UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Informatica e dell'Automazione

TESI DI LAUREA

Valutazione di simulatori di guida autonoma e di volo basati sull'Intelligenza Artificiale

Evaluation of AI-based autonomous driving and flight simulators

Relatore

Prof. Domenico Ursino

Candidato

Simone Di Battista

ANNO ACCADEMICO 2022-2023

Sommario

Questa tesi si propone di esaminare in maniera approfondita l'integrazione dell'Intelligenza Artificiale nel mondo dei simulatori di guida autonoma e di volo. Attraverso una iniziale esplorazione dei settori chiave dell'Intelligenza Artificiale, come Deep Learning e Reinforcement Learning, la ricerca si concentra sulla comprensione delle dinamiche alla base di tale integrazione. L'analisi mira a evidenziare il ruolo centrale che giocano questi settori nella simulazione di scenari virtuali. La ricerca non si ferma all'analisi teorica, ma comprende esperimenti pratici eseguiti per valutare ed analizzare le prestazioni di questi simulatori. In questo modo, la tesi si pone come obiettivo la comprensione teorica e pratica dell'impatto dell'Intelligenza Artificiale nei simulatori di guida autonoma e di volo.

Keyword: Intelligenza Artificiale, Machine Learning, Deep Learning, Reinforcement Learning, Deep Reinforcement Learning, Simulatori di guida, Simulatori di volo

Indice

Introduzione

Nell'epoca attuale, l'Intelligenza Artificiale (IA) si sta affermando come un elemento indispensabile nella nostra vita, influenzando in maniera pervasiva molti aspetti della società moderna, in particolare la nostra quotidianità. L'ampio e crescente impiego dell'Intelligenza Artificiale è attribuibile alle sue avanzate capacità di calcolo, le quali si riflettono nell'evoluzione tecnologica di settori cruciali, come l'economia, la medicina, l'automotive e l'aviazione.

L'Intelligenza Artificiale (IA) rappresenta il futuro della nostra società, portando con sè opportunità e trasformazioni radicali.

L'IA, con la sua capacità di apprendimento automatico e di elaborazione dei dati, è destinata a sostituire moltissimi lavori che richiedono compiti ripetitivi. Ciò promette miglioramenti nella precisione e riduzione degli errori umani nel mondo lavorativo. Tuttavia, l'IA, sebbene portatrice di benefici evidenti, non è esente da problematiche significative. Nella società contemporanea emergono preoccupazioni etiche connesse all'implementazione dell'IA, per cui ne viene richiesto un utilizzo ponderato e responsabile. In ogni caso, un investimento in conoscenza e ricerca rappresenta la chiave per preparare la società odierna ad abbracciare in modo positivo l'evoluzione e la trasformazione guidate dall'Intelligenza Artificiale.

Quindi, studiare e investire tempo e risorse finanziarie nell'Intelligenza Artificiale non solo migliora la qualità della vita, ma contribuisce in modo tangibile al progresso sociale ed economico della società, permettendo di sfruttare appieno il suo enorme potenziale.

Questa tesi si colloca proprio in tale contesto; in essa, forniremo, inizialmente, una panoramica generale sull'Intelligenza Artificiale (IA). Successivamente, approfondiremo due aspetti chiave legati ad essa, ovvero il Deep Learning, una sottodisciplina basata su reti neurali artificiali, e il Reinforcement Learning, un paradigma di apprendimento che si basa sul concetto di reward, o ricompensa.

Questa parte iniziale, prettamente teorica, sarà di fondamentale importanza per comprende al meglio l'integrazione di queste tecnologie in simulatori dedicati alla guida autonoma e all'aviazione. Infatti, l'elaborato includerà anche l'analisi e la descrizione di vari esperimenti condotti attraverso simulatori specifici nel contesto dell'industria automobilistica e dell'aviazione. Questa analisi ci consentirà di mostrare i vantaggi pratici e il ruolo significativo che l'Intelligenza Artificiale svolge in scenari simulati di guida autonoma e di volo.

La presente tesi è suddivisa in cinque capitoli; in particolare:

- Nel Capitolo 1 viene presentata una panoramica generale sull'Intelligenza Artificiale (IA), includendo una sua descrizione dettagliata che spazia dalla definizione, alle sue applicazioni, alla sua storia, e ai vantaggi e svantaggi associati.

- Nel Capitolo 2 si approfondiscono due componenti principali dell'IA, il Deep Learning e il Reinforcement Learning. In particolare, si esplorano in maniera dettagliata le reti neurali, esaminandone i diversi tipi, e i componenti fondamentali del Reinforcement Learning, come il Markov Decision Process, gli algoritmi model-based e model-free. Per ciascun componente, vengono analizzati i corrispettivi benefici e svantaggi, descrivendo, inoltre, le applicazioni pratiche. In aggiunta, si presenta il Deep Reinforcement Learning, descrivendone alcuni algoritmi.
- Nel Capitolo 3 ci si dedica all'esecuzione e all'analisi di esperimenti condotti con simulatori relativi al mondo della guida autonoma. Ogni simulatore affronta elementi cruciali della guida, tra cui il parcheggio, il mantenimento di corsia, il cruise control e l'individuazione di autoveicoli.
- Nel Capitolo 4 vengono eseguiti e analizzati esperimenti condotti con simulatori dedicati al mondo dell'aviazione. Ciascun simulatore si concentra su fasi specifiche, quali l'atterraggio, la definizione della traccia dell'aereo e la gestione dei voli e, nel contesto dell'aeronautica militare, l'individuazione di bersagli a terra.
- Nel Capitolo 5 viene presentata una discussione in merito alle esperienze svolte. In particolare, per ciascun simulatore, vengono citati pregi e difetti soggettivi riscontrati durante la conduzione degli esperimenti.

CAPITOLO 1

Un'introduzione generale all'Intelligenza Artificiale

Nel corso di questo capitolo, esamineremo l'Intelligenza Artificiale (IA) in modo approfondito. Inizieremo con una breve introduzione al concetto di IA per poi focalizzarci su quella che è l'evoluzione storica dell'IA. Successivamente, esploreremo le applicazioni dell'IA, concentrandoci sul suo crescente utilizzo nei settori dell'automobilistica e dell'aeronautica. Analizzeremo, quindi, anche i benefici e le limitazioni dell'IA, evidenziando la necessità di una gestione etica e sicura di questa tecnologia avanzata. Inoltre, prenderemo in considerazione come diverse entità, tra cui società, aziende e team universitari, stiano affrontando l'IA, con particolare attenzione al ruolo cruciale svolto da NVIDIA in questo contesto. Questo capitolo rappresenta un punto di partenza fondamentale per comprendere in che modo l'IA sta rivoluzionando l'industria e la società.

1.1 Fondamenti dell'Intelligenza Artificiale: Definizione, Obiettivi e Categorie

1.1.1 Definizione di Intelligenza Artificiale

L'Intelligenza Artificiale è un campo relativamente giovane che, nel corso degli anni, ha apportato un contributo significativo all'evoluzione dell'informatica.Essa è un campo multidisciplinare infatti è stata profondamente influenzata da diverse discipline, tra cui la filosofia, la matematica e le scienze cognitive.

L'Intelligenza Artificiale, abbreviata come IA, rappresenta lo studio approfondito dello sviluppo di sistemi informatici capaci di emulare l'intelletto e il comportamento umano. Possiamo definire l'Intelligenza Artificiale basandoci su come i sistemi intelligenti pensano internamente e agiscono esternamente. Ciò cerca di valutare quanto il comportamento di tali sistemi somigli al comportamento razionale umano.In particolare gli obiettivi che ci si pone sono:

- *Simulare il comportamento umano:* questa definizione si applica quando i risultati delle operazioni prodotti da sistemi intelligenti sono difficilmente distinguibili da quelli di esseri umani.
- *Emulare il pensiero umano:* questa definizione trova validità quando il processo decisionale intrapreso dai sistemi per risolvere problemi è paragonabile a quello utilizzato dagli esseri umani.

L'Intelligenza Artificiale è spesso associata a immagini futuristiche di tecnologie all'avanguardia e robot in grado di comprendere e prendere decisioni autonome. In realtà, l'IA

costituisce un ramo dell'informatica che si concentra sulla programmazione e progettazione di sistemi, sia hardware che software, che conferiscono alle macchine abilità considerate tipicamente umane. L'Intelligenza Artificiale, inoltre, è un campo che abbraccia la teoria di Howard Gardner secondo cui l'intelligenza non può essere ridotta a una singola entità, ma piuttosto è composta da diverse forme di abilità.

1.1.2 Obiettivi, abilità chiave e modelli dell'IA

L'obiettivo principale dell'Intelligenza Artificiale non consiste nella duplicazione dell'intelligenza umana, bensì nella riproduzione o emulazione di alcune delle sue funzioni.

Un sistema intelligente, infatti, è concepito con l'intento di emulare una o più delle diverse forme di intelligenza. Nonostante queste forme siano generalmente associate all'essere umano, è possibile individuare comportamenti specifici che possono essere ricreati da macchine particolari.

L'IA ha, indubbiamente, rivoluzionato la dinamica delle interazioni tra gli esseri umani e le macchine, nonché tra le macchine stesse. Questo risultato è ottenuto principalmente grazie alle tre abilità cardine dell'IA:

- *Apprendimento automatico:* consiste nella capacità delle macchine di acquisire enormi quantità dati, riconoscere modelli nei dati e adattare gli algoritmi di conseguenza. Questi algoritmi guidano i dispositivi passo dopo passo a compiere compiti specifici e sono spesso basati su reti neurali artificiali, che cercano di emulare il funzionamento del cervello umano.
- *Ragionamento:* consiste nella capacità delle macchine di elaborare informazioni e trarre conclusioni cercando di simulare il pensiero umano.
- *Autocorrezione:* consiste nella capacità delle macchine di riconoscere e correggere gli errori in modo da auto-ottimizzarsi nel tempo per garantire risultati migliori.

L'Intelligenza Artificiale (IA) sfrutta due modelli a seconda degli obiettivi e delle applicazioni:

- *Antropomorfico:* l'IA è progettata per emulare il modo in cui gli esseri umani pensano e ragionano, adottando metodi simili a quelli umani.
- *Non antropomorfico:* l'IA è progettata ottenere i migliori risultati possibili, senza necessariamente seguire i modelli di pensiero umano.

1.1.3 Categorie di Intelligenza Artificiale

Oltre a quanto è stato detto va notato che ci sono tre categorie di Intelligenza Artificiale:

- *L'Intelligenza Artificiale Limitata (ANI)*, conosciuta anche come "intelligenza artificiale debole," poichè questa forma di intelligenza è limitata nella gestione di una gamma ristretta di parametri e situazioni. È la forma di IA più facilmente riconoscibile ed è già utilizzata dalla maggioranza delle persone.
- *L'Intelligenza Artificiale Generale (AGI)*, considerata un tipo di "intelligenza artificiale forte," opera a un livello avanzato. Questa forma di intelligenza, simile a quella umana, consente alle macchine di eseguire compiti intellettuali analoghi a quelli svolti dagli esseri umani.

- *La Superintelligenza Artificiale (ASI)*, pur non esistendo ancora nella pratica, rappresenta la capacità di una macchina di superare l'intelligenza umana.

Dopo aver sottolineato l'importanza cruciale dell'Intelligenza Artificiale (IA) nell'ambito dell'informatica e nella società moderna, è essenziale esaminare il suo sviluppo evolutivo nel corso del tempo. Per ottenere una comprensione completa delle origini e dell'evoluzione dell'IA, analizziamo ora il suo percorso storico.

1.2 Evoluzione storica e futuro dell'IA

1.2.1 Le origini dell'Intelligenza Artificiale

Nel 1950, il matematico inglese Alan Turing pubblicò un influente articolo intitolato "Computing Machinery and Intelligence," in cui propose un modo per valutare l'intelligenza delle macchine. Questo concetto prende il nome di "Test di Turing" e stabilisce che, per considerare una macchina intelligente, questa doveva essere in grado di:

- Comunicare in modo comprensibile.
- Memorizzare informazioni.
- Ragionare in maniera automatica, utilizzando le informazioni memorizzate per rispondere a domande e trarre nuove conclusioni.
- Apprendere in maniera automatica, adattandosi alle circostanze e scoprendo nuovi modelli.

Con l'avvento dei primi computer dopo la Seconda Guerra Mondiale, molti scienziati iniziarono a interrogarsi sulla possibilità che le macchine potessero pensare in modo simile agli esseri umani.

La storia dell'Intelligenza Artificiale come campo di ricerca inizia ufficialmente con la "Conferenza di Dartmouth" del 1956. Questo incontro rappresentò un punto di partenza significativo per il progresso dell'IA.

I progressi successivi furono promettenti, con programmi in grado di dimostrare teoremi matematici e persino giocare a scacchi. Il linguaggio di programmazione di riferimento divenne il LISP.

Nel 1966, venne inventato il primo chatbot noto come ELIZA, un passo rivoluzionario nella storia dell'IA poiché segnò la prima interazione significativa tra l'uomo e la macchina, mirando a simulare conversazioni umane.

1.2.2 Tra alti e bassi...

La storia dell'IA è segnata da alti e bassi. Alla fine degli anni '60, c'erano grandi aspettative riguardo ai progressi nell'Intelligenza Artificiale, ma i risultati pratici nell'applicazione quotidiana non si materializzarono prontamente. Questo periodo fu caratterizzato da una mancanza di sviluppo tecnologico e dalla riduzione dei finanziamenti pubblici, dando vita a quella che prende il nome di era dell'inverno dell'Intelligenza Artificiale, che persistette fino agli anni '80.

L'Intelligenza Artificiale ha incontrato difficoltà significative in settori che sembravano più accessibili, come la traduzione automatica e il trattamento del linguaggio naturale, le quali erano legate alla potenza di calcolo limitata, alla complessità dei problemi da risolvere e alla gestione di enormi quantità di dati. Tuttavia, nel 1980 si intravide un lieve miglioramento,

grazie alla costante ricerca e all'uso crescente dei computer aziendali, che portò al successo di alcuni sistemi di Intelligenza Artificiale nel settore industriale nella seconda metà degli anni '80.

Questi progressi rinnovarono l'interesse e l'investimento nell'IA ma, in verità, questa nuova ondata di entusiasmo portò ad una bolla economica che iniziò a manifestarsi nel 1987, con la crisi di diverse imprese focalizzate nella produzione di hardware per l'IA. Tutto ciò portò a un nuovo declino nello studio dell'IA che si protrasse fino alla metà degli anni '90. Dopo questo ulteriore insuccesso, l'approccio allo studio dell'Intelligenza Artificiale subì una significativa trasformazione.

1.2.3 La rinascita dell'IA

L'Intelligenza Artificiale ha incontrato difficoltà in settori più praticabili, come la traduzione automatica e il trattamento del linguaggio naturale. Queste difficoltà erano legate principalmente alla potenza di calcolo limitata, alla complessità dei problemi da risolvere e alla gestione di enormi quantità di dati.

Dal 1995 in poi, si è verificato un notevole rinascimento dell'Intelligenza Artificiale, un fenomeno che persiste fino ai giorni nostri. In particolare:

- *1997:* l'IA ha guadagnato una fama globale quando il computer Deep Blue è riuscito a sconfiggere il campione del mondo di scacchi in carica.
- *2002:* l'IA ha fatto la sua comparsa nelle case delle persone sotto forma di Roomba, un robot aspirapolvere autonomo.
- *2006:* l'IA ha fatto il suo ingresso nel mondo aziendale, con aziende come Facebook, Twitter e Netflix, che hanno cominciato a sfruttarla.
- *2011:* Sebastian Thrun e Peter Norvig hanno organizzato un corso online chiamato "Artificial Intelligence" ottenendo un naturale successo.
- *2019:* i modelli generativi, capaci di produrre testo, immagini e altro partendo da un input testuale, hanno iniziato a guadagnare popolarità.
- *OGGI:* si sono sviluppate tecnologie avanzate di machine learning, sistemi di data mining e sistemi predittivi.

Attualmente, l'IA è presente in tutti gli aspetti della vita quotidiana: i nostri telefoni conoscono le nostre preferenze, si sbloccano con il riconoscimento del volto e ci forniscono indicazioni stradali e stime di tempo di percorrenza. Questi progressi sono stati resi possibili da due fattori principali:

- *Progressi scientifici*, tra cui l'aumento della potenza di calcolo dei computer (come previsto dalla legge di Moore per i microprocessori) e l'evoluzione di algoritmi più complessi.
- *Cambiamenti nella società*, in cui la popolazione ha contribuito in modo inconsapevole a migliorare l'Intelligenza Artificiale.

Grazie all'evoluzione storica dell'Intelligenza Artificiale, oggi possiamo osservarne l'applicazione in una varietà di campi, rappresentando una svolta in tutto il mondo.

1.3 Applicazioni dell'Intelligenza Artificiale

L'Intelligenza Artificiale è ampiamente utilizzata in vari settori, come la medicina, la finanza, la robotica, la legge e la ricerca scientifica. In alcune situazioni, l'Intelligenza Artificiale è così inserita nella società da non essere più riconosciuta come IA. È, inoltre, impiegata nelle fabbriche per svolgere operazioni pericolose o faticose per gli esseri umani.

Ci sono varie categorie principali di utilizzo dell'IA:

- *Elaborazione intelligente dei dati*: questa categoria comprende algoritmi che analizzano dati specifici per ottenere informazioni e compiere azioni in risposta. Un esempio è il rilevamento di frodi.
- *ChatBot*: questa categoria include software che possono eseguire azioni in base a comandi vocali o testuali. Sono spesso utilizzati nel customer care aziendale per fornire assistenza ai clienti, un esempio che rientra in questa categoria è ChatGPT.
- *Computer Vision*: é un campo che si concentra su algoritmi per consentire ai computer di comprendere il contenuto di immagini o video. In questo campo, l'IA é ampiamente utilizzata nella videosorveglianza per il riconoscimento di persone.
- *Soluzioni fisiche*: questa categoria comprende applicazioni come veicoli autonomi che possono operare su strada, mare, fiumi o aria.

1.3.1 L'Applicazione dell'IA nel settore automobilistico

L'Intelligenza Artificiale ha trovato applicazione nel settore automobilistico, specialmente nell'ambito delle auto a guida autonoma. Un veicolo autonomo è un'automobile che sfrutta tecnologie avanzate per operare senza la necessità di un conducente umano. Questi veicoli fanno uso di sensori, telecamere, radar e Intelligenza Artificiale per monitorare le condizioni della strada e per muoversi in modo indipendente tra diverse destinazioni, anche su strade non appositamente predisposte.

Da ciò derivano sei livelli di automazione nelle auto (Figura 1.1):

- *SAE 0*: veicoli senza alcun sistema di assistenza alla guida, completamente manuali, dove il conducente è totalmente responsabile della guida.
- *SAE 1*: veicoli con sistemi di assistenza al conducente, come il riconoscimento della corsia e dei segnali stradali.
- *SAE 2*: veicoli con sistemi di automazione parziale, come il cruise control.
- *SAE 3*: veicoli in grado di controllare autonomamente il traffico in contesti come l'autostrada.
- *SAE 4*: veicoli ad elevata automazione, che non richiedono l'intervento del conducente nella maggior parte delle situazioni, ma consentono il controllo manuale.
- *SAE 5*: veicoli completamente autonomi, che possono operare senza un conducente a bordo.

Alcune importanti case automobilistiche, come Mercedes-Benz, Audi e, soprattutto, Tesla, stanno affrontando la sfida dello sviluppo di auto autonome. Questo implica la creazione di un software connesso che sfrutta i sensori dell'IoT, consentendo ai veicoli autonomi di comunicare in tempo reale con i sensori integrati nei segnali stradali, semafori e addirittura

nelle cosiddette "Smart Road", come quella che ci sarà tra Salerno e Reggio Calabria. Questo software utilizza molto il machine learning e l'Intelligenza Artificiale per rilevare in maniera costante ciò che lo circonda, stabilire con precisione la posizione del veicolo e fare scelte intelligenti basate sui dati raccolti.

In effetti, Tesla ha recentemente introdotto la funzione di guida completamente autonoma tramite abbonamento. L'impiego di radar, sensori e Intelligenza Artificiale nel settore automobilistico offre notevoli benefici, come la significativa riduzione del numero di incidenti stradali, dal momento che si stima che il 94% degli incidenti gravi sia causato da errori umani.

Tuttavia, questa tecnologia comporta anche svantaggi, ad esempio la possibilità che i veicoli autonomi siano soggetti a possibili attacchi informatici da parte di hacker, che potrebbero prendere il controllo dei veicoli.

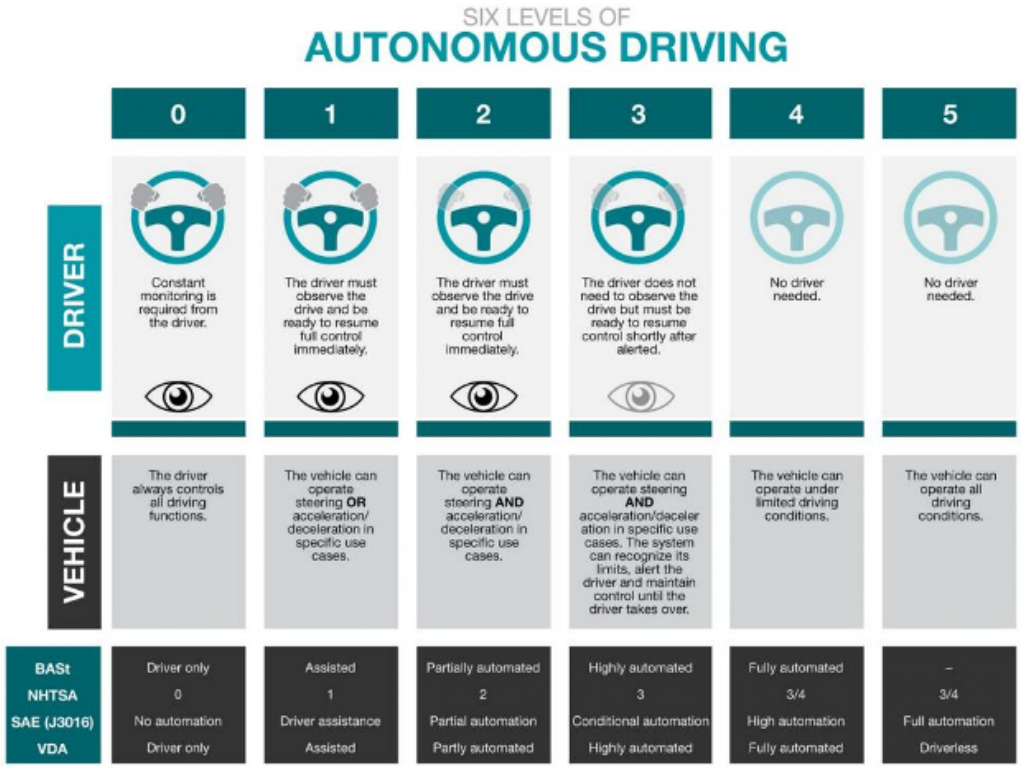

Figura 1.1: Rappresentazione grafica dei livelli di automazione nelle auto

1.3.2 L'Applicazione dell'IA nel settore aeronautico

L'Intelligenza Artificiale è fondamentale nei settori dell'aeronautica moderna. Questa tecnologia è stata sempre più utilizzata negli ultimi anni dall'industria aeronautica.

L'IA offre opportunità in ambiti come la sicurezza, la sanificazione e il controllo del traffico aereo, con recenti studi che mettono in evidenza il suo ruolo cruciale in tecnologie anticollisione, sicurezza del pilota virtuale, cybersecurity e volo autonomo.

La disponibilità di dati, algoritmi avanzati e una notevole capacità di calcolo stanno aprendo nuove prospettive, consentendo alle compagnie aeree di sfruttare l'IA per scopi come il miglioramento del servizio clienti tramite chatbot e l'ottimizzazione della manutenzione degli aeromobili.

L'IA offre il potenziale per sostenere la ripresa del settore aeronautico e ci porta verso un futuro in cui la tecnologia intelligente diventa sempre più comune nell'aviazione.

Un rapporto effettuato dalla European Union Aviation Safety Agency (EASA) ha individuato tre principali settori in cui sfruttare l'Intelligenza Artificiale nell'ambito dell'aviazione:

- Automatizzare compiti ripetitivi per ridurre la necessità di personale umano.
- Integrare il lavoro dell'uomo con quello delle macchine.
- Risolvere i problemi legati alle prestazioni umane, data la mancanza di piloti nell'industria aeronautica.

Anche le grandi aziende produttrici di aerei, come Airbus e Boeing, stanno utilizzando l'Intelligenza Artificiale e stanno sfruttando le opportunità offerte da essa. Gli aeromobili attuali sono dotati di un sistema di pilota automatico, ma per le fasi di decollo e atterraggio richiedono ancora l'intervento diretto del pilota. Airbus sta lavorando su tecnologie di Intelligenza Artificiale per consentire agli aerei commerciali di eseguire autonomamente queste operazioni, inclusa la capacità di rilevare e gestire ostacoli durante il volo. Boeing, invece, utilizza l'Intelligenza Artificiale nell'industria aeronautica, utilizzando algoritmi per migliorare l'efficienza nella produzione di aeromobili.

D'altro canto, le compagnie aeree utilizzano dati standard forniti dai produttori degli aeromobili. Tuttavia, col passare del tempo, le prestazioni degli aeroplani si deteriorano, causando una riduzione dell'efficienza e un aumento del consumo di carburante. Questo può portare a una maggiore emissione di CO_2.

Per risolvere tale problematica, la Commissione Europea ha avviato il progetto "PERF-AI", che sfrutta l'Intelligenza Artificiale per raccogliere dati sulle prestazioni dei singoli aeromobili in modo da ottimizzare le rotte di volo in base alle prestazioni effettive degli aeroplani, riducendo al minimo il consumo di carburante. Il progetto di ricerca PERF-AI ha sviluppato un modello il quale ha permesso di identificare rotte più efficienti. Queste rotte ottimizzate riducono la necessità di rifornimenti frequenti e migliorano le prestazioni degli aeromobili e hanno anche un impatto positivo sulla gestione del traffico aereo.

La gestione del traffico aereo è fondamentale per la sicurezza dei voli e l'introduzione di sistemi di automazione ha migliorato l'efficienza e la sicurezza in questo ambito. L'Intelligenza Artificiale è ora utilizzata per coordinare i voli e ridurre la dipendenza dall'intervento umano a causa del crescente traffico aereo.

Una volta che sono state esaminate le aree in cui l'Intelligenza Artificiale ha trovato applicazione possiamo iniziare a parlare dei benefici e delle limitazioni del suo impiego.

1.4 Benefici e limitazioni dell'impiego dell'Intelligenza Artificiale

La continua evoluzione dell'Intelligenza Artificiale e la sua crescente adozione in diversi settori della società moderna hanno messo in evidenza sia i vantaggi che le limitazioni associate a questa nuova tecnologia. Come spesso accade con le innovazioni, l'IA ha suscitato opinioni contrastanti e ha dato origine a due correnti di pensiero.

In maniera obiettiva, esaminiamo i benefici che l'IA sta portando al mondo contemporaneo:

- *Risparmio.* L'IA aiuta a risparmiare denaro riducendo i costi nella produzione e gli sprechi inoltre permette di risparmiare tempo automatizzando procedure lunghe.
- *Facilità di utilizzo.* Ogni persona può trarre vantaggio dall'IA nella vita di tutti i giorni.

- *Maggior qualità del lavoro.* L'IA migliora la qualità del lavoro in settori come la medicina, la ricerca scientifica, l'agricoltura e l'industria. Essa permette di rilevare malattie, aumentare la resa agricola e prevenire guasti e interruzioni nelle catene di montaggio nei settori menzionati.

- *Velocità, efficienza e meno errori.* L'IA consente di processare dati ad alta velocità con una precisione notevole in modo molto più efficiente rispetto al lavoro umano. Con l'IA è possibile automatizzare compiti ripetitivi, garantendo la precisione e la capacità di far funzionare le apparecchiature per lunghi periodi di tempo mentre i lavoratori umani, di solito, svolgono compiti di produzione per 8-10 ore al giorno. L'Intelligenza Artificiale, quindi riveste un ruolo cruciale in quei settori in cui la massima precisione e accuratezza sono fondamentali.

Questi sono solo alcuni dei vantaggi che l'IA offre. Oltre ad una serie di vantaggi, sicuramente l'Intelligenza artificiale presenta dei problemi; esaminiamone alcuni:

- *Società schiava.* Un utilizzo troppo massiccio dell'IA può aumentare la dipendenza dalla tecnologia, rendendo la società sempre più influenzata da quest'ultima.

- *Disoccupazione.* L'IA potrebbe sostituire lavoratori umani, specialmente in lavori manuali o lavori con compiti ripetitivi, sollevando preoccupazioni per la perdita di posti di lavoro in alcuni settori.

- *Privacy e sicurezza.* L'uso dell'Intelligenza Artificiale solleva preoccupazioni legate alla privacy e alla sicurezza dei dati. Un esempio di preoccupazione riguardo a ciò è il caso di ChatGPT per lo stato italiano.

