Cityscapes dataset [2]

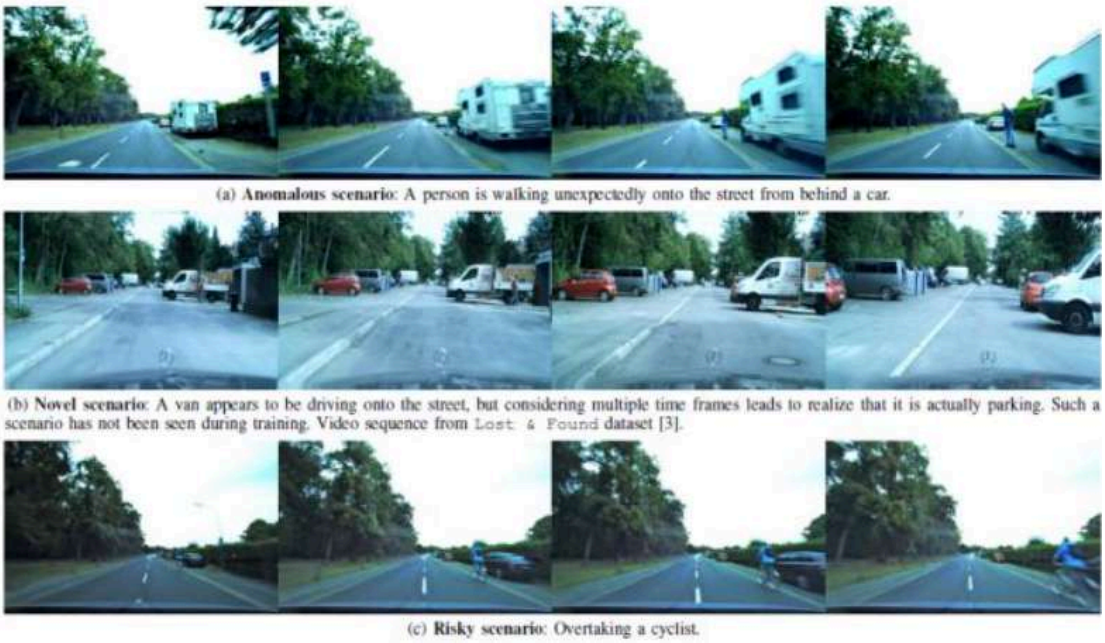
(c) A **domain-level** corner case: Snowy street scene from WildDash dataset [1]

(d) An **object-level** corner case: A carriage on the road from WildDash dataset [1]

(e) A **contextual anomaly on scene level**: Traffic cones on the street from Lost & Found dataset [3]

(f) A **collective anomaly on scene level**: Large crowd of people in an image from Cityscapes dataset [2]

Figura 10: Esempi di corner cases



(a) **Anomalous scenario**: A person is walking unexpectedly onto the street from behind a car.

(b) **Novel scenario**: A van appears to be driving onto the street, but considering multiple time frames leads to realize that it is actually parking. Such a scenario has not been seen during training. Video sequence from Lost & Found dataset [3].

(c) **Risky scenario**: Overtaking a cyclist.

Figura 11: Corner cases a livello di scenario

Una volta definiti i diversi livelli di complessità di corner cases [37], possiamo fare una breve panoramica dei metodi esistenti per il rilevamento di questi. La maggior parte dei metodi di rilevamento dei corner cases in letteratura riguardano il sensore della fotocamera e mirano a rilevarli in immagini o sequenze di immagini.

- Protopadakis et al. [41] mappano i dati RADAR marittimi tramite codificatori automatici impilati in uno spazio di funzionalità, dove utilizzano clustering basato sulla densità per rilevare valori anomali.

- Chakravarthy et al. [42] applicano una CNN ai dati RADAR grezzi per estrarre le caratteristiche e classificarle utilizzando un metodo di classificazione a set aperto per rilevare le incognite nella forma d'onda.

- Lis et al. [43] rigenerano immagini sintetiche dalle maschere di segmentazione per identificare i casi angolari del livello di contenuto tramite l'errore residuo tra l'immagine reale e quella rigenerata.

- Esistono anche metodi di rilevamento dei corner cases sui dati LiDAR, come Wong et al. [44], che introducono una rete di segmentazione di istanze open-set su nuvole di punti che identifica i punti sconosciuti in uno spazio di inclusione e li raggruppa in istanze sconosciute.

- Capellier et al. [45] propongono un metodo per rilevare oggetti noti e sconosciuti nei dati LiDAR. Bolte et al. [46] identificare i casi angolari dello strato temporale per il sensore della fotocamera se l'errore residuo tra l'immagine reale e quella prevista, ponderato dalla criticità della posizione nell'immagine, supera una soglia.

In generale però si mira al raggiungimento di approcci alla fusione di sensori per poter permettere ai veicoli autorizzati di avere una maggiore percezione dell’ambiente circostante.

WORKFLOW 3

In questo capitolo verranno descritti quali sono i passi che sono stati fatti per arrivare alla generazione di un dataset, dalla definizione e generazioni degli scenari, fino al loro riconoscimento grazie all'utilizzo delle reti neurali.

3.1 Toolchain

L'aspetto principale di questa tesi è quello di riuscire a creare un dataset di scenari etichettati e soprattutto molto parametrizzabili attraverso l'utilizzo di una toolchain che, dal punto di vista dello stato dell'arte, per quanto riguarda la cattura di dati come visto nel caso di ambienti reali, permetterebbe a chi si avvicina a questo argomento di poterlo fare in maniera più agile e chiara.

Figura 12: Workflow

Come è possibile vedere dalla Figura[12], il workflow è diviso in 5 fasi principali:

- *Specifica*: In questo primo passo si definiscono igli scenari, quindi che tipi di scenari si vogliono realizzare, quali sono gli attori, che manovre devono svolgere ecc. più tutta una serie di parametri per cercare di renderli più vari possibili. Questi parametri sono all'interno di un file di configurazione JSON che viene letto durante la generazione degli scenari e sono parametri che non sono relativi solamente al comportamento degli attori all'interno dello scenario ma hanno anche influenza su condizioni atmosferiche, di orario dello scenario e sul numero di ripetizioni che uno scenario deve produrre.

- *Generazione*: Generare in maniera ricorsiva una serie di scenari tramite ScenarioGeneration che da come output N scenari .XOSC(OpenSCENARIO).

- *Riproduzione: U*na volta generati i diversi scenari, si passa alla riproduzione di ciascuno di essi su un simulatore che vedremo a breve di nome Carla e gli scenari sono riprodotti uno di seguito all'altro in maniera automatica.

- *Ispezione*: Durante la riproduzione di ciascuno scenario, si verifica che gli scenari rappresentino correttamente la nostra idea di scenario che avevamo in mente e che non ci fossero anomalie all'interno di essi.

- *Addestramento*: Il passo finale è l'addestramento di una rete neurale che ci possa permettere di verificare il riconoscimento di ciascuno scenario.

Attraverso le prime 4 fasi sono stati generati otto classi di scenari che andremo poi ad analizzare maggiormente nel dettaglio più avanti. Le classi sono:

- AppoachingStaticObject
- ChangeLane
- Overtake
- CutIn
- FollowLeadingVheicle
- Freeride
- Pedestrian
- Traffic

Come già detto precedentemente, lo scopo, oltre a quello di creare più classi di scenario possibili, è soprattutto quello di creare per ogni classe di scenario un grande numero di scenari che siano fra loro il più diversi possibili grazie al setting di determinati parametri, in modo da avere dataset che possano essere più vari possibili. Ovviamente maggiore è il numero di classi di scenari, maggiore sarà la difficoltà per una rete neurale di classificare in maniera corretta tutti gli scenari forniti.

AMBIENTE SPERIMENTALE 4

Uno scenario è la descrizione di un ambiente caratterizzato da due tipi di elementi, quelli statici e quelli dinamici. Nel nostro caso, in cui gli scenari sono scenari automobilistici, gli elementi statici sono le strade, gli edifici, la vegetazione e tutto ciò che definisce l'ambiente, mentre gli elementi dinamici sono le condizioni climatiche, le manovre e i partecipanti ovvero tutto ciò che può cambiare. I due tipi di formato file che sono stati utilizzati per la descrizione di contenuti statici e dinamici sono rispettivamente OpenDRIVE e OpenSCENARIO. Successivamente, una volta definite le caratteristiche, ci serve un ambiente di simulazione che ci possa permettere di mostrare gli scenari descritti nei due diversi formati. Un simulatore[47] è un modello della realtà che consente di valutare e prevedere lo svolgersi dinamico di una serie di eventi o processi susseguenti all'imposizione di certe condizioni da parte dell'analista o dell'utente.
La scelta è ricaduta su Carla simulator che è un tool opensource che permette l'utilizzo di OpenSCENARIO, realistico e con al seguito una grande community di ricercatori che lo utilizza e che continua a lavorarci per aggiornarlo e svilupparlo fornendo nuove capacità.

3.1 OpenDRIVE

Il formato OperDRIVE [48] è un formato di file opensource per la descrizione geometrica di contenuti statici come strade, edifici, segnali ecc. di uno scenario. Il formato è descritto in XML (Extensible Markup Language). Lo scopo è quello di fornire una descrizione della rete stradale e, queste descrizioni, possono essere scambiate tra i diversi simulatori. Questo permette alle industrie di ridurre i costi di progettazione e conversione per gli scopi di test e di sviluppo. Quindi i dati stradali possono essere creati manualmente, convertendo dati cartografici oppure attraverso scansioni convertite di strade reali. Per esempio la rete stradale viene definita attraverso una linea di riferimento, che sarebbe la parte centrale di ogni strada. Gli oggetti invece, che caratterizzano la strada come i cartelli stradali o la vegetazione, possono essere posizionati o utilizzando la linea di riferimento oppure usando il sistema di coordinate globale in cui è posizionata la strada. Nella Figura[13] è possibile avere un idea di quello che è stato appena descritto. Nel nostro caso OpenDRIVE viene utilizzato dal nostro simulatore per la descrizione delle mappe.

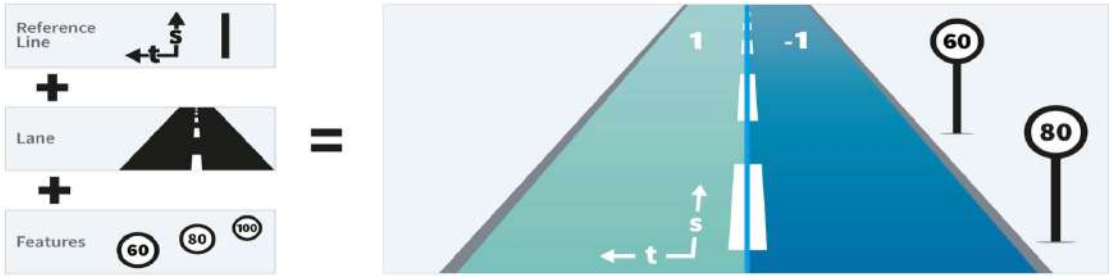


Figura 13: Strada definita dalla linea di riferimento

3.2 OpenSCENARIO

Il formato OpenSCENARIO[49] è anch'esso un formato di file XML ed è per la descrizione invece di contenuti dinamici come meteo, manovre e entità.

Nella Figura[14] sottostante possiamo vedere gli elementi descritti dai due formati:

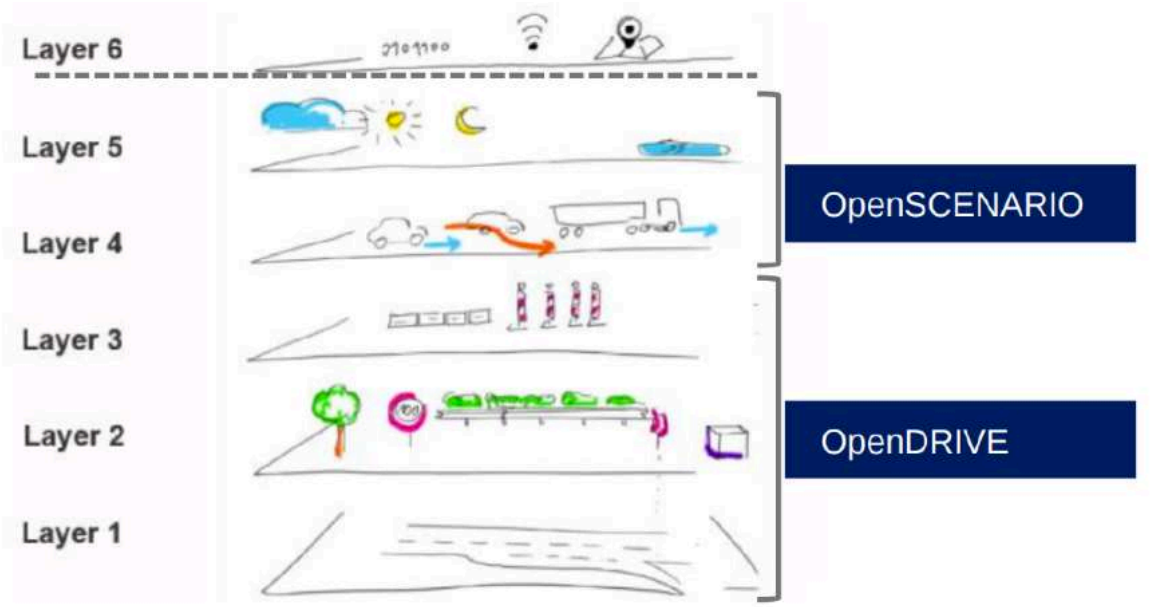


Figura 14: Livelli di astrazione di OS e OD

Entrando più nel dettaglio, ogni scenario è caratterizzato da tre concetti chiave che ne permettono la costruzione e la modifica. I tre concetti chiave sono:

- *RoadNetwork*: Infrastruttura di guida statica inclusi i segnali stradali
- *Entity*: Sono istanze come i veicoli, pedoni ecc. e popolano la RoadNetwork
- *Storyboard*: Istruzioni grazie alle quali interagiscono le istanze delle Entity

Da considerare in secondo luogo lo *Storyboard* di uno scenario contiene almeno un'istanza di *Story* e gli elementi di una storia sono collocati all'interno di una struttura specifica.

