



UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

ANALISI DI SISTEMI A GUIDA AUTONOMA IN AMBIENTE SIMULATO

RELATORE

Prof. Dario Di Nucci

CANDIDATO

Mihai Alexandru Burlacu

Matricola: 0512106533

C'è vero progresso solo quando i vantaggi di una nuova tecnologia diventano per tutti.

Henry Ford

Sommario

Negli ultimi anni l'interesse verso i sistemi guida autonoma è aumentato considerevolmente, anche per merito di aziende di punta in questo settore che sono entrate nella cultura di massa. Uno dei principali strumenti di questa rivoluzione nel campo dei trasporti sono gli Advanced Driver Assistance Systems (ADAS). Questi sistemi comprendono tecnologie hardware e software il cui obiettivo è quello di minimizzare e correggere l'errore del conducente negli istanti di guida e di parcheggio, utilizzando sensori e radar per analizzare ed elaborare l'ambiente esterno. I modelli software che gestiscono la componente decisionale degli ADAS necessitano di essere allenati e testati in modo esaustivo per accertarne un corretto funzionamento. Questo è difficile nell'ambiente reale per motivi di sicurezza e mancanza di risorse, per cui sono impiegati ambienti di co-simulazione, in cui in caso di errori, non viene causato alcun danno. L'obiettivo di questa tesi è assemblare strumenti software per sviluppare un ambiente simulato in cui analizzare il comportamento di ADAS all'interno di scenari di guida realistici. Gli strumenti presi in analisi sono: Automated Driving Toolbox di MathWorks, Carla Simulator e Udacity Simulator. Tra questi, Udacity Simulator è stato giudicato il tool più user friendly e pronto all'uso. In questo ambiente è stato possibile allenare e testare con successo dei modelli di guida autonoma a cui è stato richiesto di guidare una vettura all'interno di alcuni circuiti ricevendo come input le immagini di tre camere a bordo. Questi modelli sono basati su reti neurali convoluzionali progettate con Tensorflow e Keras.

Indice

Indice	ii
Elenco delle figure	iii
1 Introduzione	1
1.1 Motivazioni e Obiettivi	1
1.2 Risultati	2
1.3 Struttura della tesi	3
2 Stato dell'arte	4
2.1 Analisi degli standard relativi ad Automated Driving System	4
2.2 Strumenti per la co-simulazione di scenari di test per Automated Driving System	14
3 Generazione di scenari di co-simulazione di casi di test per ADS	17
3.1 MathWorks - Automated Driving Toolbox	17
3.2 CARLA Simulator	22
3.3 Udacity's Self-Driving Car Simulator	24
4 Conclusioni e sviluppi futuri	29
Bibliografia	31

Elenco delle figure

2.1	Visione schematica delle funzioni di guida che mette in risalto il ruolo del DDT. ¹	7
2.2	Use case sequence al livello 3 che mostra il coinvolgimento di un ADS, il veicolo che ha un malfunzionamento e il conducente che ne prende il controllo. ¹	8
2.3	Use case sequence al livello 3 che mostra il coinvolgimento di un ADS, l'ADS che ha un malfunzionamento e il conducente che prende il controllo del veicolo. ¹	8
2.4	Use case sequence al livello 3 che mostra il coinvolgimento di un ADS, l'uscita dall'ODD e il conducente che prende il controllo del veicolo. ¹	9
2.5	Use case sequence al livello 4 che mostra il coinvolgimento di un ADS, l'ADS che ha un malfunzionamento e il sistema che raggiunge una condizione di rischio minimale. Le linee tratteggiate sono opzionali. ¹	9
2.6	Use case sequence al livello 4 che mostra il coinvolgimento di un ADS che sta per uscire dall'ODD e il sistema che raggiunge una condizione di rischio minimale. Le linee tratteggiate sono opzionali. ¹	10
2.7	Semplice diagramma di flusso per suddividere i livelli di automazione ¹	11
2.8	Use case sequence per una funzione di livello 3 che mostra l'ADS inserito, il verificarsi di un guasto o di una condizione fuori dall' ODD, e il fallback-ready user che esegue il fallback, o, se il fallback-ready user non riesce a farlo, una strategia di mitigazione dell'errore, come l'arresto in corsia. ¹	13
2.9	Use case sequence al livello 4 che mostra ADS inserito, un evento catastrofico (ad esempio, un'interruzione completa dell'alimentazione) e il sistema che raggiunge una condizione di rischio minimale. ¹	13

2.10 Esempi di funzioni/tipi di sistemi di automazione della guida che potrebbero essere disponibili durante un determinato viaggio. ¹	14
3.1 Vista a volo d'aquila dello scenario di parcheggio. ³	18
3.2 vehicleCostMap. ³	19
3.3 Punti di passaggio. ³	19
3.4 Interpolazione. ³	20
3.5 Configurazione Simulink. ⁶	21
3.6 Lane Detector in MCity. ⁶	21
3.7 Schema ad alto livello del sistema DAVE-2. ¹¹	25
3.8 Editor di Unity. ¹⁴	27

CAPITOLO 1

Introduzione

1.1 Motivazioni e Obiettivi

Gli studi di Charlton e Starkey del 2011 [2] e di Roy e Liersch nel 2013 [6] hanno dimostrato che la maggior parte delle persone sono convinte di avere una maggiore competenza alla guida rispetto alla media: si tratta di un fenomeno diffuso in molti altri ambiti ed è noto come fallacia della Superiorità Illusoria. Questo tipo di sopravvalutazione delle proprie capacità ha, inevitabilmente, gravi conseguenze (più di 1,2 milioni di persone ogni anno perdono la vita a causa di incidenti stradali [5]) e, seppur inconsciamente, si estende definendosi come una sovrastima delle abilità dell'uomo.

La soluzione a questo problema è la completa o parziale automatizzazione del processo di guida tramite sistemi di assistenza all'autista chiamati ADAS e ADS. Essi sono un insieme di strumenti informatici ed elettronici atti ad assistere l'autista nella fase di guida e di parcheggio attraverso sensori esterni come camere, radar o LiDAR, o mediante il controllo vero e proprio della vettura in alcuni momenti. Per convenzione, con ADAS si indicano i sistemi di automazione di livello 1 e 2, e con ADS quelli di livello 3, 4 e 5; il concetto di livello di automazione e le differenze tra di essi verrà approfondito in seguito. Di fondamentale importanza è anche sensibilizzare le persone sull'importanza di mantenere l'attenzione alla guida e sulla pericolosità dell'uso di dispositivi cellulari [3].

Uno dei più grandi ostacoli alla diffusione dell’automazione è proprio la percezione negativa del pubblico nei confronti dell’automazione e delle AV in particolare [7]. Un caso storico analogo a questo si è verificato negli anni ’40 del XX secolo. Ricordando che fino ad allora tutti gli ascensori erano guidati da un operatore, l’introduzione, da parte della Otis Elevator, dell’ “Autotronic Elevating System” generò una paura collettiva nell’utilizzare gli ascensori. Il tutto si risolse con lo sciopero generale degli operatori di New York del 1945.

L’obiettivo di questo studio, nei limiti delle proprie competenze, è quelli di portare il proprio contributo nello sviluppo di sistemi a guida autonoma e dimostrare che possono essere affidabili almeno quanto un conducente umano; nella pratica, sarà analizzata l’efficienza di un sistema di guida autonoma in un environment virtuale che simuli il più possibile le caratteristiche dell’ambiente reale in cui il sistema verrà ipoteticamente utilizzato.

1.2 Risultati

Nel corso di questa tesi sono stati analizzati tre tool di simulazione: Automated Driving Toolbox, Carla Simulator e Udacity Simulator. Dopo un’attenta analisi, si è deciso che le prime due opzioni rendevano il processo di simulazione e di testing eccessivamente complesso, quindi la scelta del simulatore da usare per lo studio dei casi di test è ricaduta su Udacity Simulator. Sono stati presi in analisi due use case. Il primo richiede all’agente di guidare il veicolo all’interno di uno di due percorsi senza uscire fuori dalla carreggiata. Per allenare l’agente è stata usata una rete neurale convoluzionale allenata usando Tensorflow e Keras attraverso apprendimento supervisionato fornendo come valori in input la ripresa delle camere a bordo, e come output sterzo, accelerazione e freno. Il comportamento dell’agente è buono sul percorso su cui è stato allenato, tuttavia non si è riusciti ad allenare un modello che riuscisse a completarli entrambi. Si ipotizza che questa mancanza di generalizzazione sia imputabile alla scarsa variazione del dataset utilizzato oppure alla necessità di utilizzare un numero più alto di parametri per la rete neurale. Il secondo use case richiede all’agente di guidare su un’autostrada a velocità prossime al limite, superando i veicoli presenti quando è necessario e senza causare collisioni. Per questo scopo è stato utilizzato un modello già allenato fornito dalla stessa Udacity che si comporta come previsto.