- *Errori e mancanza di creatività.* L'Intelligenza Artificiale può, al contrario di come si potrebbe pensare commettere errori e presentare contenuti poco creativi.

- *Investimenti, manutenzione e aggiornamenti.* L'IA richiede investimenti significativi, con costi di sviluppo, aggiornamento e manutenzione che possono superare le migliaia di euro, rendendo difficile l'accesso per piccole imprese e imprese con capitali limitati.

1.5 L'Innovazione nel settore automobilistico: l'avvicinamento di società, aziende, team universitari e il ruolo chiave di NVIDIA nell'IA

1.5.1 Aziende automobilistiche e tecnologiche insieme

Come accennato nella sezione 1.3.2, l'importanza dell'Intelligenza Artificiale, supportata da tecnologie come radar, lidar e sensori, nel settore automobilistico è stata notevole ed ha portato a un significativo interesse e coinvolgimento da parte delle società e delle aziende.

Di seguito verranno elencati alcuni esempi di collaborazioni tra aziende automobilistiche e aziende tecnologiche che hanno abbracciato il tema dell'Intelligenza Artificiale:

- Partnership Waymo (Alphabet) e Fiat Chrysler. Waymo, la casa madre di Google, ha stretto una partnership con Fiat Chrysler Automobiles per sviluppare veicoli autonomi. Questa collaborazione ha portato a produrre veicoli Waymo, utilizzati per test su strada che hanno contribuito notevolmente all'avanzamento della tecnologia di guida autonoma (Figura 1.2).

- Partnership Intel e Mobileye con Case automobilistiche. Intel e Mobileye, un'azienda specializzata in tecnologia di guida autonoma, hanno collaborato con diversi produttori automobilistici, tra cui BMW e Volkswagen, per sviluppare sistemi di assistenza alla guida e guida autonoma.

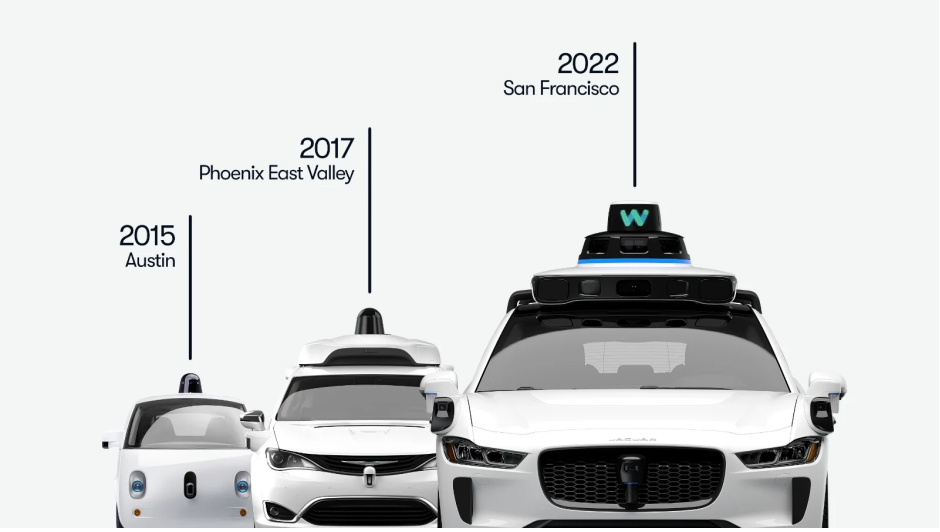

Figura 1.2: Evoluzione auto Waymo

1.5.2 L'importanza di Lunewave e NVIDIA per il futuro

Tuttavia non solo case automobilistiche ma anche aziende come Lunewave e NVIDIA, stanno contribuendo all'innovazione del settore.

In particolare NVIDIA è impegnata nello sviluppo di tecnologie avanzate per la guida autonoma, offrendo soluzioni di alta qualità che contribuiscono all'accelerazione dell'adozione di veicoli autonomi più sicuri ed efficienti.

Lunewave è una startup che ha recentemente sviluppato una nuova tecnologia radar destinata all'impiego nei veicoli autonomi. Il sistema radar creato da Lunewave si basa sulla tecnologia Luneburg lens radar, a differenza di Mobileye. Il radar sviluppato da Lunewave rappresenta un notevole avanzamento nella tecnologia di rilevamento per veicoli autonomi offrendo una maggiore precisione e affidabilità nel rilevamento dell'ambiente circostante, migliorando, così, la sicurezza e le prestazioni dei veicoli autonomi.

È proprio Lunewave che si sta occupando di sviluppare NVIDIA DRIVE.

1.5.3 NVIDIA DRIVE

NVIDIA DRIVE è una piattaforma aperta progettata per veicoli autonomi che consente ai produttori di veicoli di personalizzare e sviluppare soluzioni di guida autonoma in modo flessibile senza compromettere sicurezza o qualità. NVIDIA DRIVE Hyperion, versione avanzata di questa piattaforma, è una sottopiattaforma pronta per l'uso in produzione per veicoli autonomi.

NVIDIA DRIVE Hyperion presenta due componenti software:

- DRIVE AV. La piattaforma DRIVE Chauffeur comprende diverse funzionalità, tra cui sistemi di percezione avanzati, mappatura dettagliata e reti neurali addestrate con dati reali di guida. Questa piattaforma è realizzata utilizzando NVIDIA DRIVE Perception, progettato per rilevare e classificare oggetti, spazi percorribili, corsie e segnaletica

orizzontale, nonché semafori e segnali, in modo tale da gestire con successo una varietà di situazioni di traffico, sia in contesti urbani che autostradali, fornendo così, una soluzione completa per la vera e propria guida autonoma del veicolo.

- DRIVE IX. L'applicazione DRIVE Concierge funge da assistente intelligente per garantire la sicurezza e l'attenzione del conducente; inoltre si focalizza sull'esperienza dei passeggeri e offre servizi digitali avanzati. Essa consente ai passeggeri di personalizzare la loro esperienza a bordo, inclusi i display e l'audio nelle diverse zone del veicolo, per garantire un'esperienza di viaggio su misura. DRIVE Concierge semplifica il parcheggio, mostrando segnali stradali e aiutando a individuare spazi di parcheggio disponibili (Figura 1.3).

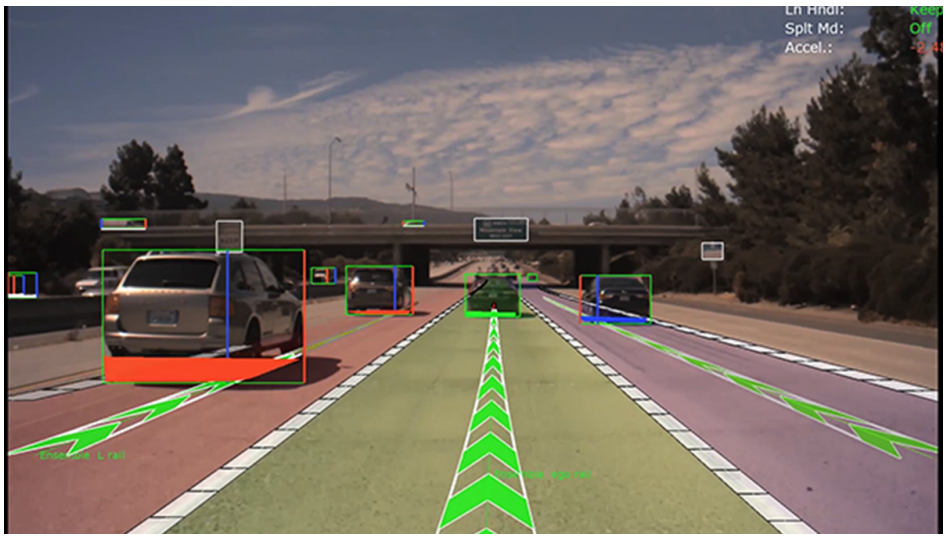

Figura 1.3: Grafica NVIDIA DRIVE Perception

Grazie all'integrazione di queste piattaforme, il guidatore può verificare le intenzioni dell'Intelligenza Artificiale attraverso un interfaccia 4D a 360 gradi.

Oltre a tutte queste aziende anche alcuni team universitari stanno cercando di utilizzare l'Intelligenza Artificiale per migliorare le prestazioni delle vetture e sviluppare sistemi di guida autonoma, come sarà richiesto dalle competizioni.

1.5.4 L'IA nel contesto del team SAE di UNIVPM

La Formula SAE (Society of Automotive Engineers) è una competizione universitaria internazionale in cui gli studenti hanno come obiettivo quello di progettettare e costruire una vettura da corsa per gareggiare in delle competizioni. Le competizioni che affrontano stanno progressivamente includendo categorie di vetture senza conducente, note come "driverless", al punto da inserire specifiche sezioni dedicate a questo tema all'interno del regolamento (FS-Rules).

Come discusso nelle sezioni precedenti, abbiamo visto che la guida autonoma, conosciuta come "driverless", è resa possibile grazie all'impiego di sensori avanzati e sistemi di Intelligenza Artificiale (IA). Alla luce di questa conoscenza, abbiamo cercato di capire come i membri del team della formula SAE dell'Università Politecnica delle Marche stiano attualmente utilizzando o pianificano di sfruttare l'IA per sviluppare vetture in grado di guidare autonomamente.

Dopo un colloquio approfondito con i responsabili del team, abbiamo potuto notare che, al momento, l'utilizzo dell'Intelligenza Artificiale (IA) per sviluppare vetture da corsa

autonome non è parte integrante delle attività del team Formula SAE; tale scelta è sicuramente dettata dalla complessità che l'IA rappresenta.

Tuttavia, durante il colloquio, è emerso un interesse crescente verso l'utilizzo dell'IA per la guida autonoma, motivato anche dal fatto che, in futuro, le competizioni potrebbero richiedere veicoli senza conducente come requisito per ottenere punteggi massimi (Figura 1.4). Ad esempio, potrebbe essere previsto che le monoposto tradizionali ottengano fino a 800 punti su 1000, mentre i veicoli driverless abbiano l'opportunità di guadagnare il punteggio massimo.

Questo interesse in crescita deriva dal fatto che, a partire da febbraio o in un momento analogo, verrà costituito un nuovo team specificamente dedicato alla ricerca e allo sviluppo nell'ambito dell'Intelligenza Artificiale, con l'obiettivo di esplorare questo campo.

Durante la discussione, i responsabili hanno condiviso la loro idea iniziale che è quella di utilizzare radar, lidar e sensori, supportati da sistemi di Intelligenza Artificiale di dimensioni ridotte (micro IA), al fine di eseguire diverse operazioni (ad esempio per garantire una pulizia completa dei filtri per avere dei risultati ottimi e precisi), con l'obiettivo principale di consentire al veicolo di guidare autonomamente lungo un percorso specificamente tracciato da birilli gialli e blu, in conformità con il regolamento della competizione.

Un aspetto negativo è che, mediante analisi dettagliate, sono stati individuati dei limiti che l'Intelligenza Artificiale potrebbe incontrare. Un esempio si ritrova nei sistemi di gestione delle batterie (BMS), poiché l'impiego dell'IA potrebbe comportare una complessità aggiuntiva rispetto all'utilizzo di BMS tradizionali.

Nonostante ciò l'obiettivo futuro è quello di implementare queste innovazioni anche sulle attuali vetture a combustione, per cercare, poi, di adattarle alla futura vettura elettrica. Data la limitata conoscenza nel campo dell'IA per il driverless, le consulenze e il supporto da parte degli sponsor saranno fondamentali.

In sintesi, il team UNIVPM Formula SAE sta considerando seriamente l'introduzione dell'IA e della guida autonoma come parte integrante, dimostrando di volersi impegnare nell'affrontare le difficoltà che ciò comporta.

Figura 1.4: Monoposto formula SAE Driverless

CAPITOLO 2

Il Deep Learning e il Reinforcement Learning

Nel corso di questo capitolo, esamineremo il Deep Learning e il Reinforcement Learning in modo approfondito. All'interno del vasto panorama dell'Intelligenza Artificiale, emergono questi due approcci di notevole importanza. Questo capitolo serve come punto di partenza per una comprensione approfondita di entrambe le discipline, offrendo una visione dei principi fondamentali, delle diverse applicazioni, dei vantaggi e delle limitazioni associate a ciascuna di esse. Tale capitolo è di fondamentale importanza per acquisire una solida base di conoscenza su come il Deep Learning e il Reinforcement Learning possano affrontare le sfide dell'Intelligenza Artificiale.

2.1 Introduzione generale al Deep Learning e al Reinforcement Learning

2.1.1 Definizioni e cenni storici

Il Deep Learning e il Reinforcement Learning rappresentano due approcci distinti nell'ambito dell'Intelligenza Artificiale. In particolare, il Machine Learning comprende queste due tecniche. Il Deep Learning è un sottocategoria dell'Intelligenza Artificiale che sfrutta reti neurali artificiali con strati multipli, spesso composte da tre o più livelli. Queste reti neurali, ispirate alla struttura del cervello umano, sono in grado di apprendere e rappresentare grandi quantità di dati complessi attraverso una gerarchia di strati di rappresentazione.

L'obiettivo principale del Deep Learning è quello di estrarre caratteristiche significative dai dati in modo automatico.

L'aspetto che contraddistingue il Deep Learning è la sua capacità di creare rappresentazioni di dati stratificate formando una piramide dell'informazione dove ogni strato successivo delle reti neurali costruisce concetti più astratti basandosi su quelli dei livelli inferiori. Ogni strato della rete consente di comprendere e rappresentare dati in modo più accurato e complesso.

Il Reinforcement Learning, noto anche come apprendimento per rinforzo, è un paradigma nell'ambito dell'apprendimento automatico che si concentra sulla creazione di agenti autonomi in grado di prendere decisioni per raggiungere obiettivi specifici interagendo con l'ambiente circostante. Questo approccio rappresenta uno dei tre principali pilastri dell'apprendimento automatico, insieme all'apprendimento supervisionato e a quello non supervisionato. Tuttavia, si distingue dagli altri due perché riguarda decisioni che prendiamo in sequenza, una dopo l'altra, in cui le azioni sono influenzate dallo stato corrente del sistema e hanno impatto sul suo futuro sviluppo.

Il processo di apprendimento nel Reinforcement Learning coinvolge l'interazione tra un agente e un ambiente dinamico senza il coinvolgimento diretto dell'essere umano; nel Reinforcement Learning, quando un agente prende una decisione o compie un'azione, riceve una specie di punteggio chiamato "ricompensa". Questo punteggio è un premio o un'incoraggiamento che l'agente ottiene quando esegue qualcosa di corretto. L'obiettivo è fare in modo che l'agente capisca come ottenere punteggi più alti, cioè compiere azioni che portano a risultati migliori attraverso cicli di tentativi ed errori. Quindi, le ricompense sono il modo in cui l'agente impara cosa fare per ottenere buoni risultati nel contesto in cui si trova.

Entrambe le tecniche del Deep Learning e del Reinforcement Learning hanno radici storiche profonde. Esaminiamo alcune tappe significative nella storia di entrambi questi approcci.

La storia del Deep Learning può essere suddivisa in tre fasi distinte: la fase cibernetica, quella del connettivismo e il moderno Deep Learning, il cui termine è stato coniato di recente:

- *La fase della cibernetica:* questa fase si riferisce ai primi momenti della storia del Deep Learning, quando iniziarono a emergere i primi modelli matematici lineari nel tentativo di comprendere il funzionamento del cervello.

  Tra gli anni più importanti in questa fase ricordiamo:

  - *1943:* Warren McCulloch e Walter Pitts creano un modello per reti neurali basato sulla matematica.
  - *1958:* Frank Rosenblatt inventa il perceptron, algoritmo che si basa su una rete neurale di computer a due strati e che utilizza operazioni di somma e sottrazione per il riconoscimento di schemi.

  Tuttavia nel 1969 gli algoritmi come il Perceptron iniziano ad essere molto criticati e la ricerca si ferma.

- *La fase del connessionismo:* durante questa fase, la quale si riferisce al periodo a cavallo tra gli anni '80 e 90, sono state sviluppate reti neurali profonde e stratificate che costituiscono la base dell'attuale Deep Learning.

  Tra gli anni cruciali di questa fase, ricordiamo:

  - *1980:* Kunihiko Fukushima ha sviluppato il Neoconitron, un tipo di rete neurale artificiale con una struttura a più livelli la quale è stata utilizzata per insegnare a un computer come riconoscere la scrittura a mano.
  - *1990:* nascono le prime reti Long Short Term memory, molto più sviluppate delle precedenti.
  - *1992:* Weng introduce Cresceptron per effettuare il riconoscimento automatico di oggetti tridimensionali in ambienti caotici.

  I limiti relativi alla potenza di calcolo dell'epoca causano una nuova fase di disinteresse e disinvestimento per le reti neurali.

- *Il Deep Learning:* l'interesse verso il deep learning è rinnovato negli anni duemila grazie all'hardware più potente, alla disponibilità di grandi quantità di dati e allo sviluppo di algoritmi più sofisticati.

  Elenchiamo alcuni anni importanti:

  - *2006:* la popolarità del termine "Deep Learning" inizia a crescere quando Geoffrey Hinton dimostra come sia possibile addestrare una rete neurale strato per strato.

- *2015:* Facebook utilizza DeepFace, una tecnologia di apprendimento profondo per identificare e "taggare" automaticamente gli utenti.
- *2016:* l'algoritmo AlphaGo sviluppato da Google sconfigge il campione professionista di Go.

La storia del Reinforcement Learning condivide alcune tappe importanti con quella del Deep Learning. Esaminiamo quindi alcuni anni fondamentali che hanno contribuito all'evoluzione di questo paradigma di apprendimento.

Le radici del Reinforcement Learning risalgono agli '50 e '60, periodo in cui Bellman introduce il termine di programmazione dinamica per descrivere la risoluzione di problemi sequenziali.

Successivamente, nel 1980, Watkins propone il metodo Q-learning, uno degli algoritmi più noti del Reinforcement Learning.

Nel corso del decennio che va dal 1990 al 2000, l'utilizzo del Reinforcement Learning è aumentato in molti settori. Nasce durante questa fase, l'algoritmo SARSA, che si focalizza su come gli agenti apprendano a compiere scelte in ambienti dinamici.

Nel decennio tra il 2000 e il 2010, grazie all'aumento della potenza di calcolo e alla disponibilità dei dati, il Reinforcement Learning ha ottenuto numerosi successi in molti settori, un esempio è quello della guida autonoma.

Ad oggi, il Reinforcement Learning è una delle componenti dell'IA più promettenti che, strettamente connesso con il Deep Learning, forma il Deep Reinforcement Learning, il quale è fondamentale nella risoluzione di problemi complessi, grazie alla sua capacità di addestrare agenti per produrre risultati migliori che lo rende di crescente importanza in molti settori (Figura 2.1).

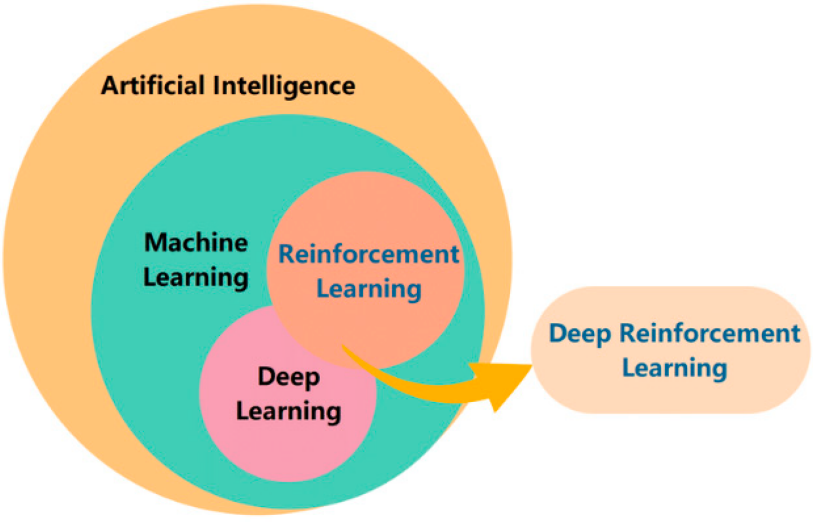

Figura 2.1: Relazione tra IA, Machine Learning, Deep Learning e Reinforcement Learning

Terminata questa introduzione iniziale, analizziamo in dettaglio singolarmente il Deep Learning e il Reinforcement Learning.

2.2 Concetti di base del Deep Learning

Come già visto nella Sezione 2.1.1, il punto centrale del Deep Learning sono le reti neurali artificiali. Cerchiamo di analizzarle.

2.2.1 Neuroni artificiali e reti neurali

Un neurone artificiale, conosciuto come Perceptron, è un'unita fondamentale di calcolo o di elaborazione all'interno di una rete artificiale di neuroni. L'obiettivo con cui è stato creato, è quello di imitare in maniera molto semplificata ma efficiente il funzionamento dei neuroni biologici presenti nel cervello dell'uomo (Figura 2.2).

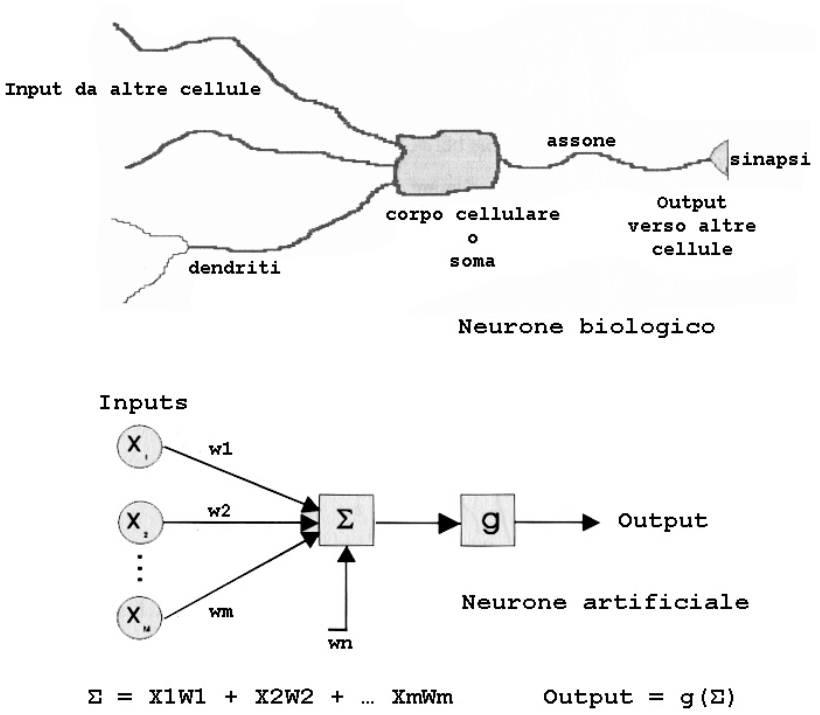

Figura 2.2: Neurone biologico e artificiale a confronto

Il funzionamento del neurone artificiale, di fondamentale importanza all'interno delle reti neurali, può essere diviso in varie fasi:

- *Input:* ogni neurone artificiale riceve uno o più segnali d'ingresso noti come input. A ciascun segnale viene assegnato un peso che ne indica l'importanza; infatti un peso maggiore assegna maggior importanza all'input associato mentre un peso minore assegna minor importanza al segnale. Questa fase è fondamentale in quanto aiuta la rete neurale a scegliere le informazioni più rilevanti in funzione del compito che sta svolgendo.
- *Somma Ponderata:* a partire da ogni segnale, si ottiene un punteggio che è ottenuto moltiplicando il peso dell'input all'input stesso. Questi punteggi vengono tutti sommati in quella che prende il nome di somma ponderata. Questa fase è importante poichè influisce fortemente su quello che sarà il segnale di uscita o output del neurone.
- *Bias:* valore costante specifico per ciascun neurone all'interno di una rete neurale. Il bias in funzione alla sua impostazione influenza la sensibilità che il neurone ha nei confronti degli input.

- *Funzione di Attivazione:* dopo aver calcolato la somma ponderata, entra in gioco la funzione di attivazione la quale determina se un neurone deve diventare attivo e produrre un output (Figura 2.3). Esistono diversi tipi di funzioni di attivazione, tra le più comuni abbiamo:

  – *Funzione Sigmoide:* una funzione matematica che trasforma un numero reale in un intervallo tra 0 e 1. Se l'input è grande e positivo, il neurone si "accende" con un valore vicino all'1, altrimenti rimane spento con un valore vicino a 0. La sua formula matematica è la seguente:

  $$S(X) = \frac{1}{1 + e^{-x}}$$

  Questa funzione di attivazione presenta benefici e limitazioni; in particolare:

  * La funzione sigmoide è una curva a forma di "S".
  * È differenziabile, quindi è possibile calcolarne il gradiente, risultando, per questo, molto utile per l'addestramento delle reti.
  * Soffre del problema della "sparizione del gradiente" e, per questo, è stata sostituita con la funzione ReLU.

  – *Funzione ReLU:* restituisce l'input se è positivo, zero se l'input è negativo. La sua espressione matematica è la seguente:

  $$f(X) = max(0, x)$$

  Anche questa funzione di attivazione presenta varie caratteristiche; in particolare:

  * È una funzione non lineare per valori negativi, e questo la rende essenziale per catturare le relazioni complesse nei dati.
  * Non presenta il problema della "sparizione del gradiente"; quindi, durante l'addestramento, la retropropagazione del gradiente funziona in maniera più efficiente.
  * Introduce il concetto di sparsità dei neuroni che implica l'attivazione di alcuni neuroni migliorando, così, l'efficienza computazionale.
  * Soffre del problema noto come "morte del neurone", in cui un neurone rimane inattivo per tutti gli input negativi e non contribuisce all'apprendimento della rete.

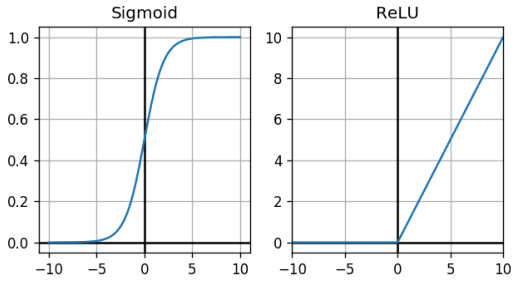

Figura 2.3: Grafici relativi alle funzioni di attivazione

- *Output:* è il risultato finale del calcolo che avviene all'interno del neurone. Questo risultato viene trasmesso agli altri neuroni della rete. L'output di un neurone è essenziale per la comunicazione e la collaborazione tra i neuroni all'interno di una rete neurale.

Un insieme di neuroni artificiali interconnessi tra loro che elabora dati prende anche il nome di rete neurale artificiale. Le reti neurali sono note anche come SNN (Simulated Neural Network) e ANN (Artificial Neural Network).

Queste reti vengono addestrate per apprendere e migliorare le loro abilità nel compiere specifici compiti. Addestrare una rete neurale significa modificare i parametri allenabili, come i pesi delle connessioni tra i neuroni e i bias, in modo che la rete possa apprendere dai dati e ottimizzare le sue prestazioni. Una volta ottimizzate, esse sono degli strumenti molto potenti nel campo dell'IA; infatti sono utilizzate nel riconoscimento delle immagini e nell'elaborazione del linguaggio naturale poichè sono in grado di organizzare e analizzare dati in maniera veloce ed efficiente.

Le reti neurali artificiali (ANN) sono strutturate in diversi strati di nodi detti layer, che includono un livello di input, uno o più livelli nascosti e un livello di output.

Una rete neurale, nella maggior parte dei casi, viene chiamata fully-connected ed è un tipo di rete neurale artificiale in cui ciascun neurone in uno strato è strettamente interconnesso a tutti i neuroni nello strato successivo.

Una rete neurale semplice ha un singolo strato nascosto, mentre le reti neurali avanzate, o Deep Neural Networks (DNN), possono contenere molti strati nascosti (di solito almeno tre o più), da qui il termine "Deep" (Figura 2.4).