In terzo, le azioni degli *Actors* (istanze delle *Entity* coinvolte nelle azioni) sono attivate dalle *Conditions*. Più in generale, le *Conditions* vengono utilizzate nei *Trigger* per avviare *Acts* ed *Events* o per interrompere *Acts* e *Storyboard*. In questo senso, le *Conditions* sono elementi costitutivi di base per definire il comportamento dinamico e le interazioni.

Ci sono due concetti aggiuntivi, che hanno lo scopo di rendere gli scenari facili da riutilizzare per diversi casi d'uso. I *Catalogs* sono raccolte di elementi OpenSCENARIO. Più scenari possono fare riferimento agli elementi definiti all'interno di un *Catalogs*, non considerando così la necessità di definire più volte lo stesso elemento. Inoltre, una

ParameterDeclaration fornisce i mezzi per definire i parametri simbolicamente all'interno di uno scenario o di un *Catalog*.

Andiamo ora ad analizzare e scoprire più nel dettaglio i componenti di uno scenario:

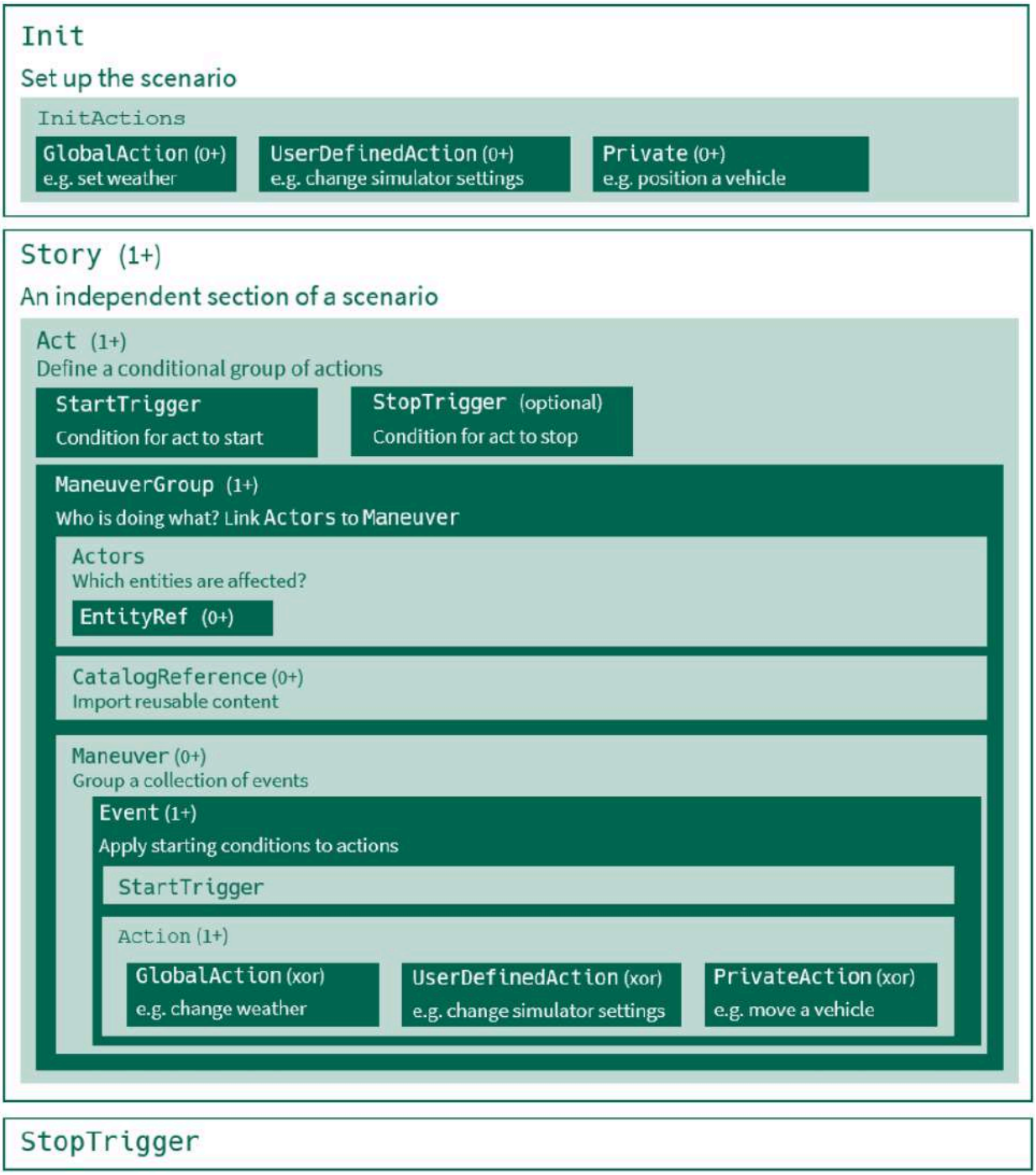


Figura 15: Struttura Generale di uno scenario

Storyboard

In OpenSCENARIO, la *Storyboard*[49] non è altro che la descrizione completa dello scenario. La struttura e la denominazione del concetto di *Storyboard* è simile a quella della narrazione classica nella narrativa, ad es. in uno spettacolo teatrale. Lo Storyboard fornisce le risposte alle domande "chi" sta facendo "cosa" e "quando" in uno scenario. Contiene un elemento di inizializzazione (*Init*) seguito da uno o più elementi *Story*.

Init viene utilizzato per impostare le condizioni iniziali per lo scenario, come la posizione e la velocità delle istanze di *Entity*. Non è possibile specificare il comportamento condizionale in questa sezione.

Story consente agli autori di scenari di raggruppare diversi aspetti in una gerarchia di livello superiore e quindi fornire una struttura in scenari di grandi dimensioni. Le istanze di Storia in OpenSCENARIO, come nella narrativa, contengono atti che definiscono gruppi condizionali di azioni. Ogni atto dovrebbe concentrarsi sulla risposta alla domanda "quando" accade qualcosa nella sequenza temporale di una Storia corrispondente. La risposta a questa domanda è fornita dagli *startTriggers* e *stopTriggers* di un atto. Se uno *startTrigger* restituisce true, allora e solo allora vengono eseguiti i *ManeuverGroups* inclusi.

Un *ManeuverGroup* fa parte di un atto e risponde alla domanda "chi" sta facendo qualcosa nello scenario assegnando istanze di entità come attori alle manovre. I *ManeuverGroups* possono anche includere riferimenti al catalogo per riutilizzare le manovre esistenti.

Le *Maneuvers* definiscono "cosa" sta accadendo in uno scenario. Sono contenitori per *Events* che devono condividere un ambito comune, mentre gli *Events* controllano il mondo simulato o le istanze corrispondenti di *Entity*. Ciò si ottiene attivando azioni, date condizioni definite dall'utente.

Entities

In uno scenario, le istanze di Entity[49] sono quegli oggetti che possono, ma non devono, cambiare la propria posizione in modo dinamico nel tempo. Le istanze di entità che non sono Veicoli o Pedoni sono chiamate *MiscObjects*. Di questa lista fanno parte ad esempio pali, alberi, vegetazione, barriere Jersey ecc.

Le istanze di entità possono essere specificate nel formato dello scenario ma le proprietà sono specifiche del loro tipo. Ad esempio, un veicolo è un'istanza di *Entity* che fornisce proprietà come *VehicleCategory* e *Performance*. Al contrario, un pedone è specificato da proprietà come *modello*, *massa* e *nome*.

Le *Actions* possono modificare lo stato di un'entità, ad es. la sua posizione, velocità o controller. D'altra parte, è possibile interrogare lo stato di un'entità per attivare un’azione.

In OpenSCENARIO si distinguono due gruppi principali di istanze di Entity:

- L'*Entity* descrive un oggetto specifico

- *EntitySelections* descrive un elenco di istanze di Entity

Il movimento di un'entità può essere controllato tramite azioni, controller assegnati dall'utente o un controller predefinito. Si presume che ogni Entità abbia un Controller predefinito che si fa carico di un dominio di movimento (laterale e/o longitudinale) quando mancano Azioni o Controller assegnati dall'utente.

Si prevede che il Controller predefinito mantenga la velocità e l'offset di corsia dell'Entità. Nei seguenti casi, il Controller predefinito sovrintende al dominio di movimento di un'Entità (laterale e/o longitudinale):

- Nessuna azione e nessun controller assegnato dall'utente è in esecuzione.

- Le azioni e/o i controller assegnati dall'utente sono in esecuzione e un dominio di movimento, laterale o longitudinale, non è indirizzato.

Actions

Le Actions[49] servono a creare o modificare elementi dinamici di uno scenario, ad es. cambiamento nella dinamica laterale di un veicolo o cambiamento dell'ora del giorno. Le azioni sono suddivise in tre categorie:

- Azioni private

- Azioni globali

- Azioni definite dall'utente

Nella fase di inizializzazione di uno scenario, le azioni sono responsabili dell'impostazione degli stati iniziali di oggetti dinamici, ambiente, infrastruttura, ecc. In qualsiasi fase successiva dello scenario le azioni vengono eseguite quando vengono attivati gli eventi. Ci sono vari tipi di *Actions* definiti in OpenSCENARIO.

Le *PrivateActions* devono essere assegnate alle istanze di Entity. Con *PrivateActions* è possibile descrivere il movimento, la posizione e la visibilità di un'entità nello scenario. Inoltre, possono definire il comportamento dinamico longitudinale o laterale delle istanze dell' *Entities*, come la velocità o il cambio di corsia. Le *PrivateActions* principali sono:

- *LongitudinalAction*: Controllo della velocità o della distanza relativa a un target

- *LateralAction*: Utilizzando LaneChangeAction o LaneOffsetAction, è possibile individuare una posizione laterale all'interno di una corsia.

- *VisibilityAction*: Abilitazione/disabilitazione della rilevabilità di un' entità da parte di sensori o altri partecipanti al traffico e visibilità nel generatore di immagini.

- *ActivateControllerAction*: Disattiva in maniera esplicita un controller. Questo può essere fatto per i domini longitudinali, laterali o entrambi.

- *ControllerAction*: Assegna un controller ad un attore all'interno della Storyboard. Il ControllerAction può essere utilizzato in alternativa per ignorare i segnali di controllo.

- *TeleportAction*: Definisce una posizione o destinazione di un'entità nello scenario. La posizione di destinazione può essere descritta come coordinate assolute o relative ad altre istanze di entità.

Oltre alle *PrivateActions esistono anche le GlobalActions* che vengono utilizzate per impostare o modificare quantità non correlate a entità.

Le *GlobalActions* principali sono:

- *EnvironmentAction*: Impostazione delle condizioni meteorologiche, delle condizioni stradali e dell'ora.

- *EntityAction*: Rimozione o aggiunta di istanze di Entity.

- *ParameterAction*: Impostazione/modifica dei valori dei parametri.

- *InfrastructureAction*: Impostazione/modifica dello stato di un semaforo o di una fase del controller del semaforo.

Events

Le azioni[49] sono elementi singolari che potrebbero dover essere combinati per creare un comportamento significativo in uno scenario. Questo comportamento è creato dagli eventi che fungono da contenitori per le azioni. Gli eventi incorporano anche *startTriggers*. Questi ultimi non solo determinano l'inizio dell'Evento. Servono anche per avviare le Azioni contenute. Le azioni devono sempre essere racchiuse da eventi con una sola eccezione: nella fase di inizializzazione, le azioni vengono dichiarate singolarmente.