1.3 Struttura della tesi

Nella prima sezione di questa tesi sono stati illustrati i motivi e gli obiettivi dietro ad essa. Nel secondo capitolo si esporranno le caratteristiche dei sistemi di automazione in termini più tecnici secondo gli standard internazionali, le differenze tra i vari livelli di automazione e i tool più utilizzati per simulare un ADAS. Nel terzo capitolo alcuni di questi software verranno analizzati in dettaglio per comprendere come utilizzarli nella pratica per simulare casi di test. Nel quarto e ultimo capitolo verranno illustrati possibili sviluppi futuri del sistema che è stato implementato.

CAPITOLO 2

Stato dell'arte

In questo capitolo verrà analizzato lo stato dell'arte nell'ambito dei sistemi di guida autonoma per meglio comprendere quanto il corrente stato della tecnologia sia lontano dalla possibilità di automatizzare completamente l'infrastruttura dei trasporti.

2.1 Analisi degli standard relativi ad Automated Driving System

Per riuscire a contestualizzare al meglio le informazioni che verranno a breve riportate, è utile mostrare che il grado di automazione di un sistema è convenzionalmente suddiviso in sei livelli. Essi sono descritti dettagliatamente dallo standard J3016 dell'ente normativo SAE International [8]. SAE International, precedentemente chiamata Society of Automotive Engineers, è un'associazione professionale con sede negli Stati Uniti, attiva a livello globale e un'organizzazione che sviluppa standard per i professionisti dell'ingegneria in vari settori. L'enfasi principale è posta sulle industrie di trasporto globale come l'aerospaziale, l'automotive e quella dei veicoli commerciali. Oltre ai suoi sforzi di standardizzazione, SAE International dedica anche risorse a progetti e programmi di educazione STEM, certificazione professionale e concorsi di design collegiali.

In questa sede si analizzeranno le differenze e le evoluzioni dello standard J3016, dovuto alla sempre crescente diffusione di queste tecnologie e a una conseguente necessità di standardizzazione. E' bene comunque precisare che i seguenti documenti siano da considerarsi descrittivi piuttosto che normativi, e tecnici piuttosto che legali.

Standard J3016 - 2014

Il documento del 2014 rappresenta la prima forma dello standard J3016. Esso fornisce una tassonomia per i veicoli a motore automatizzati che spaziano da totale assenza di automazione ad automazione completa. Tuttavia, soltanto i tre livelli più alti vengono descritti in modo dettagliato, mentre le definizioni dei livelli più bassi sono da intendersi come punti di riferimento per delineare i gradi di automazione. Verranno qui descritti i ruoli e le responsabilità del conducente e del sistema in funzione del livello di automazione.

Livello	Conducente	Sistema
Livello 0 Nessuna automazione	Monitora l'ambiente ed esegue le funzioni di guida dinamiche.	Non è dotato di alcuna funzione se non, optionalmente, quella di mostrare spie di avviso.
Livello 1 Guida Assistita	Esegue funzioni di guida longitudinali (accelerazione, freno) o laterali (sterzo) e decide quando attivare e disattivare l'assistenza di guida; deve comunque essere sempre in grado di prendere il controllo del veicolo in caso di necessità.	Può eseguire funzioni di guida parziali nel caso in cui venga attivato e disattivarsi immediatamente su richiesta del conducente.
Livello 2 Automazione Parziale	Monitora l'ambiente di guida e supervisiona le funzioni di guida eseguite dal sistema di automazione parziale. Inoltre, decide quando attivare e disattivare il sistema; deve comunque essere sempre in grado di prendere il controllo del veicolo in caso di necessità.	Può eseguire funzioni di guida sia longitudinali che laterali e può disattivarsi immediatamente su richiesta del conducente.

Livello 3 Automazione Condizionale	Decide quando è necessario attivare il sistema di guida autonoma; può comunque decidere di prendere il controllo della vettura dopo che la sua richiesta è stata accettata dal sistema.	Monitora l'ambiente di guida ed esegue funzioni di guida sia longitudinali che laterali. Potrebbe rinviare momentaneamente la disattivazione se un immediato passaggio di controllo al conducente possa compromettere la sicurezza del veicolo.
Livello 4 Alta Automazione	Decide quando è necessario attivare il sistema di guida autonoma; può comunque decidere di prendere il controllo della vettura dopo che la sua richiesta è stata accettata dal sistema.	Permette l'attivazione solo sotto particolari condizioni e permette la disattivazione solo quando queste condizioni non sussistono.
Livello 5 Automazione Completa	Potrebbe avere la possibilità di disattivare il sistema. E' contemplata la possibilità della totale assenza di un conducente umano.	Esegue tutte le funzioni di guida e si disattiva solo su richiesta del conducente o quando il veicolo è arrivato a destinazione.

Viene definita condizione di rischio minimale, una condizione di funzionamento a basso rischio di un veicolo a motore a cui un sistema di guida automatica ricorre automaticamente in caso di un guasto del sistema o l'incapacità del conducente umano di rispondere in modo appropriato alla richiesta di eseguire le funzioni di guida. I livelli sono da considerarsi mutualmente esclusivi.

Standard J3016 - 2016

Il documento del 2016 rappresenta un'importante evoluzione, introducendo nuovi concetti e approfondendo quelli già menzionati nel documento del 2014. L'attività di guida riguarda una varietà di azioni che vengono suddivise in tre categorie: strategiche, tattiche e operazionali [4]. Le strategiche riguardano la pianificazione del viaggio, quando e dove andare, come viaggiare e i migliori percorsi da percorrere. Le tattiche riguardano la guida del

veicolo, sorpassare, cambiare corsia, scegliere una velocità adeguata e così via. Le operazioni riguardano i riflessi, le reazioni e le scelte prese in una frazione di secondo che spaziano da micro allineamenti dello sterzo all'evitare improvvisi pericoli sulla carreggiata.

Questa nuova versione dello standard introduce il concetto di DDT, ovvero "Dynamic Driving Task". Si tratta dell'insieme delle operazioni in tempo reale e delle funzioni richieste per guidare un veicolo su una strada trafficata; include, quindi, solo le funzioni strategiche e tattiche. Altri elementi inclusi nel DDT possono essere:

1. Movimento laterale del veicolo mediante sterzo;
2. Movimento longitudinale del veicolo mediante accelerazione e decelerazione;
3. Monitoraggio dell'ambiente attraverso il rilevamento di oggetti, riconoscimento, classificazione e preparazione alla risposta;
4. Esecuzione della risposta a oggetti ed eventi;
5. Pianificazione di manovra.

Per semplicità, i punti (3) e (4) costituiscono l'OEDR, ovvero "Object and Event Detection and Response".

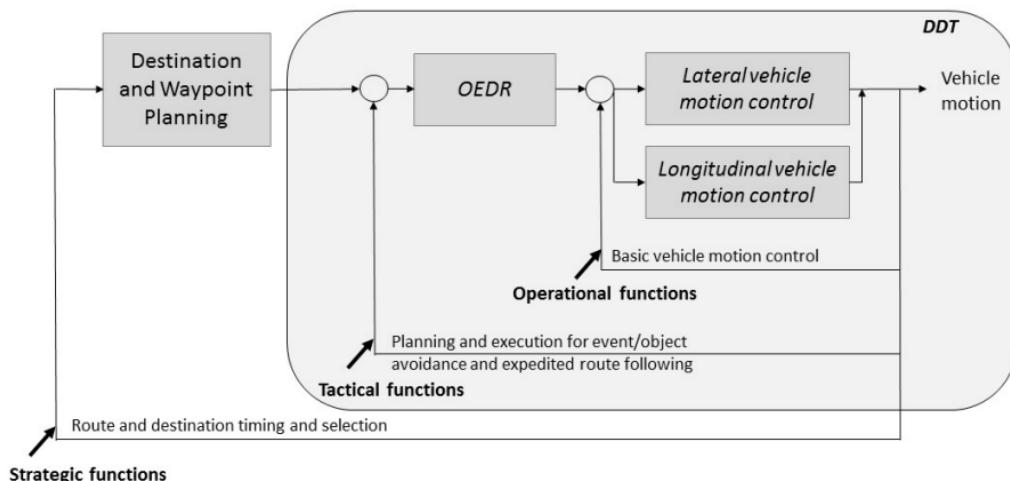


Figura 2.1: Visione schematica delle funzioni di guida che mette in risalto il ruolo del DDT.¹

Si definisce DDT fallback la reazione da parte del conducente o dell'ADS a prendere il controllo del DDT nel caso in cui si manifestassero fallimenti di sistema rilevanti a livello di performance dello stesso DDT. Il DDT e il DDT fallback sono due sistemi distinti: un sistema

¹Figura derivata da [8]

in grado di gestire un DDT potrebbe non essere dotato di fallback. Per esempio, un ADS di livello 3 che è in grado di eseguire il DDT all'interno del suo dominio di applicazione, potrebbe non essere in grado di eseguire fallback in ogni situazione in cui se ne presenta la necessità e quindi rende necessario l'intervento da parte del conducente. Al contrario, gli ADS di livello 4 e 5 dovrebbero essere sempre in grado di eseguire fallback.