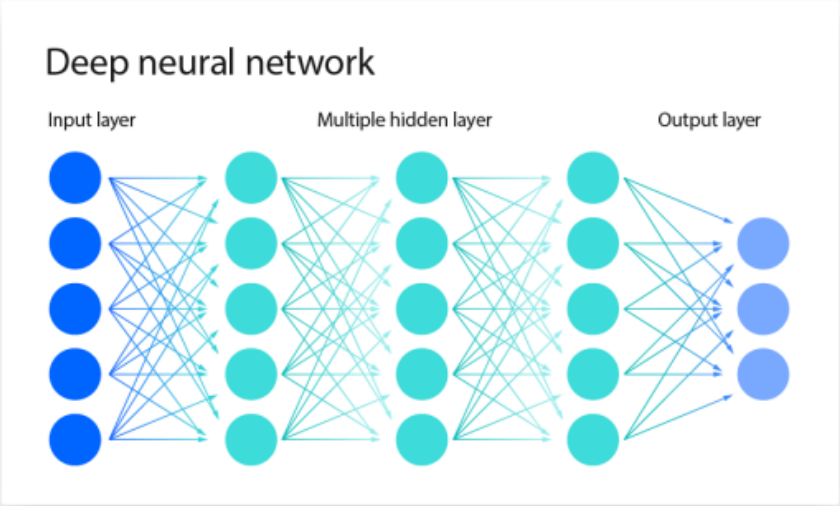

Figura 2.4: Organizzazione a strati di una rete neurale profonda

Analizziamo i vari strati delle reti neurali:

- *Strato di input:* l'input layer è il punto iniziale della rete neurale, noto come strato di input; esso riceve l'informazione iniziale da un insieme di dati o da una fonte esterna. Ciascun nodo (neurone) rappresenta una caratteristica o un'elemento di dati di input.
- *Strato nascosto:* l'hidden layer, chiamato così poichè i calcoli svolti in questi strati non sono direttamente visibili dall'utente, è il punto centrale della rete neurale; infatti si trova tra lo strato di input e di output. Gli strati nascosti hanno un'importante funzione nell'elaborazione dei dati e nell'acquisizione di conoscenze nel processo di apprendimento delle reti neurali.
- *Strato di output:* l'output layer è il punto finale della rete neurale; noto come strato di output, fornisce il risultato finale della rete. La configurazione dello strato di output,

ossia il numero degli strati di output e le funzioni di attivazione utilizzate, può variare a seconda del tipo di compito che la rete è progettata per affrontare.

Architetture di una Deep Neural Network

Esistono diverse architetture che una rete neurale profonda può presentare, dalle più semplici alle più avanzate.

- *Architetture semplici:*
  - *FFNN (FeedForward Neural Network:)* è un tipo di rete neurale artificiale in cui il flusso di informazioni si muove sempre in avanti, partendo dallo strato di input e finendo nello strato di output, passando per lo strato nascosto.
- *Architetture avanzate:* tra i sistemi di Deep Learning più importanti e diffusi abbiamo le RNN e le CNN. Esaminiamo in dettaglio tali architetture nella prossima sottosezione.

Recurrent Neural Networks e Convolutional Neural Networks
Le RNN (Recurrent Neural Networks), o reti neurali ricorrenti, sono un tipo di reti neurali artificiali progettate per lavorare con dati sequenziali. Questa architettura di DL è molto utilizzata in applicazioni che coinvolgono dati che si presentano nel tempo o in sequenza, come la traduzione linguistica, il riconoscimento di immagini o l'elaborazione del linguaggio naturale; infatti sono parte integrante di applicazioni come Siri o Google Translate. Le RNN presentano due caratterstiche che le distinguono dalle altre reti; esse sono:

- *Memoria:* le *RNN* sono in grado di memorizzare informazioni a partire dai segnali di input passati ed utilizzarle sia per input che output ricorrenti; infatti tengono conto delle relazioni tra input e output a differenza delle tradizionali DNN, rendendo possibile la previsione di risultati futuri basandosi su eventi precedenti.
- *Condivisione dei pesi:* a differenza delle reti FeedForward e Convolutional in cui ogni nodo ha un peso diverso, nelle RNN, il peso è identico per tutti i nodi dello stesso layer di rete, favorendo, così, l'apprendimento e l'adattamento alle sequenze di dati. Per addestrare le RNN, e quindi per aggiustare i pesi, viene utilizzato un algoritmo di retropropagazione nel tempo (BPTT - Backpropagation Through Time) specifico per dati sequenziali.

  Il calcolo dei gradienti ci consente di conoscere la pendenza della *funzione di perdita* (è una misura dell'errore tra le previsioni del modello e i valori di output attesi) rispetto ai pesi della rete e, di conseguenza, aggiornare i pesi in modo che la rete possa migliorare le sue prestazioni nel compito specifico per cui è stata addestrata. Il funzionamento del BPTT si suddivide in tre parti:
  - *Forward Pass:* è un processo iterativo; i dati in una RNN vengono passati all'interno della rete a partire dall'input per arrivare all'output, strato per strato, uno step temporale alla volta, detto anche "slice" dei dati sequenziali.
  - *Calcolo degli errori:* dopo aver eseguito il Forward Pass il modello produce un output previsto per un determinato step temporale. Il risultato di output associato a ciascun elemento in una sequenza di dati ha un valore atteso. A questo punto viene calcolato l'errore attraverso la funzione di perdita la quale attribuisce un punteggio che indica quanto il modello si è avvicinato o allontanato dal valore atteso.

– *Backward Pass:* inizia dalla fine della sequenza e si sposta indietro in maniera iterativa; per ciascun step temporale calcola i gradienti degli errori che rappresentano quanto l'errore dell'output del modello in quel punto specifico è influenzato dalla variazione dei pesi della rete. L'obiettivo principale è ridurre l'errore misurato con la funzione di perdita; per farlo si utilizza in maniera ricorsiva l'algoritmo di "discesa del gradiente" per aggiornare i pesi fin tanto che l'errore diventa accettabile.

In conclusione possiamo affermare che le RNN utilizzano il BPTT perchè permette di calcolare i gradienti e, successivamente anche di aggiornare i pesi e il bias della rete evitando di incappare in problemi come "gradienti troppo piccoli", che impediscono all'algoritmo di apprendere, e "gradienti troppo grandi", che provocano l'instabilità dei pesi.

Le Recurrent Neural Network si dividono in alcune tipologie, tra cui ricordiamo:

– *BRNN (Bidirectional Recurrent Neural Networks):* rappresentano una variante delle RNN e, a differenza di queste, sono in grado di ottenere informazioni a partire da input passati e futuri, migliorando l'accuratezza delle previsioni. Riescono a guardare avanti e indietro nel tempo.
– *LSTM (Long Short-Term Memory):* sono utilizzate per risolvere il problema delle dipendenze a lungo termine in una sequenza di dati. Le LSTM presentano una struttura con delle celle nei livelli nascosti, dove ogni cella è, a sua volta, suddivisa in tre porte, le quali controllano il flusso di informazioni fondamentale per effettuare previsioni circa l'output nella rete. Le tre porte sono le seguenti:
  * *Porta di input:* detta anche *input gate*, determina quali dati dell'input sequenziale sono pertinenti per il contesto in cui la rete agisce.
  * *Porta di output:* detta anche *output gate*, controlla quali informazioni nella cella dovrebbero essere utilizzate per generare l'output attuale e quali dovrebbero essere utilizzate per effettuare previsioni future.
  * *Porta di dimenticanza:* detta anche *forget gate*, determina quali informazioni dovrebbero essere dimenticate.
– *GRU (Gated Recurrent Unit):* sono utilizzate per affrontare il problema della memoria a breve termine delle RNN; a differenza delle LSTM che utilizzano uno "stato di cella", queste reti utilizzano degli stati nascosti in cui sono presenti due porte:
  * *Porta di reimpostazione:* detta anche *reset gate*, determina quali informazioni dall'input sequenziale dovrebbero essere considerate.
  * *Porta di aggiornamento:* detta anche *update gate*, indica come lo stato corrente deve essere aggiornato in funzione degli input passati e futuri.

Le Convolutional Neural Networks o CNN sono reti neurali artificiali, notevoli per le loro prestazioni superiori nell'elaborazione di immagini, dati vocali e segnali audio. Queste reti includono tre tipi di strati principali:

– *Strato convoluzionale:* è una parte fondamentale in una CNN ed è responsabile della maggior parte dei calcoli.
  Funziona grazie a tre elementi che sono i dati in input, un filtro (o kernel) e una mappa delle caratteristiche. Supponiamo di avere un'immagine a colori come input; questa è una matrice tridimensionale di pixel, con altezza, larghezza e

profondità. Il filtro è un piccolo array bidimensionale di pesi che rappresenta una parte dell'immagine. Di solito ha dimensioni 3x3.

Il processo di convoluzione inizia quando il filtro viene sovrapposto su un'area dell'immagine. Per calcolare il valore nella mappa delle caratteristiche ossia l'output, viene eseguito il prodotto scalare tra i valori dei pixel dell'immagine sottostante e i pesi del filtro corrispondenti. Il risultato di questo prodotto scalare è il valore in quella posizione specifica della mappa delle caratteristiche. Questo procedimento continua fino a quando il filtro ha attraversato l'intera immagine; questo processo di convoluzione viene eseguito per ciascun filtro e ognuno di questi è progettato per rilevare una caratteristica diversa nell'immagine. La mappa delle caratteristiche risultante rappresenta le caratteristiche estratte dall'immagine e viene utilizzata nei passaggi successivi della CNN.

– *Strato di Pooling:* noto anche come sottocampionamento, permette la riduzione della dimensionalità dell'input, riducendo il numero di parametri in esso. In questo livello, un filtro, detto kernel, viene applicato a tutti i segnali di input, ed effettua un'aggregazione di questi valori producendo un unico valore di output. Esistono due tipi di pooling:
  * *Pooling massimo:* durante l'applicazione del filtro, viene selezionato il valore massimo tra i pixel aggregati da inviare all'output.
  * *Pooling medio:* in questo caso il filtro calcola la media dei valori aggregati e invia questo valore medio all'array di output.

  Il livello di Pooling presenta dei benefici e degli svantaggi per la rete. Sicuramente contribuisce alla riduzione della complessità della rete, migliora l'efficienza, e la riduzione della dimensionalità aiuta a gestire meglio le informazioni rilevanti nelle immagini o nei dati di input, concentrandosi su caratteristiche chiave; d'altro canto provoca la perdita di alcune informazioni.
– *Strato fully-connected:* in questo livello, ogni nodo nel layer di output è direttamente collegato a ciascun nodo nel layer precedente, creando una rete densamente connessa. A differenza dei livelli precedenti che utilizzano una funzione di attivazione detta ReLU, questo strato utilizza una funzione di attivazione, detta SoftMax, la quale assegna etichette e classi ai segnali in input. In conclusione, quindi, questo strato svolge un ruolo chiave nella fase di classificazione, contribuendo a prendere una decisione finale sulla categoria o l'etichetta da assegnare ai dati di input in funzione delle rappresentazioni di questi ultimi nei layer precedenti.

Le reti neurali, grazie alla loro capacità di apprendimento di informazioni complesse, hanno rivoluzionato molti settori. Andiamo, quindi, ad osservare in quali settori trova applicazione il Deep Learning.

2.3 Applicazioni del Deep Learning

Il Deep Learning rappresenta, quindi, una della principali ragioni di successo dell'Intelligenza Artificiale. Come già accennato nelle sezioni precedenti, il DL, attraverso l'utilizzo di rete neurali, è molto importante per il riconoscimento di immagini, vocale, il trattamento del linguaggio naturale, e molto altro. Oltre a questo sono molti i settori in cui questa tecnica sta ottenendo molto successo.

Uno dei campi in cui si fa uso del Deep Learning, è sicuramente, quello dei *ChatBot*, per l'assistenza clienti in cui sta permettendo di liberare risorse umane da compiti ripetitivi.

Un altro settore di rilievo è rappresentato dagli *assistenti virtuali*, tra i quali troviamo noti esempi come Alexa e Siri. Questi assistenti sfruttano tecnologie di Deep Learning per apprendere e migliorare continuamente tramite le interazioni degli utenti.

Il Deep Learning è, inoltre, molto impiegato nel settore della *sicurezza* e delle *operazioni finanziarie*, con l'obiettivo di rilevare violazioni di sicurezza dei dati. Questi sistemi antifrode identificano possibili attacchi alla sicurezza bancaria, sia nelle transazioni con le carte di credito che attraverso i dispositivi POS.

Un'altra area di importanza notevole in cui il DL trova ampio utilizzo è quello della *medicina*, per migliorare la precisione e l'efficienza della diagnosi medica, in particolare per la diagnosi precoce (un esempio è il cancro), poiché è in grado di individuare piccole anomalie che potrebbero sfuggire all'occhio umano.

In un settore altrettanto fondamentale, come quello *automobolistico*, il Deep Learning viene ampiamente impiegato nei veicoli a guida autonoma, per il rilevamento di segnaletica stradale e ostacoli, migliorando la sicurezza nelle strade.

Guardando al futuro, il Deep Learning rappresenta un potenziale enorme per l'IA, e le sue applicazioni continueranno a crescere e a evolversi.

2.4 Limiti e vantaggi del Deep Learning

Nella parte conclusiva di questa trattazione del Deep Learning, approfondiamo i benefici e gli svantaggi che questa tecnica presenta.

Tra i principali benefici troviamo i seguenti:

- Viene applicato in moltissimi settori.
- È in grado di apprendere e rilevare immagini e pattern basandosi su caratteristiche varie.
- Rende automatizzate operazioni complesse, portando alla riduzione degli errori dell'uomo.
- Riesce ad adattarsi e ad apprendere nuove informazioni in maniera continua.

Invece, tra i principali svantaggi, abbiamo i seguenti:

- C'è rischio di overfitting, ossia la rete neurale apprende troppi dati perdendo la capacità di generalizzazione.
- Richiede notevoli quantità di risorse e dati etichettati molto difficili da ottenere, determinando, così, un costo elevato per l'addestramento.
- Le scelte prese dalle reti neurali sono paragonate a una "black box".
- Richiede manutenzione e aggiornamenti in maniera continua.

Conclusa la trattazione del Deep Learning, nelle sezioni successive ci concentreremo sull'analisi approfondita del Reinforcement Learning.

2.5 Fondamenti del Reinforcement Learning

Come già evidenziato nella Sezione 2.1.1, il Reinforcement Learning rappresenta un ramo essenziale del Machine Learning. Questa disciplina si concentra sul processo di addestramento di agenti autonomi affinché possano prendere decisioni ottimali attraverso l'interazione con il mondo che li circonda. Per comprendere appieno il Reinforcement Learning, è cruciale approfondire le sue fondamenta di base, che costituiscono il fulcro su cui poggia questa disciplina.

2.5.1 Agenti, ambiente e la loro interazione

I primi elementi chiave, del Reinforcement Learning che analizzeremo sono gli agenti e l'ambiente. Questi rappresentano il cuore pulsante di questa tecnica.

- *Gli agenti:* rappresentano le entità che prendono decisioni nell'ambiente circostante. Essi possono presentarsi in una vasta gamma di forme, che vanno da entità puramente software a veri e propri robot fisici. Questa diversità di natura consente al Reinforcement Learning di essere applicato in una molteplicità di contesti; gli agenti software possono essere implementati su computer o dispositivi mobili, mentre i robot fisici rappresentano agenti che operano nel mondo reale, compiendo azioni tangibili e interagendo con l'ambiente fisico che li circonda. La loro funzione primaria è quella di agire ed apprendere, orientando la scelta di azioni e decisioni in base alle esperienze pregresse con l'obiettivo di ottimizzare le ricompense.
- *L'ambiente:* rappresentano il luogo o lo spazio circostante in cui gli agenti operano. Così come gli agenti, anche l'ambiente può manifestarsi in varie forme diverse, che spaziano dal mondo reale all'ambiente simulato al computer, garantendo una flessibilità che consente applicazioni in vari settori.
- *L'interazione:* l'agente interagisce costantemente con l'ambiente circostante attraverso un processo di "tentativi ed errori". Cerchiamo di interpretare meglio la loro interazione (Figura 2.5): in un momento specifico, chiamato *istante t*, sia l'agente che l'ambiente si trovano in una condizione ben precisa, chiamata *stato s*.

  Lo stato descrive una situazione specifica includendo, ad esempio la posizione dell'agente, la temperatura dell'ambiente e altre informazioni rilevanti. Una volta che l'agente acquisisce informazioni circa lo stato s, prende una decisione e compie un'azione passando all'istante temporale successivo, cioè $t + 1$, ricevendo una *ricompensa* da parte dell'ambiente. Si genera, così, la "coppia stato-azione", che indica quale azione dovrebbe essere intrapresa in un particolare stato.

2.5.2 Altri componenti chiave del Reinforcement Learning

Esistono altri quattro componenti chiave che operano in modo cooperativo per guidare l'agente nel suo processo decisionale.

- *Politica:* detta anche *policy*, rappresenta la strategia decisionale, ovvero determina come l'agente reagisce agli stimoli dell'ambiente. Essa è una funzione matematica che specifica le azioni che l'agente dovrebbe eseguire in risposta ad uno stato dell'ambiente circostante. Esistono diversi tipi di policy; ovvero:

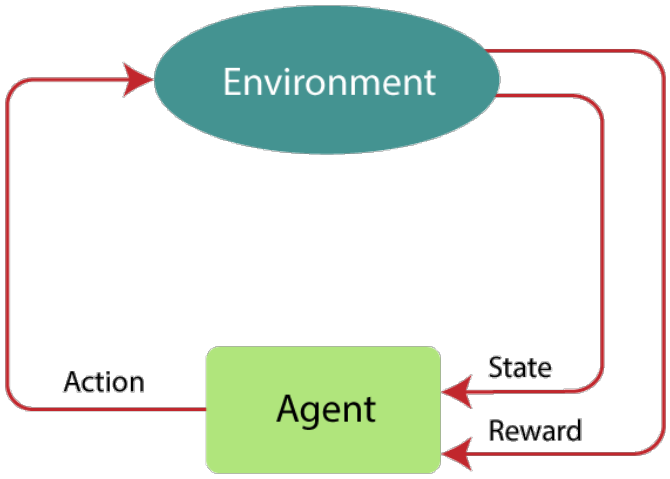

Figura 2.5: Ciclo di interazione agente-ambiente nel Reinforcement Learning

  - *Policy deterministica:* ad ogni stato specifico dell'ambiente è associata una singola azione, in modo tale che l'agente prende delle decisioni in maniera prevedibile.
  - *Policy stocastica:* ad ogni stato specifico dell'ambiente è associata una distribuzione di probabilità di diverse possibili azioni, in modo tale che l'agente quando si trova in uno stato particolare, ha la possibilità di scegliere tra diverse azioni, ciascuna con una probabilità associata.

- *Segnale di ricompensa:* detto anche *reward signal*, è un punteggio numerico che valuta la qualità delle azioni eseguite dall'agente in uno stato specifico, il quale può essere positivo o negativo in base ai risultati prodotti dalle azioni dell'agente. Questo reward è molto importante in quanto influenza le future decisioni dell'agente. Se un'azione ha ottenuto un punteggio positivo, l'agente sarà incline a ripetere azioni simili. Il segnale di reward definisce così quali sono i buoni ed i cattivi comportamenti per l'agente. Questo segnale è la base di partenza per aggiornare la policy utilizzata; se un'azione selezionata dalla policy è seguita da un reward basso, allora la policy potrebbe cambiare per selezionare qualche altra azione in una situazione futura.

- *Funzione di valore:* detta anche *value function*, è una funzione che indica quanto sia buono essere in uno stato oppure compiere una determinata azione in uno stato specifico. Esistono due tipi di funzioni di valore:

  - *Funzione di valore di stato:* assegna un valore a ciascuno stato; permette all'agente di individuare gli stati più convenienti.
  - *Funzione di valore di azione:* indica quanto sia ottimale eseguire una determiata azione in uno stato specifico; aiuta il processo decisionale dell'agente.

  Mentre il reward indica ciò che è buono nell'immediato, la funzione valore specifica cosa è buono nel lungo termine.

- *Modello dell'ambiente:* detto anche *Model of the environment*, non è presente in tutti i sistemi di Reinforcement Learning; è fondamentale per prevedere le reazioni dell'ambiente alle azioni dell'agente prima che quest'utlimo le compia effettivamente. I modelli sono usati per pianificare, ovvero decidere prima quali azioni saranno eseguite sulla base degli stati che potranno essere raggiunti.

Nella Figura 2.6 viene fornita una semplificazione grafica del workflow del Reinforcement Learning.

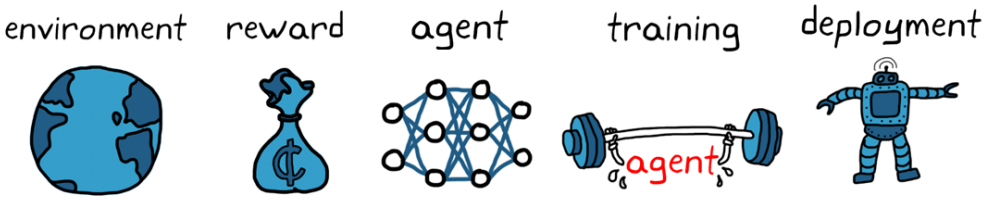

Figura 2.6: Workflow del Reinforcement Learning

2.5.3 Markov Decision Process

Nel Reinforcement Learning è necessario dare una descrizione formale all'ambiente circostante ed è per questo che si definisce l'ambiente come un MDP (processo di decisione Markoviano). Un MDP è definito da:

- Un insieme finito di stati S.
- Un insieme finito di azioni A.
- Una funzione di transizione di stato $P_a(s, s')$, la quale indica la probabilità che una determinata azione a effettuata dall'agente in uno stato s all'istante t porterà allo stato s' all'istante di tempo $t+1$.
- una funzione di reward $R_a(s, s')$, che assegna un valore numerico ad ogni possibile transizione dallo stato s a s'.

Come in parte già accennato, l'interazione stato-ambiente avviene nel seguente modo: all'istante t l'agente individua lo stato dell'ambiente s_t. Sulla base di questo stato, l'agente sceglie quale azione a_t eseguire. Una volta eseguita l'azione, prima di passare allo stato successivo s_{t+1}, l'ambiente risponde fornendo all'agente una ricompensa immediata (signal reward) r_{t+1}. Diremo, quindi, che un'ambiente soddisfa *la proprietà di Markov* se:

$$P(s_{t+1}, r_{t+1} | s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, \ldots, s_0, a_0, r_1) = P(s_{t+1}, r_{t+1} | s_t, a_t)$$

La proprietà Markoviana implica che quello che succede all'istante $t+1$, (s_{t+1}, r_{t+1}), dipende solo da quello che è successo all'istante precedente (s_t, a_t), in particolare dipende soltanto dallo stato e dall'azione e non da tutto ciò che è successo in precedenza $(s_t, a_t, r_t, \ldots, r_1, s_0, a_0)$. Se la proprietà di Markov è rispettata significa che l'agente esegue una strategia decisionale in funzione dello stato e dell'azione corrente.

2.5.4 Valore di ritorno

L'obiettivo di un agente di Reinforcement Learning è quello di scegliere una policy che massimizzi la somma dei reward signal, ossia massimizzi il valore di ritorno nel lungo termine. La somma dei reward viene chiamata *valore di ritorno* (G_t) ed è data da:

$$G_t = r_t + r_{t+1} + r_{t+2} + \ldots$$

Esiste anche il *valore di ritorno scontato*, che applica uno sconto alle ricompense future in quanto hanno meno valore rispetto a quelle immediate.

$$G_t = r_t + \gamma * r_{t+1} + \gamma^2 * r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

dove γ è detto *tasso di sconto* ed è un valore tra 0 e 1.

2.5.5 Funzione di ritorno

Come accennato in precedenza, esistono due tipi di funzione valore, una di stato e l'altra di azione. Andiamo a scriverle sotto forma di formule matematiche e vediamo entrambe a cosa corrispondono.

La funzione valore di stato è data dalla somma dei reward attesa (E_π) che l'agente riceverà dall'ambiente utilizzando una determinata policy π, partendo da uno stato s:

$$V_\pi(s) = E_\pi(G_t|s_t = s) = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| s_t = s \right]$$

D'altro canto, *la funzione valore di azione*, o *Q-function*, è data dalla somma dei reward prevista (E_π) che l'agente riceverà dall'ambiente utilizzando una determinata policy π, eseguendo un'azione a in uno stato s:

$$Q_\pi(s,a) = E_\pi(G_t|s_t = s, a_t = t) = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| s_t = s, a_t = a \right]$$

2.5.6 Equazioni di Bellman

Le equazioni di Bellman descrivono il problema di ottimizzazione, e quindi di massimizzazione della somma dei reward attesa. Una policy π è migliore di una policy π' se, per tutti gli stati $s \in S$, il ritorno atteso della policy π è maggiore di quello della policy π'. Da ciò si ottiene che:

$$V_\pi(s) > V_{\pi'}(s) \rightrightarrows V_*(S) = \max_\pi V_\pi(s)$$

Analogalmente per la funzione valore di azione.

In seguito a molteplici passaggi matematici si ottengono due equazioni di Bellman, per la funzione valore di stato e per la funzione valore di azione, conosciute come *equazioni di ottimalità di Bellman*.

Esprimiamole sotto forma di formule matematiche: per la funzione valore di stato si ha:

$$V_*(s) = \max_a \sum_{s'} p(s'|s,a)[r_t + \gamma V_*(s')]$$

dove $p(s'|s,a)$ indica la probabilità di arrivare nello stato s' partendo dallo stato s intraprendendo l'azione a.

Per la funzione valore di azione si ha:

$$Q_*(s,a) = \sum_{s'} p(s'|s,a)[r_t + \gamma \max_{a'} Q_*(s',a')]$$

Se sono note le probabilità di transizione da uno stato all'altro e le funzioni di ricompensa, le equazioni di Bellman vengono risolte tramite la *programmazione dinamica* in modo iterativo.

Ci sono algoritmi conosciuti come algoritmi *model-based* i quali presuppongono che queste probabilità siano conosciute o stimate al momento. Al contrario, gli algoritmi noti come *model-free*, tramite *rollout* del sistema tentano di simulare l'interazione tra l'agente con l'ambiente per cercare di capire cosa accade effettivamente.

2.5.7 Algoritmi model-based

Gli algoritmi model-based cercano di costruire un modello dell'ambiente in cui agisce l'agente. Tale modello contiene le probabilità di transizione da uno stato all'altro, i premi relativi alle azioni ed altre informazioni. Questi algoritmi sfruttano, come citato nella Sezione 2.5.2, il *model of the enveiroment*, calcolando le sequenze di azioni ottimali.

2.5.8 Algoritmi model-free

Gli algoritmi model-free non cercano di costrire un modello dell'ambiente bensì si basano sull'osservazione diretta delle interazioni dell'agente con l'ambiente per migliorare le policy. Questa tecnica prende nome di *rollout*. Questi algoritmi possono essere classificati come *on-policy* oppure come *off-policy*. In particolare:

- *on-policy*, gli algoritmi utilizzano la policy corrente per eseguire azioni, osservare i risultati e, successivamente, aggiornare la policy stessa in base all'esperienza.
- *off-policy*, gli algoritmi possono appartenere a due diverse categorie, ovvero:
  - *policy comportamentale*, utilizzata dall'agente per raccogliere dati.
  - *policy target*, che l'agente vuole ottimizzare.

  L'agente, a partire dai dati raccolti attraverso la policy comportamentale, tende ad aggiornare la policy target.

I più importanti algoritmi del mondo model-free sono i seguenti:

- *Metodi Monte Carlo (MC):* questi metodi utilizzano l'idea della *Generalized Policy Iteration (GPI)*. Questa rappresenta un processo iterativo il quale si divide in due sotto-processi:
  - *Policy evaluation:* detto anche valutazione della politica, si concentra sulla costruzione di un'approssimazione della funzione valore basandosi sulla politica corrente. Questo prevede l'utilizzo di tecniche come il rollout; si fa eseguire alla politica corrente un certo numero di episodi completi in un ambiente terminando una volta raggiunto uno stato finale. L'agente, interagendo con l'ambiente circostante, ottiene delle ricompense, in particolare, al termine di ogni episodio, si sommano tutte le reward ottenute nell'episodio stesso. Tale somma fornisce un'approssimazione che indica il valore dello stato iniziale quando viene utilizzata la politica corrente.
  - *Policy improvement:* detto anche miglioramento della politica, si focalizza su migliorare la politica dopo averla valutata. Coinvolge l'approccio *greedy*, il quale implica la scelta di azioni che sono ottimali e che portano a una maggiore accumulazione di ricompense. Spesso, però, questo approccio non rappresenta la scelta migliore, e, per questo, si sceglie di utilizzare una tecnica di esplorazione *epsilon-greedy*, che consiste in esplorazioni mirate per garantire che l'agente non rimanga bloccato in soluzioni sotto-ottimali.

I metodi Monte Carlo si basano sulla raccolta di dati da episodi completi, da sequenze di interazioni agente-ambiente, e, per questo motivo, si adattano bene a situazioni in cui è possibile eseguire simulazioni complete di un'intera sequenza di azioni. Per evitare stime meno accurate e grande varianza, che potrebbero essere determinate da ambienti caotici, è necessario eseguire un gran numero di episodi. Inoltre, questi ultimi non possono assicurare una stima perfettamente precisa dei valori ottimali con un numero fisso di episodi. La funzione di aggiornamento della funzione valore è data da:

$$V(s) \leftarrow V(s) + \alpha[G - V(s)]$$

dove:

- $V(s)$ rappresenta il valore stimato dello stato s prima dell'aggiornamento.
- α rappresenta il *learning rate*, che indica l'entità dell'aggiornamento, valori più elevati implicano aggiornamenti più aggressivi, mentre valori più bassi rendono gli aggiornamenti più conservativi.
- G è il ritorno totale ottenuto alla fine dell'episodio.

L'obiettivo principale di questa formula è aggiornare il valore stimato del singolo stato in modo che si avvicini gradualmente al valore ottimale, ciò permette al sistema di apprendere dai valori di ritorno ottenuti e migliorare le stime.