L'impostazione *maximumExecutionCount* specifica quante volte deve essere eseguito un evento, dove il numero di esecuzioni viene incrementato di uno ogni volta che viene raggiunto *endTransition.*

Un Evento viene anche parametrizzato con una definizione di priorità rispetto ad Eventi che coesistono nello stesso ambito (Manovra). Ogni volta che viene avviato un Evento, il parametro di priorità viene preso in considerazione per determinare cosa accadrà agli Eventi già in corso nella stessa Manovra. Le tre scelte relative alla Priorità corrispondente sono:

- Overwrite: Tutti gli altri eventi nell'ambito vengono interrotti e l'evento viene avviato.
- Skip: L'Evento non lascia lo stato di standby fino al termine degli altri Eventi.
- Parallel: L'Evento inizia senza considerare gli Eventi già in corso.

Ogni Evento definito in uno scenario corrisponde a una singola istanza di runtime il che implica che non possono esserci più istanze dello stesso Evento in esecuzione contemporaneamente. A sua volta, ciò significa che gli *startTrigger* non hanno alcun significato a meno che l'evento non sia in uno standbyState, al contrario di ogni startTrigger che avvia una nuova istanza dell'evento.

Maneuver

Una Manovra[49] raggruppa gli Eventi. La definizione di una Manovra può essere esternalizzata a un Catalogo e parametrizzata per un facile riutilizzo in una varietà di scenari. Esempi di manovre sono le manovre di guida, come un (doppio) cambio di corsia o un sorpasso. Tuttavia, combinazioni generiche di Azioni possono essere raggruppate in Manovre, ad es. per simulare un cambiamento climatico.

Conditions and Triggers

Uno scenario[49] può essere considerato come un insieme di *Actions* significative la cui attivazione è regolata da *Trigger*. Questi Trigger giocano un ruolo importante su come si evolve uno scenario poiché lo stesso insieme di azioni può portare a una moltitudine di risultati diversi e tutto dipende da come le azioni vengono attivate in relazione l'una all'altra. Un trigger in OpenSCENARIO è il risultato derivante da una combinazione di condizioni e valuterà sempre come vero o falso.

In OpenSCENARIO una *Condition* è un'espressione logica che viene valutata durante il runtime e restituisce sempre true o false. Una condizione è un contenitore di espressioni logiche e viene valutata durante il runtime. La condizione opera sulle valutazioni correnti e precedenti delle sue espressioni logiche per produrre un output booleano che viene utilizzato dai trigger.

Una singola condizione potrebbe non essere sufficiente per rappresentare un trigger desiderato. In scenari complicati, può invece essere richiesto che la relazione tra un insieme di Condizioni funga da unico Trigger. Un *ConditionGroup* è un'associazione di condizioni che viene valutata in fase di esecuzione e può essere valutata come vera solo se e solo se tutte le condizioni associate sono vere, altrimenti verrà valutata come falsa. Un ConditionGroup è quindi un modo per raggruppare un dato numero di condizioni in un singolo trigger.

Un trigger viene quindi definito come un'associazione di ConditionGroup. Un trigger restituisce true se almeno uno dei ConditionGroup associati restituisce true, altrimenti

restituisce false (operazione OR). Data la natura dei singoli ConditionGroup (AND tra le sue Condizioni) e le associazioni di ConditionGroup (OR tra i suoi membri), un Trigger incarna una mappatura completa della relazione (AND, OR) tra le singole Condizioni. I trigger vengono utilizzati per avviare o arrestare gli elementi dello scenario in corso e sono indicati rispettivamente come *startTrigger* e *stopTrigger*.

startTrigger

Un *startTrigger* viene utilizzato per spostare un'istanza di runtime di un elemento Storyboard da standbyState a runningState. Solo startTriggers host Act ed Event e qualsiasi elemento che non contiene uno startTrigger eredita startTrigger dal relativo elemento padre. Ad esempio, l'avvio di un atto avvia anche i suoi gruppi di manovre e manovre, ma non avvia gli eventi poiché hanno i propri startTrigger. Inoltre, nessun Evento può iniziare se non appartiene ad un Atto che si trova nello Stato in esecuzione.

stopTrigger

Un *stopTrigger* viene usato per forzare un'istanza di runtime di uno StoryboardElement a passare dal relativo standbyState o runningState a completeState. Solo gli elementi Story e Act ospitano stopTriggers. Qualsiasi StoryboardElement eredita stopTrigger dal padre. Questo è vero anche se lo StoryboardElement in esame ha il proprio stopTrigger. Ad esempio, se una Storia è interessata da uno stopTrigger, lo sono anche tutti i suoi Atti, anche se hanno il proprio stopTrigger.

Condition Type

Il tipo di condizione di base contiene tre elementi di base: name, delay e conditionEdge. Mentre il primo elemento è si spiega da solo, gli altri richiedono un chiarimento.

Delay

Questo elemento si riferisce alla quantità di tempo che deve trascorrere tra il soddisfacimento della Condizione e la segnalazione come soddisfatta. Indipendentemente da altri parametri che possono essere utilizzati per definire la Condizione, questo elemento definisce un puro ritardo sulla sua uscita.

conditionEdge

- Questo elemento può essere utilizzato per introdurre un componente dinamico nella verifica della condizione, poiché gli stati precedenti della sua espressione logica ora giocano un ruolo nell'output della condizione:

- Una condizione con un fronte di salita restituisce vero se la sua espressione logica precedentemente valutata come falsa, ma ora vale come vera.

- Una condizione impostata con un fronte discendente restituisce vero se la sua espressione logica precedentemente valutata vera ma ora valuta falsa.

- Una condizione impostata con un fronte di salita o di discesa restituirà vero se viene verificato un fronte di salita o di discesa.

- Infine, una condizione impostata con none restituirà vero se la sua espressione logica è vera e falso se la sua espressione logica è falsa.

Se il parametro *RisingEdge* è impostato su *rising*, *falling* o *risingOrFalling*, una condizione non viene definita la prima volta che viene verificata poiché la precedente valutazione dell'espressione logica non è definita. Per risolvere questo problema, ci si aspetta che tutte le condizioni definite con *rising*, *falling* o risingOrFalling restituiscano false la prima volta che vengono controllate da un motore di simulazione. Tutti gli altri elementi di una condizione dipenderanno dal suo sottotipo, di cui ce ne sono due, *ByEntityCondition* e *ByValueCondition.*

ByEntityConditions

ByEntityConditions utilizzerà gli stati delle istanze di Entity per eseguire la valutazione condizionale. Le valutazioni condizionali possono dipendere dal valore di un singolo stato, o da come il valore di un dato stato si rapporta ad un altro stato (all'interno dell'Entità, tra istanze dell'Entità, e tra l'Entità e le corrispondenti caratteristiche del RoadNetwork). Le condizioni dell'entità richiedono la definizione di TriggeringEntities i cui stati vengono utilizzati nella valutazione condizionale. Nel caso in cui sia definita più di un'entità di attivazione, all'utente vengono fornite due alternative per determinare quando la condizione restituisce true; o tutte le entità Triggering verificano l'espressione logica o almeno un'entità verifica l'espressione logica.

ByValueConditions

ByValueConditions rappresentano espressioni logiche che dipendono da valori non direttamente correlati alle istanze di Entity. Ad esempio, possono essere stati degli scenari, orari o informazioni sui segnali stradali. ByValueConditions fornisce anche un wrapper per condizioni esterne che possono dipendere da valori non accessibili dallo scenario e disponibili solo per l'implementazione dell'utente. Esempi di questi sono la pressione di pulsanti e segnali o comandi personalizzati.

4.3 CARLA

Tutte le simulazioni che sono caratterizzate dai partecipanti e dalle loro manovre, sono ampiamente gestite e controllate da CARLA. Questo perché essa fornisce un pacchetto chiamato ScenarioRunner che permette di gestire uno scenario determinando manovre comuni come ad esempio un taglio di una strada, oppure un cambio di corsia. ScenarioRunner converte le manovre descritte nel file OpenSCENARIO in un file piramidale che possa permettere cosi lo svolgimento di diverse azioni o manovre in maniera sequenziale o parallela.

CARLA (Car Learning to Act) [50] è un simulatore di guida autonoma open source che si basa su Unreal Engine per eseguire la simulazione e utilizza lo standard OpenDRIVE per definire strade, ambienti urbani e condizioni ambientali, inclusi il tempo e l'ora del giorno Figura[16]. Il controllo sulla simulazione è garantito da un'API gestita in Python e C++ che cresce costantemente con il progetto.

Figura 16: Scenari con orari e condizioni meteo differenti(https://najibghadri.com/assets/mscthesis/)

Entrando più nello specifico introduciamo le principali e funzionalità e moduli di CARLA[51]:

World and Client

Il *Client* è il modulo che viene eseguito per gestire la simulazione e viene eseguito attraverso un IP e una porta specifica. Il client comunica con il server tramite terminale e inoltre possono esserci più client in esecuzione contemporaneamente, ma l'esecuzione multiclient richiede una conoscenza più approfondita dei CARLA. Il *World* è l'oggetto dove viene eseguita la simulazione e contenente tutti i metodi utilizzati per generare tutti i

contenuti statici e dinamici, oltre che le manovre eseguite. Il mondo viene distrutto nel momento in cui si decide di cambiare mappa, ricreandolo con un'altra.

Actors and blueprints

Gli *Actors* sono tutto ciò che hanno un ruolo nella simulazione in esecuzione come veicoli, pedoni, sensori, segnali stradali e semafori. A differenza degli actors, i *blueprints* sono modelli già creati che presentano una serie di attributi, alcuni dei quali possono essere personalizzati altri no.

Maps and navigation

La mappa rappresenta il la città simulata e sono disponibili otto mappe le cui strade sono descritte dallo standard OpenDRIVE 1.4. Le strade stesse, come anche le corsie e gli svincoli sono gestiti dalle Python API e sono usati insieme alla classe waypoint per fornire paths di *navigation.* Inoltre gli oggetti come *carla.Landmark* permettono di gestire sia segnali stradali che i semafori. I file OpenDrive forniscono quindi informazioni anche sul comportamento da parte dei veicoli in prossimità della segnaletica stradale come un limite di velocità, uno stop in prossimità di un incrocio, precedenze e dei semafori a seconda delle condizioni.

Sensors and data

I sensori sono collegati al veicolo che raccolgono informazioni sull'ambiente che li circonda e sono definiti dai blueprints nella libreria Blueprint. I sensori principali sono il lidar, radar, IMU, camera, collision detector. Grazie ai sensori i veicoli hanno la possibilità di conoscere meglio l'ambiente che li circonda, sia per quanto riguarda gli elementi statici che dinamici e di catturare un numero più elevato di dati e informazioni. Questo è dovuto anche al numero che è in continua crescita di sensori disponibili sul mercato, a prezzi non eccessivamente alti e con performance sempre più elevate. Per esempio l'incremento del numero dei sensori attualmente installati sui veicoli ha portato ad una diminuzione del numero di incidenti beneficiandone i conducenti, ma anche gli altri veicoli guidati da terzi [SherinAbdelhamid, HossamS.Hassanein, GlenTakaharab: Vehicle as a Mobile Sensor. In: ScienceDirect]. Qui sotto è possibile vedere due esempi di sensori usati dal veicolo nella Figura[16] e nella Figura[17].

Figura 17: Sensore LIDAR

Figura 18: Sensore RADAR

4.4 Scenario Runner

Scenario Runner è il modulo che ci permette l'esecuzione degli scenari automobilistici all'interno del simulatore CARLA. Gli scenari vengono descritti attraverso script Python oppure tramite lo standard OpenSCENARIO. Grazie a Scenario Runner abbiamo potuto riprodurre all'interno di CARLA ogni scenario definito in formato XOSC.

4.5 Pyoscx e Scenario Generation

Lo scopo adesso è quello di creare una serie di scenari in formato XOSC definiti da una serie di parametri per poi costruire vari dataset. Questo è stato fatto utilizzando il modello scenariogeneration del tool pyoscx che permette di generare una lista di parametri grazie alla quale è stato possibile generare molti tipi diversi di scenari caratterizzati ad esempio dalle condizioni meteorologiche, ora del giorno, traffico più meno intenso ecc.

Scenariogeneration è un wrapper Python per la generazione di file xml OpenSCENARIO e OpenDRIVE.Il pacchetto Python scenariogeneration è una raccolta di librerie per la generazione di file XML OpenSCENARIO (.xosc) e OpenDRIVE (.xodr). Questo pacchetto combinato (che include i precedenti pyoscx, pyodrx) può essere utilizzato per generare insieme scenari basati su OpenSCENARIO con mappe della rete stradale basate su OpenDRIVE interconnesse. Il pacchetto consiste nel modulo scenario_generator e due sottopacchetti Python, xosc (OpenSCENARIO) e xodr (OpenDRIVE) [52].