Le seguenti figure mostrano vari tipi di fallback a seconda del livello di automazione.

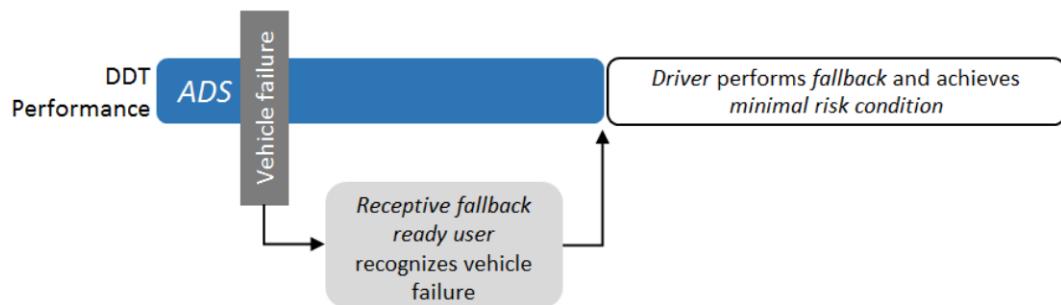


Figura 2.2: Use case sequence al livello 3 che mostra il coinvolgimento di un ADS, il veicolo che ha un malfunzionamento e il conducente che ne prende il controllo.¹

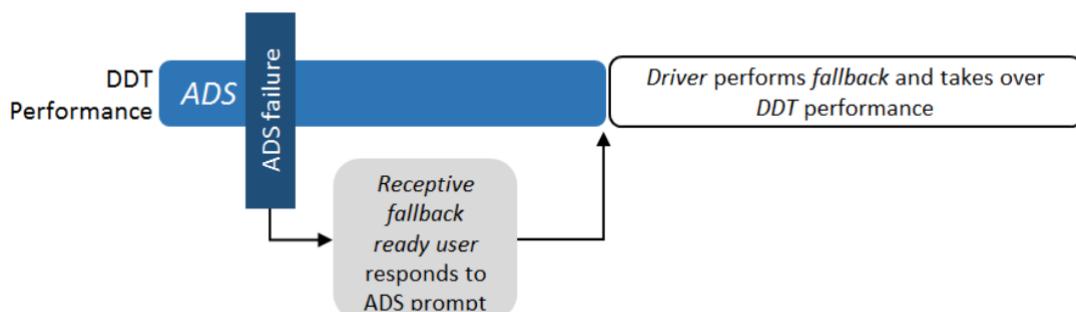


Figura 2.3: Use case sequence al livello 3 che mostra il coinvolgimento di un ADS, l'ADS che ha un malfunzionamento e il conducente che prende il controllo del veicolo.¹

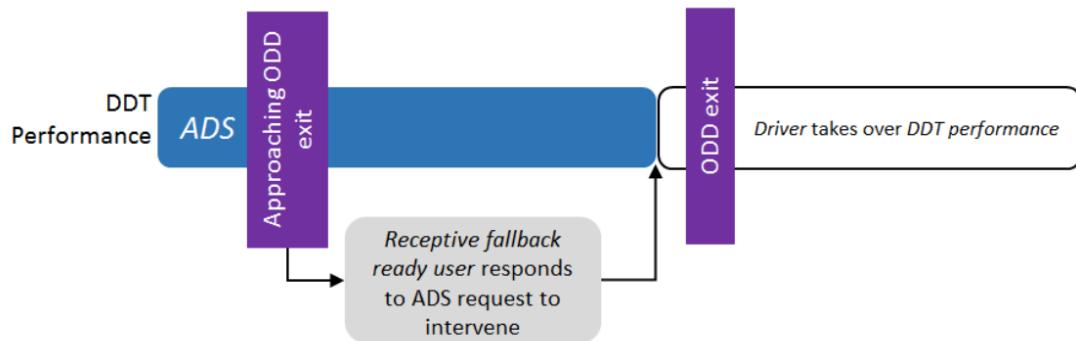


Figura 2.4: Use case sequence al livello 3 che mostra il coinvolgimento di un ADS, l’uscita dall’ODD e il conducente che prende il controllo del veicolo.¹

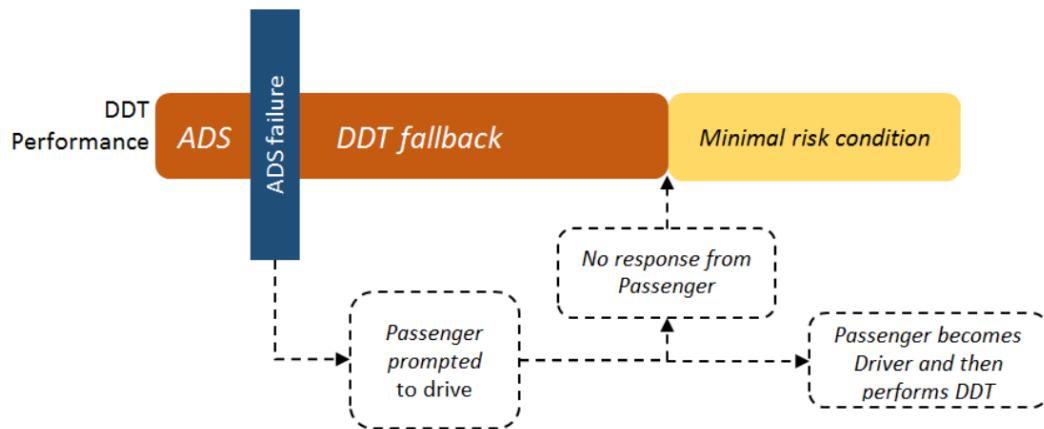


Figura 2.5: Use case sequence al livello 4 che mostra il coinvolgimento di un ADS, l’ADS che ha un malfunzionamento e il sistema che raggiunge una condizione di rischio minimale. Le linee tratteggiate sono opzionali.¹

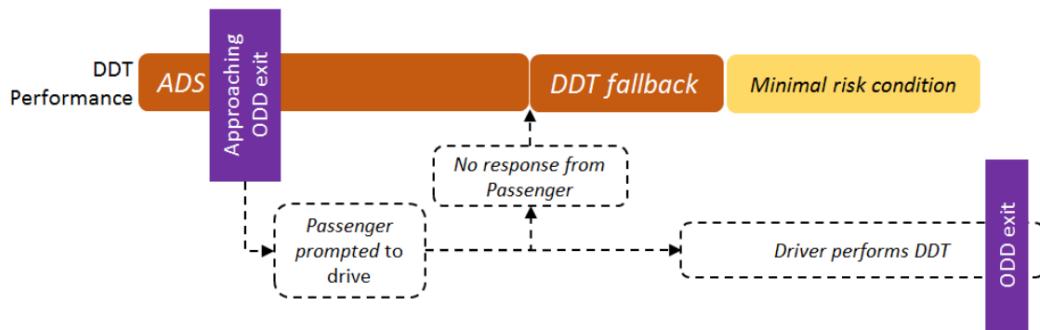


Figura 2.6: Use case sequence al livello 4 che mostra il coinvolgimento di un ADS che sta per uscire dall'ODD e il sistema che raggiunge una condizione di rischio minimale. Le linee tratteggiate sono opzionali.¹

Un altro importante concetto definito da questa versione è quello di ADS, ovvero Automated Driving System. Esso rappresenta l'insieme di hardware e software che insieme sono capaci di eseguire il DDT in modo continuativo, indipendentemente se sia limitato a uno specifico dominio di applicazione, ovvero le specifiche condizioni nelle quali un sistema di guida autonoma è progettato per lavorare; questo termine è usato in modo specifico per descrivere sistemi di guida autonoma di livello 3, 4 o 5. In contrapposizione a ADS, il termine generico "sistema di guida autonoma" si riferisce a sistemi di livello compreso tra 1 e 5 o capaci di eseguire parte o tutto il DDT in modo continuativo.

Il concetto di condizione di rischio minima viene ulteriormente approfondito specificando che si tratta di una condizione raggiunta generalmente dopo aver eseguito un DDT fallback. Ai livelli 1 e 2 il conducente potrebbe non essere in grado di raggiungere una condizione di rischio minima in seguito a un fallimento del sistema di guida autonoma. Al livello 3, dato un fallimento di sistema rilevante a livello di performance del DDT, l'utente dovrebbe essere in grado di eseguire un DDT fallback e raggiungere una condizione di rischio miniale quando lo ritiene opportuno. Ai livelli 4 e 5, l'ADS è in grado di raggiungere autonomamente una condizione di rischio miniale che può comportare una sosta di emergenza lungo il percorso prestabilito, una manovra più complessa atta a rimuovere il veicolo dalla carreggiata o farlo tornare alla struttura di spedizione.