- *Metodi Temporal Difference (TD):* anche questi metodi sfruttano l'idea della GPI ma, a differenza del metodi Monte Carlo, essi aggiornarno i valori ad ogni step temporale senza attendere la conclusione dell'episodio. Nella fase di Policy evaluation non utilizzano la somma dei reward, bensì l'errore temporale, ovvero la differenza tra la nuova stima e la vecchia stima della funzione valore; tale errore viene utilizzato per aggiornare la funzione valore. Tutto ciò genera una situazione di *trade-off* in cui si riduce la varianza (si avranno stime più stabili e coerenti) ma aumenta il bias, perché l'agente sta facendo previsioni sui valori degli stati futuri basandosi su stime incomplete (non ottenute alla fine di un intero episodio). L'equazione di aggiornamento della funzione valore è data da:

  $$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$$

  dove:

  - $V(s)$ rappresenta il valore stimato dello stato s prima dell'aggiornamento;
  - α rappresenta il *learning rate*;
  - r è la ricompensa ottenuta dalla transizione dallo stato s allo stato successivo s';
  - γ è il fattore di sconto che tiene conto delle ricompense future;
  - $V(s')$ è il valore stimato del prossimo stato s'.

  La principale differenza tra i metodi Temporal Difference e i metodi Monte Carlo sta nell'aggiornamento: i metodi TD aggiornano i valori a ogni step temporale basandosi sul ritorno immediato della transizione di stato, mentre i metodi Monte Carlo aspettano la fine dell'episodio per l'aggiornamento basandosi sul ritorno totale dell'intero episodio. I due algoritmi di TD più diffusi sono *SARSA* e *Q-Learning*.

  - *SARSA (State-Action-Reward-State-Action):* è un algoritmo on-policy che cerca di migliorare la policy corrente in modo incrementale. È un algoritmo basato su tabella, cioè presenta una tabella di valori $Q(s, a)$ per ogni coppia stato-azione.

Implementazione dell'algoritmo SARSA

```
Inizializza Q(s,a) random
repeat
 Osserva stato iniziale s1
 Selezione dell'azione a1 utilizzando la policy derivata da Q,
 ad esempio con una strategia epsilon-greedy.
 for t=1 to T do
    Esegui azione a1
    Osserva il reward rt e il nuovo stato st+1
    Selezione dell'azione at+1 utilizzando la policy derivata da Q,
    ad esempio con una strategia epsilon-greedy.
    Aggiorna Q
 end for
until terminazione
```

L'aggiornamento dei valori $Q(s, a)$ è dato dalla formula:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

dove:

* $Q(s_t, a_t)$ rappresenta il valore stimato della coppia stato-azione;
* α rappresenta il *learning rate*;
* r è la ricompensa ottenuta dalla transizione dallo stato s allo stato successivo s' dallo stato con l'azione a;
* γ è il fattore di sconto che tiene conto delle ricompense future;
* $Q(s_{t+1}, a_{t+1})$ è il valore stimato della coppia stato-azione successiva.

– *Q-learning:* è un algoritmo off-policy, che cerca di apprendere il valore ottimo indipendentemente dalla policy. Anche questo è un algoritmo basato su tabella.

Implementazione dell' algoritmo Q-learning

```
Inizializza Q(s,a) random
repeat
 Osserva stato iniziale st
 for t=1 to T do
    Selezione dell'azione at utilizzando la policy derivata da Q,
    ad esempio con una strategia epsilon-greedy.
    Esegui azione at
    Osserva il reward rt e il nuovo stato st+1
    Aggiorna Q
 end for
until terminazione
```

L'aggiornamento della funzione valore di azione $Q(s, a)$ è dato dalla formula:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

dove:

* $\max_a Q(s_{t+1}, a)$ rappresenta il valore stimato della migliore azione a nello stato successivo s_{t+1}

Gli algoritmi sopracitati sono di fondamentale importanza per capire le basi del Reinforcement Learning; esistono anche algoritmi più avanzati che vengono spesso utilizzati in problemi più complessi. Si tratta di algoritmi che sfruttano la fusione del Deep Learning e del Reinforcement Learning; per questo motivo sono chiamati algoritmi di *Deep Reinforcement Learning*.

2.6 Deep Reinforcement Learning

Il Deep Reinforcement Learning (DRL) è una sottodisciplina che unisce tecniche di addestramento di agenti intelligenti con l'uso di reti neurali profonde; si fa riferimento, quindi, all'utilizzo di DNN per effettuare approssimazioni della funzione valore o della policy in algoritmi di RL. In sintesi, il Deep Learning accorre in aiuto del Reinforcement Learning quando si affrontano situazioni estremamente complesse. Questo perché le reti neurali profonde vengono impiegate per elaborare stati d'ambiente complessi, consentendo all'agente di comprenderli in modo più profondo e di prendere decisioni intelligenti. Analizziamo gli algoritmi relativi a questa disciplina.

2.6.1 Algoritmi di Deep Reinforcement Learning

Gli algoritmi di DRL sono molto potenti, utilizzati per addestrare agenti in situazioni ambientali complesse. Hanno un'ottima capacità di elaborare grandi quantità di dati e di apprendere rappresentazioni complesse di questi.

Due tra questi algoritmi, che vedremo poi utilizzati nel capitolo successivo, sono:

- *Deep Q-Network (DQN):* rappresenta un'evoluzione del metodo Q-learning; si concentra sull'approssimazione della funzione valore di azione Q dove la tabella stato-azione viene sostituita da reti neurali. DQN utilizza due reti neurali, ovvero:
  - *Rete target:* viene utilizzata per fornire una stima obiettivo (target) per l'apprendimento della rete online.
  - *Rete online:* viene utilizzata per prendere decisioni durante l'interazione dell'agente con l'ambiente circostante; riceve in input lo stato e restituisce i valori della funzione $Q(s, a)$. Nella rete online l'apprendimento non consiste nell'aggiornare la tabella, ma nell'aggiustamento dei pesi dei neuroni che compongono tale rete in modo da migliorare le stime dei valori Q. Questo processo è guidato dalla funzione di perdita, che, come già citato nella Sezione 2.2.1, misura l'errore tra le previsioni della rete e i target.

  La funzione di perdita (loss function) utilizzata per l'addestramento è definita come:

  $$L_t = (E[r + \gamma \max_{a'} Q(s_{t+1}, a_t)] - Q(s_t, a_t))^2$$

  dove $E[r + \gamma \max_{a'} Q(s_{t+1}, a_t)]$ è il valore di ritorno ottimo atteso, mentre $Q(s_t, a_t)$ è il valore stimato della rete.

  Gli errori derivati dalla funzione di perdita vengono retropropagati attraverso la rete tramite il processo di backpropagation, il quale sfrutta il gradiente, che ci indica in quale direzione la funzione di perdita cresce di più. Muovendoci nella direzione opposta al gradiente, lavoriamo per ridurre al massimo l'errore, aggiornando i pesi della rete e ottimizzando quest'ultima. DQN utilizza una tecnica chiamata *Experience Replay*; in essa, l'esperienza dell'agente viene presa ad ogni time step t e salvata in un dataset D chiamato *replay memory*. L'addestramento viene eseguito utilizzando una tecnica di *mini-batch*, dove si estrae casualmente un gruppo di campioni di esperienze dalle replay memory. Questo approccio consente di riutilizzare le esperienze passate in più cicli di addestramento, migliorando l'efficienza, consentendo alle esperienze di influenzare più di un aggiornamento della rete e diminuendo la varianza tra gli aggiornamenti.

- *Deep Deterministic Policy Gradient (DDPG):* si concentra sulla selezione di azioni in uno spazio continuo di esse. L'agente sceglie un'azione specifica invece di campionarla probabilisticamente. Il DDPG utilizza due funzioni neurali, ovvero:

  – *funzione Q:* stima il valore di ritorno atteso in relazione ad una coppia stato-azione.
  – *funzione valore:* stima il valore di ritorno atteso dello stato in funzione di una certa policy.

  Questo algoritmo sfrutta un *replay buffer* simile a quello utilizzato da DQN, in cui vengono memorizzate esperienze passate le quali vengono campionate casualmente durante l'addestramento.

  Anche in questo caso vengono utilizzate due reti neurali, ovvero:

  – *attore:* utilizzato per generare azioni deterministiche.
  – *critico:* utilizzato per stimare i valori delle coppie stato-azione.

  DDPG utilizza la tecnica di ottimizzazione del gradiente per aggiornare le reti attore e la rete critico. Nel primo caso l'aggiornamento si traduce in un miglioramento della policy dell'agente, ovvero della strategia con cui seleziona le azioni. Nel secondo caso l'aggiornamento mira a ridurre l'errore tra le stime Q e i target calcolati.

La funzione di perdita del critico (*critic loss function*) utilizzata per l'addestramento è definita come:

$$L_t = \frac{1}{2} E[(Q(s,a) - [r + \gamma Q(s', a(s'))])]^2$$

dove

- $Q(s,a)$ è la stima della funzione Q;
- r è il reward;
- s' è il nuovo stato;
- $a(s')$ l'azione scelta dall'attore per lo stato s'.

La funzione esprime l'errore quadratico medio tra la stima e il target e mira a minimizzare questa differenza.

2.7 Applicazioni del Reinforcement Learning

Il Reinforcement Lerning, con l'avanzare del tempo sta trovando sempre più spazio in una vasta gamma di settori. Nel seguito, ne citiamo alcuni:

- *Gaming:* un esempio molto famoso in questo settore è quello di AlphaGo, un software realizzato da DeepMind addestrato, appunto, con l'apprendimento per rinforzo. Ulteriore esempio in questo campo è rappresentato dal gioco di PacMan, utilizzato come ambiente di apprendimento.
- *Finanza:* in questo settore il Reinforcement Learning ha trovato applicazione poichè, grazie ad esso, si riesce ad effettuare l'ottimizzazione dei portafogli e ad implementare algoritmi di strategie di trading.

- *Robotica:* in cui il Reinforcement Learning viene usato per addestrare robot. Un esempio è quello di IRoomba che sfrutta l'apprendimento per rinforzo per mappare e navigare all'interno di una casa.
- *Guida autonoma:* nel settore self-driving car, il Reinforcement Learning viene impiegato per addestrare gli *Advanced Driver Assistance Systems* (ADAS). Ogni fase di guida può essere modellata come una policy:
  - cambio di corsia;
  - parcheggio automatico;
  - cruise control automatico;
  - sorpasso;
  - mantenimento della carreggiata.

  L'obiettivo è quello di rendere i veicoli autonomi nel prendere delle decisioni corrette, migliorando, così, la sicurezza stradale. Un esempio è l'autopilot di Tesla i cui veicoli sono dotati di sensori avanzati con cui raccolgono dati in tempo reale: attraverso questi dati vengono, poi, addestrati i modelli per l'assistenza alla guida.
- *Assistenza sanitaria:* in cui il Reinforcement Learning è di frequente utilizzo per la capacità di individuare alcuni tipi di patologie croniche. Un esempio è l'applicazione Streams, che sfrutta il Reinforcement Learning per individuare segni di deterioramento nei pazienti per garantire l'intervento tempestivo e preventivo da parte dei medici.

Dato il vasto utilizzo, ciò ha portato a far emergere i limiti, ma anche i benefici, che il Reinforcement Learning porta con sè.

2.8 Benefici e svantaggi del Reinforcement Learning

In conclusione a questo capitolo, analizziamo i limiti e i vantaggi che il Reinforcement Learning presenta. Tra i principali limiti abbiamo i seguenti:

- é complesso addestrare un agente;
- addestrare un agente è un processo molto lungo;
- per riuscire a prendere decisioni ottimali, il Reinforcement Learning richiede un'enorme quantità di dati i quali risultano essere molto costosi;
- c'è il rischio che un agente apprenda comportamenti errati.

Tra i principali vantaggi abbiamo i seguenti:

- un'agente è capace di apprendere in maniera automatica e riesce ad adattarsi a cambiamenti continui;
- un'ottimale gestione delle risorse per migliorare l'efficienza di allocazione in ambiti come la logistica;
- può essere utilizzato per l'automazione di compiti complessi.

CAPITOLO 3

Analisi dei simulatori di guida autonoma

All'interno di questo capitolo, esamineremo, in maniera dettagliata, il contesto della guida autonoma, con un focus specifico sull'analisi dei simulatori di guida autonoma. In un periodo caratterizzato da un'evoluzione del settore automobilistico, i simulatori di guida rivestono un ruolo fondamentale nel processo di sviluppo, apprendimento e perfezionamento delle tecnologie legate all'Intelligenza Artificiale impiegate nella guida autonoma.

Nel corso di questo capitolo esamineremo diverse categorie di simulatori, ognuno dei quali si dedica ad un aspetto cruciale della fase di guida autonoma.

3.1 Introduzione generale dei simulatori

La fase di simulazione svolge un ruolo di fondamentale importanza nello sviluppo e nell'addestramento di sistemi autonomi avanzati. In particolare, l'ambiente di sviluppo *Matlab* offre una piattaforma per creare e testare simulatori di guida autonoma molto sofisticati. Questi simulatori permettono di migliorare ed ottimizzare le capacità dei veicoli autonomi in una serie di scenari complessi, rendendo possibile il progresso in microsettori cruciali della guida autonoma, come il parcheggio automatico, il mantenimento della corsia e il controllo della velocità adattivo.

Esamineremo quattro simulatori di guida autonoma sviluppati in Matlab, ciascuno dei quali si concentra su specifiche funzionalità e fasi chiave della guida autonoma. Questi simulatori sono i seguenti:

- *Automatic Parking Valet with Unreal Engine Simulation:* esso si concentra sulla capacità di parcheggio automatico dei veicoli autonomi, utilizzando la potente piattaforma di sviluppo *Unreal Engine* per creare scenari realistici.
- *Train RL Agent for Lane Keeping Assist with Constraint Enforcement:* esso è dedicato all'addesstramento di agenti intelligenti basati sul Reinfocercement Learning per il mantenimento di corsia, con una particolare attenzione alle regole di sicurezza.
- *Train DDPG Agent for Adaptive Cruise Control:* esso si concentra sul cruise control, utilizzando l'algoritmo DDPG per addestrare gli agenti a gestire la velocità del veicolo in relazione alle condizioni del traffico.
- *Object Detection Using SSD Deep Learning:* esso effettua la rilevazione di oggetti utilizzando il modello Single Shot Detector.

Questi simulatori consentono agli ingegneri di migliorare le tecnologie relative all'IA, agendo in ambienti simulati, diminuendo i rischi e i costi relativi ad esperienze reali in strada.

Nelle fasi successive del capitolo analizzeremo in dettaglio i simulatori elencati in precedenza.

3.2 Automatic Parking Valet with Unreal Engine Simulation

Questo simulatore illustra il funzionamento di un sistema ibrido di parcheggio automatico che combina il controllo predittivo del modello adattivo (MPC) con il Reinforcement Learning (RL). Questo sistema è in grado di eseguire una manovra di parcheggio in ambienti virtuali simulati utilizzando il motore grafico Unreal Engine.

L'obiettivo principale del sistema di controllo è parcheggiare un veicolo partendo da una posizione iniziale e posizionarlo in uno spazio di parcheggio vuoto. Il sistema gestisce anche spazi di parcheggio stretti, occupati e deve evitare ostacoli lungo il percorso.

Il funzionamento del sistema prevede due fasi principali:

- *Controllore MPC Adattivo*: il sistema utilizza il modello predittivo adattivo (MPC) per guidare il veicolo a una velocità costante mentre cerca un posto vuoto per il parcheggio. Il controllore MPC ha una conoscenza preliminare dell'ambiente, inclusa la posizione dei posti vuoti e dei veicoli parcheggiati.
- *Agente di RL:* individuato un posto libero, l'agente di Reinforcement Learning assume il controllo e implementa una manovra di parcheggio preaddestrata.

Vediamo le varie fasi di implementazione del seguente simulatore.

3.2.1 Implementazione del Simulatore 1

Parcheggio

L'ambiente del parcheggio utilizzato in questo esempio è una sezione del "Large Parking Lot" fornito dal Automated Driving Toolbox.

Il parcheggio è rappresentato dall'oggetto "*ParkingLotEnvironment*", che contiene informazioni sul veicolo ego, sui posti del parcheggio vuoti e sugli ostacoli fermi (auto parcheggiate e limiti). Ogni posto auto ha un numero di indice unico e un indicatore luminoso che può essere verde, ad indicare che il posto è libero, o rosso, per indicare che il posto è occupato. Le auto parcheggiate sono rappresentate in nero, mentre gli altri limiti sono evidenziati in azzurro.

Definiamo un intervallo di campionamento e un tempo di simulazione, entrambi espressi secondi.

```
Ts = 0.1;
Tf = 50;
```

Creiamo un percorso di riferimento per il veicolo ego utilizzando la funzione ausiliaria "*createReferenceTrajectory*".

```
xRef = createReferenceTrajectory(Ts,Tf);
```

Creiamo un oggetto "*ParkingLotEnvironment*" con un posto auto libero all'indice 32 e utilizziamo il percorso di riferimento specificato xRef.

```
freeSpotIndex = 32;
map = ParkingLotEnvironment(freeSpotIndex,"Route",xRef);
```

Definiamo una posizione iniziale (X_0, Y_0, θ_0) per il veicolo ego. Le unità di misura per X_0 e Y_0 sono metri, mentre θ_0 è espresso in radianti.

```
egoInitialPose = [40 -55 pi/2];
```

Calcoliamo la posizione di destinazione per il veicolo utilizzando la funzione "*createTargetPose*". La posizione di destinazione corrisponde alla posizione indicata da "*freeSpotIdx*" che indica l'indice del posto auto libero.

```
egoTargetPose = createTargetPose(map,freeSpotIndex);
```

Sensori: camere e lidar

Il campo visivo di una telecamera montata sul veicolo ego è rappresentato dall'area ombreggiata in verde. Mentre il veicolo ego avanza, il modulo telecamera rileva i parcheggi all'interno del campo visivo e determina se un posto è libero o occupato. Il sensore Lidar, mostrato in Figura 3.1, è modellato utilizzando segmenti di linee radiali che partono dal centro geometrico del veicolo. Lungo questi segmenti di linea vengono misurate le distanze dagli ostacoli. L'agente utilizza queste letture per determinare la vicinanza del veicolo ego agli altri veicoli nell'ambiente.

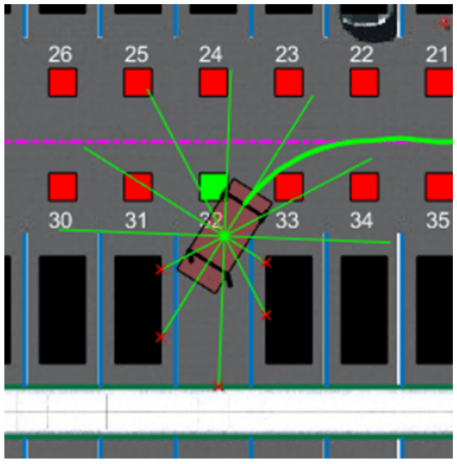

Figura 3.1: Rappresentazione del lidar del Simulatore 1

Modello

Definiamo il modello per il parcheggio automatico e creiamo l'oggetto controller MPC adattivo per il tracciamento della traiettoria di riferimento utilizzando lo script "*createMPCForParking*".

```
autoParkingValetParams3D
mdl = "rlAutoParkingValet3D";
open_system(mdl)
createMPCForParking3D;
```

Ambiente

L'ambiente per l'addestramento è la regione ombreggiata in rosso come rappresentato nella Figura 3.2.

Creiamo le specifiche di osservazione e azione per l'ambiente.

```
nObs = 16;
nAct = 1;
obsInfo = rlNumericSpec([nObs 1]);
obsInfo.Name = "observations";
actInfo = rlNumericSpec([nAct 1],LowerLimit=-1,UpperLimit=1);
actInfo.Name = "actions";
```

Creiamo l'interfaccia dell'ambiente Simulink.

```
blk = mdl + "/Controller/RL Controller/RL Agent";
env = rlSimulinkEnv(mdl,blk,obsInfo,actInfo);
```

La funzione "*autoParkingValetResetFcn*" ripristina la posizione iniziale del veicolo ego su valori casuali all'inizio di ogni episodio.

```
env.ResetFcn = @autoParkingValetResetFcn3D;
```

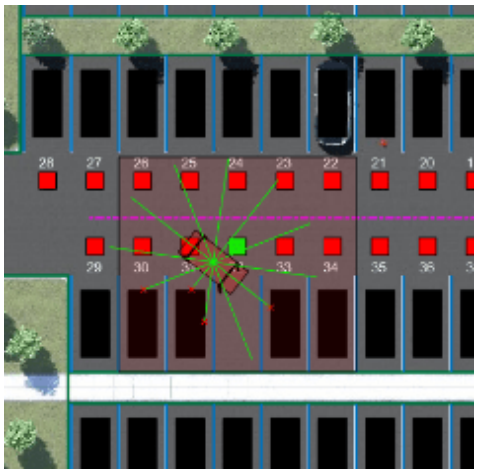

Figura 3.2: Rappresentazione dell'ambiente del simulatore 1

Agente

L'agente utilizzato in questo esempio è un agente con gradiente di policy deterministico profondo noto come TD3 (Twin Delayed Deep Deterministic Policy Gradient). Gli agenti TD3 si basano su due componenti fondamentali: l'attore e il critico.

Inizializziamo le reti attore e critico in modo casuale, e definiamo i parametri delle reti critico.

```
rng(0)
% Main path
mainPath = [
    featureInputLayer(nObs,Name="StateInLyr")
    fullyConnectedLayer(128)
    concatenationLayer(1,2,Name="concat")
    reluLayer
    fullyConnectedLayer(128)
    reluLayer
    fullyConnectedLayer(1,Name="CriticOutLyr")
];

% Action path
actionPath = [
    featureInputLayer(nAct,Name="ActionInLyr")
    fullyConnectedLayer(128,Name="fc2")
];

criticNet = layerGraph(mainPath);
criticNet = addLayers(criticNet, actionPath);
criticNet = connectLayers(criticNet,"fc2","concat/in2");
```

Effettuiamo la conversione in un oggetto '*dlnetwork*'.

```
criticdlnet = dlnetwork(criticNet);
summary(criticdlnet)
```

Creiamo le reti critico.

```
critic1 = rlQValueFunction(criticNet,obsInfo,actInfo,...
ObservationInputNames="StateInLyr",ActionInputNames="ActionInLyr");
critic2 = rlQValueFunction(criticNet,obsInfo,actInfo,...
ObservationInputNames="StateInLyr",ActionInputNames="ActionInLyr");
```

Definiamo i parametri della rete neurale attore ed effettuiamo la conversione in un oggetto "*dlnetwork*".

```
anet = [
    featureInputLayer(nObs)
    fullyConnectedLayer(128)
    reluLayer
    fullyConnectedLayer(128)
    reluLayer
    fullyConnectedLayer(nAct)
    tanhLayer
];
actorNet = layerGraph(anet);
actordlnet = dlnetwork(actorNet);
summary(actordlnet)
```

Creiamo l'attore.

```
actor = rlContinuousDeterministicActor(actordlnet,obsInfo,actInfo);
```

Definiamo i parametri dell'agente TD3.

```
agentOpts = rlTD3AgentOptions(SampleTime=Ts, ...
DiscountFactor=0.99, ...
ExperienceBufferLength=1e6, ...
MiniBatchSize=128);

agentOpts.ExplorationModel.StandardDeviation = 0.1;
agentOpts.ExplorationModel.StandardDeviationDecayRate = 1e-4;
agentOpts.ExplorationModel.StandardDeviationMin = 0.01;
```

Definiamo le opzioni per l'addestramento dell'attore e del critico.

```
agentOpts.ActorOptimizerOptions.LearnRate = 1e-3;
agentOpts.ActorOptimizerOptions.GradientThreshold = 1;
agentOpts.ActorOptimizerOptions.L2RegularizationFactor = 1e-3;

agentOpts.CriticOptimizerOptions(1).LearnRate = 2e-3;
agentOpts.CriticOptimizerOptions(2).LearnRate = 2e-3;
agentOpts.CriticOptimizerOptions(1).GradientThreshold = 1;
agentOpts.CriticOptimizerOptions(2).GradientThreshold = 1;
```

Creiamo l'agente utilizzando l'attore, il critico e l'oggetto agente "*agentOpts*".

```
agent = rlTD3Agent(actor,[critic1 critic2], agentOpts);
```

In conclusione, definiamo le caratteristiche e la funzione per il training.

```
trainOpts = rlTrainingOptions(...
MaxEpisodes=10000,...
MaxStepsPerEpisode=200,...
ScoreAveragingWindowLength=200,...
Plots="training-progress",...
StopTrainingCriteria="AverageReward",...
StopTrainingValue=120);
doTraining = false;
if doTraining
    trainingResult = train(agent,env,trainOpts);
else
    load("rlAutoParkingValetAgent.mat","agent");
end
```

Prima di procedere con l'analisi dei risultati degli esperimenti condotti utilizzando questo simulatore, è importante identificare alcuni elementi chiave che saranno discussi in tutti gli esperimenti, ovvero:

- *Episodi:* si riferiscono ai singoli tentativi, ossia le interazioni dell'agente con l'ambiente circostante. Ogni episodio parte da una condizione iniziale e termina al raggiungimento di una condizione finale o di un obiettivo. Gli episodi possono variare in lunghezza e complessità.
- *Average reward* o *ricompensa media*: indica la media delle ricompense ottenute dall'agente durante un determinato numero di episodi. È un valore che indica il rendimento medio dell'agente nell'ambiente, infatti, un aumento di quest'ultimo indica un miglioramento delle prestazioni dell'agente.
- *Episode reward* o *ricompensa periodica*: si riferisce alla ricompensa ottenuta dall'agente nel singolo episodio. Può avere brusche variazioni da episodio a episodio.
- *Episode QO:* si riferisce al valore stimato dell'azione dell'agente per una coppia episodio-azione, ossia rappresenta quanto l'agente ritiene che sia vantaggioso intraprendere una determinata azione in un dato episodio. Questo parametro guida l'agente all'apprendimento di una politica ottimale.

3.2.2 Analisi degli esperimenti relativi al Simulatore 1

Tutti gli esperimenti effettuati hanno in comune l'obiettivo di ottenere una ricompensa media di valore pari a 120 entro il numero massimo di episodi prefissato a 10000.

Esperimento 1

Senza apportare alcuna modifica al codice del simulatore, abbiamo condotto un esperimento, che ha prodotto i seguenti risultati.

La situazione dell'esperimento in seguito ad un campionamento effettuato dopo un numero di 974 episodi è rappresentata in Figura 3.3.

In seguito al campionamento, dal grafico e dai dati possiamo notare che l'obiettivo in termine di reward average, non è stato ancora raggiunto.

Le curve di "episode reward" e quella di episode $Q0$ sono in una fase di stabilità, nè crescente e nè decrescente. L'oscillazione della curva "episode reward" tra i +150 e i -50 indica che l'agente sta sperimentando delle variazioni nelle sue prestazioni durante gli episodi. Questa variabilità potrebbe indicare un agente che sta cercando nuove strategie e sta apprendendo dai successi e dai fallimenti.

Nonostante l'ampia oscillazione delle prestazioni nei singoli episodi, notiamo che la curva di "average reward" è caratterizzata da una tendenza generale alla stabilità, la quale si alterna a una tendenza crescente. Questo suggerisce che l'agente sta ottenendo un miglioramento medio delle prestazioni, nonostante le fluttuazioni degli episodi. Questa tendenza positiva è un segnale che potrebbe indicare che l'addestramento sta procedendo nella giusta direzione.

Inolte la curva di "Episode $Q0$" oscilla tra valori pari circa a 50 e -40; ciò indica incertezza nell'agente riguardo alle azioni da intraprendere.

Dopo aver esaminato i risultati ottenuti durante il campionamento, procediamo ora all'analisi dei dati alla conclusione dell'esperimento.

Il risultato dell'esperimento 1 è mostrato in Figura 3.4. L'obiettivo di raggiungere un valore di "average reward" di 120 è stato conseguito dopo 1278 episodi ed una durata temporale pari a circa 6 ore.