Nel nostro caso abbiamo utilizzato solo le funzionalità xosc per OpenSCENARIO perché l'ambiente era già generato dal simulatore. Nella Figura[18] qua sotto sono riportati tutti i passaggi che portano alla generazione completa di uno scenario

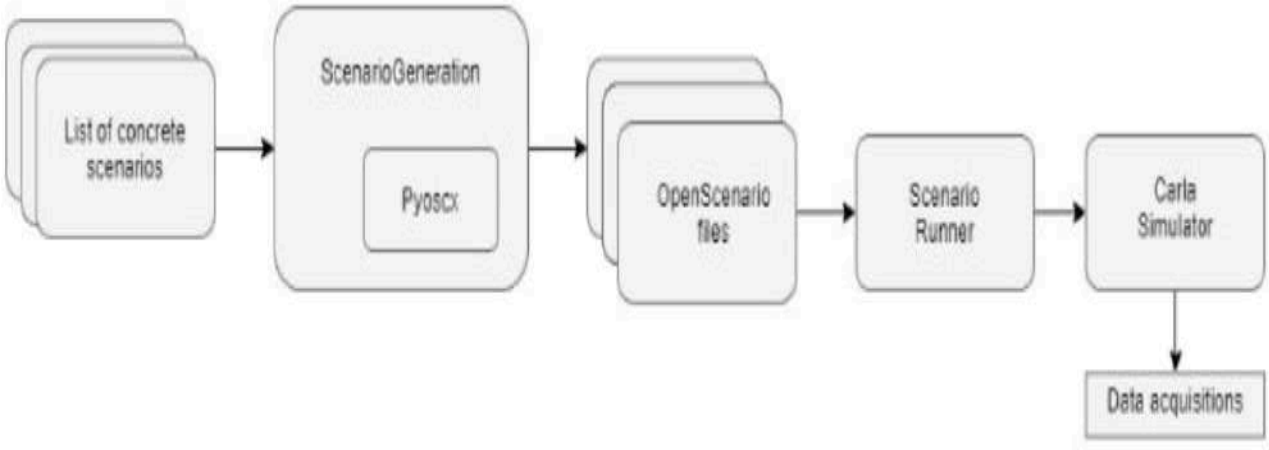


Figura 19: Passaggi per arrivare alla generazione di uno scenario su Carla

SCENARI AUTOMOBILISTICI 5

In questo capitolo si parla degli scenari che sono stati generati/migliorati e dei metodi e delle funzioni che sono state utilizzate per la loro implementazione. L'architettura utilizzata è visibile nella Figura[19].

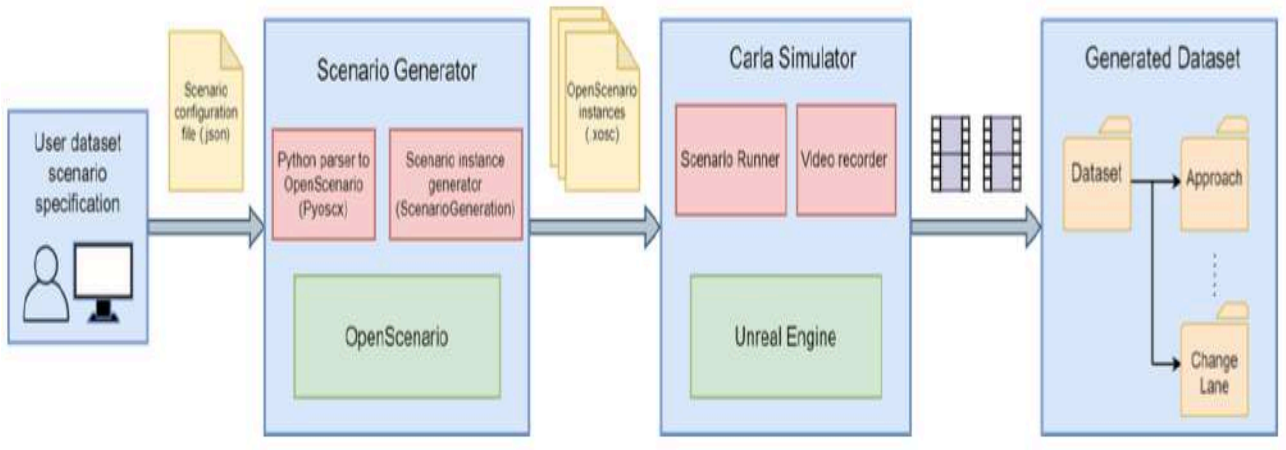


Figura 20: Architettura

5.1 Scenari generati

Come prima cosa abbiamo visto quali sono gli scenari che sono stati realizzati, la loro lista di parametri e quali potessero essere i nuovi parametri e scenari da poter aggiungere . uno dei. Ovviamente l'architettura del sistema sviluppato non preclude la possibilità di creare scenari più complessi. Gli attori principali presenti in ogni scenario sono:

- *Ego_vehicle*: l'ego è il soggetto di tutti gli scenari e quello che esegue o subisce l'azione da parte di un altro veicolo. Inoltre è il veicolo da il quale sono stati registrati tutti i video.

- *Adversary_vehicle*: l'adversary è il veicolo con il quale l'ego interagisce in tutti gli scenari.

- *Npc_vehicle*: gli NPC sono veicoli di contorno, servono per generare traffico e non hanno compiti specifici ma seguono le regole della strada. Sono stati inoltre definiti alcuni parametri per poter generare un traffico più o meno intenso.

- *Pedestrian*: Sono i pedoni che vengono generati all'interno della città per rendere lo scenario il più realistico possibile e, come gli Npc_vehicle, non hanno compiti specifici ma seguono le regole della strada.

Gli scenari che sono stati presi in considerazione sono il Freeride, Follow, CutIn, Approach e ChangeLane mentre i nuovi scenari aggiunti sono il Traffic e l'Overtake:

- ***Freeride*:** L'ego_vehicle si muove all'interno dello scenario in maniera libera, senza interagire direttamente con altri veicoli. In questo particolare scenario si è deciso di non generare altri veicoli se non l'ego, in modo da creare uno scenario semplice e di facile interpretazione, potendo permettere a chi si avvicina a questo argomento di iniziare a capire di cosa si sta trattando. Esempio nella Figura [20].

Figura 21: Freeride

- ***ApprocchingStaticObject*:** In questo scenario l'ego_vehicle accelera e quando si trova in prossimità dell'avversary comincia a frenare fino a fermarsi dietro di lui. In tutto questo sono stati creati veicoli dintornò alla scena principale, in modo da poter rendere il tutto più reale. Esempio nella Figura [21].

Figura 22: ApprocchingStaticObject

- ***Change Lane*:** L'ego vehicle, in questo scenario, si sta muovendo quando ad un certo punto, in prossimità del veicolo adversary che si muove lentamente, decide di cambiare corsia per superarlo. Esempio nella Figura [22].

Figura 23: Change Lane

- **Overtake:** A differenza del Change Lane precedente, l'ego vehicle non compie un sorpasso nei confronti dell'avversary perché quest'ultimo si muove lentamente, ma compie semplicemente un cambio di corsia nel traffico quando lo ritiene più opportuno. Esempio nella Figura [23].

Figura 24: Change Lane2

- ***Cut in*:** In questo scenario l'ego prosegue in maniera lineare nella sua corsia in cui è stato generato. L'adversary, invece, viene posizionato nella corsia di fianco e dietro all'ego ad una distanza che scegliamo noi nel file JSON dei parametri. L'adversary ha una velocità maggiore dell'ego_vehicle e quando il primo si trova in prossimità del secondo, decide di cambiare bruscamente corsia, invadendo quella dell'ego e proseguendo dritto una volta compiuta la manovra. Esempio nella Figura [24].

Figura 25:Cut in

- ***FollowLeadingVehicle*:** Anche qui, nello scenario sono presenti un ego_vehicle e un adversary. L'adversary è posizionato davanti all'ego ed esegue delle manovre, mentre l'ego lo segue alla stessa velocità ed a una distanza di sicurezza. Esempio nella Figura [25].

Figura 26: FollowLeadingVehicle

- ***Pedestrian*:** Nello scenario dei Pedestrian, l'ego vehicle si muove per la città liberamente seguendo le regole della strada, stando attento però anche al comportamento dei pedoni che possono per esempio attraversare la strada in prossimità di un attraversamento. Esempio nella Figura [26].

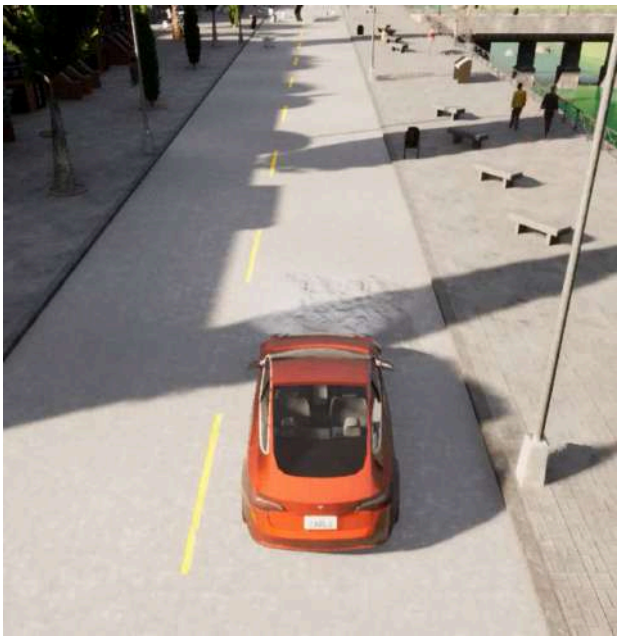


Figura 27: Pedestrian

- ***Traffic*:** In questo ultimo scenario abbiamo l'ego vehicle che si trova in una situazione di traffico con i veicoli attorno che si muovono seguendo le regole della strada ma a velocità diverse in modo da creare più dinamicità. Esempio nella Figura [27]. Lo scenario di traffico ha dato un contributo fondamentale alla variabilità degli scenari, principalmente per il nuovo modo di generare gli npc che vedremo nelle prossime slide.

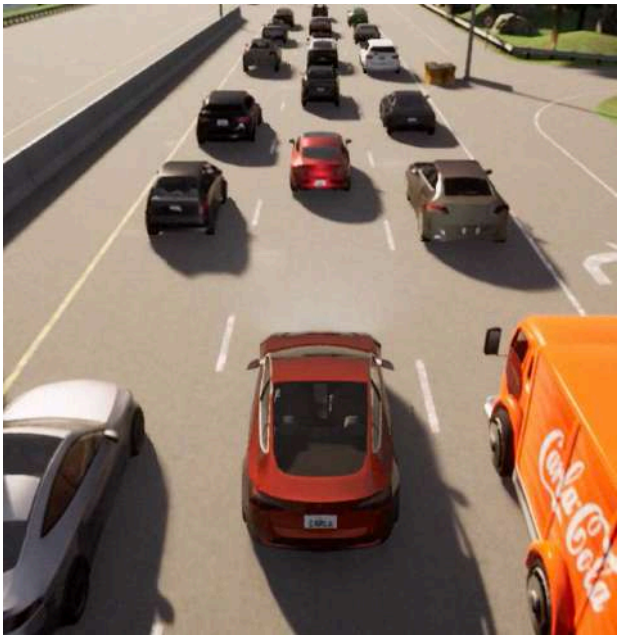


Figura 28: Traffic

5.2 Metodi e funzioni

Questi sono tutti gli scenari che sono stati, prima di essere generati, definiti e occorre un modo per poterli generare e per ottenere la descrizione OpenScenario di ognuno e inoltre, anche un modo per generarli in situazioni e circostanze diverse, in modo da non ottenere sempre lo stesso scenario, tramite il setting di alcuni parametri. Iniziamo parlando di pyxosc che, come già accennato nel capitolo precedente, è un python wrapper per Openscenario e quindi permette di descrivere tramite un file python uno scenario. Il modulo xosc gestisce la parte relativa a OpenSCENARIO e il vantaggio di questo modulo è che permette di evitare di dover definire manualmente il file XML descrittivo dello scenario. Questo è stato possibile grazie all'utilizzo e alla modifica del tool di ScenarioGeneration che implementa il modulo xosc di pyxosc e ne permette la parametrizzazione. Proprio grazie a ScenarioGenerator che è possibile generare e parametrizzare i file OpenSCENARIO e per utilizzarlo è stata creata una classe che eredita ScenarioGenerator e lo inizializza. Essendo però gli scenari poco parametrizzabili come già accennato precedentemente si è deciso di, modificare quelli già esistenti per permettere l'inserimento di più valori e di aggiungerne altri per rendere le scenari di una stessa classe più dinamici e vari. I parametri utilizzati/ modificati sono:

- Town: è una stringa che specifica il nome della mappa che il simulatore Carla deve caricare durante la riproduzione dello scenario. Sono disponibili al momento 7 Town, dalla 0 alla 6.

- isRaining: è un valore booleano che ha come scopo l'attivazione o disattivazione della pioggia.

- hourOfDay: Specifica l'ora del giorno in cui si svolge lo scenario.

- RandomPosition: Anch'essa è una variabile booleana che specifica se lo scenario avrà posizioni di spawn degli attori randomiche oppure no.