Alla luce di questi nuovi concetti, la suddivisione in livelli in funzione del ruolo del sistema può essere schematizzata in modo più sintetico nel seguente modo.

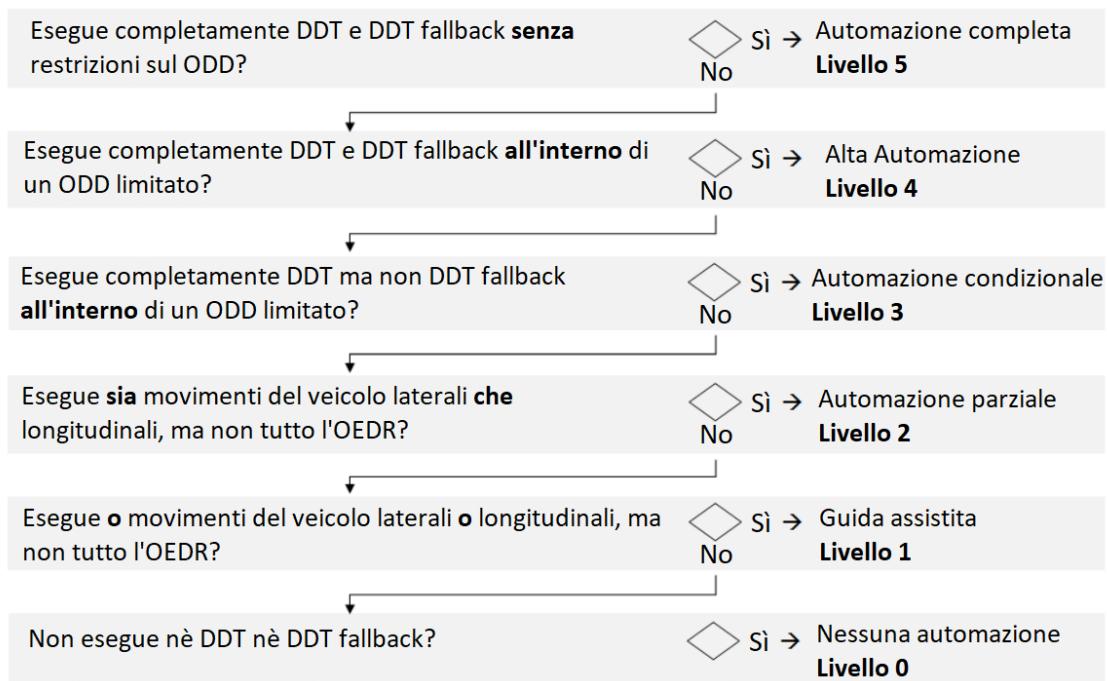


Figura 2.7: Semplice diagramma di flusso per suddividere i livelli di automazione¹

Si definisce fallback-ready user l’utente di un veicolo dotato di una funzione ADS di livello 3 inserita, che è in grado di guidare il veicolo ed è ricettivo alle richieste di intervento emesse dall’ADS e a evidenti guasti del sistema rilevanti per le prestazioni DDT nel veicolo che lo costringono a eseguire il DDT fallback.

Standard J3016 - 2018

La versione dello standard pubblicata nel 2018 si configura come un semplice aggiornamento della precedente introducendo molti meno concetti inediti rispetto a quest’ultima. Tra le principali terminologie introdotte ora:

Dispatching entity. Un’entità che invia veicoli dotati di ADS in funzionamento senza conducente. Le funzioni svolte da una dispatching entity possono essere suddivise tra uno o più agenti, a seconda della specifica d’uso del veicolo o dei veicoli dotati di ADS in questione.

Dispatch. Mettere in servizio in una operazione senza conducente un veicolo dotato di ADS, attivando quest’ultimo.

Supporto di guida. E’ un termine generico usato per indicare le funzioni di un sistema di guida autonoma di livello 1 o 2. Esse hanno la capacità di eseguire solo parte del DDT, e

quindi richiedono che il conducente esegua il resto del DDT e supervisioni la performance delle funzioni stesse. Per questo motivo, esse supportano ma non sostituiscono il conducente.

Operazione senza conducente. Operazioni di un veicolo dotato di ADS in cui non è presente alcun utente oppure gli utenti presenti non sono conducenti o fallback-ready users.

Veicolo dual-mode. Un tipo di veicolo dotato di ADS progettato per operazioni sia senza conducente che con conducente. Una caratteristica ADS che è utilizzabile solo durante una parte di un viaggio, come una caratteristica progettata per eseguire il completo DDT durante gli ingorghi sulle autostrade, non sarebbe sufficiente a classificare il suo veicolo ospitante come un dual-mode perché non sarebbe in grado di funzionare senza conducente per un viaggio completo.

I veicoli dotati di funzioni di automazione di livello 2 e 3 possono avere un’ulteriore strategia di mitigazione dei guasti progettata per portare il veicolo a un arresto controllato ovunque il veicolo si trovi, se il conducente non riesce a supervisionare le performance del sistema (livello 2), o se il fallback-ready user non riesce ad eseguire il fallback quando richiesto (livello 3). Per ad esempio, se il fallback-ready user di una feature per ingorghi di livello 3 non risponde a una richiesta di intervento dopo che il traffico si è liberato (una condizione fuori dall’ODD), il veicolo può eseguire una strategia di mitigazione del guasto progettata per portarlo ad un arresto controllato, fermarsi nella sua attuale corsia di marcia e accendere le luci di emergenza. I veicoli equipaggiati con ADS di livello 4 e 5 possono anche avere una strategia di mitigazione dei guasti di stop-in-place in alcune rare condizioni di grave guasto che rendono l’ADS non funzionale attraverso, ad esempio, la perdita di potenza di backup dopo un iniziale guasto di alimentazione. La mitigazione dei guasti eseguita dal veicolo è diversa dal raggiungimento della condizione di rischio minimo e non fa parte della funzione di fallback assegnata a un ADS di livello 4 o 5, perché si verifica dopo che l’ADS si è disinserito o è stato inabilitato da un raro evento catastrofico e, come tale, non rientra nell’ambito della SAE J3016.

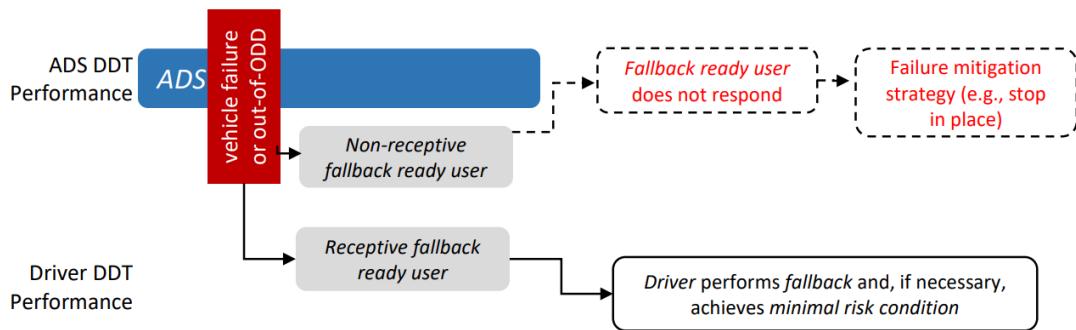


Figura 2.8: Use case sequence per una funzione di livello 3 che mostra l’ADS inserito, il verificarsi di un guasto o di una condizione fuori dall’ ODD, e il fallback-ready user che esegue il fallback, o, se il fallback-ready user non riesce a farlo, una strategia di mitigazione dell’errore, come l’arresto in corsia.¹

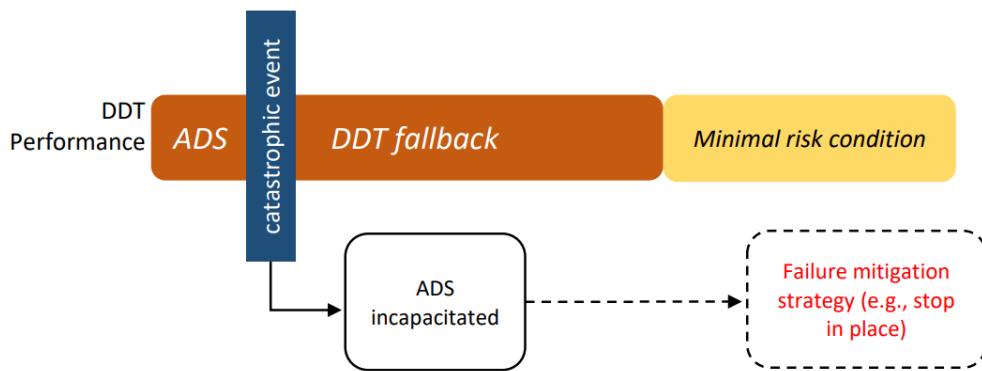


Figura 2.9: Use case sequence al livello 4 che mostra ADS inserito, un evento catastrofico (ad esempio, un’interruzione completa dell’alimentazione) e il sistema che raggiunge una condizione di rischio minimale.¹

Standard J3016 - 2021

Come per lo standard del 2018, anche la versione del 2021 rappresenta un aggiornamento che introduce qualche nuovo termine. Tra quelli degni di nota, sono da annoverare:

Funzione sub-trip. Una funzione del sistema di automazione di guida equipaggiata su un veicolo convenzionale che richiede che un guidatore umano esegua il DDT completo per almeno una parte di ogni viaggio.