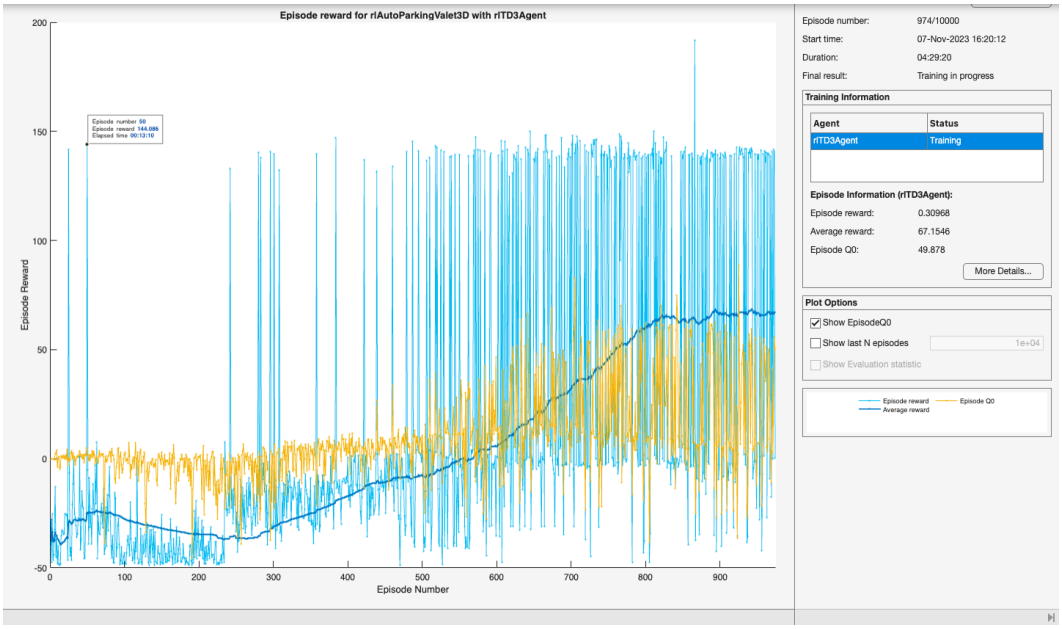

Figura 3.3: Situazione dell'Esperimento 1 del Simulatore 1 dopo 974 episodi

Di seguito, elenchiamo le diverse fasi rilevate nell'andamento della curva *average reward* in maniera approssimativa:

- *Fase iniziale (Episodi 0-75):* nella fase iniziale, la curva presenta picchi negativi che toccano anche valori pari a -35, con delle fluttuazioni sia positive che negative.
- *Stabilizzazione preliminare (Episodi 75-275):* in questa fase, la curva sembra essersi stabilizzata su valori pari a -25 con una tendenza decrescente.
- *Cambio di tendenza (Episodi 275-875):* intorno all'Episodio 275, si verifica un notevole cambiamento nella tendenza. La curva inizia a mostrare un incremento costante, passando da una tendenza negativa a una positiva che prosegue fino all'Episodio 875 raggiungendo un valore pari a circa 65.
- *Stabilizzazione intermedia (Episodi 875-1110):* in questo intervallo, la curva rimane relativamente stabile.
- *Cambio di tendenza finale (Episodi 1110-1278):* durante questa fase si verifica un secondo notevole cambiamento di tendenza il quale determina un aumento della curva. Questo culmina nel raggiungimento dell'obiettivo di reward average di 120 all'Episodio 1278.

Oltre questo, è interessante notare l'andamento della curva *episode reward*. Inizialmente, la curva mostra valori costantemente bassi intorno a -50, indicando prestazioni inizialmente non ottimali da parte dell'agente. C'è una svolta notevole che si verifica a partire intorno all'Episodio 500, quando iniziano ad apparire picchi costanti a valori intorno a 145 che rappresentano il raggiungimento di obiettivi positivi da parte dell'agente, indicando un miglioramento significativo nelle sue prestazioni. Nonostante questi miglioramenti, è interessante notare che continuano ad esserci picchi negativi, con valori di -50, anche se con una frequenza minore rispetto ai picchi positivi e con una tendenza a diminuire in maniera progressiva con l'avanzare dell'esperimento. In particolare, in alcuni tratti, i picchi negativi assumono un valore di 0 anziché -50, indicando un cambiamento significativo nell'andamento dell'episode reward in quella fase. Questi picchi negativi possono indicare situazioni in cui l'agente non riesce a raggiungere gli obiettivi o compie azioni subottimali.

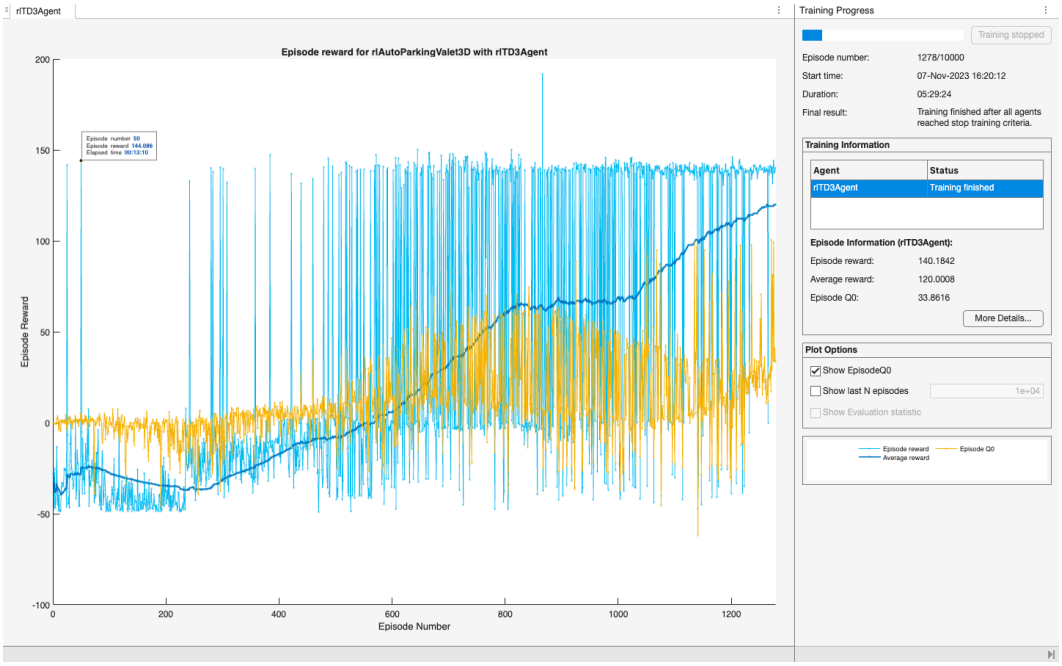

Figura 3.4: Risultato finale dell'Esperimento 1 del Simulatore 1

Per quanto riguarda, invece, l'andamento della curva *episode* Q0 mostra un cambiamento significativo nel corso dell'esperimento. Inizialmente, la curva appare stretta, contenuta tra una gamma limitata di valori che oscilla attorno allo 0. Tuttavia, con l'avanzare degli episodi, la curva si è notevolmente allargata, con una frequenza di picchi di valori pari a 50, notevolmente maggiore rispetto ai valori negativi. La curva, pertanto, nel grafico risulta essere molto densa in corrispenza del valore 25 circa.

Dopo aver condotto un'analisi sui risultati generati dall'Esperimento 1 possiamo ora affrontare i risultati dell'Esperimento 2.

Esperimento 2

Per condurre l'Esperimento 2 sono state apportare delle modifiche al codice standard del Simulatore 1. Nel seguito esamineremo tali modifiche.

Le seguenti aumentano la complessità della rete neurale utilizzata per l'agente, aumentando il numero di neuroni, passando da 128 neuroni a 256 neuroni in entrambi gli strati fully connected.

```
mainPath = [
    featureInputLayer (nObs, Name="StateInLyr")
    fullyConnectedLayer(256)
    concatenationLayer(1, 2, Name="concat")
    reluLayer
    fullyConnectedLayer(256)
    reluLayer
    fullyConnectedLayer(1, Name="CriticOutLyr")
    ];
```

Le seguenti modifiche sono state effettuate sui tassi di apprendimento dell'attore e del critico dell'agente. Il tasso di apprendimento è un parametro che regola la velocità con cui i pesi del modello vengono aggiornati durante l'addestramento.

```
agentOpts.ActorOptimizerOptions.LearnRate = 5e-4;
agentOpts.CriticOptimizerOptions(1).LearnRate = 1e-3;
agentOpts.CriticOptimizerOptions(2).LearnRate = 1e-3;
```

Le modifiche riportate di sotto riguardano i parametri relativi all'esplorazione dell'agente; esse comportano una maggiore esplorazione da parte dell'agente durante l'addestramento, con azioni più variabili.

```
agentOpts.ExplorationModel.StandardDeviation=0.2;
agentOpts.ExplorationModel.StandardDeviationDecayRate=1e-5;
agentOpts.ExplorationModel.StandardDeviationMin=0.05;
```

I cambiamenti seguenti sono relativi al fattore di regolarizzazione L2 applicato all'ottimizzatore dell'attore.

```
agentOpts.ActorOptimizerOptions.L2RegularizationFactor=1e-4;
```

Ulteriori modifiche sono state apportate ai limiti di apprendimento, i quali rappresentano una soglia che può essere utilizzata per limitare il valore massimo del gradiente durante l'aggiornamento dei pesi del modello, ed al minibatch, il quale è un sottoinsieme dei dati di addestramento che vengono selezionati casualmente per calcolare l'aggiornamento dei pesi.

```
agentOpts.ActorOptimizerOptions.GradientThreshold=0.5;
agentOpts.CriticOptimizerOptions(1).GradientThreshold=0.5;
agentOpts.CriticOptimizerOptions(2).GradientThreshold = 0.5;
agentOpts.MiniBatchSize=64;
```

Una volta introdotte le modifiche, analizziamo i risultati ottenuti dall'Esperimento 2, rappresentati in Figura 3.5. Con l'Esperimento 2, l'obiettivo di raggiungere un valore di "average reward" di 120 è stato conseguito dopo 1383 episodi ed una durata temporale pari a circa 5 ore.

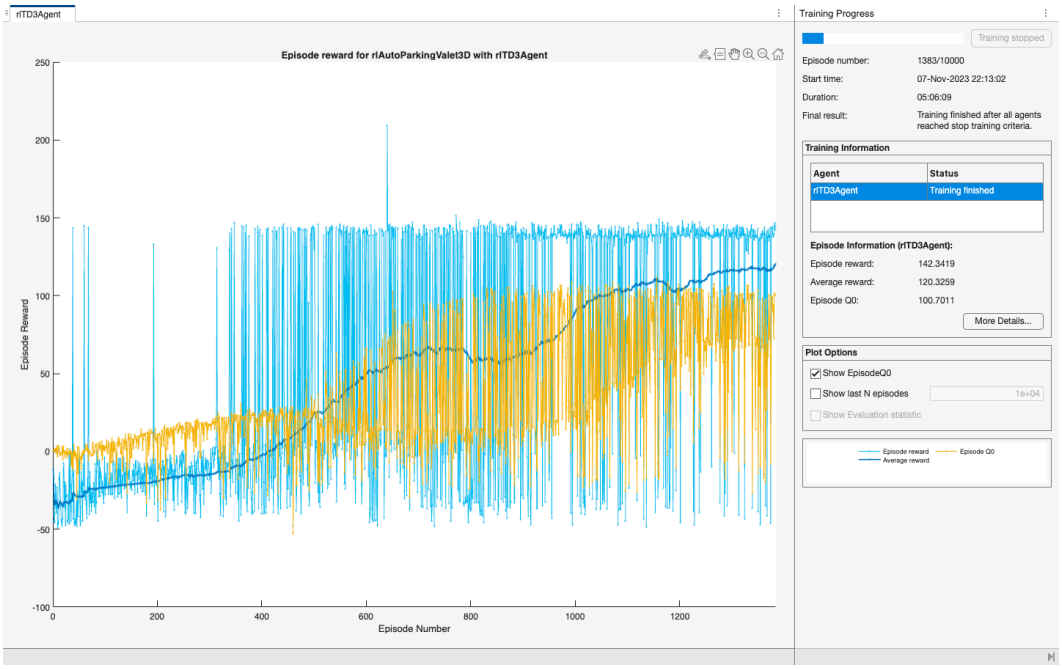

Figura 3.5: Risultato finale dell'Esperimento 2 del Simulatore 1

Di seguito, elenchiamo le diverse fasi rilevate nell'andamento della curva *average reward* in maniera approssimativa:

- *Apprendimento iniziale (Episodi 0-35):* nella fase iniziale, l'agente inizia l'addestramento senza una conoscenza approfondita dell'ambiente e, per questo motivo, la curva presenta valori negativi pari a -25.

- *Crescita graduale (Episodi 40-725):* in questa fase, la curva presenta una tendenza crescente molte graduale, con delle piccole flessioni decrescenti, ma poco significative.
- *Stabilità intermedia (Episodi 725-775):* intorno all'Episodio 725, la curva inizia a stabilizzarsi attorno ad un valore pari a 65.
- *Caduta temporanea (Episodi 775-860):* in questo intervallo, la curva subisce una caduta temporanea fino a toccare valori di circa 55. La caduta è provocata probabilmente da fluttuazioni casuali. L'agente deve riadattarsi per tornare a migliorare le sue prestazioni.
- *Crescita importante (Episodi 860-1175):* in questi episodi, la curva torna ad aumentare in maniera graduale fino a picchi che toccano i +100, sinonimo che l'agente ha appreso una politica molto efficace. È importante notare anche come, tra gli Episodi 950 e 1000, si verifica una brusca accelerazione nella crescita della curva.
- *Fase finale (Episodi 1175-1383):* in quest'ultima fase la curva continua a crescere con alcune fluttuazioni; nel complesso mostra una tendenza di crescita costante fino al termine dell'esperimento.

Dopo la curva di "average reward" analizziamo l'andamento della curva *episode reward*. Nelle fasi iniziali la curva mostra valori costantemente bassi, oscillando tra -50 e 0, riflettettendo le difficoltà dell'agente ad ottenere ricompense positive. Verso l'Episodio 300, si presenta un cambio di tendenza; infatti si determinano picchi ripetuti con valori intorno a 145 indicando che l'agente sta costantemente migliorando le sue prestazioni. . Anche in questo caso è possibile notare che ci sono ancora picchi negativi, con valori di -50, sebbene con una frequenza minore rispetto ai picchi positivi che, verso la fine dell'esperimento, tendono a sparire.

L'andamento della curva episode $Q0$ cambia durante l'esperimento. Inizialmente, la curva appare stretta, con picchi positivi che raggiungono valori pari a 25 e picchi negativi che sfiorano i -25. Successivamente, la curva si allarga; ciò indica valori positivi e negativi maggiori. La frequenza con cui si verificano picchi positivi è maggiore rispetto a quella con cui si verificano picchi negativi. I valori positivi sono molto vicini a 100 mentre quelli negativi a -25; ciò determina una densità maggiore in corrispondenza del valore 50.

Le modifiche effettuate potrebbero aver provocato alcune differenze tra gli elementi chiave dei due esperimenti; analizziamole:

- Nella curva *episode reward* del secondo esperimento si sono raggiunti valori fissi pari a 145 in 300 episodi, mentre, nell'Esperimento 1, lo stesso risultato si raggiunge in 600 episodi; questo potrebbe essere dovuto alle modifiche effetuate ai parametri dell'agente, in particolare alla diminuzione del learning gate dell'attore, che potrebbe aver consentito all'agente di apprendere in maniera più lenta ma più stabile.
- Nella curva *average reward* la riduzione del tasso di apprendimento dell'attore può aver causato un fase di dimuzione delle prestazioni visualizzata nell'Esperimento 2 tra gli Episodi 775 e 860. Successivamente, la riduzione del rumore di esplorazione potrebbe aver aiutato a ripristinare la tendenza positiva.
- Gli aumenti di valore dell'*episode* $Q0$ sono dovuti alla riduzione del rumore di esplorazione.
- Le modifiche effettuate al minibatch e ai gradienti potrebbero aver reso l'addestramento più lento in termini di numero di episodi richiesti, ma più stabile.

Ulteriori modifiche viste in precedenza hanno introdotto strati aggiuntivi nella rete neurale, portando a un aumento significativo nel numero di parametri apprendibili (learnables) da 35.4k nell'Esperimento 1 a 103.4k nell'Esperimento 2.

Concludo l'analisi dei risultati dicendo che le modifiche apportate per l'Esperimento 2 sembrano aver reso il training più efficiente; infatti, l'agente è riuscito ad apprendere con un tempo seppure minore seppure il numero di episodi risulta essere maggiore.

3.3 Train RL Agent for Lane Keeping Assist with Constraint Enforcement

Questo simulatore si focalizza sull'addestramento di un agente di Reinfrocement Learnig per mantenere un veicolo all'interno di una corsia, utilizzando il Lane Keeping Assist e applicando vincoli. Attraverso le regolazioni all'angolo di sterzata anteriore, l'obiettivo dell'agente è quello di far rimanere il veicolo allineato con la linea centrale della corsia.

L'agente dovrà cercare di prendere decisioni sull'angolo di sterzata in funzione delle informazioni che riceve dai sensori. I vincoli sono un elemento chiave di questo processo; essi rappresentano le regole della strada che l'agente deve rispettare affinchè il sistema sia un sistema sicuro.

Vediamo come viene implementato il seguente simulatore.

3.3.1 Implementazione del Simulatore 2

Veicolo

Definiamo le caratteristiche del veicolo.

```
rng(0);
m = 1575;           % Massa del veicolo
Iz = 2875;          % Momento di inerzia di imbardata
lf = 1.2;           % Distanza longitudinale dal baricentro ai pneumatici anteriori
lr = 1.6;           % Distanza longitudinale dal baricentro ai pneumatici posteriori
Cf = 19000;         % Rigidità in curva dei pneumatici anteriori
Cr = 33000;         % Rigidità in curva dei pneumatici posteriori
Vx = 15;            % Velocità longitudinale
Ts = 0.1;           % Tempo di campionamento
T = 15;             % Durata
rho = 0.001;        % Curvatura stradale
e1_initial = 0.2;   % Deviazione laterale iniziale dalla linea centrale
e2_initial = -0.1;  % Errore iniziale dell'angolo di imbardata
steerLimit = 0.2618;% Massimo angolo di sterzata per il comfort del conducente
```

Ambiente e Agente

Creiamo un ambiente utilizzando il modello "*rlLearnConstraintLKA*". Definiamo il modello.

```
mdl = 'rlLearnConstraintLKA';
open_system(mdl)
```

Creiamo le specifiche di osservazione e azione per l'ambiente.

```
obsInfo = rlNumericSpec([6 1]);
actInfo = rlFiniteSetSpec((-15:15)*pi/180);
```

Creiamo l'ambiente.

```
agentblk = [mdl '/RL Agent'];
env = rlSimulinkEnv(mdl,agentblk,obsInfo,actInfo);
```

La funzione "*localResetFcn*" ripristina la deviazione laterale e il relativo angolo di imbardata all'inizio di ogni episodio.

```
env.ResetFcn = @(in)localResetFcn(in);
```

Creiamo un agente DQN.

```
agent = createDQNAgentLKA(Ts,obsInfo,actInfo);
```

Vincoli

Carichiamo i dati presalvati.

```
collectData = false;
if collectData
    count = 1050;
    data = collectDataLKA(env,agent,count);
else
    load trainingDataLKA data
end
```

Estraiamo dati in input, output e definiamo i vincoli.

```
% Estrarre lo stato e i dati di input
inputData = data(1:1000,[2,5,4,6,7]);
% Estrarre i dati per la deviazione laterale nella fase temporale successiva.
outputData = data(1:1000,8);
% Calcolare la relazione dallo stato e dall'input alla deviazione laterale..
relation = inputData\outputData;
% Estrarre le componenti dei coefficienti della funzione di vincolo.
Rf = relation(1:4)';
Rg = relation(5);
```

Training

Definiamo le opzioni per l'addestramento, usiamo il modello "*rlLKAwithConstraint*".

```
mdl = 'rlLKAwithConstraint';
open_system(mdl)
agentblk = [mdl '/RL Agent'];
env = rlSimulinkEnv(mdl,agentblk,obsInfo,actInfo);
env.ResetFcn = @(in)localResetFcn(in);
```

Definiamo le caratteristiche e la funzione per il training.

```
maxepisodes = 5000;
maxsteps = ceil(T/Ts);
trainingOpts = rlTrainingOptions(...
'MaxEpisodes',maxepisodes,...
'MaxStepsPerEpisode',maxsteps,...
'Verbose',false,...
'Plots','training-progress',...
'StopTrainingCriteria','EpisodeReward',...
'StopTrainingValue',-1);
trainAgent = true;
if trainAgent
    trainingStats = train(agent,env,trainingOpts);
else
    load rlAgentConstraintLKA agent
end
```

Anche in questo caso, è importante identificare alcuni elementi chiave che saranno discussi in tutti gli esperimenti, ovvero episodi, average reward, episode reward e episode $Q0$.

3.3.2 Analisi degli esperimenti relativi al Simulatore 2

Tutti gli esperimenti effettuati hanno in comune l'obiettivo finale che è quello di ottenere una ricompensa periodica di valore pari a -1 entro il numero massimo di episodi prefissato a 5000.

Esperimento 1

Per effettuare il primo esperimento abbiamo utilizzato il codice standard del simulatore senza aver effettuato alcuna modifca. Il risultato dell'Esperimento 1 è mostrato in Figura 3.6. L'obiettivo dell'addestramento di raggiungere una ricompensa periodica pari a -1 è stato ottenuto in seguito a 2255 episodi e ad una durata temporale pari a circa 2 ore.

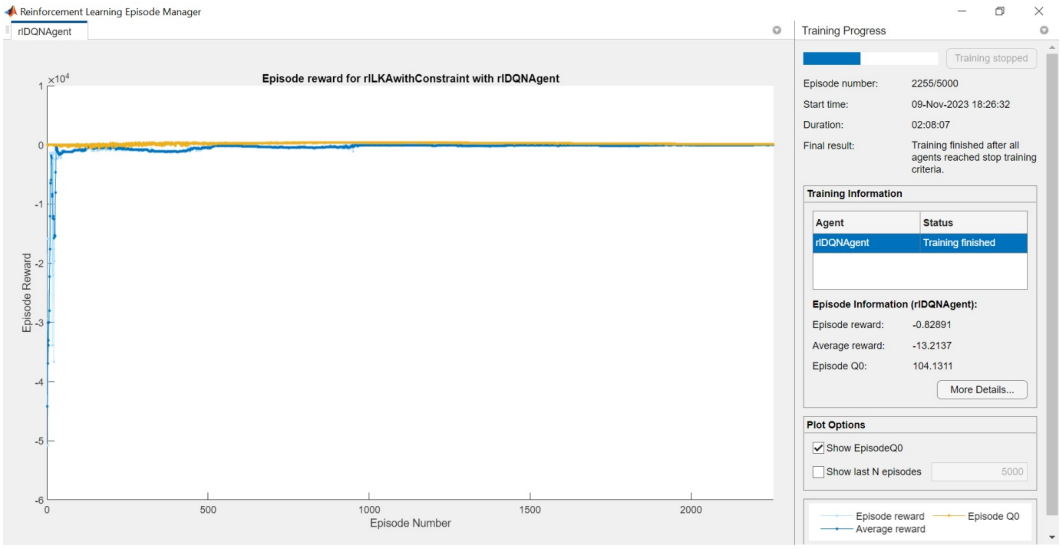

Figura 3.6: Risultato finale dell'Esperimento 1 del Simulatore 2

Di seguito analizziamo le fasi relative all'andamento della curva interessata per il raggiungimento dell'obiettivo:

- *Momenti iniziali (Episodi 0-25):* nell'Episodio 0, l'agente mostra una prestazione molto negativa con un valore di circa -50.000 che, nei successivi 25 episodi, migliora gradualmente fino a toccare un valore di circa -500.
- *Miglioramenti e crolli (Episodi 25-50):* tra gli episodi 25 e 50 la curva subisce un crollo, scendendo a circa -15.000. Questa fase di tendenza decrescente è seguita da un miglioramento fino a valori prossimi a -200.
- *Prima stabilità (Episodi 50-125):* in questa fase la curva mostra una maggiore stabilità che si mantiene intorno a -1.200. Tuttavia si osserva un leggero incremento verso lo zero.
- *Fase intermedia (Episodi 125 - 500):* tra questi episodi, la curva mostra stabilità fino all'Episodio 250, quando inizia a decrescere nuovamente, toccando valori di circa -1.110 all'Episodio 500. Da qui inizia una fase di crescita che continua fino all'Episodio 750.
- *Nuova fase di decrescita (Episodi 750 - 1000):* in questa fase possiamo osservare una tendenza della curva a decrescere raggiungendo valori prossimi a -1.000.
- *Crescite e conclusione stabile (Episodi 1000 - 2255):* nella fase conclusiva la curva mostra una tendeza stabile e gradualmente crescente fino all'Episodio 2255, quando raggiunge finalmente un valore di -1. Il risultato dimostra il successo dell'addestramento, con l'agente che raggiunge un livello di performance accettabile nell'ambito del mantenimento della corsia.

La curva di "*average reward*" rispecchia l'andamento appena analizzato della curva di *episode reward*.

La curva di episode *Q*0 presenta variazioni più significative nelle fasi iniziali dell'esperimento, suggerendo un grado di incertezza nelle stime dei Q-value. Questo indica che l'agente ha una visione meno precisa del valore associato alle azioni.

Man mano che l'esperimento prosegue, la curva mostra una notevole tendenza a ridurre la sua ampiezza. Dopo l'Episodio 500, si stabilizza attorno a valori prossimi allo zero, indicando che l'agente è in grado di effettuare stime migliori.

Dopo aver analizzato i risultati del primo esperimento passiamo ad analizzare i risultati del secondo esperimento.

Esperimento 2

Anche con questo simulatore è stato effettuato un secondo esperimento, per farlo sono state apportate delle modifiche al codice. Nel seguito esamineremo tali modifiche:

Le seguenti modifiche sono state apportate ad alcune caratteristiche del veicolo, come, per esempio, la sua velocità iniziale, la sua massa, il limite di sterzata, la rigidtà dei pneumatici anteriori, la devizione laterale iniziale e l'angolo di imbardata.

```
Vx = 25;
m = 1700;
steerLimit = 0.2;
Cf = 20000;
e1_initial = -0.3;
e2_initial = 0.2;
```

La modifica più interessante fatta per l'Esperimento 2 è quella riportata in seguito, in cui si può notare come sia stato aggiunto ai dati di input, che includono lo stato e l'azione dell'agente, del rumore gaussiano randomico.

```
inputData = data(1:1000,[2,5,4,6,7]);
inputData = inputData + 0.01 * randn(size(inputData));
```

Dopo aver specificato le modifiche apportate, possiamo approfondire i risultati dell'Esperimento 2.

La prima cosa molto importante da evidenziare è che nell'Esperimento 2 non si è raggiunto l'obiettivo prefisatto; infatti l'esperimento è terminato una volta raggiunto il numero massimo di 5000 episodi dopo circa 5 ore.

Il risultato dell'esperimento è rappresentato in Figura 3.7.

Esaminiamo le diverse fasi nel progresso della curva *episode reward*:

- *Fase 1 (Episodi 1-1000):* nella fase iniziale si osservano valori molto negativi, intorno a circa -12.500, ad evidenza di una prestazione iniziale difficoltosa. Durante questa fase, la curva manifesta ampie variazioni, con picchi sia su valori maggiori, come -2.000, che su valori minori, come -22.500. La densità maggiore di punti si ha attorno a -7.500.

- *Fase 2 (Episodi 1000-1650):* in questi episodi la curva inizia a toccare picchi di maggior valore, circa -500, indicando una progressiva crescita delle performance dell'agente. Rimangono, comunque, presenti picchi a valori molto negativi. A partire dall'Episodio 1650, si nota una diminuzione dell'ampiezza della curva.

- *Fase 3 (Episodi 1650-2200):* durante questo periodo, la curva diminusce drasticamente la propria ampiezza, mantenendo una sorta di stabilità. Tra gli Episodi 1800 e 2000, la curva alterna picchi molto negativi, pari a -8000, a picchi che toccano i -500. Negli episodi compresi tra 2000 a 2200, si osserva una fase di stabilizzazione, con valori relativamente costanti, ma con la presenza, seppur con bassa frequenza, di picchi profondamente negativi.

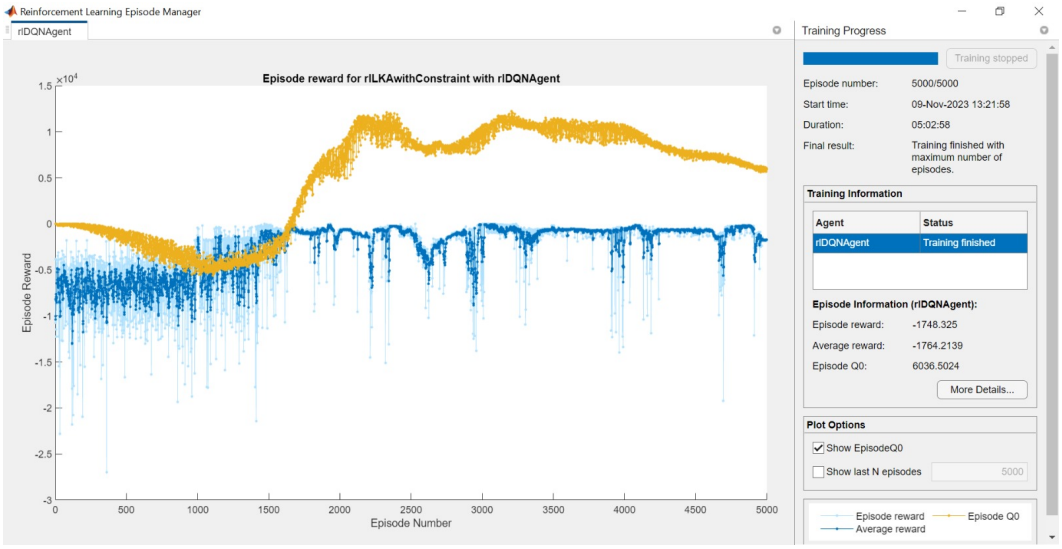

Figura 3.7: Risultato finale dell'Esperimento 2 del Simulatore 2

- *Fase 4 (Episodi 2200-5000):* tra gli Episodi 2200 e 2300, si verifica una brusca decrescita dei valori seguita da una brusca ricrescita, con una stabilizzazione di questi fino all'Episodio 2500. In corrispondenza dell'Episodio 3000, si manifesta una decrescita graduale fino a valori di -13000. Da 3000 fino a 5000, la curva mantiene valori stabili di -800, con alcune decrescite tra gli Episodi 3250 e 3500, e 3750 e 4000, dove si raggiungono picchi negativi di -12.500.