- InitialOffset: è una lista di int, in modo da avere diverse posizioni in ogni sceanrio, che definisce la distanza alla quale l'adversary viene generato rispetto alla posizione dell'ego.

- Repetition: è un numero intero che specifica il numero di ripetizioni dello stesso scenario.

- Npc_number: è un int che specifica il numero massimo di npc che vengono generati entro un raggio dalla posizione dell'ego.

- Scenario_lane: nel caso di scenari che hanno bisogno di più corsie per essere eseguiti (es. ChangeLane o CutIn) questo parametro specifica quale siano le corsie interessate. In particolare viene preso il punto in cui viene generato l'ego e sulla base di di quel punto si considera la corsia o le corsie che servono per eseguire lo scenario.

Invece i nuovi parametri inseriti sono:

- ApproachSpeed/speed_adv: è una lista di int che specifica la velocità alla quale un veicolo ego o adv si muove.

- change_offset/change_dist: Distanza tra ego e adv per fare il cambio corsia/ l'ego cambia corsia dopo aver percorso una certa distanza

- npc_lane: é un array che ci permette di definire un range di veicoli di npc da generare per ogni corsia tra quelle presenti vicino, compresa dell'ego stesso, all'ego. Questo parametro ci consente di generare un traffico più o meno intenso a seconda delle esigenze dello scenario. Possibile, è anche inserire come valori [0,0] non generando alcun veicolo. Se per esempio scriviamo [10,15], per ogni corsia verranno generati tra i 10 e i 15 npc, se invece scriviamo [15,15] verranno generati in ogni corsia 15 npc.

- lane_number: è un array che ci permette di definire il numero di lane che vogliamo nei nostri scenari. Questo ci permette di risparmiare tempo in termini di filtraggio di scenari, perché se ci servono solo scenari, per esempio, a una corsia e quindi [1,1], basta inserire

i parametri e avremo tutti scenari così. Se mettiamo invece [2,3], avremo scenari a due o tre corsie.

- distance_cut: array che ci permette di ottenere scenari di cut in in cui il taglio della strada è più o meno brusco a seconda del valore scelto nel range inserito. Il valore indica la distanza dell'adversary dall' ego dalla quale iniziare la manovra di cut da parte dell'avversar

- radius_offset: Questo parametro ci permette di definire un raggio intorno all'ego, dentro il quale non vengono generati npc. La non generazione non riguarda per forza tutte le corsie ma solo quelle non utili allo scenario. In questo modo è possibile popolare la mappa solamente nelle vicinanze dello scenario senza così sovraccaricare il server.

Esempio di una lista di parametri nella Figura [28]:

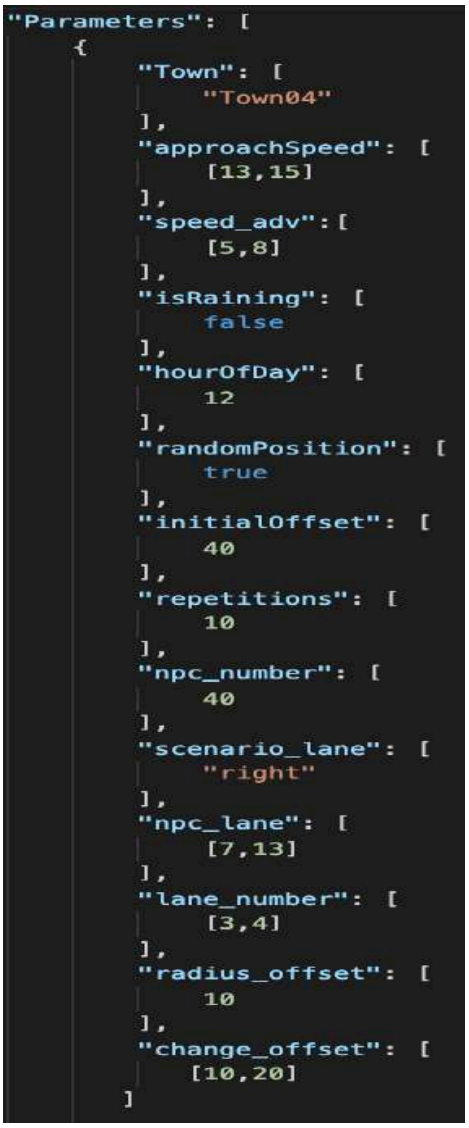


Figura 29: Lista parametri

Un argomento adesso interessante, da prendere in considerazione, è approfondire come sono stati definiti i punti in cui generare i diversi attori degli scenari. I metodi di Carla che sono stati utilizzati sono:

- ***carla.Transform*:** Questa è una classe che definisce una oggetto di tipo transform e, attraverso la combinazione di posizione e rotazione, indica una coordinata nella spazio.

- ***carla.Waypoint*:** I waypoint in CARLA sono descritti come punti 3D. Hanno una carla.Transform che individua il waypoint in una strada e lo orienta in base alla corsia. Memorizzano anche le informazioni stradali appartenenti a tale punto per quanto riguarda la sua corsia e la sua segnaletica orizzontale.

- ***carla.Map*:** Questa classe contiene le informazioni stradali e la gestione dei waypoint della mappa che è correntemente caricata sul server e i dati vengono recuperati dal file OpenDRIVE che descrive la strada.

- ***get_waypoint(self,location,project_to_road=True,lane_type=carla.LaneType.Driving)* :** Restituisce un waypoint che può essere localizzato in una posizione esatta o traslato al centro della corsia più vicina. Tale tipo di corsia può essere definito utilizzando flag come LaneType.Driving & LaneType.Shoulder. Il metodo restituirà Nessuno se il waypoint non viene trovato, il che può accadere solo quando si tenta di recuperare un waypoint per una posizione esatta. Ciò facilita il controllo se un punto si trova all'interno di una determinata strada, altrimenti restituirà il waypoint corrispondente.All'interno di un waypoint sono immagazzinate anche le informazioni relative alle corsie (waypoint.lane_change), cioè se dato un waypoint esistono o no delle corsie laterali.

Le funzioni che sono state modificate e utilizzate sono:

- ***check_town(town)*:** Per prima cosa occorre che il server di Carla sia aperto con la mappa corretta caricata. La mappa che vogliamo caricare è specificata nei parametri del file Json e questo metodo controlla se la mappa caricata è corretta e nel caso sia sbagliata carica quella giusta, prima di procedere alla generazione dei file.

- ***get_offset_waypoint(waypoint_ego, offset)*:** Ottenuta la posizione dell'Ego, si procede con questa funzione per ottenere quella dell'Adversary, che si ottiene in base alla posizione dell' waypoint associato all'Ego e ad un altro parametro che è l'offest. Decidendo di utilizzare il waypoint, ci permette di calcolare una posizione sulla stessa lane dell'Ego che sia spostata in avanti o indietro pari al valore dell'offset.

- ***get_random_npc_spawn(current_map, waypoint_list, center_position, radius, offset, scenario_lane, radius_offset):*** Questa funzione, grazie al passaggio di alcuni parametri dal file JSON e altri trovati grazie alle altre funzioni descritte precedentemente, ci permette di definire il raggio, a partire dall'ego, in cui non considerare gli waypoint del dizionario trovato in get_waypoint_npc. Gli waypoint non considerati sono quelli

all’interno del raggio e che si trovano nella lane uguale a quella dell’ego e uguale a scenario_lane, cosi lo scenario si può svolgere tranquillamente senza interefernze da parte di alcuni veicoli nonostante questi ultimi siano generati in tutte le altre parti per rendere lo scenario più reale possibile.

- ***get_random_vehicles()*:** Questa classe ci permette di generare una lista di tutti i veicoli disponibili filtrando quelli indesiderati (es biciclette e motocicli).

- ***carla2pyxosc(spawn)*:** Questo metodo serve per convertire le posizioni in formato Carla in quello pyxosc.

Le funzioni che sono state create per permettere :

- ***get_way(waypoint)***: Questa funzione, prende in ingresso il waypoint dell’ego e sulla base di quello crea sullo stesso suo livello il numero di waypoint pari al numero di corsie disponibili al suo fianco, escluse le corsie di emergenza e ovviamente i marciapiedi, creando cosi un dizionario di waypoint.

- ***get_random_spawn_points: (offset, scenario_lane, lane_number, npc_lane =“None”, radius_offset = 0)*:** Questa è la funzione che viene chiamata dal file generate in cui vengono descritti tutti gli scenari e che al suo interno sono richiamate tutte le altre funzioni. alla fine restituisce il waypoint dell’ego, dell’avversar e infine di tutti i veicoli di contorno che si vogliono generare. Se nel file generate non passiamo alcun argomento di npc_lane e e di radius_offset, in automatico non verranno generati veicoli e il radius_offset sarà uguale a 0.

- ***get_waypoint_npc(waypoint_map, scenario_lane, npc_lane)*:** Questa funzione prende in ingresso il dizionario calcolato nella funzione get_way, più gli altri parametri che si trovano nel file JSON e replica un numero di waypoint per ogni corsia, partendo dal waypoint che si trova allo stesso livello dell’ego e che è stato salvato nel dizionario, pari al numero dentro npc_lane. Questa funzione, oltre a restituire waypoint_list(la lista di tutti gli waypoint), è molto importante perché in teoria si sarebbero potuti generare i veicoli tramite gli spawn point (pallini rossi) di Carla ma, come possiamo vedere nella Figura [29], in alcuni punti non ci sono e questo avrebbe portato a casi di scenari in cui il waypoint dell’ego veniva preso in zone in cui non erano presenti spawn point non generando così alcun traffico.

- Pallini rossi: tutti i possibili punti di spawn della Town selezionata, in questo caso la Town3.
- Pallino blu: il punto di spawn dell’Ego.
- Pallino azzurro: punto di spawn dell’Adversary.
- Cerchio verde: raggio esterno di spaws dei veicoli npc.
- Cerchio azzurro: raggio interno di spawn dei veicoli npc per il filtro sulle lane.

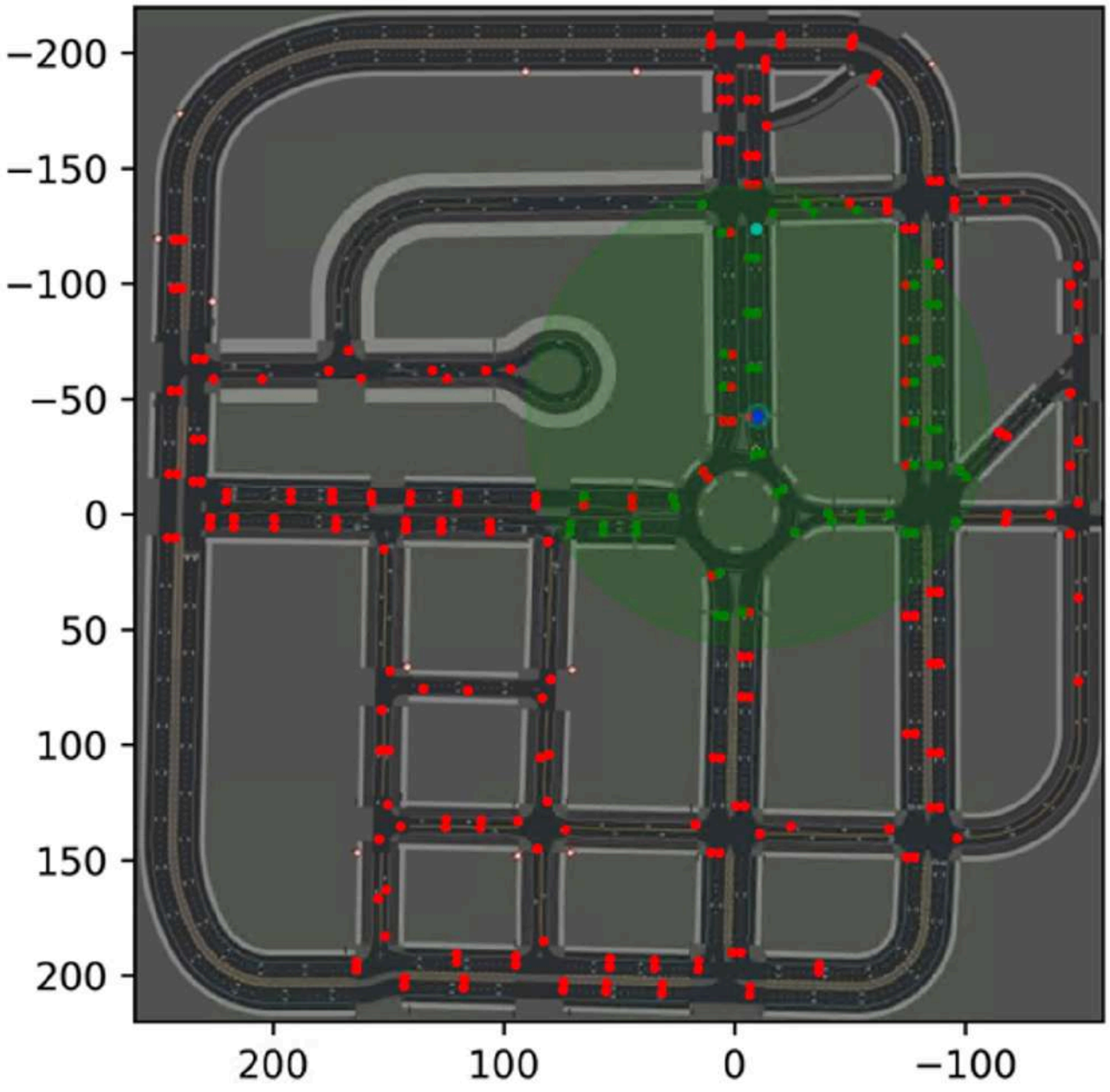


Figura 30: Town3

5.3 Scenario Runner e Carla

Come già detto nei capitoli precedenti, Scenario Runner è il modulo che ci permette l'esecuzione degli scenari automobilistici all'interno del simulatore CARLA. Per eseguire uno scenario basato sul formato OpenSCENARIO, si deve eseguire ScenarioRunner come segue:

python scenario_runner.py --openscenario <path / to / xosc-file>

Così sul server verrà eseguito lo scenario e muovendo la telecamera liberamente all'interno di Carla è possibile lo svolgimento dello scenario da qualsiasi punto di vista e vedere ogni volta il veicolo o i veicoli che vengono distrutti e creati nuovamente in un altra zona. L'obbiettivo però è quello di avere una camera fissa posta sulla parte frontale dell' Ego vehicle per questo è stato necessario eseguire il seguente comando mostrato sotto, in modo da creare un'istanza client di carla che permettesse di prendere possesso dell'Ego tramite una finestra pygame. python manual_control.py.
Quello che a noi interessa è poter eseguire una serie di file OpenScenario uno di seguito all'altro e per farlo sono stati creati degli script python che rendevano il processo automatico. E' possibile eseguire tutto con un unico comando specificando il nome dello scenario in esecuzione e il path della cartella che contiene i file degli scenari da eseguire:

python loop_scenario_runner.py SceanarioName <path / to / xosc-folder>

Lo script loop_scenario_runner.py esegue tramite subprocess gli script scenario_runner.py e manual_control.py in sequenza e li ripete per un numero di volte uguale al numero di file XOSC contenuti nel path specificato. Specificare il nome dello scenario come parametro è fondamentale per permettere il salvataggio corretto del video che viene registrato ad ogni esecuzione e questo ci sarà utile in secondo momento per il riconoscimento degli scenari.
Gli script più importanti sono:
• scenario_runner.py: è lo script principale da eseguire durante l'esecuzione di uno scenario. Esegue nell'ordine:

- Il caricamento della configurazione dello scenario.
- Il caricamento dello scenario e del manager(classe di gestione).
- L'attivazione dell'esecuzione dello scenario scelto.

• openscenario_parser.py: è il modulo che fornisce un parser per i file di configurazione dello scenario basato su formato OpenSCENARIO.
• scenario_manager.py: Implementa ScenarioManager, la classe di gestione degli scenari. Questa classe contiene tutte le funzionalità necessarie per iniziare, eseguire e analizzare uno scenario.
Alcune modifiche a questi script sono risultate necessarie per realizzare funzionalità mancanti o non ancora supportate da Carla e OpenScenario. Uno dei punti fondamentali per uno svolgimento più realistico degli scenari è stato l'utilizzo dell'autopilot. L'autopilot ci ha

aiutato ad evitare collisioni involontarie con altri veicoli o pedoni e rispettare la segnaletica stradale. Questo non succede se al posto dell'autopilot viene data direttamente una velocità ai veicoli, infatti così non viene rispettata la segnaletica e nel caso di veicoli davanti, questi non si fermano portando a delle collisioni.
Un'altra problematica dell'utilizzo dell'autopilot è che esso può prendere decisioni in autonomia e quindi può succedere che un veicolo decida autonomamente di cambiare corsia. Questo problema è stato risolto imponendo un vincolo all'autopilot, ovvero non dare la possibilità a tutti i veicoli con autopilot di fare cambio di corsia a meno che non siamo noi a specificarlo. L'ennesimo problema dell'autopilot è stato quello delle velocità, infatti tutti i veicoli sotto l'azione dell'autopilot raggiungono una velocità e la mantengono. Se gli scenari sono lunghi, i veicoli raggiungono tutti la stessa velocità e si crea un effetto di movimento a blocco dei veicoli poco realistico. Per ovviare a questo problema sono state fatte alcune modifiche ad un modulo di python che è il traffic_Manager per permettere ai veicoli di muoversi in maniera più dinamica. Il Traffic_manager permette di modificare alcune azioni dell'autopilot come la lista di veicoli che guidano ad una percentuale di velocità maggiore o minore del limite di velocità o la distanza minima da tenere tra un veicolo e l'altro.

Figura 31 Esempio di inquadratura

Un aspetto importante di Carla è la possibilità di avere diverse inquadrature del veicolo e noi possiamo decidere di metterne una e quando parte la registrazione dello scenario, viene registrata con l'inquadratura scelta. Un esempio è nella Figura [31] che come possiamo vedere ha un inquadratura dall'alto del veicolo permettendoci di vedere meglio anche il comportamento degli altri veicoli.

Però l'inquadratura migliore è quella in cui abbiamo la telecamera sulla parte frontale del veicolo come possiamo vedere nella Figura[32] in modo da poterci permettere di catturare le immagini migliori per definire in un secondo momento il riconoscimento degli scenari. Per la realizzazione dei video è stata utilizzata l'implementazione Opencv di python, in

particolare opencv.VideoWriter che permette, a partire da immagini, nel nostro caso quelle prodotte da pygame.surfarray ogni tick, di costruire un video e salvarlo.

Figura 32: Esempio di telecamera frontale

RICONOSCIMENTO SCENARI 6

6.1 Dataset prodotto

Una volta essere riusciti ad implementare tutto, si è passati alla generazione del dataset. Si è cercato di creare un dataset che potesse essere il più vario possibile. Ricordando che le classi di scenario che sono state utilizzate sono:

- AppoachingStaticObject
- ChangeLane
- ChangeLane2
- CutIn
- FollowLeadingVheicle
- Freeride
- Pedestrian
- Traffic

Lo scopo, oltre a quello di creare un dataset di modeste dimensioni (creare un dataset di grandi dimensioni avrebbe richiesto molto tempo), è stato anche quello di lavorare molto sulla lista di parametri del file .json citato nel Capitolo 5 in modo da ottenere scenari diversi tra di loro. Per ogni classe di scenario sono stati generati 600 scenari (eccetto per il Pedestrian che alla fine non è stato generato a causa di problemi dovuti alla versione di Carla troppo vecchia) per un totale di 6600 video ai quali si aggiungono altri 600 video che spiegheremo tra poco per arrivare infine ad un dataset di 7200 video.

Cambiando i diversi parametri della lista come le ore del giorno, diverse condizioni meteo, traffico più o meno intenso etc. abbiamo generato un dataset molto vario.

Principalmente per ogni classe di scenario abbiamo ottenuto 600 video di cui 200 video di giorno(alle 12:00 perché abbiamo il massimo della luce e questo è fattore molto importante per vedere quanto la luce influenzi il riconoscimento degli scenari successivamente), 200 video di sera (alle ore 21:00) e 200 video di giorno ma con pioggia e nebbia.

Dato che lo scopo finale è testare il dataset attraverso le reti neurali per verificarne il corretto riconoscimento è stato deciso di dividere il dataset in 2 parti perché si è ritenuto che alcuni scenari avessero bisogno di pochi secondi mentre altri di qualcuno in più per il riconoscimento:

- *Approach*, *FollowLeadingVheicle, Traffic ChangeLane*, *ChangeLane2*, *CutIn* e *Freeride* per un totale di 4200 video da 4 secondi ciascuno. Abbiamo scelto 4 secondi perché è un numero giusto di secondi che permette alla rete di riconoscerli.

- *ChangeLane*, *ChangeLane2*, *CutIn* e *Freeride* per un totale di 2400 video da 2 secondi ciascuno. Qui invece, è sono stati scelti solo 2 secondi perché è un numero giusto di secondi che permette alla rete di riconoscerli in quanto diversi tra loro.

Inoltre per i video da 2 secondi abbiamo creato un'ulteriore classe chiamata *Other*, contenente altri 600 video tra *Approach*, *FollowLeadingVheicle* e *Traffic.* Quindi l'ammontare del numero di video da 2 secondi è di 3000 video.

La generazione del dataset ha richiesto circa 9 giorni di lavoro, con una media di circa 800 video generati in 10 ore al giorno. Le tabelle riportate qua sotto, mostrano quali sono le caratteristiche per la generazione dei dataset

Dataset 4 sec	Numero totale	Giorno	Notte	Pioggia e Nebbia	Filtraggio %	Tempo
Approach	600	200	200	200	~10%	8 ore
Follow	600	200	200	200	~10%	8 ore
Traffic	600	200	200	200	~10%	8/9 ore
ChangeLane	600	200	200	200	~10%	8 ore
Overtake	600	200	200	200	~10%	8 ore
CutIn	600	200	200	200	~15%	7 ore
Freeride	600	200	200	200	~1%	5/6 ore

Dataset 2 sec	Numero totale	Giorno	Notte	Pioggia e Nebbia	Filtraggio %	Tempo
ChangeLane	600	200	200	200	~10%	7 ore
Overtake	600	200	200	200	~10%	7 ore
CutIn	600	200	200	200	~15%	7 ore
Freeride	600	200	200	200	~1%	5/6 ore
Other	600	200	200	200	~10%	7 ore

I parametri che influenzano il tempo di generazione delle classi di scenari sono:

- Numero di scenari che si vogliono generare.

- Numero di veicoli generati: Maggiore è il numero di veicoli, maggiore sarà il tempo che viene impiegato per distruggerli, generarli e assegnare ad ognuno le azioni da compiere.

- Percentuale di filtraggio: Il caso del CutIn è quello maggiore perché essendoci diversi tipi di tagli della strada, dai più netti a quelli più smooth, nel caso di quelli più netti se il veicolo dell'avversary scelto randomicamente è lungo, rischia di toccare il veicolo dell'ego.

I risultati della tabella, confrontati con quelli del dataset di partenza per certi aspetti sono migliori. Infatti siamo passati da un filtraggio di circa il 50% per la generazione del dataset precedente, ad ora che applichiamo un filtraggio massimo che è di circa il 15% per il CutIn. Anche se il filtraggio è diminuito, i tempi di generazione sono rimasti quasi li stessi e questo è dovuto prevalentemente al modo in cui vengono generati degli npc. Ora, come spiegato precedentemente, gli npc non vengono più generati tramite gli spawn point, ma mediante l'utilizzo degli waypoint. Questa tecnica, oltre a permetterci di generare npc in qualsiasi punto della mappa, ci permette di crearne quanti ne vogliamo e, di conseguenza, maggiore è il numero di veicoli da generare, maggiore sarà il tempo che Carla impiegherà a eliminare tutti i veicoli e generarli nuovamente per lo scenario successivo (con gli spawn point invece si poteva creare un numero massimo di npc, nelle zone dove ovviamente era possibile farlo).