Funzione full-trip. Funzione ADS che guida un veicolo per un intero percorso. La seguente figura mostra come un viaggio potrebbe essere completato utilizzando varie combinazioni di livelli di guida autonoma.

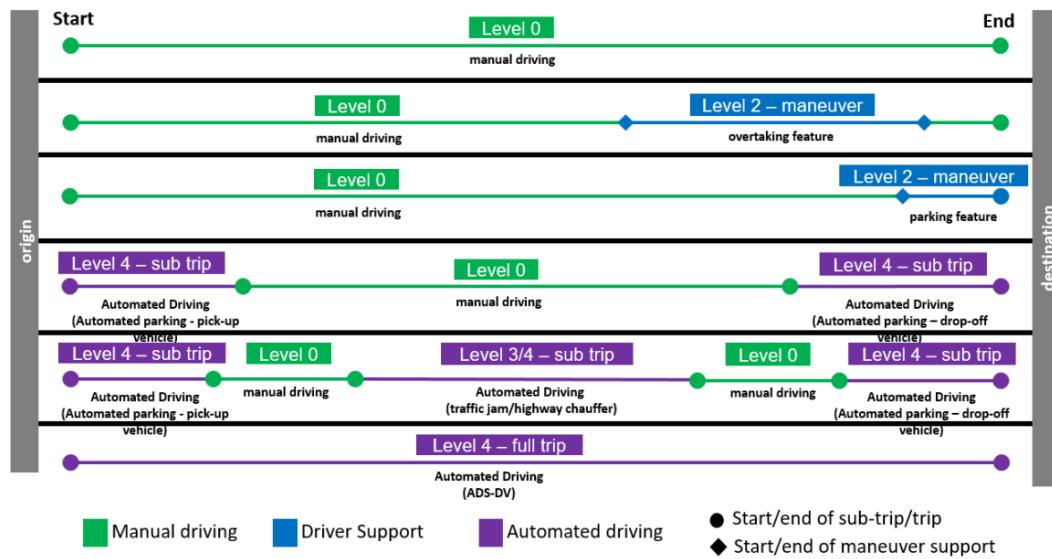


Figura 2.10: Esempi di funzioni/tipi di sistemi di automazione della guida che potrebbero essere disponibili durante un determinato viaggio.¹

2.2 Strumenti per la co-simulazione di scenari di test per Automated Driving System

Data la vastissima gamma di tool e framework disponibili, verranno qui elencati e analizzati brevemente i più promettenti.

Software per la generazione di mappe

ASAM OpenDRIVE fornisce una base comune per la descrizione delle reti stradali con la sintassi del linguaggio XML, utilizzando l'estensione del file xodr. I dati memorizzati in un file ASAM OpenDRIVE descrivono la geometria delle strade, le corsie e gli oggetti, come i segnali stradali. Lo scopo principale di ASAM OpenDRIVE è fornire una descrizione della rete stradale che possa essere usata nelle simulazioni per sviluppare e convalidare le funzionalità ADAS.

OpenStreetMap è un progetto collaborativo per la creazione di un database geografico libero e modificabile del mondo. L'output principale del progetto sono i geodati alla base delle mappe. I dati di OSM possono essere utilizzati in vari modi, tra cui la produzione di mappe cartacee ed elettroniche, la geocodifica di indirizzi e nomi di luoghi e la pianificazione di itinerari.

HERE HD Map è un sistema cloud-based che fornisce dati dettagliati riguardo mappe in tutto il mondo. Fornisce dati in tempo reale mediante attraverso dati sensoriali provenienti da veicoli interconnessi nel sistema. Questo tool è caratterizzato da tre layer sovrapposti: il primo riguarda la topologia della strada e i sensi di marcia, il secondo i tipi di carreggiata e i limiti di velocità e il terzo la segnaletica e luoghi di interesse.

Simulatori

CARLA Simulator è un simulatore di guida open source. Oltre a fare affidamento sull'infrastruttura stradale fornita da OpenDRIVE e OpenStreetMap, offre la possibilità di creare un ambiente tridimensionale usando Unreal Engine 4 come motore grafico.

ADORe è un toolkit modulare in grado di pianificare, controllare e simulare sistemi di guida automatica e utilizza sull'infrastruttura stradale fornita da OpenDRIVE. Permette, inoltre, di gestire il comportamento della vettura facendola comunicare con quelle vicine attraverso comunicazione radio V2X basata su ETSI ITS-G5. La simulazione avviene in modo bidimensionale con una vista a volo d'aquila.

MathWorks Automated Driving Toolbox fornisce algoritmi e strumenti per la progettazione di sistemi di guida assistita e guida autonoma. È possibile implementare algoritmi di pianificazione e sistemi di percezione LIDAR e di visione. Il toolbox può lavorare con dati importati da HERE HD Live Map e le reti stradali di OpenDRIVE. La visualizzazione avviene a volo d'uccello per la coverage dei sensori e in prima persona per LIDAR.

Udacity's Self-Driving Car Simulator è un simulatore di ambienti di guida basato sul motore grafico Unity. Era inizialmente usato nel Udacity's Self-Driving Car Nanodegree per insegnare agli studenti come allenare modelli di auto a guida autonoma basati su deep learning, ed è poi stato reso open-source.

Sistemi di assistenza alla guida

Autoware è uno stack software open-source per veicoli a guida autonoma, costruito su Robot Operating System (ROS). Include le funzionalità necessarie per guidare un veicolo autonomo, dalla localizzazione e rilevamento degli oggetti, al controllo e alla pianificazione del percorso.

openpilot è un sistema di assistenza alla guida open source. Include sistemi di Cruise Control Adattivo, Lane Centering Automatico, Avviso di Collisione Frontale e Lane Departure Warning. Riceve in input solo il video fornito da una camera montata sul cruscotto e, pur

essendo il sistema più facile da implementare, proprio questa sua caratteristica rappresenta il suo punto debole. Il sistema è, infatti, sensibile alla visibilità della camera ed eventualità quali particolari condizioni atmosferiche, forti sorgenti luminose e temperature molto alte o molto basse possono intaccarne la performance.

CAPITOLO 3

Generazione di scenari di co-simulazione di casi di test per ADS

L'obiettivo di questo capitolo è quello di trovare i tool necessari per riuscire a simulare un sistema ADAS all'interno di uno scenario di guida. Gli strumenti che si andranno ad analizzare sono: Automated Driving Toolbox di MathWorks, Carla Simulator e Udacity's Self-Driving Car Simulator.

3.1 MathWorks - Automated Driving Toolbox

I primi software analizzati sono stati Automated Driving Toolbox di MathWorks e i tool affini dell'ecosistema, quali MATLAB, Simulink e Cuboid. La versione utilizzata di questi tools è la R2022a.

Simulazione di un sistema di parcheggio

Parcheggiare un veicolo può essere un procedimento complesso. E' necessario avere consapevolezza dello spazio intorno a sè, identificare dei punti di passaggio intermedi tra la posizione attuale e la destinazione, calcolare una traiettoria che li unisca tutti, e seguirla senza commettere errori. Per comprendere il come costruire un sistema autonomo che faccia questo sono stati seguiti i seguenti tutorial: Automated Parking Valet in Simulink¹ e Visualize

¹<https://www.mathworks.com/help/driving/ug/automated-parking-valet-in-simulink.html>

Automated Parking Valet Using Unreal Engine Simulation².



Figura 3.1: Vista a volo d'aquila dello scenario di parcheggio.³

Questa immagine rappresenta una vista dall'alto della mappa Large Parking Lot in cui la figura azzurra rappresenta la posizione iniziale del veicolo e quella arancione la posizione finale. Il primo passo da fare è capire quali parti di questa mappa sono percorribili e quali portano a una collisione.

Il comando `sceneImageBinary = helperCreateCostmapFromImage(sceneImage);` permette di convertire l'immagine di cui sopra nella seguente.