La curva di *awerage reward* segue l'andamento della curva *episode reward*. Mentre la curva di episode $Q0$ tra gli Episodi 0 e 1000, ha un andamento decrescente fino a toccare valori di -5000 circa. In corrispondenza dell'Episodio 1000 vi è un punto di flesso in cui la curva cambia andamento, crescendo fino all'Episodio 2400 circa, toccando valori positivi pari a +10000. Tra gli Episodi 2500 e 3250 la curva assume un andamento convesso seguito da un andamento concavo tra gli Episodi 3250 e 3500. Nelle fasi finali si stabilizza.

Le modifiche apportate potrebbero aver generato divergenze tra gli elementi chiave dei due esperimenti; Esaminiamo più in dettaglio tali modifiche:

- L'aumento della velocità longitudinale, della massa, della rigidità in curva dei pneumatici anteriori e la modifica delle condizioni iniziali potrebbero aver reso la guida più complessa, richiedendo una maggiore precisione alle azioni dell'agente. Questo potrebbe aver influito nella instabilità della curva di *episode reward*.
- La riduzione del limite di sterzata potrebbe aver influenzato la capacità dell'agente di mantenere la corsia, a causa di curve più strette, non permettendo il raggiungimento dell'obiettivo.
- L'inserimento di rumore gaussiano ha sicuramente influenzato l'Esperimento 2 nel seguente modo:
  - I dati di input, possono variare a causa del rumore anche se l'agente si trova sempre nella medisima situazione; ciò rende l'ambiente circostante imprevedibile.
  - La presenza del rumore gaussiano può rendere più difficile per l'agente apprendere relazioni tra azioni e risultati.
  - La presenza di tale rumore può portare ad una maggiore variabilità poichè l'agente può ottenere buoni risultati, oppure risultati pessimi, per colpa della casualità del rumore.

3.4 Train DDPG Agent for Adaptive Cruise Control

Questo simulatore si concentra sull'addestramento di un agente DDPG per il controllo di velocità adattivo, noto anche come cruise control. L'obiettivo dell'addestramento è far viaggiare l'auto ego a una velocità predefinita dovendo mantenere una distanza sicura dall'auto leader. L'agente dovrà cercare di prendere decisioni circa il controllo dell'accelerazione e del freno.

Vediamo il codice standard per implementare il simulatore.

3.4.1 Implementazione del Simulatore 3

Veicolo e modello Simulink

Definiamo la posizione e la velocità iniziale per i due veicoli. Specifichiamo lo spazio di arresto, l'intervallo temporale e la velocità impostata che l'ego car deve raggiungere. Limitiamo i valori di accelerazione e impostiamo un intervallo di campionamento e un tempo di simulazione.

```
x0_lead = 50;   % posizione iniziale per l'auto in testa
v0_lead = 25;   % velocità iniziale per l'auto in testa
x0_ego = 10;    % posizione iniziale per il veicolo ego
v0_ego = 20;    % velocità iniziale per il veicolo ego
D_default = 10;
t_gap = 1.4;
v_set = 30;
amin_ego = -3;
amax_ego = 2;
Ts = 0.1;
Tf = 60;

mdl = "rlACCMdl";
open_system(mdl)
agentblk = mdl + "/RL Agent";
```

Ambiente

Creiamo le specifiche di osservazione e azione per l'ambiente.

```
obsInfo = rlNumericSpec([3 1], ...
LowerLimit=-inf*ones(3,1), ...
UpperLimit=inf*ones(3,1));

obsInfo.Name = "observations";
obsInfo.Description = ...
"velocity error and ego velocity";

actInfo = rlNumericSpec([1 1], ...
LowerLimit=-3,UpperLimit=2);

actInfo.Name = "acceleration";
```

Creiamo l'interfaccia dell'ambiente Simulink.

```
env = rlSimulinkEnv(mdl,agentblk,obsInfo,actInfo);
```

La funzione "*localResetFcn*" alla fine dell'episodio resetta in maniera randomica la posizione iniziale della lead car.

```
env.ResetFcn = @(in)localResetFcn(in);
```

Agente

L'agente utilizzato in questo esempio è un agente DDPG. Inizializziamo le reti attore e critico in modo casuale.

```
L = 48; % numero di neuroni
mainPath = [
    featureInputLayer( ...
        prod(obsInfo.Dimension), ...
        Name="obsInLyr")
    fullyConnectedLayer(L)
    reluLayer
    fullyConnectedLayer(L)
    additionLayer(2,Name="add")
    reluLayer
    fullyConnectedLayer(L)
    reluLayer
    fullyConnectedLayer(1,Name="QValLyr")
];

actionPath = [
    featureInputLayer( ...
        prod(actInfo.Dimension), ...
        Name="actInLyr")
    fullyConnectedLayer(L,Name="actOutLyr")
];
```

Definiamo i parametri della rete critico e effettuiamo la conversione in un oggetto "*dlnetwrok*".

```
criticNet = layerGraph(mainPath);
criticNet = addLayers(criticNet, actionPath);
criticNet = connectLayers(criticNet,"actOutLyr","add/in2");
criticNet = dlnetwork(criticNet);
summary(criticNet)
```

Grazie al comando seguente possiamo visualizzare la struttura schematica della rete critico mostrata in Figura 3.8.

```
plot(criticNet)
```

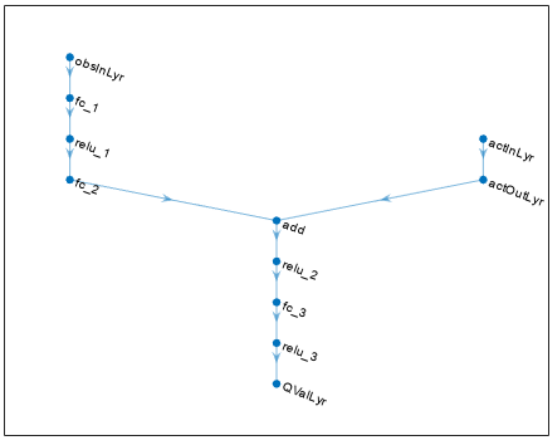

Figura 3.8: Schema della rete critico

Creiamo il critico.

```
critic = rlQValueFunction(criticNet,obsInfo,actInfo,...
ObservationInputNames="obsInLyr",ActionInputNames="actInLyr");
```

Definiamo i parametri della rete neurale attore ed effettuiamo la conversione in un oggetto "*dlnetwork*".

```
actorNet = [
    featureInputLayer(prod(obsInfo.Dimension))
    fullyConnectedLayer(L)
    reluLayer
    fullyConnectedLayer(L)
    reluLayer
    fullyConnectedLayer(L)
    reluLayer
    fullyConnectedLayer(prod(actInfo.Dimension))
    tanhLayer
    scalingLayer(Scale=2.5,Bias=-0.5)
];
actorNet = dlnetwork(actorNet);
summary(actorNet)
```

Creiamo l'attore.

```
actor = rlContinuousDeterministicActor(actorNet, ...
obsInfo,actInfo);
```

Definiamo le opzioni per l'addestramento dell'attore e del critico.

```
criticOptions = rlOptimizerOptions( ...
LearnRate=1e-3, ...
GradientThreshold=1, ...
L2RegularizationFactor=1e-4);

actorOptions = rlOptimizerOptions( ...
LearnRate=1e-4, ...
GradientThreshold=1, ...
L2RegularizationFactor=1e-4);
```

Definiamo i parametri dell'agente DDPG.

```
agentOptions = rlDDPGAgentOptions(...
SampleTime=Ts,...
ActorOptimizerOptions=actorOptions,...
CriticOptimizerOptions=criticOptions,...
ExperienceBufferLength=1e6);

agentOptions.NoiseOptions.Variance = 0.6;
agentOptions.NoiseOptions.VarianceDecayRate = 1e-5;
```

Creiamo l'agente utilizzando l'attore, il critico e l'oggetto agente "*agentOptions*".

```
agent = rlDDPGAgent(actor,critic,agentOptions);
```

Training

Definiamo le caratteristiche e la funzione per il training.

```
maxepisodes = 5000;
maxsteps = ceil(Tf/Ts);
trainingOpts = rlTrainingOptions(...
    MaxEpisodes=maxepisodes,...
    MaxStepsPerEpisode=maxsteps,...
    Verbose=false,...
    Plots="training-progress",...
    StopTrainingCriteria="EpisodeReward",...
    StopTrainingValue=260);

doTraining = true;
if doTraining
    trainingStats = train(agent,env,trainingOpts);
else
    load("SimulinkACCDDPG.mat","agent")
end
```

Simulazione

In questo simulatore è possibile simulare l'agente addestrato.

```
simOptions = rlSimulationOptions(MaxSteps=maxsteps);
experience = sim(env,agent,simOptions);
x0_lead = 80;
sim(mdl)
```

Come nei precedenti simulatori, in questo avremo degli elementi chiave ricorrenti nell'analisi dei simulatori: episodi, *average reward*, *episode reward* e episode *Q*0.

3.4.2 Analisi degli esperimenti per il Simulatore 3

All'interno di questo simulatore, gli esperimenti effettuati hanno come obiettivo finale una ricompensa periodica di valore pari a 260 entro il numero massimo di episodi prefissato a 5000.

Esperimento 1

Come nei precedenti casi, il primo esperimento è stato effettuato senza apportare alcuna modifica al codice. Il risultato dell'esperimento è mostrato in Figura 3.9. L'addestramento ha raggiunto l'obiettivo dopo 1214 episodi e circa 3 ore di tempo.

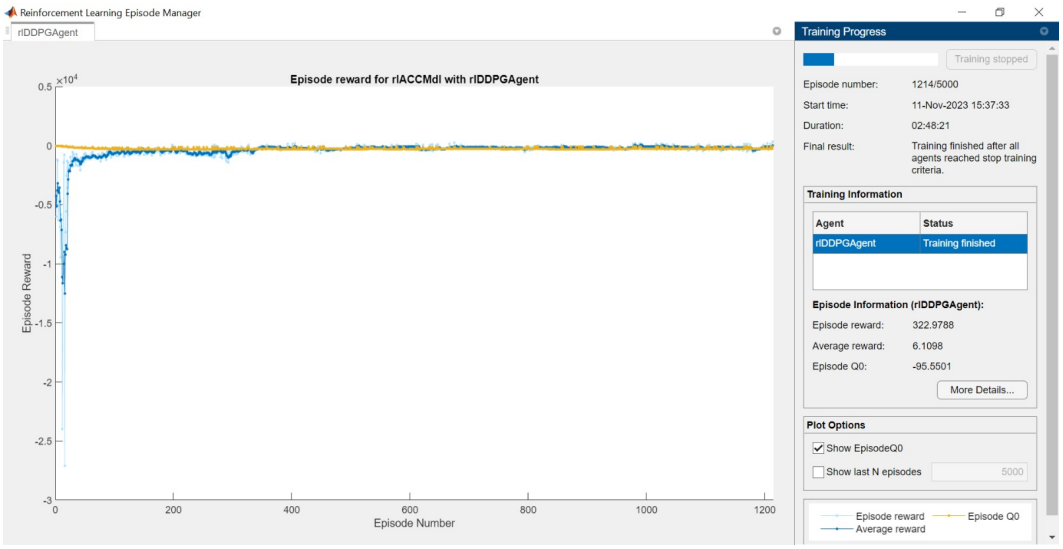

Figura 3.9: Risultato finale dell'Esperimento 1 del Simulatore 3

Analizziamo l'andamento relativo alla curva *episode reward*, responsabile del raggiungimento dell'obiettivo.

- 1^a *Fase (Episodi 1-50):* inizialmente la curva tocca valori molto negativi tra -1000 e -10000 con alcuni picchi su -20000 a dimostrazione del fatto che l'agente sta effettuando una esplorazione iniziale senza, però, ottenere risultati significativi.
- 2^a *Fase (Episodi 50-225):* nella fase successiva, la curva inizia a stabilizzarsi su valori pari a -2500 procedendo, però, con un andamento lento ma crescente fino all'Episodio 225. L'agente sta cercando di apprendere strategie più prestazionali.
- 3^a *Fase (Episodi 225-325):* tra gli Episodi 225 e 325, la curva si stabilizza nuovamente su valori negativi ma maggiori rispetto a prima a circa -1100, con un piccolo picco decrescente tra gli Episodi 275 e 300.

- 4ª *Fase (Episodi 325-1214):* una volta che l'esperimento ha superato l'Episodio 325, la curva determina un andamento simile ad una retta parallela all'asse su valori prossimi allo 0, indicando che l'agente ha imparato una strategia che si avvicina all'obiettivo. Infine, soltanto negli ultimi episodi, la curva raggiunge valori positivi, superando l'obiettivo richiesto a dimostrazione del fatto che l'agente ha raggiunto livelli di prestazione ottimali.

Anche in questo caso, essendone la media, la curva di *average reward* segue l'andamento della curva appena descritta. Mentre la curva di episode $Q0$ segue un andamento lineare per tutta la durata dell'esperimento attorno a valori prossimi allo zero.

Oltre all'addestramento, il simulatore fornisce la possibilità di visualizzare l'evoluzione dell'agente attraverso tre schemi, rappresentati nelle Figure 3.10, 3.11 e 3.12, queste analizzano alcuni parametri nel corso dell'esperimento, ovvero:

- velocità della lead car;
- velocità della ego car;
- velocità obiettivo della ego car;
- accellerazione della ego car;
- distanza relativa e di sicurezza.

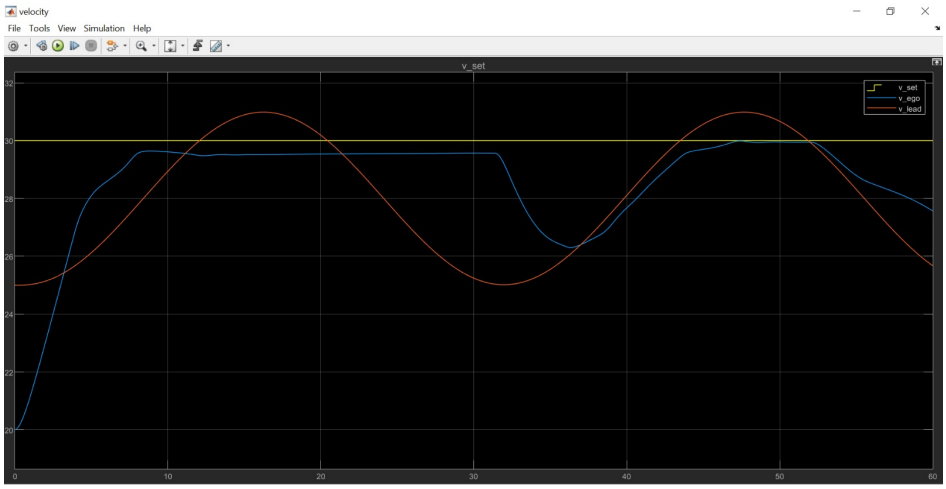

Figura 3.10: Andamento delle velocità per l'Esperimento 1 del Simulatore 3

In breve, analizziamo l'interazione degli schemi.

Nei primi 32 secondi, la distanza relativa è maggiore della distanza di sicurezza, quindi l'ego car raggiunge e mantiene la velocità impostata. Fintanto che l'ego car raggiunge la velocità impostata, l'accelerazione è positiva; in seguito, una volta raggiunta tale velocità, l'accellerazione si stabilizza sullo zero. A t=32, la distanza relativa e di sicurezza si incrociano, con la distanza relativa che diventa inferiore rispetto a quella di sicurezza. Da questo momento in poi, l'ego car diminuisce la propria velocità rispetto a quella impostata. Per rallentare, l'accelerazione diventa negativa da 32 a circa 37 secondi. Successivamente, l'ego car regola l'accelerazione a seconda che la distanza relativa sia considerata sicura o meno per cercare di ristabilire la velocità obiettivo.

Passiamo ora ad analizzare i risultati del secondo esperimento.

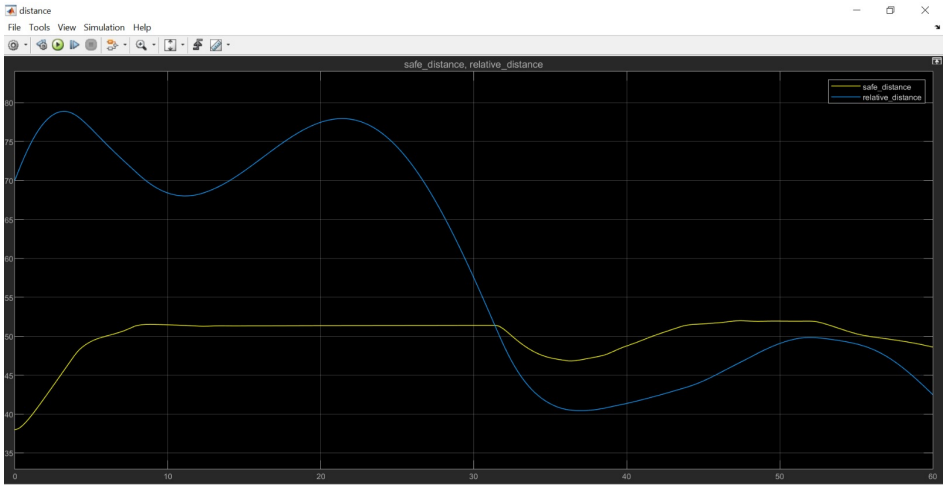

Figura 3.11: Andamento delle distanze per l'Esperimento 1 del Simulatore 3

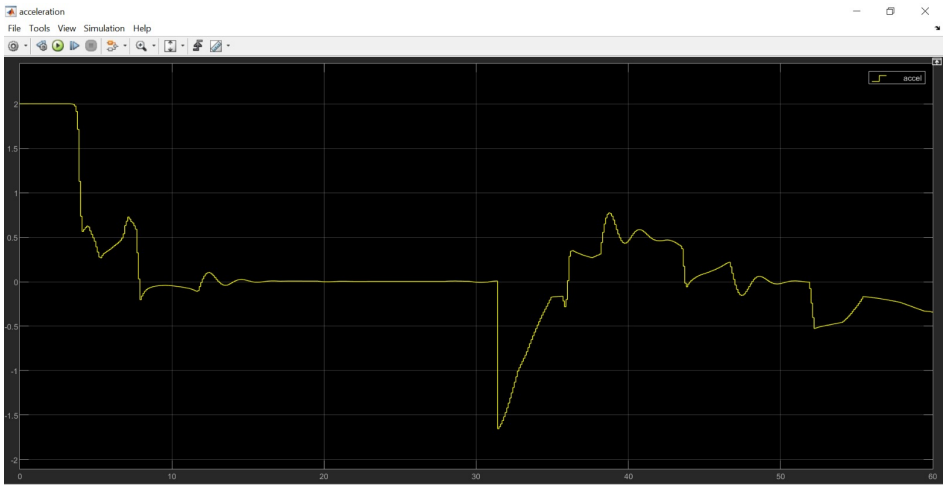

Figura 3.12: Andamento dell'accelerazione per l'Esperimento 1 del Simulatore 3

Esperimento 2

Per il secondo esperimento con questo simulatore abbiamo apportato alcune modifiche al codice per poter determinare delle differenze con il primo esperimento. Nel seguito esamineremo tali modifiche. Le prime modifiche sono state effettuate ad alcune caratteristiche dei due veicoli, come, per esempio, la loro velocità iniziale, la velocità obiettivo della ego car, lo spazio di arresto, l'intervallo temporale e i vincoli dell'accelerazione.

```
x0_lead = 30;   % posizione iniziale per l'auto in testa
v0_lead = 30;   % velocità iniziale per l'auto in testa
x0_ego = 25;    % posizione iniziale per il veicolo ego
v0_ego = 25;    % velocità iniziale per il veicolo ego
D_default = 15;
t_gap = 1.8;
v_set = 35;
amin_ego = -5;
amax_ego = 3;
```

Le seguenti modifiche sono state effettuate sui tassi di apprendimento, sul fattore di regolarizzazione L2, sui limiti di apprendimento dell'attore e del critico dell'agente. Inoltre abbiamo effettuato delle modifiche ad alcuni parametri dell'agente, come la lunghezza del buffer di esperienza e la varianza del rumore aggiunto alle azioni dell'agente.

```
criticOptions = rlOptimizerOptions( ...
```

```
    'LearnRate', 1e-4, ...
    'GradientThreshold', 0.5, ...
    'L2RegularizationFactor', 1e-5);
actorOptions = rlOptimizerOptions( ...
    'LearnRate', 5e-5, ...
    'GradientThreshold', 0.5, ...
    'L2RegularizationFactor', 1e-5);
agentOptions = rlDDPGAgentOptions(...
    'ExperienceBufferLength', 1e5);
agentOptions.NoiseOptions.Variance = 0.2;
```

Dopo aver descritto le modifiche apportate ai vari parametri, analizziamo i risultati ottenuti grazie all'Esperimento 2, i quali sono rappresentati in Figura 3.13. È fondamentale evidenziare che, nell'Esperimento 2 è stato raggiunto l'obiettivo prefisatto dopo soltanto 182 episodi e 45 minuti di tempo.

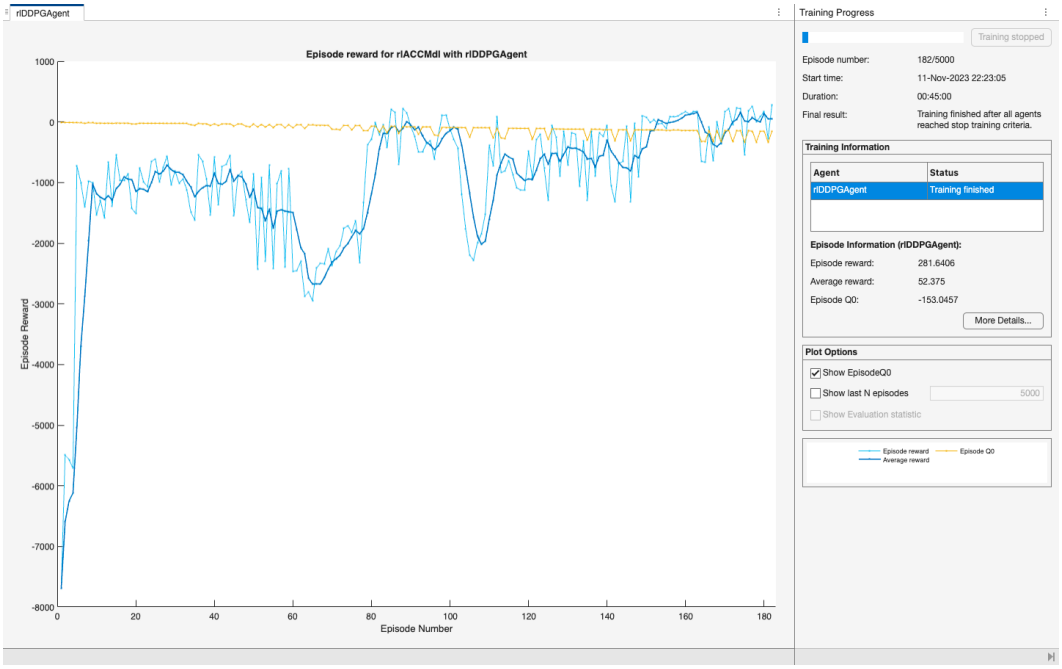

Figura 3.13: Risultato finale dell'Esperimento 2 per il Simulatore 3

Analizziamo la tendenza della curva delle ricompense degli episodi, che è determinante per il raggiungimento dell'obiettivo.

- *Fase Iniziale (Episodi 1-60):* durante l'episodio iniziale, la curva aveva dei valori molto negativi, circa -8000, ma da subito, all'incirca in corrispondenza dell'Episodio 5, ha fatto notare una tendenza positiva toccando valori di -1000. Essa ha mantenuto questi valori indicativamente fino all'Episodio 60. Tutto ciò indica una fase di esplorazione iniziale più efficace rispetto a quella dell'Esperimento 1.
- *Prima fase di crollo (Episodi 60-70):* verso l'Episodio 60, si è verificato un improvviso crollo della curva di *episode reward* a circa -2000, giustificabile da una sfida inaspettata per l'agente.
- *Seconda fase di crollo e risalita (Episodi 70-106):* tra questi episodi, la curva ha manifestato una forte risalita, raggiungendo valori positivi pari 200. Tuttavia, tra l'Episodio 100 e l'Episodio 106, si è verificato un piccolo crollo, seguito da una rapida risalita verso valori vicini allo zero all'Episodio 110; ciò potrebbe indicare una fase di apprendimento più avanzata per l'agente che sta cercando politiche più sofisticate.

- *Stabilità (Episodi 110-150):* tra gli Episodi 110 e 150, la curva di *episode reward* è rimasta relativamente stabile, con fluttuazioni di poco sotto allo zero.
- *Aumenti e crolli (Episodi 150-170):* in questa fase la curva ha fatto notare un aumento e un crollo, mantenendosi comunque vicina allo zero.
- *Fase finale (Episodi 170-182):* nell'ultima fase c'è stato l'aumento decisivo che ha portato ad ottenere l'obiettivo, indicando che l'agente ha ottimizzato le strategie apprese.

Per l'Esperimento 2, esamineremo in modo analogo a come fatto per l'Esperimento 1 gli schemi per valutare l'evoluzione dell'agente. Essi sono rappresentati nelle Figure 3.14, 3.15 e 3.16.