6.2 Reti utilizzate e risultati

Come già accennato più volte, per testare la variabilità dei dataset che vengono creati, è necessario utilizzare le reti neurali. Per apprendere informazioni significative di alto livello dai dati, i sistemi ML supervisionati richiedono dataset contenenti campioni etichettati. In particolare l'estrazione di informazioni da video tramite reti neurali (NN) è stata oggetto di numerosi lavori. Le reti neurali convoluzionali 2D (CNN), sono adatte per il rilevamento di oggetti ma non sono sufficienti per l'elaborazione di video, poiché non modellano informazioni temporali e schemi di movimento. Di conseguenza, quelle in grado di fornirci maggiori informazioni sono quelle 3D. Nel nostro caso particolare, ho deciso di addestrare i due dataset ottenuti utilizzando[53] una rete 3D e una rete (2+1)D su 4 condizioni (solo Daily, solo Night, solo F&R e Total che comprende tutte e tre i casi insieme). La differenza della (2+1)D rispetto alla 3D è che la (2+1)D fattorizza esplicitamente una convoluzione 3D in due operazioni separate e successive, una convoluzione spaziale 2D e una convoluzione temporale 1D. Gli aspetti sui quali ci siamo concentrati sono stati:

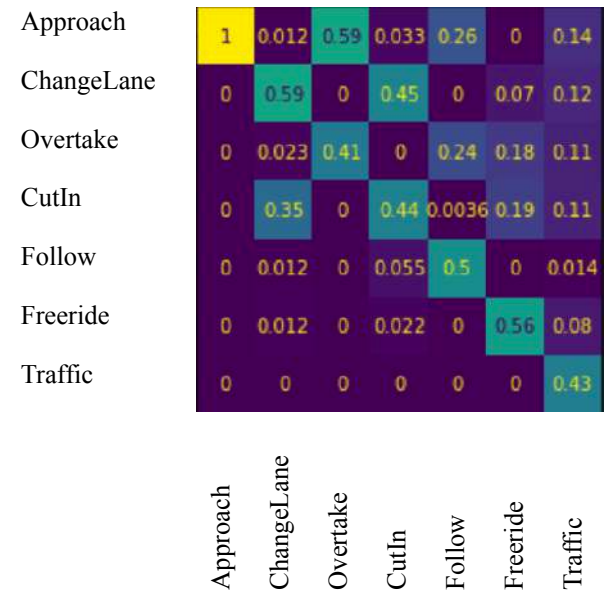
- Quanto influisce il numero di FPS in una clip sulle prestazioni degli NN
- Qual è l'effetto delle condizioni meteorologiche e di luce
- In che modo le prestazioni vengono influenzate dalla variazione della lunghezza del video clip in ingresso

Le due reti hanno i loro vantaggi a seconda del tipo di dataset che viene analizzato:

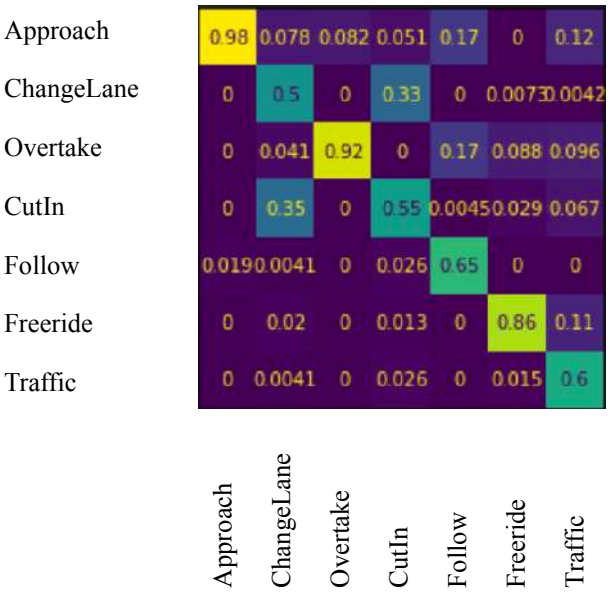
- Per video di breve durata e che cambiano velocemente, sembra più adatta la rete (2+1)D
- Per video di più lunga durata e che cambiano lentamente e con condizioni meteorologiche e luci peggiori, sembra più adatta la rete 3D

Quindi, in base al tipo di dataset generato, è stato ritenuto più logico addestrare la rete 3D per i video da 4 secondi e la (2+1)D per i video da 2 secondi (tenendo conto che in entrambe le tipologie di video sono presenti anche condizioni atmosferiche e di luce più scarse). Qui sotto vengono riportati i risultati descritti da 3 confusion matrix ottenute nel caso Total a 1 FPS.

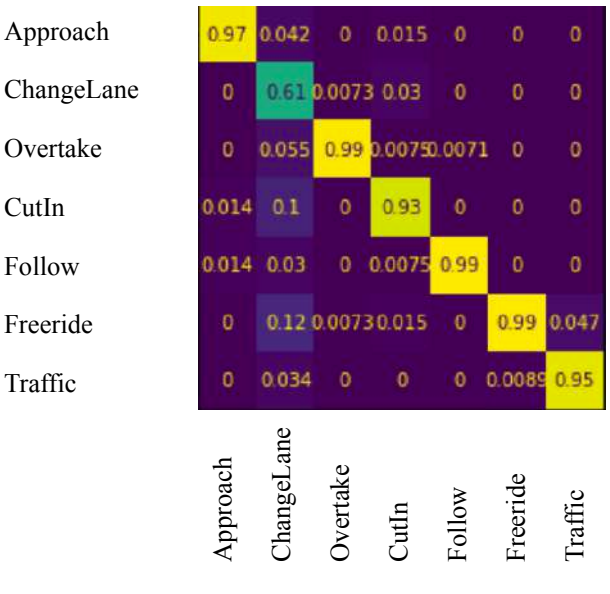
3D: 1 sec 4 FPS
Accuracy: 0.49
Precision: 0.56



3D: 2 sec 2 FPS
Accuracy: 0.66
Precision: 0.72

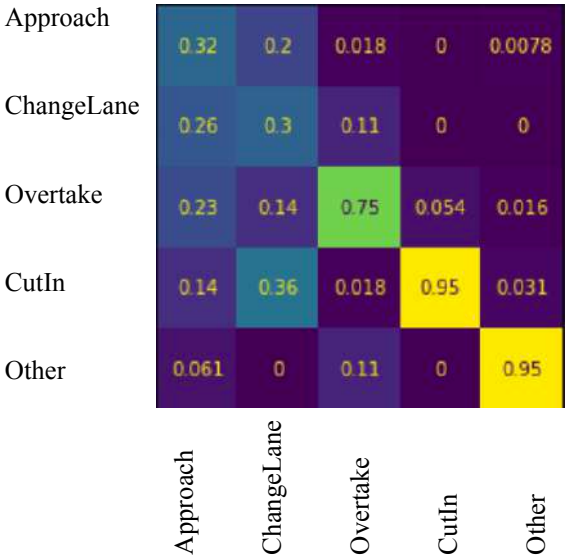


3D: 4 sec 1 FPS
Accuracy: 0.88
Precision: 0.91

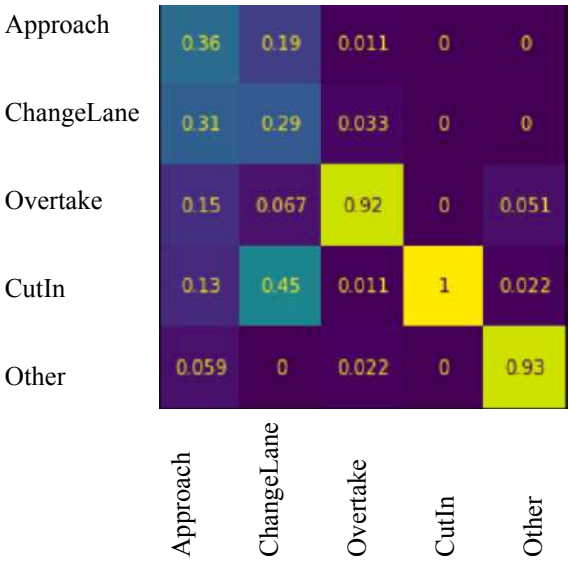


Questi qui sopra sono 3 risultati nel caso della rete 3D ottenuti considerando 1,2 e 4 secondi a 1 FPS e, con queste condizioni, l'accuratezza e la precisione aumentano all'aumentare dei secondi considerati. Avendo a disposizione 4 secondi di scenario la rete, anche a pochi frame, riesce a raggiungere livelli di *Precision* e *Accuracy* abbastanza alte. I risultati dell'addestramento della rete (2+1)D per Total a 1 FPS sono invece riportati qui sotto:

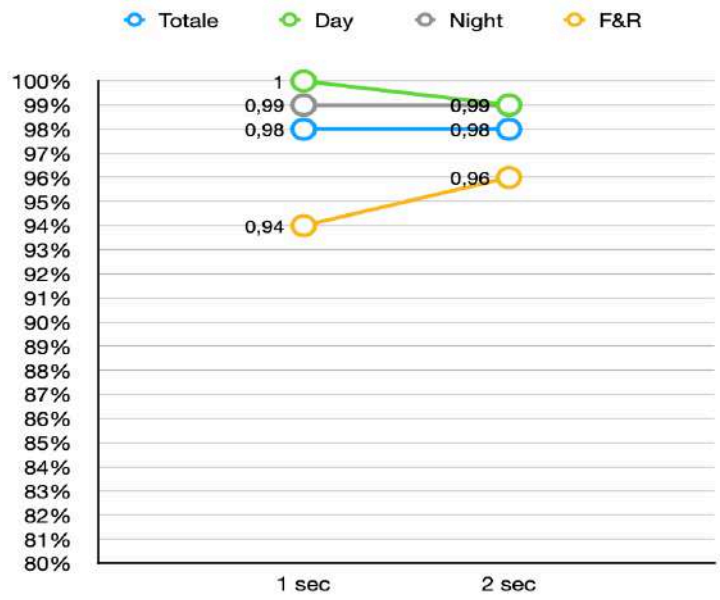
(2+1)D: 1 sec 2 FPS
Accuracy: 0.48
Preision: 0.65



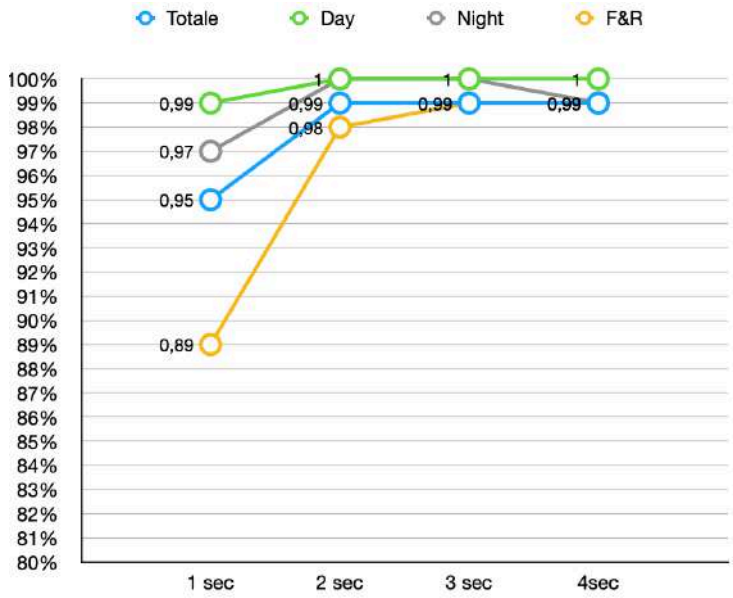
(2+1)D: 2 sec 1 FPS
Accuracy: 0.54
Preision: 0.70



In questo caso la rete, considerando solo 2 secondi, a bassi frame non riesce a capire di che scenario si tratti ma, aumentando il numero di FPS, si sono ottenuti valori di *Accuracy* e *Precision* maggiori. A 3 FPS infatti la rete raggiunge già livelli di accuratezza superiori a 0.80.



CNN (2+1)D
Training: 2 sec
Test: 1,2 sec



CNN 3D
Training: 4 sec
Test: 1,2,3,4 sec

I seguenti grafici riportano i risultati delle performance in termini di accuratezza delle due reti convoluzionali addestrate con i nuovi dataset da 4 e 2 secondi. Come è possibile vedere le reti sono state testate a diverse condizioni meteo e di luce. Nel caso della (2+1)D, addestrata con il dataset da 2 secondi i video clip sono stati testati a 1 e 2 secondi, mentre nel caso della rete convoluzionale 3D, addestrata con il dataset da 4 secondi, i video clip sono stati testati da 1 a 4 secondi. Entrambe le reti hanno ottenuto ottime performance sia con il test sui video completi che anche con un solo secondo a disposizione risultando robuste anche alle diverse condizioni meteo e di luce

CONCLUSIONI E SVILUPPI FUTURI 7

Il tema della guida autonoma è uno dei più trattati e discussi degli ultimi anni da parte di ricercatori e case automobilistiche di tutte il mondo e così sarà anche in futuro.

Con questa tesi credo di aver contribuito al mondo della guida autonoma per arrivare un giorno ad un livello massimo di automazione dei veicoli stessi, permettendoci così di percorrere qualsiasi strada senza alcun intervento manuale da parte nostra. Si suppone però che per arrivare a raggiungere un traguardo simile ci vogliano ancora alcune decine di anni.

Essendo stato in grado di raggiungere ed ottenere l'obiettivo da me prefissato ho maturato un'esperienza e una consapevolezza della materia costruttiva ed ispirante. Lo studio mi ha permesso di migliorare lo strumento per la generazione di scenari automobilistici, dalla possibilità di generare quanti npc vogliamo attorno all'ego vehicle per creare traffico utilizzando gli waypoint al posto degli spawn point, fino alla modifica dei parametri già esistenti e l'inserimento di nuovi, oltre che all'aggiunta di due ulteriori scenari da me realizzati, permettendo così la creazione di dataset maggiormente parametrizzabili e contenenti scenari molto diversi anche all'interno di una stessa classe di scenario. Sono stati creati due dataset con 7 classi di scenari, uno da 4200 video della durata di 4 secondi e uno da 3000 video della durata di 2 secondi per un totale di 7200 video (solitamente vengono generati circa 800 video in una giornata di lavoro), passando da un filtraggio del 50% circa per la generazione del dataset precedente, ad un filtraggio massimo che è di circa il 15% per il CutIn.