²<https://www.mathworks.com/help/driving/ug/visualize-automated-parking-valet-using-3d-simulation.html>

³ Figura derivata da <https://it.mathworks.com/help/driving/ug/visualize-automated-parking-valet-using-3d-simulation.html>

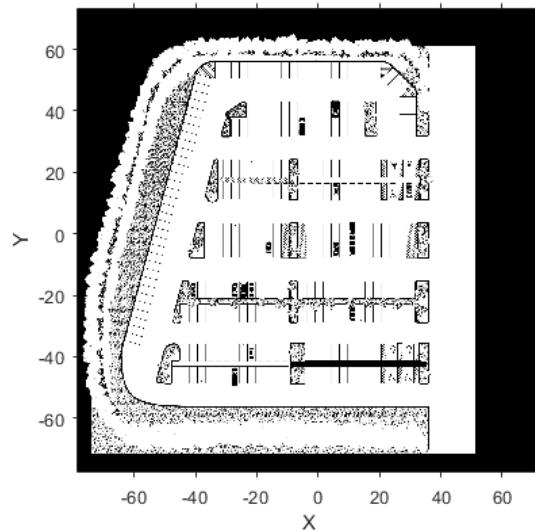


Figura 3.2: vehicleCostMap.³

L’oggetto `vehicleCostmap` crea una mappa dei costi che rappresenta lo spazio di ricerca della pianificazione intorno a un veicolo. La mappa dei costi contiene informazioni sull’ambiente, come ostacoli o aree che il veicolo non può attraversare. A questo punto bisogna salvare all’interno di una tabella le coordinate dei punti di passaggio che si è deciso di seguire. La seguente figura ne esegue un plotting per facilitarne la visualizzazione.

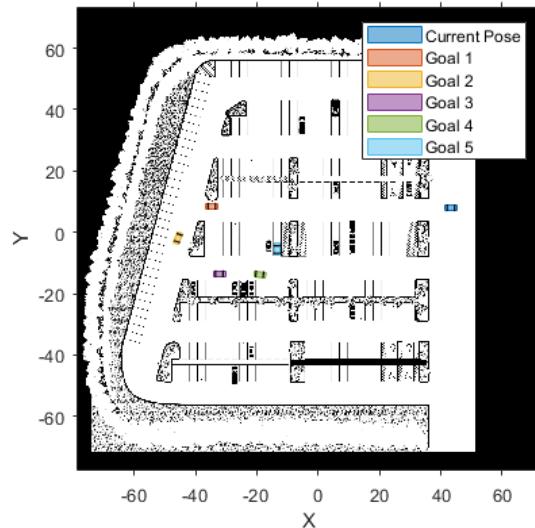


Figura 3.3: Punti di passaggio.³

Utilizzando il modello preconfigurato `APVWith3DSimulation` è possibile interpolare i punti di passaggio, il cui risultato rappresenta la traiettoria finale che il veicolo dovrà

seguire. Il modello permette, inoltre, di eseguire una rappresentazione in 3D dell’operazione in ambiente Unreal Engine 4.

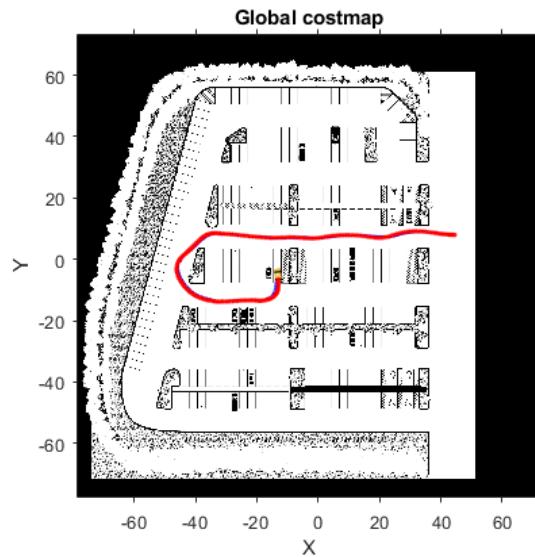


Figura 3.4: Interpolazione.³

Progettare un Lane Marker Detector

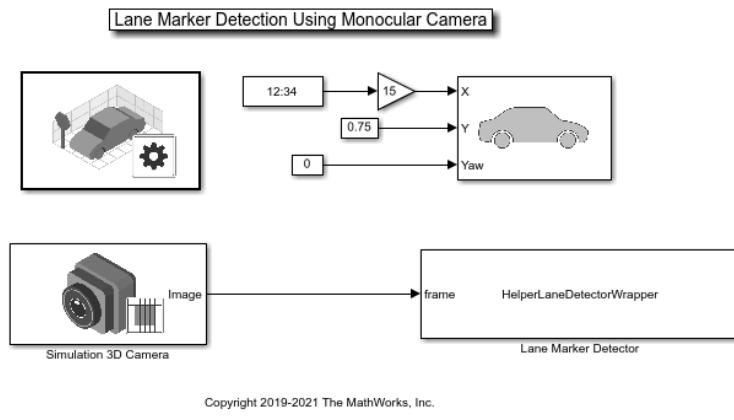
Per sviluppare un sistema di percezione per ADAS è possibile utilizzare modelli di rilevazione della segnaletica orizzontale. Essi devono essere affidabili e in grado di operare nelle situazioni più diverse. I tutorial Visual Perception Using Monocular Camera⁴ e Design Lane Marker Detector Using Unreal Engine Simulation Environment⁵ illustrano come configurare un setup adatto ad allenare e testare modelli di questo tipo in un ambiente realistico.

Il primo passo consiste nella configurazione di uno scenario di guida semplice che riguarda un veicolo che percorre una strada dritta attraverso il comando:

`helperStraightRoadMLTest.`

⁴<https://www.mathworks.com/help/driving/ug/visual-perception-using-monocular-camera.html>

⁵<https://www.mathworks.com/help/driving/ug/design-of-lane-marker-detector-in-3d-simulation-environment.html>



Copyright 2019-2021 The MathWorks, Inc.

Figura 3.5: Configurazione Simulink.⁶

Questo schema rappresenta la configurazione Simulink dello scenario appena descritto e rende possibile selezionare diverse scene nell'engine di visualizzazione 3D, muovere i veicoli nella scena e posizionare diversi tipi di sensori sulle vetture. Permette inoltre di registrare i movimenti dell'auto in un file mp4 attraverso una camera monoculare montata a bordo per poter allenare il modello di predizione in un momento in cui l'elaboratore non sia impegnato a gestire la simulazione dello scenario. Per generalizzare l'abilità predittiva del modello è necessario testarlo anche in altri scenari più complessi, come nel caso seguente.

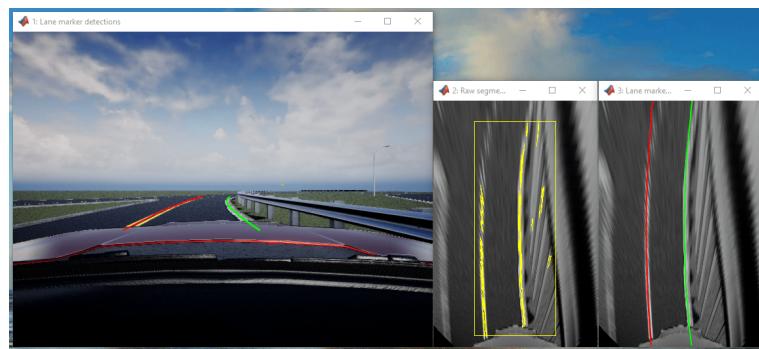


Figura 3.6: Lane Detector in MCity.⁶

Dopo un'attenta analisi di questi e altri tutorial, e della documentazione ad essi riguardante è stato deciso che l'approccio per la costruzione e simulazione di ADAS utilizzato dall'ecosistema MatWorks sia troppo di base e quindi molti aspetti esulano dall'obiettivo di questa tesi. Verranno ora analizzate alcune alternative.

⁶Figura derivata da <https://it.mathworks.com/help/driving/ug/design-of-lane-marker-detector-in-3d-simulation-environment.html>

3.2 CARLA Simulator

CARLA Simulator consiste in una server application che si occupa della simulazione vera e propria, e di una componente client accessibile attraverso l'API Python che si basa sul framework Pygame e permette di manipolare la simulazione. Queste due componenti possono essere eseguite anche su macchine diverse per alleggerire il carico di lavoro e differenziare la GPU che calcola la simulazione da quella lato client che verosimilmente sta allenando una rete neurale. Un'altra informazione utile per la gestione di questo tipo di simulazione è il modo in cui viene gestito il tempo. Il sistema è impostato di default in modalità asincrona: questo vuol dire che il server, senza aspettare il client, cerca di simulare lo scenario più velocemente possibile, quindi la durata della simulazione è diversa della durata dello scenario effettivamente simulato. Il sistema può essere impostato in modalità sincrona, in cui il server aspetta il tick del client. In caso di architettura multiclient, solo un client deve comunicare il tick al server.

Con l'aiuto del tutorial Getting started with CARLA⁷ presente nella documentazione, si vogliono esplorare le funzionalità principali offerte dal tool. Per prima cosa è necessario avviare l'applicazione lato server eseguendo CarlaUE4.exe (in ambiente Windows).



Per avere la possibilità di manipolare lo scenario, si deve eseguire il seguente codice Python con cui si instaura la connessione tra il client e il server attraverso la porta 2000.