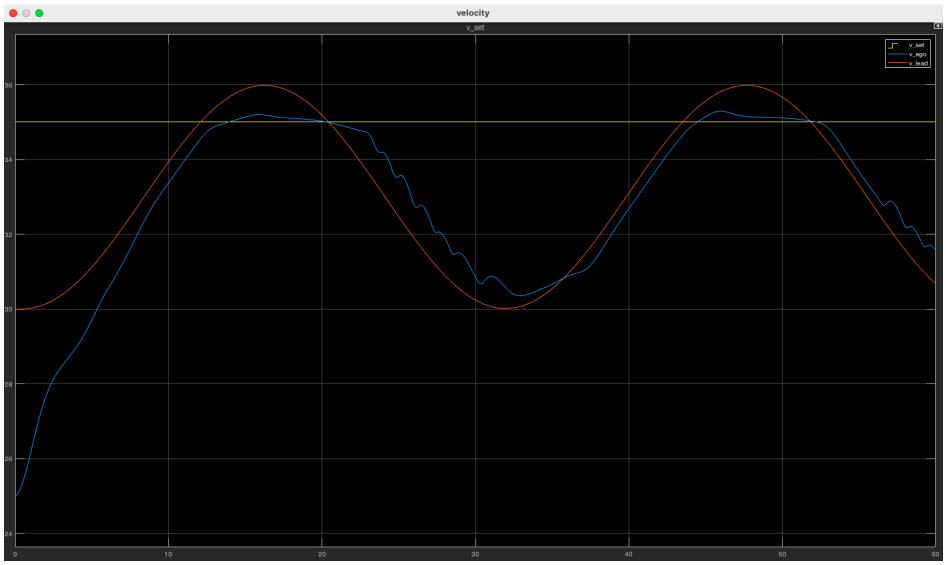

Figura 3.14: Andamento delle velocità per l'Esperimento 2 del Simulatore 3

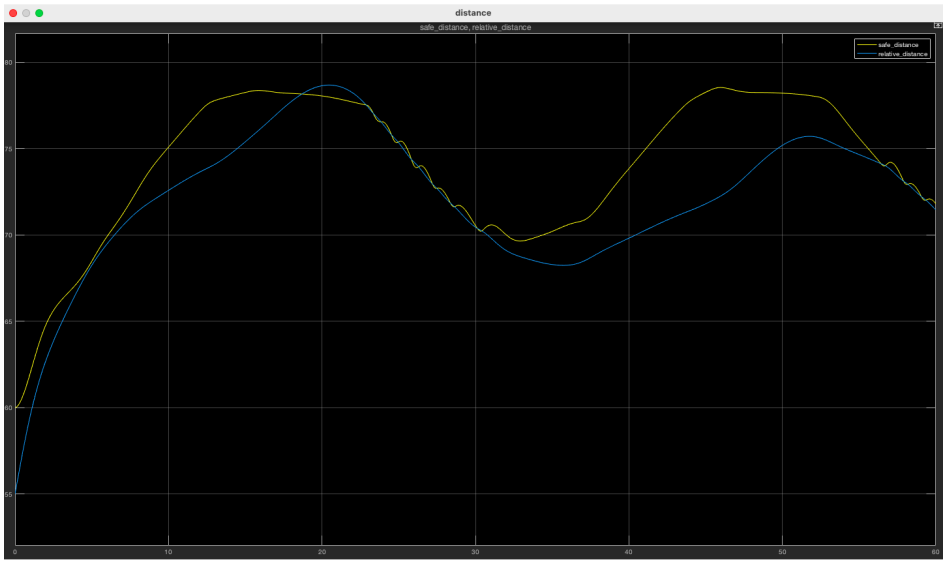

Figura 3.15: Andamento delle distanze per l'Esperimento 2 del Simulatore 3

In sintesi, esaminiamo come interagiscono i diversi schemi. Nella fase iniziale, la distanza relativa e quella di sicurezza aumentano entrambe, e sono pressocchè identiche; per questo

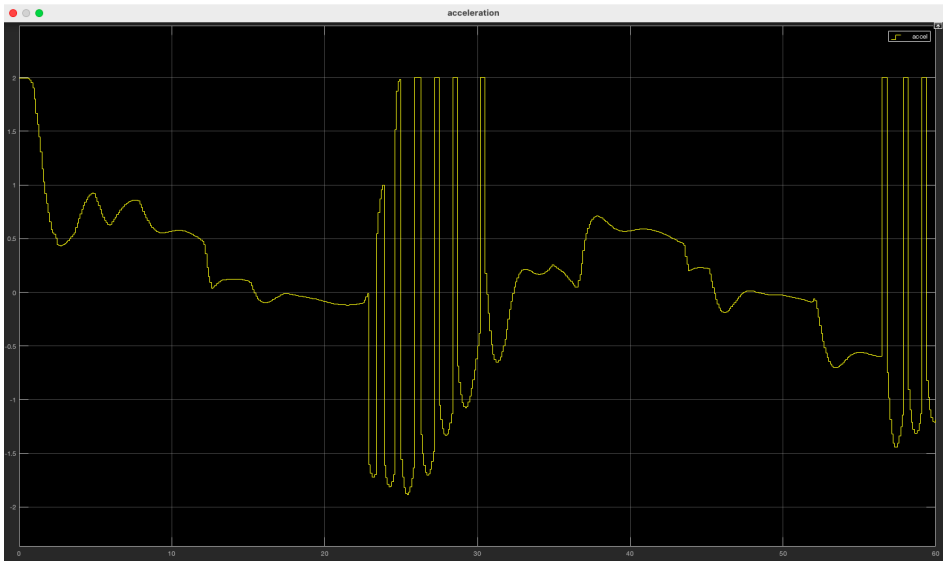

Figura 3.16: Andamento dell'accelerazione per l'Esperimento 2 del Simulatore 3

motivo l'accellerazione nei primi 12 secondi è positiva in modo che l'ego car possa raggiungere la velocità definita. Tra 20 e 30 secondi, la velocità della lead car diminuisce, la distanza di sicurezza e quella relativa anche; questo spiega la diminuzione della velocità dell'ego car; l'accellerazione alterna valori negativi e positivi (con una frequenza maggiore per quelli negativi). In seguito, l'auto ego regola l'accelerazione in base alla sicurezza della distanza relativa, cercando di ristabilire la velocità desiderata.

In conclusione analizziamo come le modifiche apportate avrebbero determinato delle divergenze tra i due esperimenti:

- Le condizioni iniziali dell'esperimento, la lead car e l'ego car partono entrambe con posizioni e velocità più vicine tra loro, potrebbero aver reso l'apprendimento più rapido per aver diminuito la complessità dell'ambiente iniziale.
- La varianza del rumore minore potrebbe aver ridotto la possibilità di esplorazioni casuali, consentendo all'agente di sviluppare strategie efficaci.
- L'aumento della distanza di sicurezza può aver permesso all'agente di apprendere strategie più sicure.
- La riduzione del gap time potrebbe aver portato l'agente a concentrarsi maggiormente sulla lead car.
- La riduzione dei coefficienti di regolarizzazione nei parametri dell'agente potrebbe aver favorito un apprendimento meno vincolato.

Concluso il terzo esperimento ci apprestiamo ad analizzare il quarto.

3.5 Object Detection Using SSD Deep Learning

Il seguente simulatore si concentra sull'addestramento di un rilevatore di veicoli utilizzando l'architettura *SSD (Single Shot Multibox Detector)*. La funzione chiave è *trainSSDObjectDetector*, sviluppata per semplificare la fase di addestramento di rilevatori di oggetti basati su SSD.

L'obiettivo principale del simulatore, quindi, è creare un modello in grado di identificare e delimitare correttamente veicoli in un'immagine. L'output finale sarà un rilevatore di veicoli addestrato, pronto per l'uso su delle immagini.

Per ragioni di brevità, eviteremo di entrare nei dettagli implementativi del codice e ci concentreremo sull'analisi dei risultati degli esperimenti.

3.5.1 Analisi degli Esperimenti per il Simulatore 4

L'esperimento condotto con il Simulatore 4 non ha un obiettivo specifico, se non quello di massimizzare la precisione con cui viene rilevata l'automobile. Tuttavia, è stabilito un limite da raggiungere di 60 epoche per ciascun test. È importante notare che in questo simulatore si parla di epoche, un concetto più vicino all'apprendimento automatico, e non di episodi, come nei precedenti simulatori; il concetto di episodio, infatti, rappresenta un concetto più vicino al Reinforcement Learning.

Nella Figura 3.17 è rappresentata l'immagine utilizzata per verificare l'addestramento del detector.

Figura 3.17: Immagine di test

L'immagine di test è stata aggiunta al codice standard del simulatore grazie alle righe di codice che seguono con le quali viene anche visualizzato il risultato dell'apprendimento del detector.

```
ImagePath = '/Users/simone/Desktop/TESI_SIMONE/SIMULATORE 4/test3.png';
I = imread(yourImagePath);
I = imresize(I, inputSize(1:2));
[bboxes, scores] = detect(detector, I);
I = insertObjectAnnotation(I, 'rectangle', bboxes, scores);
figure;
imshow(I);
title('Risultati della detection sull'immagine di test');
```

Nell'esperimento, inoltre , vengono generati e forniti in output anche diversi parametri contenuti in una tabella. Questi sono i seguenti:

- *Epoch:* rappresenta un ciclo completo attraverso l'intero set di addestramento.
- *Iteration:* rappresenta un singolo passaggio attraverso un mini-batch durante l'epoca. L'intero set di addestramento viene utilizzato per addestrare la rete maggiormente se le epoche sono in un numero maggiore. Maggiore è anche il numero di aggiornamenti

di peso che vengono eseguiti durante l'addestramento se il numero di iterazioni è maggiore.

- *Time Elapsed:* indica il tempo di addestramento.
- *Mini-batch Loss:* indica quanto l'output previsto dalla rete neurale si discosta dai valori effettivi presenti nel mini-batch di dati. È ideale, con l'avanzare dell'esperimento, avere una riduzione di questo valore, indicando un aumento delle capacità di effettuare previsioni del modello.
- *Mini-Batch Accuracy:* rappresenta l'accuratezza, la quale è calcolata con la percentuale delle previsioni effettuate correttemente dalla rete in rapporto al totale delle previsioni. Il valore di questo parametro è un indicatore della percentuale con cui il modello sta effettuando previsioni corrette.
- *Mini-Batch RMSE:* è una misura che indica la differenza tra le previsioni del modello e i valori effettivi nel mini-batch. Un valore più basso per questo indicatore si traduce in una maggiore precisione nelle previsioni. Seppur simile al parametro Mini-Batch Loss, ci sono delle differenze tra i due, ossia:
  - La Mini-Batch Loss si riferisce alla perdita, la quale viene calcolata utilizzando una funzione di perdita. È utilizzata per l'ottimizzazione.
  - La Mini-batch RMSE si riferisce alla valutazione della performance che viene calcolata utilizzando la radice quadrata della media degli errori quadratici. È utilizzata per valutare la precisione.
- *Base Learning Rate:* indica il tasso di apprendimento utilizzato durante l'addestramento, il quale regola l'ampiezza degli aggiornamenti dei pesi della rete.

Esperimento unico

L'esperimento è stato effettuato inserendo la propria immagine per il test finale del rilevatore.

Analizziamo i vari output del simulatore.

In Figura 3.18 è mostrato il risultato del test effettuato sul detector addestrato. Come si può notare, il rilevatore individua e delinea il veicolo con un rettangolo giallo, inserendo anche un valore che indica il grado di certezza, il quale è pari al 99.871%. L'addestramento è effettuato solamente sull'identificazione di automobili e per tale motivo il ciclista sulla destra e i cartelli stradali non sono evidenziati.

L'esperimento ha prodotto in output dei valori specifici relativi ai parametri già descritti in generale nella Sottosezione 1.5.1. La tabella presente in Figura 3.19 esprime tali valori.

Analizziamo in dettaglio l'andamento di questi valori durante l'esperimento.

- Il tempo trascorso durante l'addestramento cresce con l'aumentare delle epoche e ciò rappresenta la normalità. Tuttavia, è importante notare che il modello sta apprendendo in maniera efficiente poiché il tempo rimane contenuto.
- Nella colonna relativa ai valori del mini-batch loss possiamo notare una diminuzione significativa, da un valore iniziale elevato (50.6666), a valori finali più bassi (0.6865), a dimostrazione del fatto che sta apprendendo efficacemente dai dati con l'avanzare delle epoche.

Figura 3.18: Risultato del test dell'esperimento

- La mini-batch accuracy aumenta da un valore iniziale (47.80%) a valori molto elevati (fino al 99.97%), indicando che il rilevatore sta individuando le automobili in maniera sempre più accurata.
- L'andamento della mini-batch RSME da valori iniziali più elevati (1.88) a valori più bassi (0.39) dimostra una maggiore precisione nelle previsioni del modello rispetto ai dati reali.

Per valutare le prestazioni del rilevatore addestrato viene utilizzata la metrica chiamata *precisione media*, la quale combina la capacità del rilevatore di fare previsioni corrette *(precision)* e la sua abilità ad individuare tutti gli oggetti presenti *(recall)*. Il grafico della curva Precision-Recall visualizza quanto il rilevatore sia preciso a vari livelli di recall. L'obiettivo è che la precisione rimanga alta anche quando si sta inziando ad individuare un numero crescente di oggetti. L'esperimento ha prodotto il grafico, mostrato in Figura 3.20.

Diamo, nel seguito, una piccola descrizione a questo grafico:

- Nella parte superiore del grafico possiamo notare che è indicato $AveragePrecision = 0.97$, questo valore rappresenta la media delle precisioni calcolate a diverse soglie di confidenza.
- La curva inizia da (0, 1), indicando una precisione massima (1) e un recall nullo (0), a dimostrazione del fatto che inizialmente il rilevatore fa poche previsione ma corrette.
- In seguito sono presenti delle discese e delle salite che potrebbero indicare un punto di trade-off in cui si cerca di bilanciare i valori di precision e recall.

In conclusione, questo simulatore possiamo dire che è possibile condurre ulteriori esperimenti per migliorare le prestazioni del detector mediante l'adozione di architetture *(You Only Look Once)*, in particolare YOLOv2 o YOLOv4, invece di SSD.

```
Training on single CPU.
Initializing input data normalization.
|========================================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |     Loss     |   Accuracy   |     RMSE     |      Rate       |
|========================================================================================================|
|       1 |           1 |       00:00:30 |       50.6666 |       47.80% |         1.88 |          0.0010 |
|       5 |          50 |       00:18:17 |       3.6199 |       99.71% |         1.11 |          0.0010 |
|      10 |         100 |       00:34:13 |       2.1974 |       99.82% |         0.88 |          0.0010 |
|      14 |         150 |       00:48:45 |       1.2246 |       99.89% |         0.59 |          0.0010 |
|      19 |         200 |       01:05:09 |       1.0407 |       99.92% |         0.55 |          0.0010 |
|      23 |         250 |       01:20:18 |       1.0247 |       99.93% |         0.45 |          0.0010 |
|      28 |         300 |       01:38:54 |       0.8679 |       99.96% |         0.42 |          0.0010 |
|      32 |         350 |       01:59:04 |       0.8737 |       99.96% |         0.40 |          0.0008 |
|      37 |         400 |       02:18:05 |       0.6485 |       99.96% |         0.39 |          0.0008 |
|      41 |         450 |       02:36:18 |       0.6749 |       99.95% |         0.35 |          0.0008 |
|      46 |         500 |       02:53:57 |       0.5593 |       99.97% |         0.33 |          0.0008 |
|      50 |         550 |       03:12:00 |       0.6865 |       99.94% |         0.39 |          0.0008 |
|========================================================================================================|
Training finished: Max epochs completed.
Detector training complete.
*********************************************************************
```

Figura 3.19: Valori dei parametri in output dell'esperimento

La differenza, sostanzialmente sta nel fatto che ognuna di queste due architetture porta a dei vantaggi; infatti, con SSD il livello di precisione è maggiore mentre con YOLO aumenta la velocità di esecuzione.

Oltre a ciò, è possibile effettuare esperimenti sfruttando reti differenti rispetto alla *ResNet 50* che abbiamo utilizzato, come la *ResNet101*, la quale potrebbe catturare caratteristiche più sofisticate, aumentando, però, drasticamente i tempi di esecuzione.

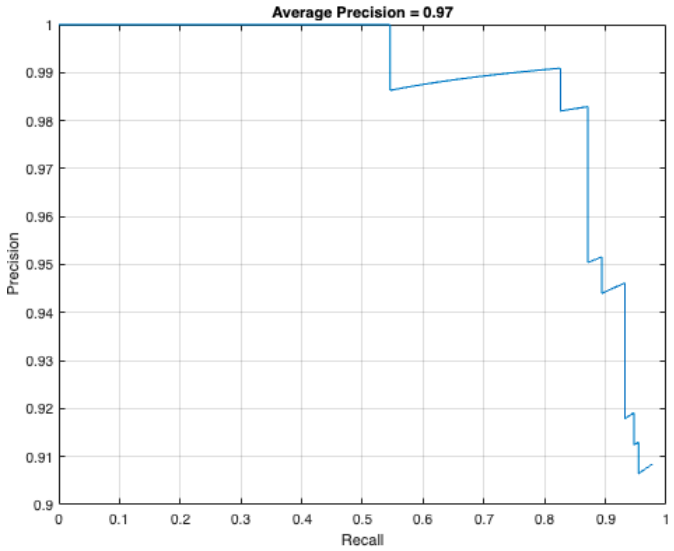

Figura 3.20: Grafico PR relativo all'esperimento

CAPITOLO 4

Analisi dei simulatori di volo

In questo capitolo ci concentreremo sul contesto dell'aviazione, riponendo particolare attenzione sull'analisi di alcuni simulatori, che affrontano diverse tematiche cruciali nell'ambito aeronautico. Esploreremo in dettaglio i risultati di esperimenti condotti con tali simulatori, potendo constatare l'importanza delle tecnologie di apprendimento automatico nell'evoluzione del settore.

4.1 Introduzione generale dei simulatori

Come già evidenziato nel precedente Capitolo 3, anche nel campo aeronautico, la fase di simulazione è di cruciale importanza poichè consente alle tecnologie aeronautiche di innovarsi e di ottimizzarsi continuamente. Nel corso di questo capitolo, focalizzeremo la nostra attenzione su tre simulatori di volo, implementati mediante l'utilizzo dell'ambiente di sviluppo Matlab.

Gli strumenti di simulazione che esamineremo sono incentrati su settori fondamentali del mondo aeronautico, ovvero l'atterraggio, la definizione delle traiettorie in volo e l'individuazione di oggetti a terra in scenari di natura militare. I simulatori che analizzeremo sono i seguenti:

- *Simulate, Detect, and Track Anomalies in a Landing Approach:* esso è progettato per individuare automaticamente deviazioni e anomalie nei velivoli durante le fasi finali di avvicinamento ad una pista di atterraggio dell'aeroporto. L'obiettivo principale è garantire un atterraggio sicuro, monitorando e segnalando eventuali deviazioni dalle regole di atterraggio prestabilite.
- *Simulate and Track En-Route Aircraft in Earth-Centered Scenarios:* esso si concentra sulla modellazione di una traiettoria di volo che si estende per migliaia di chilometri. L'obiettivo di questo simulatore è contribuire al progresso nella definizione delle traiettorie di volo e alla gestione efficiente del traffico aereo.
- *Automatic Target Recognition (ATR) in SAR Images:* ampiamente utilizzato nel mondo dell'aeronautica militare, esso addestra un detector per il riconoscimento automatico di un bersaglio in immagini radar che coprono una vasta area. Il suo obiettivo principale è contribuire alle operazioni di sorveglianza e ricerca.

Questi simulatori, come quelli per la guida autonoma, consentono di effettuare simulazioni in ambienti controllati e virtuali, diminuendo rischi e costi derivabili dalle esperienze reali.

Nelle successive sezioni del capitolo analizzeremo i simulatori elencati.

4.2 Simulate, Detect, and Track Anomalies in a Landing Approach

Questo simulatore si focalizza sulla fase di atterraggio, la quale è una fase critica per la sicurezza del volo. Un aereo in avvicinamento finale deve allinearsi con la pista, scendere gradualmente al suolo e ridurre la velocità mantenendosi al di sopra della velocità di stallo, per garantire che l'aereo tocchi delicatamente il suolo affinchè si riduca il rischio per i passeggeri e si evitino danni fisici all'aereo o alla pista.

Durante l'avvicinamento, i controllori monitorano l'aereo sulla base di sistemi di tracciamento. Di conseguenza, è necessario comunicare in modo automatico e affidabile, ai controllori del traffico aereo, l'eventuale presenza di velivoli che si avvicinano al punto di atterraggio in modo non sicuro, ovvero velivoli non ben allineati con la pista, oppure velivoli che scendono troppo velocemente o troppo lentamente.

Ciò che fa questo simulatore è proprio determinare la correttezza della traiettoria di un aereo in fase di atterraggio. L'aeroporto in questione è l'aeroporto Logan di Boston e la pista soggetta a questo esperimento è la 22L.

Affinché una traiettoria sia sicura, deve soddisfare le seguenti regole:

- essa deve essere strettamente allineata con la direzione della pista;
- la pendenza di planata deve essere compresa tra 2,5 e 4 gradi negli ultimi 20963 metri;
- a distanze superiori a 20963 metri l'altitudine deve essere almeno di 3000 piedi;
- la velocità deve essere compresa tra 120 e 180 nodi al punto di atterraggio.

Il rilevamento di anomalie nella traiettoria in tempo reale basato sui dati rappresenta una sfida continua a causa di due fattori, ovvero:

- i dati di tracciamento, spesso, sono imperfetti e soggetti a rumore;
- i sensori possono segnalare falsi rilevamenti.

In conclusione, quindi, il simulatore ha il compito di determinare la correttezza della traiettoria di atterraggio di un aeromobile, considerando la presenza di rumore nei dati di tracciamento ed evitando falsi allarmi.

Vediamo come viene implementato questo simulatore.

4.2.1 Implementazione del Simulatore 1

Traiettoria di atterraggio

Definiamo una traiettoria di avvicinamento all'atterraggio utilizzando un oggetto "*geoTrajectory*" e visualizziamola grazie all'oggetto" *helperPertScenarioGlobeViewer*". I valori positivi che definiamo per il parametro "'*ClimbRate*'" rallentano l'aereo in avvicinamento fino a raggiungere una velocità sicura per l'atterraggio e determinano una traiettoria discendente.

```
baselineApproachTrajectory = geoTrajectory
([42.7069 -70.8395 1500; 42.5403 -70.9203 950; ...
... 42.3736 -71.001 0],[0;180;400],...
'ClimbRate', [6; 3.75; 3.75]);

viewer = helperPertScenarioGlobeViewer;
viewer.TargetHistoryLength = 0;
viewer.TrackHistoryLength = 0;
```

```
viewer.TrackLabelScale = 0.75;
positionCamera(viewer, [42.3072 -70.8463 12455], [0 -34 335]);
plotTrajectory(viewer,...
... baselineApproachTrajectory, 'Color',...
... [15 255 255]/255, "Width", 1);
```

Definiamo le regole elencate in precedenza nella Sezione 4.2 attraverso "*helperTrajectoryValidationRule*" e visualizziamole sulla mappa come mostrato in Figura 4.1.

```
trajRules = defineTrajectoryRules();
showRules(viewer, trajRules)
snap(viewer)
```

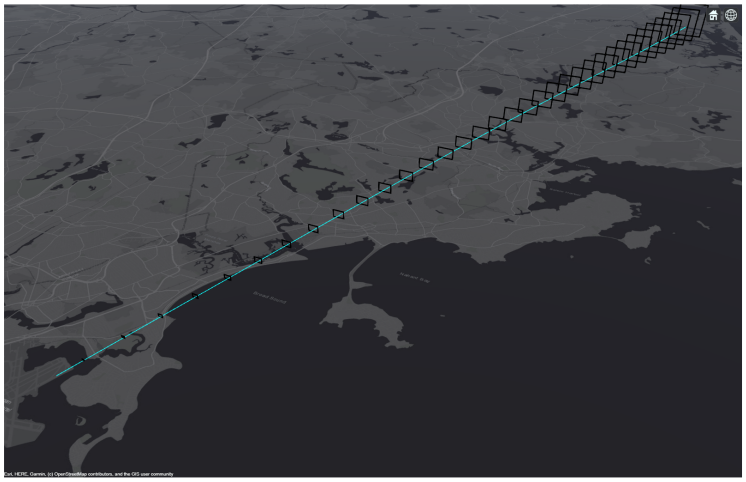

Figura 4.1: Visualizzazione traiettoria corretta

Definiamo variazioni casuali alla traiettoria di volo attorno alla traiettoria di base attraverso l'oggetto "perturbations". Procediamo con la definizione delle traiettorie e successivamente introduciamo delle perturbazioni.

```
s = rng(2021, 'twister'); % Set random noise generator for repeatable results
numTrajectories = 20;
trajectories = cell(1,numTrajectories);
for i = 1:numTrajectories
    trajectories{i} = clone(baselineApproachTrajectory);
    perturb(trajectories{i});
end
```

Per verificare quali traiettorie perturbate soddisfano le regole di un avvicinamento sicuro all'atterraggio utilizziamo la funzione ausiliaria "*validateTrajectory*". La funzione dichiara una traiettoria anomala, dal punto di vista della sicurezza, se almeno l' 1% dei punti campionati da essa non soddisfa una qualsiasi regola della traiettoria. Tracciamo le traiettorie.

```
[truthAnomalyFlags, ...
... truthPercentAnomalousSteps] =
validateTrajectory(trajectories,trajRules);
plotTrajectory(viewer, trajectories(truthAnomalyFlags),
'Color', [255 255 17]/255, "Width", 1);
plotTrajectory(viewer, trajectories(~truthAnomalyFlags),
'Color', [15 255 255]/255, "Width", 1);
positionCamera(viewer, [42.4808 -70.916 1136], [0 0 340]);
snap(viewer)
```

Scenario

Definiamo uno scenario.

```
scenario = trackingScenario('UpdateRate', 1, 'IsEarthCentered', true);
```

Pianifichiamo le traiettorie e colleghiamo ciascuna ad una piattaforma. La differenza minima di sicurezza definita tra due aerei è di un minuto.

```
for i = 1:numTrajectories
    perturbations(trajectories{i}, 'TimeOfArrival', 'Uniform',(i-1)*60, (i-1)*60+10);
    perturbations(trajectories{i}, 'Waypoints', 'None');
    platform(scenario, 'Trajectory', trajectories{i});
end
perturb(scenario);
```

Gli aeroporti utilizzano un'attrezzatura di rilevamento che prende nome di *Model X*. Semplifichiamo questo modello con l'utilizzo di un radar statistico, "*fusionRadarSensor*". Colleghiamo il modello ad un oggetto "*trackerGNN*". Configuriamo il tracker in modo conservativo sulla conferma delle traiettorie "(*ConfirmationThreshold*)". La soglia di conferma è stabilita in modo che una traiettoria sia confermata solo se riceve aggiornamenti positivi per almeno 4 su 5 volte.

```
asdex = fusionRadarSensor(1, ...
    'ScanMode', 'No Scanning', ...
    'MountingAngles', [0 0 0], ...
    'FieldOfView', [360;20], ...
    'UpdateRate', 1, ...
    'ReferenceRange', 40000,...
    'RangeLimits', [0 50000], ...
    'RangeResolution', 100, ...
    'HasElevation', true, ...
    'HasINS', true, ...
    'DetectionCoordinates', 'Scenario', ...
    'FalseAlarmRate', 1e-7, ...
    'ElevationResolution', 0.4, ...
    'AzimuthResolution', 0.4);
p = platform(scenario, 'Position', [42.3606 -71.011 0], 'Sensors', asdex);
tracker = trackerGNN("AssignmentThreshold", [100 2000], "ConfirmationThreshold", [4 5]);
tam = trackAssignmentMetrics('AssignmentThreshold', 100, 'DivergenceThreshold', 200);
```

Rilevazione traiettorie anomale

Rileviamo le traiettorie anomali.

```
positionCamera(viewer, [42.3072 -70.8463 12455], [0 -34 335]);
showRules(viewer, trajRules)
clear validateTracks
while advance(scenario)
    dets = detect(scenario);
    if ~isempty(dets) || isLocked(tracker)
        tracks = tracker(dets, scenario.SimulationTime);
    else
        tracks = objectTrack.empty;
    end
    poses = platformPoses(scenario,"Quaternion","CoordinateSystem","Cartesian");
    tam(tracks,poses);
    [assignedTrackIDs, assignedTruthIDs] = currentAssignment(tam);

    [tracks, trackAnomalyHistory] =
    validateTracks(tracks, trajRules, assignedTrackIDs, assignedTruthIDs);

    updateDisplay(viewer,scenario.SimulationTime,[scenario.Platforms{:}],dets,[],tracks);
    end
```

Verifichiamo che gli avvisi siano stati emessi per le traiettorie corrette.

```
comparisonTable = analyze(trackAnomalyHistory ,truthPercentAnomalousSteps);
disp(comparisonTable)
```

4.2.2 Analisi degli esperimenti per il Simulatore 1

Esperimento 1

L'Esperimento 1 è stato eseguito con una modifica al codice, aumentando il numero di traiettorie da 20 a 25.

Nella Figura 4.2 viene presentato uno screenshot dell'Esperimento 1, in cui è identificato un insieme di punti, ognuno dei quali è contrassegnato da un'annotazione specifica, come, ad esempio, "T710", rappresentanti gli aeromobili in fase di atterraggio. Le traiettorie sono visualizzate con punti gialli per indicare quelle anomale e punti color ciano per le traiettorie corrette. I rettangoli neri delineano la traiettoria considerata sicura, come precedentemente illustrato nella Figura 4.1.

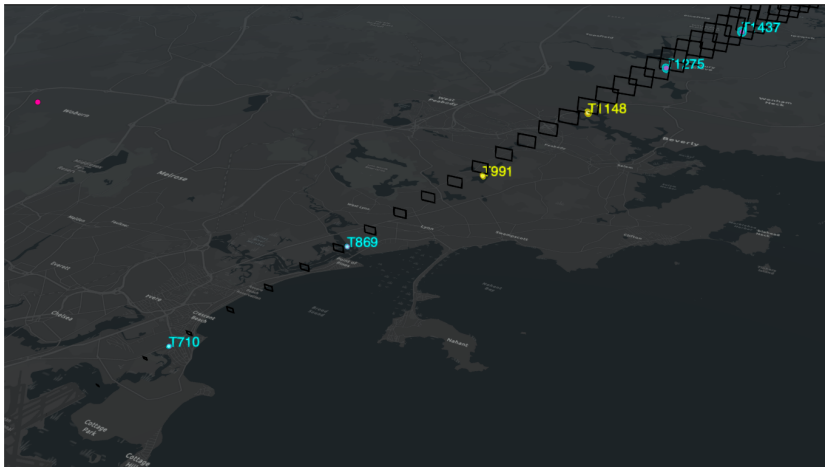

Figura 4.2: Istantanea dell'Esperimento 1

Nella Figura 4.2, è evidente che le traiettorie degli aeromobili contrassegnati come "T991" e "T1148" si collocano al di sotto o in prossimità del rettangolo nero. Questa posizione indica che l'altitudine di entrambe le traiettorie non è conforme agli standard di sicurezza stabiliti per quella specifica distanza dalla pista di atterraggio. Per le restanti traiettorie, è evidente che tutto procede secondo le regole di sicurezza definite.

Ulteriori risultati dell'Esperimento 1 sono visualizzati nella Tabella 4.1. Questa tabella fornisce gli esiti ottenuti dall'analisi delle traiettorie anomale dimostrando se il sistema di rilevamento delle anomalie sia efficiente.

All'interno della tabella troviamo diversi indicatori; analizziamoli:

- *TruthID*: indica in modo univoco ciascuna traiettoria di volo.
- *Truth Anomaly Flag:* indica se la traiettoria è anomala, basandosi sui dati di verità. Se è uguale a "true", significa che la traiettoria è considerata anomala.
- *Track Anomaly Flag:* indica se il sistema ha emesso correttamente un avviso di anomalia durante la simulazione per la traiettoria indicata.