Successivamente sono state utilizzate due reti convoluzionali che usano convoluzioni 3D una e (2+1)D l'altra, per eseguire una misura indiretta della bontà del dataset. Sono stati ottenuti ottimi risultati di predizione sulla rete(2+1)D addestrata sul dataset da 2 secondi, mentre la 3D è stata addestrata per valutare il dataset da 4 secondi con l'obbiettivo di classificare gli scenari.

Ciò nonostante bisognerebbe continuare verso questa strada, ampliando e migliorando maggiormente il dataset sulla base dell'evoluzione della mia tesi e in futuro si potrebbe provvedere all'introduzione di nuovi scenari, a migliorare ancora di più quelli già presenti e ad aumentare il numero di parametri da poter settare all'interno di uno scenario, come per esempio l'aggiunta di un parametro per il raggio di sterzata del veicolo in modo da non essere più fisso ma parametrizzabile. Inoltre si potrebbe utilizzare una versione più aggiornata di Carla (verificando che tutti gli scenari funzionino correttamente anche qui) e infine provare a utilizzare la nuova versione di OpenScenario chiamata OpenScenario 2.0, che consente consente una rivoluzione in termini di automazione, facilità d'uso, accuratezza, produttività e misurabilità e che verrà rilasciata questo Dicembre 2021.

BIBLIOGRAFIA

[1] Synopsys. The 6 Levels of Vehicle Autonomy Explained. Url: https://www.synopsys.com/automotive/autonomous-driving-levels.html.

[2] David Morris, Garikayi Madzudzo and Alexeis Garcia-Perez. in InderScienceOnline: Cybersecurity and the auto industry: the growing challenges presented by connected cars, Vol. 18, No. 2, Abstract, Jun 2018.

[3] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse MonoSLAM: Real-time single camera SLAM. In: IEEE Trans. Pattern Anal. Mach. Intell., vol. 29, no. 6, pp. 1052–1067, Jun. 2007.

[4] C. Cadena Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. In: IEEE Trans. Robot., vol. 32, no. 6, pp. 1309–1332, Dec. 2016.

[5] Q. Li, L. Chen, M. Li, S.-L. Shaw, and A. Nuchter A sensorfusion drivable-region and lane-detection system for autonomous vehicle navigation in challenging road scenarios. In: IEEE Trans. Veh. Technol., vol. 63, no. 2, pp. 540–555, Feb. 2014.

[6] D. González, J. Pérez, V. Milanés, and F. Nashashibi A review of motion planning techniques for automated vehicles. In: IEEE Trans. Intell. Transp. Syst., vol. 17, no. 4, pp. 1135–1145, Apr. 2016.

[7] L. Chen, L. Fan, G. Xie, K. Huang, and A. Nüchter Moving-object detection from consecutive stereo pairs using slanted plane smoothing. In: IEEE Trans. Intell. Transp. Syst., vol. 18, no. 11, pp. 3093–3102 Nov. 2017.

[8] Bonnefon, Jean-François, Azim Shariff, and Iyad Rahwan. The social dilemma of autonomous vehicles. In Science 352.6293 (2016): 1573-1576.

[9] J.Wei,J.M.Snider,T.Gu,J.M.Dolan,andB.LitkouhiAbehavioralplanningframework for autonomous driving. In: IEEE Intell. Vehicles Symp., Jun. 2014, pp. 458–464.

[10] L. Chen, X. Hu, T. Xu, H. Kuang, and Q. Li Turn signal detection during nighttime by CNN detector and perceptual hashing tracking. In: IEEE Trans. Intell. Transp. Syst., vol. 18, no. 12, pp. 3303–3314, Dec. 2017.

[11] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, “The Cityscapes Dataset for Semantic Urban Scene Understanding,” in Proc. of CVPR, Las Vegas, NV, USA, 2016, pp. 3213–3223.

[12] F. Yu, W. Xian, Y. Chen, F. Liu, M. Liao, V. Madhavan, and T. Darrell, “BDD100K: A Diverse Driving Video Database With Scalable Annotation Tooling,” arXiv preprint arXiv:1805.04687, 2018.

[13] A. Breuer, J.-A. Term¨ohlen, S. Homoceanu, and T. Fingscheidt “openDD: A Large-Scale Roundabout Drone Dataset,” in Proc. Of ITSC, Rhodes, Greece, 2020, pp. 1–6.

[14] G. Neuhold, T. Ollmann, S. R. Bul`o, and P. Kontschieder, “The Mapillary Vistas Dataset for Semantic Understanding of Street Scenes,” in Proc. of ICCV, Venice, Italy, 2017, pp. 4990–4999.

[15] K. Lis, K. Nakka, P. Fua, and M. Salzmann, “Detecting the Unexpected via Image Resynthesis,” in Proc. of ICCV, Seoul, Korea, 2019, pp. 2152–2161.

[16] P. Pinggera, S. Ramos, S. Gehrig, U. Franke, C. Rother, and R. Mester, “Lost and Found: Detecting Small Road Hazards for Self-Driving Vehicles,” in Proc. of IROS, Daejeon, South Korea, 2016, pp. 1099– 1106.

[17] H. Blum, P.-E. Sarlin, J. Nieto, R. Siegwart, and C. Cadena, “The Fishyscapes Benchmark: Measuring Blind Spots in Semantic Segmentation,” arXiv preprint arXiv:1904.03215, 2019.

[18] M. Braun, S. Krebs, F. Flohr, and D. M. Gavrila, “EuroCity Persons: A Novel Benchmark for Person Detection in Traffic Scenes,” IEEE Trans. on PAMI, vol. 41, no. 8, pp. 1844–1861, 2019.

[19] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision Meets Robotics: The KITTI Dataset,” IJRR, vol. 32, no. 11, pp. 1231–1237, 2013.

[20] X. Huang, X. Cheng, Q. Geng, B. Cao, D. Zhou, P. Wang, Y. Lin, and R. Yang, “The ApolloScape Dataset for Autonomous Driving,” in Proc. of CVPR, Workshop, Salt Lake City, UT, USA, 2018, pp. 1067–1073.

[21] J. Geyer, Y. Kassahun, M. Mahmudi, X. Ricou, R. Durgesh, A. S. Chung, L. Hauswald, V. H. Pham, M. M¨uhlegg, S. Dorn, T. Fernandez, M. J¨anicke, S. Mirashi, C. Savani, M. Sturm, O. Vorobiov, M. Oelker, S. Garreis, and P. Schuberth, “A2D2: Audi Autonomous Driving Dataset,” arXiv preprint arXiv:2004.06320, 2020. [Online]. Available: https://www.a2d2.audi

[22] “Waymo Open Dataset: An Autonomous Driving Dataset,” 2019, accessed 22-Jan.-2021. [Online]. Available: https://www.waymo.com/ open

[23] “PandaSet: Public Large-scale Dataset for Autonomous Driving,” 2019, accessed 22-Jan.- 2021. [Online]. Available: https://scale.com/open-datasets/pandaset

[24] J. Jeong, Y. Cho, Y.-S. Shin, H. Roh, and A. Kim, “Complex Urban Dataset with Multi-level Sensors from Highly Diverse Urban Environments,” IJRR, vol. 38, no. 6, pp. 642–657, 2019.

[25] M. Pitropov, D. Garcia, J. Rebello, M. Smart, C. Wang, K. Czarnecki, and S. Waslander, “Canadian Adverse Driving Conditions Dataset,” IJRR, pp. 1–10, 2020.

[26] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, “nuScenes: A Multimodal Dataset for Autonomous Driving,” in Proc. of CVPR, Seattle, WA, USA, 2020, pp. 11 618–11 628.

[27] M. Meyer and G. Kuschk, “Automotive Radar Dataset for Deep Learning Based 3D Object Detection,” in Proc. of EuRAD, Paris, France, 2019, pp. 129–132.

[28] W. Maddern, G. Pascoe, C. Linegar, and P. Newman, “1 Year, 1000km: The Oxford RobotCar Dataset,” IJRR, vol. 36, no. 1, pp. 3–15, 2017.

[29] M. Sheeny, E. De Pellegrin, S. Mukherjee, A. Ahrabian, S. Wang, and A. Wallace, “RADIATE: A Radar Dataset for Automotive Perception,” arXiv preprint arXiv:2010.09076, pp. 1–15, 2020.

[30] https://www.docenti.unina.it/webdocenti-be/allegati/materiale-didattico/567538

[31] Halil Beglerovic , Thomas Schloemicher , Steffen Metzner and Martin Horn. Deep Lear- ning Applied to Scenario Classification for Lane-Keep-Assist Systems. In: Appl. Sci. 2018, 8(12), 2590.

[32] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei ImageNet:A large-scale hierarchical image database. In: Proc. Eur. Conf. Comput. Vis., Jun. 2009, pp. 248–255.

[33] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba Places:A 10 million image database for scene recognition. In: IEEE Trans. Pattern Anal. Mach. Intell., vol. 40, no. 6, pp. 1452–1464, Jun. 2018.

[34] S. Ren, K. He, R. Girshick, and J. Sun Faster R-CNN: Towards real-time object detection with region proposal networks. In: IEEE Trans.Pattern Anal. Mach. Intell., vol. 39, no. 6, pp. 1137–1149, Jun. 2017

[35] Yasasa Abeysirigoonawardena; Florian Shkurti; Gregory Dudek for Generating Adversarial Driving Scenarios in High-Fidelity Simulators in: https://ieeexplore.ieee.org/abstract/document/8793740

[36] Thorn, Eric, Kimmel, Shawn C, Chaka, Michelle, Hamilton, Booz Allen. A framework for automated driving system testable cases and scenarios.. In: Tech.rep. United States. Department of Transportation. National Highway Traffic Safety, (2018).

[37] Heidecker, Florian et al. “An Application-Driven Conceptualization of Corner Cases for Perception in Highly Automated Driving.” ArXiv abs/2103.03678 (2021): n. pag

[38] C. B. Kuhn, M. Hofbauer, G. Petrovic and E. Steinbach, "Introspective Black Box Failure Prediction for Autonomous Driving," 2020 IEEE Intelligent Vehicles Symposium (IV), 2020, pp. 1907-1913, doi: 10.1109/IV47402.2020.9304844.

[39] Xinhai Zhang, Jianbo Tao, Kaige Tan, Martin Törngren, José Manuel Gaspar Sánchez, Muhammad Rusyadi Ramli, Xin Tao, Magnus Gyllenhammar, Franz Wotawa, Naveen Mohan, Mihai Nica, Hermann Felbinger et al. “Finding Critical Scenarios for Automated Driving Systems: A Systematic Literature Revie” arXiv:2110.08664 [cs.SE]

[40] J. Breitenstein, J. -A. Termöhlen, D. Lipinski and T. Fingscheidt, "Systematization of Corner Cases for Visual Perception in Automated Driving," 2020 IEEE Intelligent Vehicles Symposium (IV), 2020, pp. 1257-1264, doi: 10.1109/IV47402.2020.9304789.

[41] E. Protopapadakis, A. Voulodimos, A. Doulamis, N. Doulamis, D. Dres, and M. Bimpas, “Stacked Autoencoders for Outlier Detection in Over-the-Horizon Radar Signals,” Computational Intelligence and Neuroscience, vol. 2017, pp. 1–11, 2017.

[42] R. V. Chakravarthy, H. Liu, and A. M. Pavy, “Open-Set Radar Waveform Classification:Comparison of Different Features and Classifiers,” in Proc. of the IEEE International Radar Conference, Washington, DC, USA, 2020, pp. 542 – 547.

[43] K. Lis, K. Nakka, P. Fua, and M. Salzmann, “Detecting the Unexpected via Image Resynthesis,” in Proc. of ICCV, Seoul, Korea, 2019, pp. 2152–2161.

[44] K. Wong, S. Wang, M. Ren, M. Liang, and R. Urtasun, “Identifying Unknown Instances for Autonomous Driving,” in CoRL, Osaka, Japan, 2019, pp. 1–10.

[45] E. Capellier, F. Davoine, V. Cherfaoui, and Y. Li, “Evidential Deep Learning for Arbitrary LIDAR Object Classification in the Context of Autonomous Driving,” in Proc. of IV, Paris, France, 2019, pp. 1304 – 1311.

[46] J.-A. Bolte, A. B¨ar, D. Lipinski, and T. Fingscheidt, “Towards Corner Case Detection for Autonomous Driving,” in Proc. of IV, Paris, France, 2019, pp. 438–445

[47] https://it.wikipedia.org/wiki/Simulazione

[48] OpenDrive ASAM OpenDrive. https : https://www.asam.net/standards/detail/opendrive/.

[49] OpenScenarioASAMOpenScenario.https:https://www.asam.net/standards/detail/openscenario/.

[50] https://carla.readthedocs.io/en/latest/start_introduction/#the-simulator

[51] https://carla.readthedocs.io/en/latest/core_concepts/

[52] https://github.com/pyoscx/scenariogeneration

[53] Marianna Cossu, Jorge Leonardo Quimi Villon, Francesco Bellotti[1], Alessio Capello, Alessandro De Gloria, Luca Lazzaroni, Riccardo Berta “Classifying simulated driving scenarios from automated cars ”