Code Listing 3.1: Connessione client

```
import carla

client = carla.Client('localhost', 2000)
world = client.get_world()
```

⁷https://carla.readthedocs.io/en/latest/tuto_G_getting_started

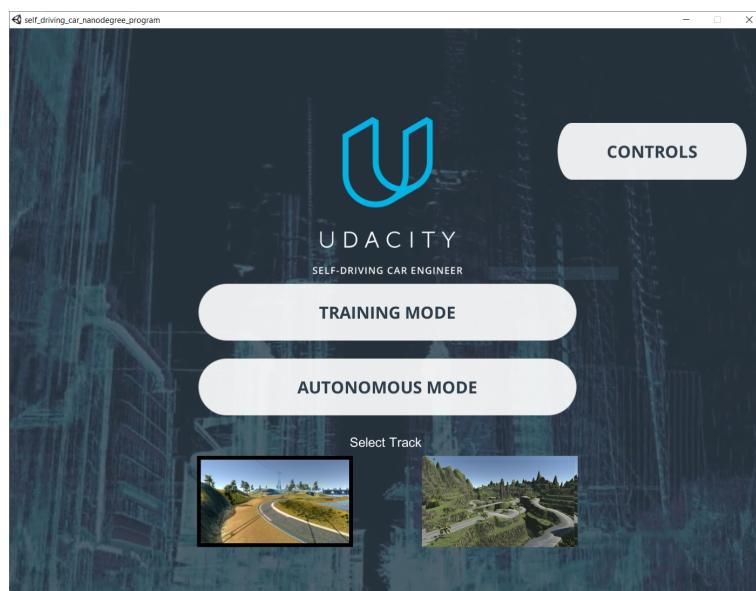
Il nostro primo obiettivo è quello di popolare la mappa con alcuni veicoli. Per fare questo, è possibile scrivere uno script Python, oppure avviare lo script `generate-traffic.py` presente nella cartella `examples` di CARLA. Verranno quindi generati 30 veicoli e 10 pedoni che si muoveranno nella mappa in modo casuale guidati dal modulo `Traffic Manager` che include anche un sistema anticollisione. E' utile notare che i veicoli non sono guidati da un sistema ADAS, in quanto non prendono decisioni in base agli input di sensori montati a bordo, ma è il motore di gioco ad imporre loro cosa fare nelle diverse situazioni. Per esempio, il sistema anticollisione consiste in una hitbox attorno alla mesh della vettura che fa scattare un evento quando viene attraversata. In computer grafica, una mesh è un reticolo che definisce un oggetto nello spazio, composto da vertici, spigoli e facce [9]. Le hitbox sono un insieme di forme 2D o 3D che un gioco utilizza per registrare le collisioni in tempo reale. Ad esempio, le hitbox registrano se si è colpito un oggetto. Lo script `manual-control.py` permette di creare un client in cui l'utente può guidare un veicolo ego, cambiare mappa, cambiare le condizioni meteorologiche, e leggere informazioni utili, quali velocità, posizione e rotazione del veicolo, eventuali collisioni con altri oggetti e così via.



È inoltre possibile creare oggetti di tipo `Sensor` (Depth, RGB, LIDAR, radar etc.) in qualsiasi posizione del veicolo in grado di captare informazioni riguardo all'ambiente circostante al veicolo e salvarle su disco oppure trasmetterle in diretta ad altri software. I dati raccolti dagli eventi di tipo `CollisionEvent`, `LaneInvasionEvent` e `ObstacleDetectionEvent` possono essere usati come label per allenare modelli per guida autonoma insieme a quelli raccolti dai sensori. Per quanto riguarda la generazione di mappe diverse da quelle di default la soluzione più promettente è usare il tool `RoadRunner` di MathWorks; esso, però, necessita di licenza apposita.

3.3 Udacity’s Self-Driving Car Simulator

Il simulatore di Udacity consiste in un ambiente di simulazione basato su Unity in cui è possibile testare i propri modelli di automazione di diversi scenari di guida. E’ possibile reperire le varie versioni del simulatore nella repository GitHub `self-driving-car-sim`⁸ di Udacity così come il codice sorgente del progetto Unity. Per questo use case è stato utilizzata la versione Term 1 Version 2 su piattaforma Windows. Per avviare il simulatore è sufficiente lanciare l’eseguibile che aprirà la seguente finestra.



Essa permette di scegliere quale tracciato usare tra i due disponibili. La prima mappa consiste in un tracciato a singola corsia di ridotte dimensioni nei pressi di un lago, ed è caratterizzato da curve molto larghe e molti tratti più o meno rettilinei. La seconda mappa, invece, consiste in un tracciato a doppia corsia all’interno di una foresta. Esso è formato da curve strette e salite e discese molto ripide, fattori che ne incrementano la difficoltà di completamento. Il menù permette anche di selezionare la modalità d’utilizzo tra training mode e autonomous mode. Il training mode permette all’utente di muovere la vettura all’interno del tracciato selezionato come in un videogioco, utilizzando le frecce direzionali o i tasti WASD. Premendo il tasto R è possibile scegliere la directory in cui salvare il dataset. Ad una nuova pressione del tasto R, il sistema registrerà le coordinate e la telemetria dell’auto ad ogni frame. Per interrompere la registrazione è necessario premere di nuovo il tasto R. A questo punto il sistema ripercorre il percorso seguito dalla vettura salvando nella directory scelta in precedenza i frame catturati dalle tre camere a bordo a cui vengono associati i dati

⁸<https://github.com/udacity/self-driving-car-sim>

telemetrici (angolazione delle ruote, accelerazione, freno e velocità) in un file CSV. Utilizzato l’autonomous mode l’applicazione si comporta come un server che invia le immagini catturate dalle camere ad un client il quale, utilizzando un modello di intelligenza artificiale, risponde inviando comandi alla vettura (angolazione delle ruote e acceleratore). Per l’implementazione pratica sono stati seguiti i tutorial di Naoki Shibuya su GitHub⁹ e sul suo sito personale¹⁰.

Attraverso il Training Mode sono stati raccolti due dataset separati. Il primo percorrendo il primo tracciato tre volte in un verso di percorrenza e altre tre volte nel verso opposto. Il secondo dataset è stato raccolto in modo analogo sul secondo tracciato. La tipologia di agente di guida si rifa al modello descritto in un paper da Nvidia in cui venne utilizzato un sistema di registrazione della telemetria DAVE-2 [1].

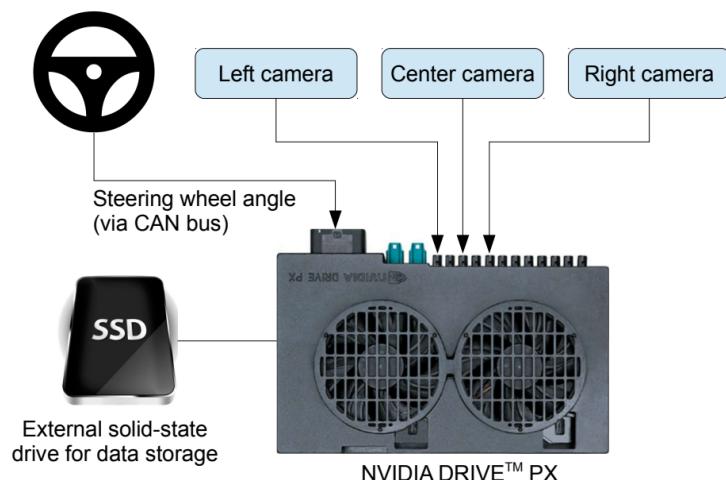


Figura 3.7: Schema ad alto livello del sistema DAVE-2.¹¹

Il sistema registra le immagini catturate dalle tre camere a bordo del veicolo insieme all’istante in cui sono state memorizzate. Analogamente viene memorizzata l’angolazione del volante applicata dal conducente in ogni istante. Questi dati di bordo sono stati usati per allenare una Rete Neurale Convoluzionale, un tipo di rete neurale particolarmente adatta al pattern recognition di dati visivi. Si è quindi dimostrato che un tale tipo di agente è in grado di generalizzare la conoscenza raccolta essendo in grado di guidare in autonomia un veicolo lungo un percorso anche al variare delle condizioni meteorologiche.

Per l’implementazione si fa riferimento alla repository `CarBehavioralCloning`¹².

⁹<https://github.com/naokishibuya/car-behavioral-cloning>

¹⁰<https://naokishibuya.medium.com/introduction-to-udacity-self-driving-car-simulator-4d78198d301d>

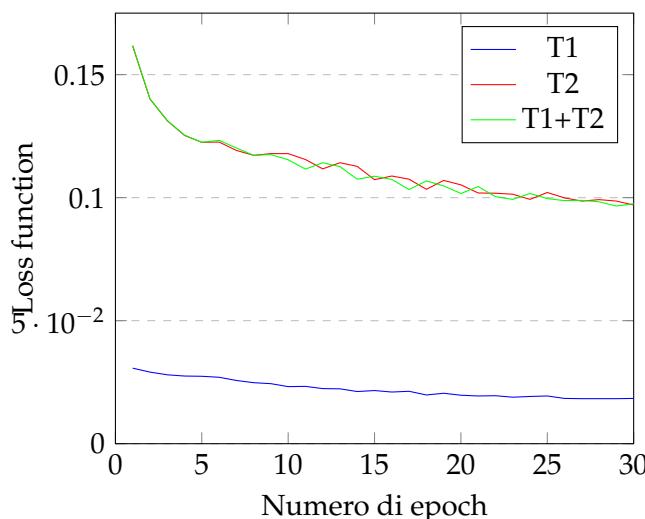
¹¹Immagine tratta da [1]

¹²<https://github.com/Alex180689/CarBehavioralCloning>

Per prima cosa è necessario installare l’ambiente Anaconda descritto dal file env.yalm che contiene tutte le dipendenze necessarie di framework come TensorFlow e Keras. Per allenare il modello si esegue lo script python con il comando `python model.py`. All’interno di questo file è possibile specificare alcuni parametri per l’apprendimento quali il learning rate e il numero di epoch. L’allenamento è stato eseguito su una CPU Intel i5-10200H e ha impiegato circa 155 secondi per ogni epoch. Per ogni epoch su cui il modello viene allenato, ne viene salvata una copia nella directory del progetto.

Per testare il modello appena generato bisogna navigare nella directory in cui si trova il modello e lanciare l’applicazione client con il comando `python drive.py model.h5` sostituendo `model` con il suo nome.

Data la semplicità del primo percorso, un agente allenato su soli 5 epoch del primo dataset mostra già ottimi risultati. Al contrario, il secondo tracciato è caratterizzato da curve più strette salite molto ripide, quindi risulta più difficile da completare. Per questo motivo, sono stati necessari 23 epoch per raggiungere delle buone prestazioni. Come è possibile immaginare, se questi modelli sono testati su percorsi di cui non hanno esperienza presentano delle pessime performance. A questo punto, l’obiettivo era quello di riuscire a sviluppare un modello capace di completare in modo soddisfacente entrambi i tracciati, sottoponendolo ad entrambi i dataset. Tuttavia, anche dopo numero si tentativi, non è stato possibile raggiungere questo risultato neanche dopo 50 epoch. Si ipotizza che la causa di questo sia la poca eterogeneità degli ambienti presenti nel dataset che non permettono al modello di generalizzare abbastanza gli ambienti con cui ha familiarità. Di seguito è illustrato il valore della loss function in funzione del numero di epoch in ognuno di questi tre casi.



Allenare il modello su un maggior numero di ambienti differenti è una buona idea per mi-

gliorarne le performance. Per creare delle nuove mappe all’interno del simulatore di Udacity bisogna scaricare il codice sorgente del progetto `self-driving-car-sim`¹³ e modificarne l’aspetto delle mappe esistenti attraverso l’editor di Unity.



Figura 3.8: Editor di Unity.¹⁴

Per il secondo use case analizzato è stato utilizzata la versione Term 3 su piattaforma Ubuntu. Una volta lanciata, l’applicazione si presenta nel seguente modo.



L’unica modalità permette di avviare l’ambiente simbolativo in cui il veicolo ego percorre una lunga autostrada percorsa da altre vetture che viaggiano a velocità compresa tra 40 e 50 MPH. Spuntando la checkbox `Manual Mode` l’auto verrà guidata dall’utente attraverso

¹³<https://github.com/udacity/self-driving-car-sim>

¹⁴Immagine tratta da <https://naokishibuya.medium.com/introduction-to-udacity-self-driving-car-simulator-4d78198d301d>

l’uso delle freccette direzionali. In caso contrario, l’applicazione si comporta come un server in attesa di una applicazione client che invii input. Per l’implementazione pratica di questo modello si è seguita la documentazione della repository p7-path-planning¹⁵.

Per questo use case non è necessario allenare un modello, visto che è già presente nella repository sopra menzionata. Per impostare l’ambiente bisogna clonare la repository ed eseguire i seguenti comandi:

```
mkdir build && cd build
cmake .. && make
./path_planning
```

Una volta avviato il server con la spunta su Manual Mode disabilitata, l’agente guiderà la vettura all’interno della mappa data la posizione della stessa, sensor fusion data e alcuni punti di percorso.



L’obiettivo dell’agente è quello di guidare la vettura a velocità più vicina possibile al limite di 50 MPH (senza superarlo), trovando la traiettoria giusta per superare gli altri veicoli senza entrare in collisione e senza superare i 10 m/s^2 di accelerazione e i 10 m/s^3 di strappo.

¹⁵<https://github.com/markmiser/udacity-self-driving-car-engineer/tree/master/p7-path-planning>

CAPITOLO 4

Conclusioni e sviluppi futuri

Per raggiungere l'obiettivo di questa tesi, ovvero riuscire a simulare con successo dei casi di test per un sistema ADAS, sono stati presi in considerazione diversi tool: Automated Driving Toolbox, Carla Simulator e Udacity Simulator. Il tool che è stato considerato più adatto per questo scopo è l'ambiente simulativo di Udacity. Esso ha permesso di allenare e testare dei modelli di intelligenza artificiale in grado di guidare un veicolo all'interno di alcuni tracciati virtuali. Il modello è basato su reti neurali convoluzionali progettate utilizzando i framework di Tensorflow e Keras.

Il principale miglioramento di cui necessita il modello sviluppato è la capacità generalizzare le proprie conoscenze e completare in sicurezza percorsi di cui non ha esperienza. Come già detto, il modo più semplice per fare questo è costruire molte mappe dall'aspetto diverso nell'editor Unity e utilizzarle per ricavarne un dataset più variegato. L'apprendimento supervisionato, tuttavia, ha mostrato avere due debolezze in particolare.

La prima relativa al fatto che l'agente imita l'utente che ha generato il dataset, quindi la sua capacità di guida rappresenta un upper bound per l'agente stesso.

La seconda relativa al fatto che al crescere del dataset necessario per il training, l'utente deve generarlo passando sempre più tempo in training mode.

Per risolvere entrambi questi problemi, è possibile valutare l'implementazione di una forma di apprendimento non supervisionato. Una buona scelta per questo scopo è l'algoritmo NEAT (NeuroEvolution of Augmenting Topologies), un tipo di algoritmo genetico applicato ad una rete neurale che permette di ottimizzare la funzione di fitness variando non solo i valori dei

pesi di una rete neurale ma anche la topologia e il numero di neuroni dei layer centrali[10]. In secondo piano, il modello utilizzato manca di un sistema di navigazione e segue semplicemente la strada nel modo che ritiene più sicuro. Come sviluppo futuro da questo punto di vista, si potrebbe implementare un algoritmo di pathfinding che possa trovare il percorso migliore dalla posizione attuale alla destinazione. Un’alternativa è usare alcuni tool già esistenti specializzati in routing come OpenStreetMap¹ e GraphHopper².

¹<https://www.openstreetmap.org/>

²<https://docs.graphhopper.com/>

Bibliografia

- [1] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars, 2016. (Citato a pagina 25)
- [2] Samuel G Charlton and Nicola J Starkey. Driving without awareness: The effects of practice and automaticity on attention and driving. *Transportation research part F: traffic psychology and behaviour*, 14(6):456–471, 2011. (Citato a pagina 1)
- [3] Tim Jannusch, Darren Shannon, Michaele Völler, Finbarr Murphy, and Martin Mullins. Smartphone use while driving: An investigation of young novice driver (ynd) behaviour. *Transportation research part F: traffic psychology and behaviour*, 77:209–220, 2021. (Citato a pagina 1)
- [4] John A. Michon. *A critical view of driver behavior models: What do we know, what should we do?*, page 485–524. Springer US, 1985. (Citato a pagina 6)
- [5] World Health Organization. *Global status report on road safety 2015*. World Health Organization, 2015. (Citato a pagina 1)
- [6] Michael M. Roy and Michael J. Liersch. I am a better driver than you think: Examining self-enhancement for driving ability. *Journal of applied social psychology*, 43(8):1648–1659, Aug 2013. (Citato a pagina 1)
- [7] Brandon Schoettle and Michael Sivak. *A survey of public opinion about autonomous and self-driving vehicles in the US, the UK, and Australia*. 2014. (Citato a pagina 2)

- [8] Jennifer Shuttleworth. Sae standards news: J3016 automated-driving graphic update. *SAE International*, 2019. (Citato alle pagine 4 e 7)
- [9] Francesco Siddi. *Grafica 3D con Blender*. Apogeo, 2014. (Citato a pagina 23)
- [10] Kenneth O Stanley and Risto Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. In *Proceedings of the 4th Annual Conference on genetic and evolutionary computation*, pages 569–577, 2002. (Citato a pagina 30)