Come possiamo notare dalla tabella, la congruenza tra le due colonne "Truth Anomaly Flag" e "Track Anomaly Flag" indica che le traiettorie anomale previste coincidono con quelle realmente anomale. Ciò denota efficacia da parte del sistema nel corso dell'esperimento. Tuttavia, notiamo che la traiettoria TruthID 9 presenta una discrepanza tra la flag di anomalia nei dati di verità e quella emessa dal sistema; infatti, i dati di verità indicano "false" per questa traiettoria, il sistema ha emesso un avviso di anomalia ("true"). Questo potrebbe

TruthID	Truth Anomaly Flag	Track Anomaly Flag
1	false	false
2	false	false
3	false	false
4	false	false
5	false	false
6	false	false
7	true	true
8	true	true
9	false	true
10	true	true
11	false	false
12	false	false
13	true	true
14	true	true
15	false	false
16	false	false
17	false	false
18	false	false
19	true	true
20	true	true
21	false	false
22	true	true
23	false	false
24	false	false
25	true	true

Tabella 4.1: Tabella dell'analisi delle traiettorie anomale dell'Esperimento 1

essere indicativo di un falso positivo da parte del sistema, che ha segnalato erroneamente un'anomalia in una traiettoria che, secondo i dati di verità, non era anomala.

L'ultimo output che l'esperimento ci fornisce è rappresentato in Figura 4.3, la quale offre un riassunto visivo di tutte le traiettorie analizzate

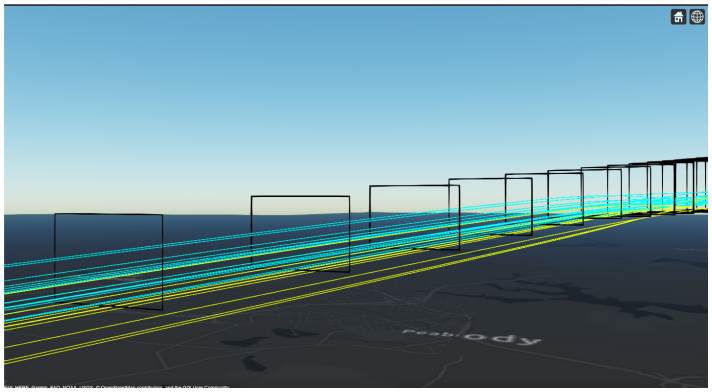

Figura 4.3: Riassunto delle traiettorie analizzate nell'Esperimento 1

Passiamo ad analizzare i risultati dell'Esperimento 2.

Esperimento 2

Per il secondo esperimento, abbiamo apportato alcune modifiche al codice originale.

È stato inserito un quarto waypoint, e di conseguenza, sono stati modificati i parametri "*ClimbRate*" e "*perturbations*". Sono state apportate modifiche al sensore radar "*asdex*" per cercare di avere una maggiore risoluzione e, un maggior dettaglio nelle misurazioni.

```
baselineApproachTrajectory =
geoTrajectory([42.7069 -70.8395 1500;
42.5403 -70.9203 950;
42.3736 -71.001 0; 42.2 -71.05 0],[0;180;400;600],...
'ClimbRate', [6; 3.75; 3.75; 0]);
perturbations(baselineApproachTrajectory,
'Waypoints', 'Normal', zeros(4,3), [0 1e-2 500; 0 2e-3 200; 0 2e-4 0; 0 1e-4 0]);

asdex = fusionRadarSensor(1, ...
'RangeResolution', 50, ...
'ElevationResolution', 0.2, ...
'AzimuthResolution', 0.2)
```

Nella Figura 4.4 viene presentato uno screenshot dell'Esperimento 2, in cui è nuovamente possibile visualizzare le traiettorie corrette e quelle anomale.

Figura 4.4: Istantanea dell'Esperimento 2

Nella Figura 4.4 è possibile notare che le traiettorie degli aeromobili contrassegnati come "T6106", "T7646" e "T10226" siano colorate di giallo e, quindi, segnalate come anomale.

Nella figura 4.5 visualizziamo il riassunto delle traiettorie analizzate nell'esperimento.

Come evidenziato nella Figura 4.5, rispetto all'Esperimento 1, l'attuale dell'esperimento presenta una notevole, quasi totale, prevalenza di traiettorie anomale rispetto a quelle corrette.

Come possiamo notare, le modifiche apportate hanno notevolmente diversificato gli esperimenti; infatti, nel primo si aveva una maggioranza di traiettorie sicure, mentre nel secondo si ha una quasi totalità di traiettorie anomale. Cerchiamo di analizzare come le modifiche potrebbero aver influenzato ciò:

- L'aggiunta di un nuovo waypoint ha alterato la forma e l'altitudine della traiettoria di avvicinamento; questo potrebbe aver influito sulla validità delle traiettorie rispetto alle regole di sicurezza.

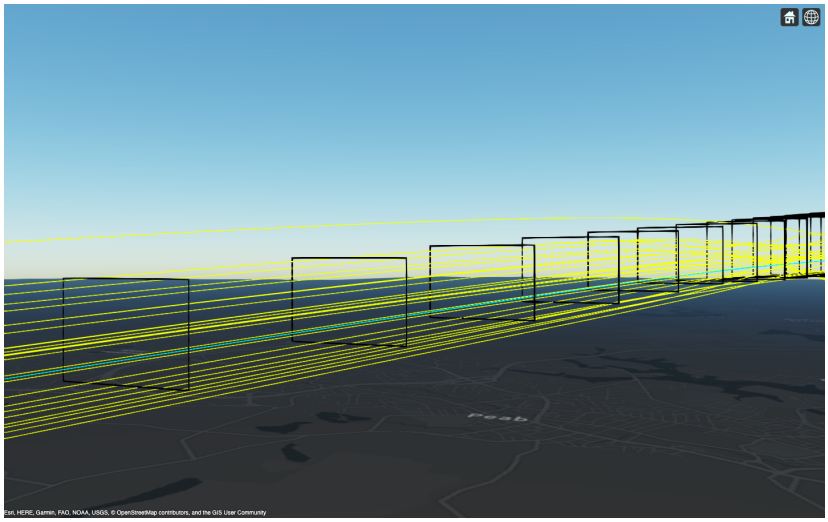

Figura 4.5: Riassunto traiettorie analizzate nell'Esperimento 2

- L'aggiunta del waypoint ha determinato l'aggiunta di un ulteriore livello di perturbazione, il quale potrebbe aver introdotto ulteriore variabilità nella traiettoria.
- Le modifiche del sensore possono aver influito sulla precisione delle rilevazioni e sulla capacità del sistema di tracciare in maniera accurata le traiettorie.
- La modifica apportata all'rng può aver comportato livelli maggiori di perturbazione e di rumore nel sistema.

Passiamo all'analisi del secondo simulatore.

4.3 Simulate and Track En-Route Aircraft in Earth-Centered Scenarios

Questo simulatore mostra come utilizzare uno scenario di tracciamento sulla Terra e un oggetto "*geoTrajectory*" per modellare una traiettoria di volo che si estende su migliaia di chilometri. Sono impiegati due modelli distinti:

- un radar monostatico.
- segnali ADS-B.

Negli Stati Uniti, la Federal Aviation Administration è responsabile della regolamentazione di migliaia di voli ogni giorno. I voli commerciali sono tracciati in ogni momento, dal loro aeroporto di partenza fino all'arrivo. A partire dal 2020, tutti gli aeroplani che volano sopra i 10.000 piedi sono tenuti ad essere equipaggiati con un transponder *Automatic Dependent Surveillance Broadcast (ADS-B)* per trasmettere la loro posizione stimata a bordo. Questo messaggio è ricevuto ed elaborato dai centri di controllo del traffico aereo.

4.3.1 Implementazione per il Simulatore 2

Scenario

Definiamo lo scenario.

```
s = rng;
rng(2020);
scene = trackingScenario('IsEarthCentered',true,'UpdateRate',1);
```

Aereo

Definiamo le caratteristiche e la traiettoria dell'aereo.

```
load('flightwaypoints.mat')
flightRoute = geoTrajectory(lla,timeofarrival);
airplane = platform(scene,'Trajectory',flightRoute);
posAccuracy = 50;
velAccuracy = 10;
gps = gpsSensor('PositionInputFormat','Geodetic',...
'HorizontalPositionAccuracy',posAccuracy,...
'VerticalPositionAccuracy', posAccuracy,'VelocityAccuracy',10);
adsbTx = adsbTransponder('MW2020','UpdateRate', 1,'GPS', gps,'Category',adsbCategory.Large);
load('737rcs.mat');
airplane.Signatures{1} = boeing737rcs;
```

Stazioni di controllo e radar

Creiamo e definiamo dei radar ARSR-4. In seguito, aggiungiamo ogni radar a ciascuna stazione di controllo.

```
% Modella un radar ARSR-4
updaterate = 1/12;
fov = [360;30.001];
elAccuracy = atan2d(0.9,463); % 900m accuratezza
elBiasFraction = 0.1;

arsr4 = fusionRadarSensor(1,'UpdateRate',updaterate,...
    'FieldOfView',[360 ; 30.001],...
    'HasElevation',true,...
    'ScanMode','Mechanical',...
    'MechanicalAzimuthLimits',[0 360],...
    'MechanicalElevationLimits',[-30 0],...
    'HasINS',true,...
    'HasRangeRate',true,...
    'HasFalseAlarms',false,...
    'ReferenceRange',463000,...
    'ReferenceRCS',0,...
    'RangeLimits', [0 463000],...
    'AzimuthResolution',1.4,...
    'AzimuthBiasFraction',0.176/1.4,...
    'ElevationResolution',elAccuracy/elBiasFraction,...
    'ElevationBiasFraction',elBiasFraction,...
    'RangeResolution', 323,...
    'RangeBiasFraction',116/323,... Accuracy / Resolution
    'RangeRateResolution',100,...
    'DetectionCoordinates','Scenario');

% Aggiungi radar ARSR-4 in ciascuna stazione di controllo
% I radar ARSR-4 sono tre come le stazioni di controllo
radarsitesLLA = [41.4228  -88.0583  0;...
    40.6989  -89.8258  0;...
    39.2219  -95.2461  0];

for i=1:3
    radar = clone(arsr4);
    radar.SensorIndex = i;
    platform(scene,'Position',radarsitesLLA(i,:),...
        'Signatures',rcsSignature('Pattern',-50),'Sensors',{radar});
end
```

Definiamo un ricevitore di messaggi ADS-B centrale.

```
    adsbRx = adsbReceiver('ReceiverIndex',2);
    adsbRange = 2e5;
```

Tracker centralizzato

Definiamo un radar centralizzato per tutte le stazioni di controllo. Uniamo le tracce ottenute dal radar con le tracce ADS-B ottenute dal ricevitore ADS-B.

```
radarGNN = trackerGNN('TrackerIndex',1,...
 'MaxNumTracks',50,...
 'FilterInitializationFcn',@initfilter,...
 'ConfirmationThreshold',[3 5],...
 'DeletionThreshold',[5 5],...
 'AssignmentThreshold',[250 2000]);
fuser = trackFuser('FuserIndex',3,'MaxNumSources',2,...
 "AssignmentThreshold",[250 500],...
 "StateFusion",'Intersection',...
 'StateFusionParameters','trace',...
 'ConfirmationThreshold',[2 3*12],...
 'DeletionThreshold',[4 4]*12);
```

A questo punto è possibile tracciare il volo utilizzando il radar e ADS-B.
Per questioni di brevità non riportiamo l'implementazione di questa parte del codice.

4.3.2 Analisi dell'esperimento per il Simulatore 3

Esperimento unico

L'esperimento è stato effettuato senza apportare alcuna modifica al codice standard del simulatore. Analizziamo gli output, grazie ad una sequenza di immagini. In Figura 4.6 è rappresentata la parte iniziale dell'esperimento.

Figura 4.6: Fase 1 dell'esperimento

La linea bianca rappresenta la traiettoria reale e la linea gialla rappresenta la traiettoria stimata dal radar. All'inizio dell'esperimento, l'aereo è lontano dalla stazione di controllo più a sud ed, inoltre, non viene trasmesso alcun messaggio ADS-B. In seguito alla fase di decollo, l'aereo inizia ad essere tracciato solo dal radar. Si noti che i rilevamenti con radar sono imprecisi in altitudine, e questo può rappresentare un problema, in quanto i controllori del traffico aereo lavorano sia orizzontalmente che verticalmente.

Poco dopo, la traccia radar viene passata al Fused, il quale determina una traiettoria mostrata in arancione. La traiettoria generata dal fusore e quella generata dal radar, seppur

si potrebbe pensare che siano le stesse, in realtà sono differenti, poichè il fusore aggiunge il proprio rumore alla traccia.

Nella Figura 4.7 è rappresentata la fase successiva dell'esperimento in cui l'aereo si trova nel raggio di comunicazione ADS-B e viene stabilita una nuova traccia ADS-B, mostrata in fucsia.

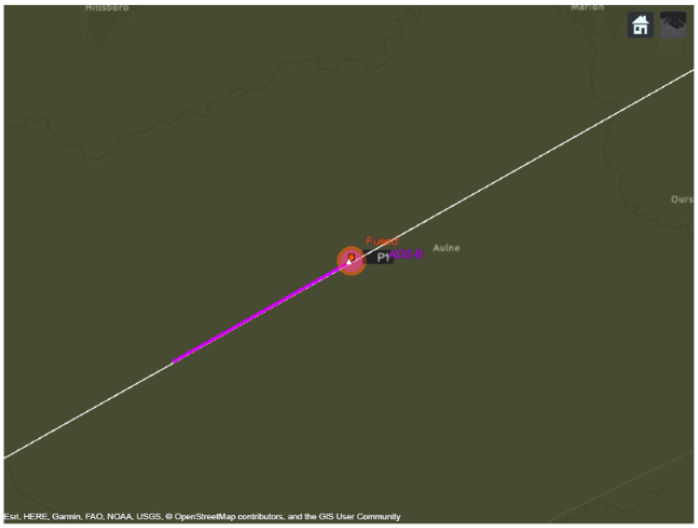

Figura 4.7: Fase 2 dell'esperimento

Nella Figura 4.8 , l'aereo entra nel cono di silenzio. Il localizzatore radar cancella la traccia dopo diversi aggiornamenti senza nuovi rilevamenti. A questo punto il Fused si affida solo alla traccia ADS-B per stimare la posizione dell'aereo.

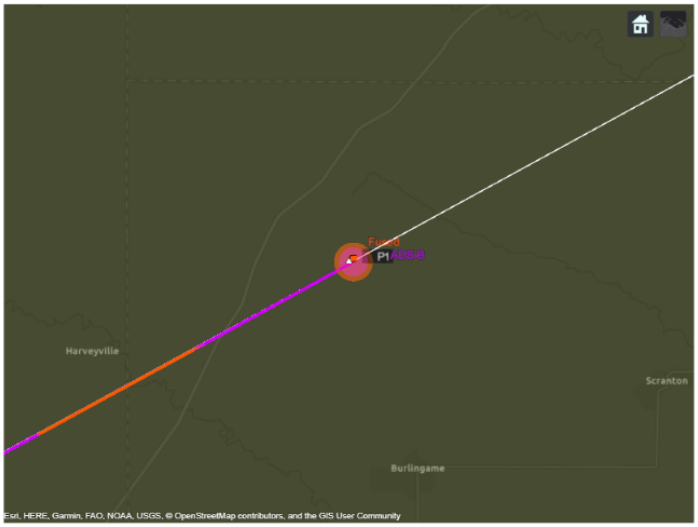

Figura 4.8: Fase 3 dell'esperimento

Nell'ultima fase, mostrata in Figura 4.9, l'aereo entra nell'area coperta dalla seconda e dalla terza stazione radar di controllo e, per tale motivo, viene stabilita una nuova traccia radar. I rilevamenti delle due stazioni radar vengono uniti dal localizzatore radar e il Fused fonde la nuova traccia radar con la traccia ADS-B.

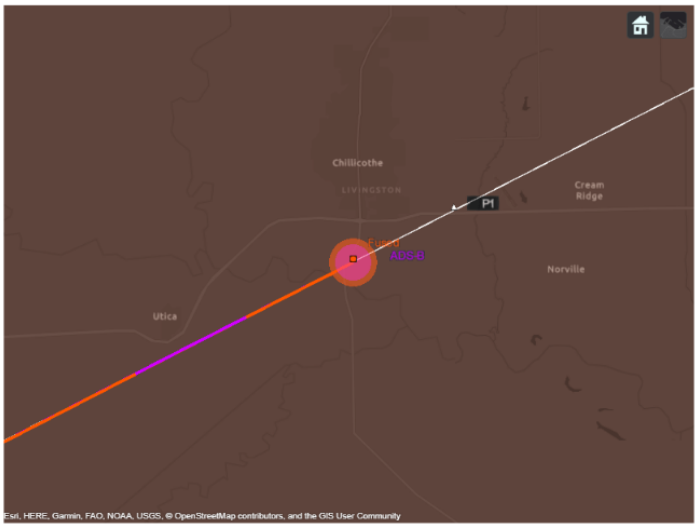

Figura 4.9: Fase 4 dell'esperimento

Nella fase di atterraggio, mostrata in Figura 4.10, possiamo notare nel dettaglio tutte le singole tracce. Per ogni traiettoria specifica, è indicato il suo tipo di generazione ossia se è stata generata da un radar, da un sistema ADS-B, o se è stata ottenuta attraverso la fusione (Fused) di dati provenienti da diverse fonti.

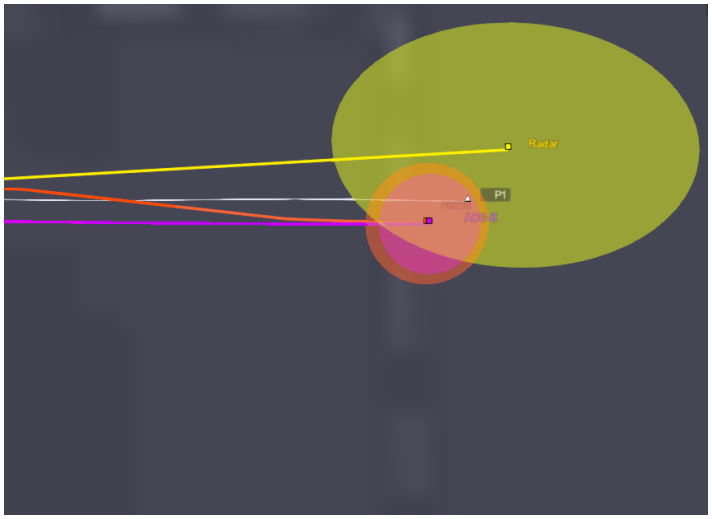

Figura 4.10: Rappresentazione delle traiettorie dell'esperimento

Durante il corso dell'esperimento, è stato fatto riferimento alle torri di controllo. La Figura 4.11 fornisce una visualizzazione della portata e della copertura delle torri di controllo, mostrando come il loro segnale si estenda nello spazio aereo circostante. Le corrispettive aree nella figura sono evidenziate con dei colori leggermente trasparenti e presentano la forma di un cerchio. Durante il corso del test, è stato discusso il concetto di "cono del silenzio" associato alle torri di controllo radar che utilizzano un'antenna a cono cieco. Questo termine si riferisce a una zona di spazio direttamente sopra il radar che non può essere sorvegliata a causa delle limitazioni della scansione dell'antenna. Per affrontare questo problema, spesso si adotta una strategia di sovrapposizione delle aree di copertura tra diverse torri di controllo. Tuttavia, anche con questa strategia, potrebbero persistere delle aree non completamente coperte dalla rete radar.

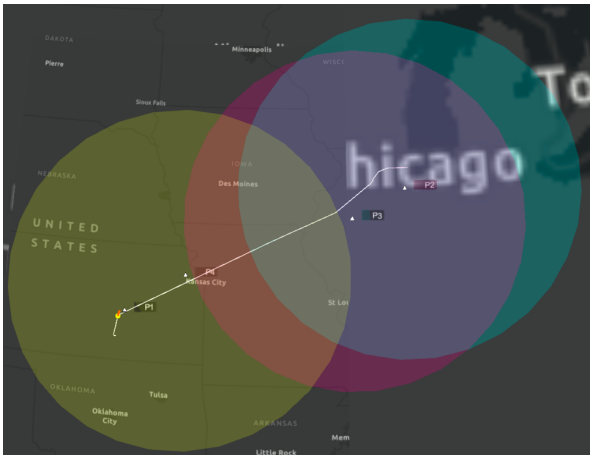

Figura 4.11: Area di copertura delle torri di controllo

Oltre alle immagini analizzate, al termine del test, viene fornito come risultato un grafico che mette in relazione il tempo e l'*OSPA*; tale grafico è mostrato in Figura 4.12.

La metrica OSPA è utilizzata per confrontare la qualità del tracciamento generato solo dal radar, solo dall'ADS-B e dal radar unito con ADS-B.

Nelle fasi iniziali, la metrica mostra due picchi iniziali, causati dalla mancata disponibilità dei segnali ADS-B. Nella fase che va dai minuti 19 a 25, l'OSPA relativo solo alle tracce radar risulta elevato, poiché l'aereo si trova sopra l'angolo cieco della rete radar.

L'introduzione dei segnali ADS-B in questa area porta a un notevole miglioramento delle prestazioni di tracciamento.

Intorno al minuto 40, l'OSPA relativo solo alle tracce ADS-B subisce un degrado a causa della perdita di disponibilità di ADS-B nella regione.

Nelle fasi finale della simulazione, sia segnali radar che ADS-B, sono disponibili. L'OSPA relativo solo alle tracce radar mostra complessivamente prestazioni inferiori rispetto a quello relativo solo alle tracce ADS-B, a causa della minore precisione verticale.

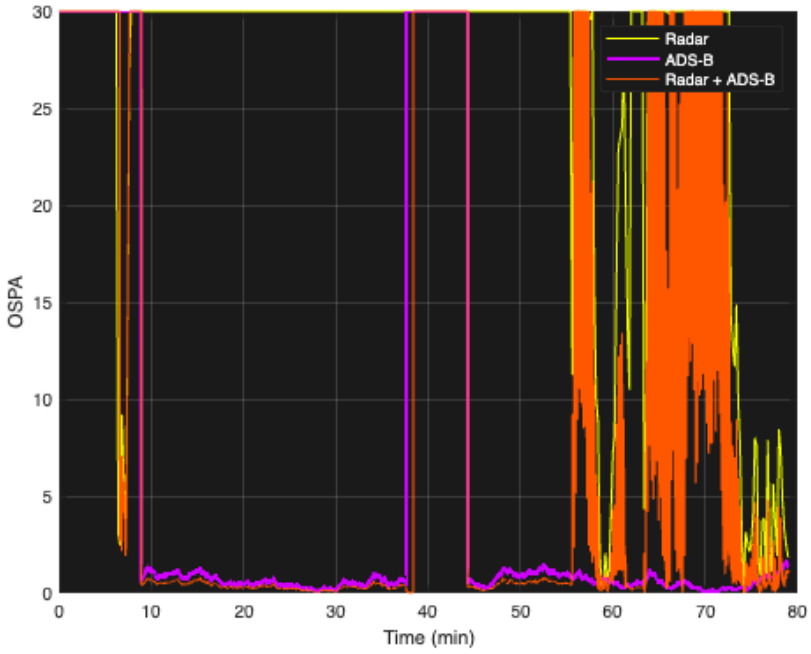

Figura 4.12: Grafico OSPA

4.4 Automatic Target Recognition (ATR) in SAR Images

Questo simulatore mostra come addestrare una rete neurale convoluzionale per il riconoscimento del bersaglio in immagini radar ad apertura sintetica su larga scala dette "*SAR*". L'obiettivo principale del simulatore, quindi, è creare un modello in grado di identificare e delimitare correttamente i bersagli. L'output finale sarà un rilevatore di bersagli addestrato, pronto per l'uso su delle immagini radar. I simulatori di questa tipologia vengono utilizzati per addestrare algoritmi di riconoscimento automatico di bersagli. Per effettuare tale addestramento, il simulatore utilizza il dataset *MSTAR* (Moving and Stationary Target Acquisition and Recognition) pubblicato dall'*Air Force Research Laboratory*. In particolare, questa tipologia di simulatori viene impiegata per svariate funzioni; elenchiamole di seguito:

- *Addestramento:* questi simulatori vengono utilizzati per generare una vasta gamma di immagini; tali immagini contengono bersagli le quali vengono utilizzate come dati di addestramento per gli algoritmi. In questo modo gli algoritmi apprendono a riconoscere i bersagli in svariate condizioni.
- *Valutazione delle prestazioni:* effettuando le simulazioni, è possibile valutare le prestazioni degli algoritmi attraverso dei parametri come, ad esempio, la precisione.
- *Miglioramenti:* le valutazioni permettono, a loro volta, di determinare lo sviluppo di algoritmi migliorati.
- *Integrazione:* gli algoritmi di questo tipo, una volta ottimizzati, possono essere integrati a bordo di aerei e velivoli militari, come testimonia il simulatore di cui parleremo.

I bersagli utilizzati in questo simulatore sono i seguenti:

- *BTR-60:* è un veicolo blindato.
- *BRDM-2:* è un veicolo da combattimento.
- *ZSU-23/4:* è una tipologia di carro armato.
- *T62:* è una tipologia di carro armato.
- *SLICY:* rappresenta una categoria di bersagli militari definiti da forme geometriche semplici e statiche.

Per ragioni di brevità, eviteremo di entrare nei dettagli implementativi del codice e ci concentreremo sull'analisi dei risultati degli esperimenti.

4.4.1 Analisi degli Esperimenti per il Simulatore 3

Nell'esperimento eseguito con il simulatore non avevamo un obiettivo specifico. L'unico requisito previsto, era quello di raggiungere il numero massimo di epoche.

Nella Figura 4.13 è rappresentata una delle molti immagini utilizzate per addestrare il detector.

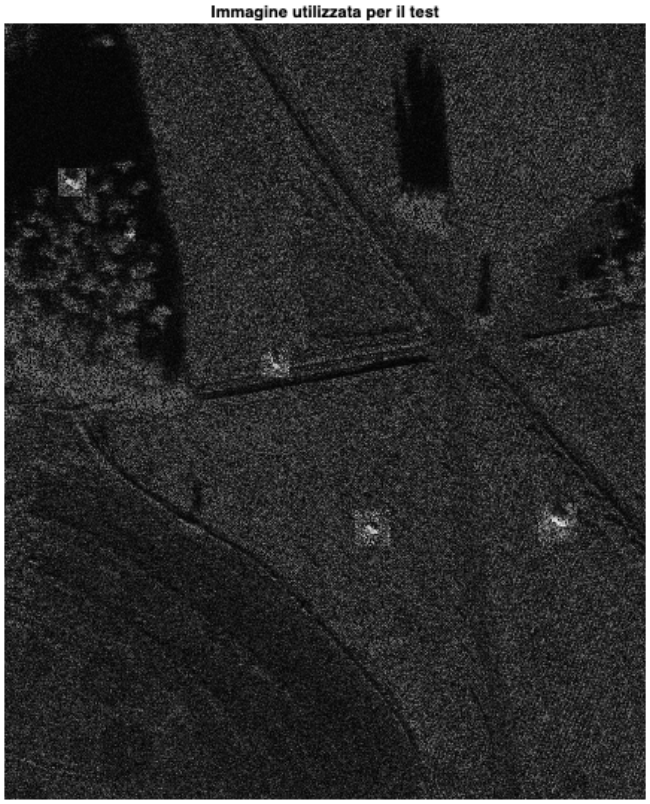

Figura 4.13: Esempio d'immagine utilizzata per l'addestramento

Esperimento unico

In seguito all'esecuzione dell'addestrsmento, analizziamo gli output determinati dal simulatore. I risultati che ci vengono forniti sono i medesimi del simulatore analizzato in Sezione 1.5. In Figura 4.14 viene visualizzato il risultato del test effettuato sul rilevatore di bersagli in seguito all'addestramento. Nell'immagine i bersagli sono stati contraddistinti da un box giallo. Inoltre, a ciascun bersaglio rilevato è associato un valore che indica il grado di confidenza; quest'ultima rappresenta la certezza con cui il detector individua un determinato bersaglio. Ciascun bersaglio presenta un grado di confidenza differente; infatti, i bersagli *BRDM-2* presentano due gradi di certezza del 96% e del 100%, il bersaglio *T62* presenta un grado di confidenza del 100%, il bersaglio *BTR60* presenta un grado di certezza del 100% , il bersaglio *ZSU-23/4* presenta un grado di certezza del 100% e il bersaglio *SLICY* presenta un grado di certezza del 100%.

Anche in questo simulatore, per valutare le prestazioni del rilevatore addestrato, viene utilizzata la metrica chiamata precisione media. Per determinare precisione e recall, consideriamo:

$$Precisione = \frac{T_P}{T_P + F_P}$$

$$Recall = \frac{T_P}{T_P + F_N}$$

dove

- T_P è il numero di veri positivi, che si hanno quando il detector individua un bersaglio quando è presente.

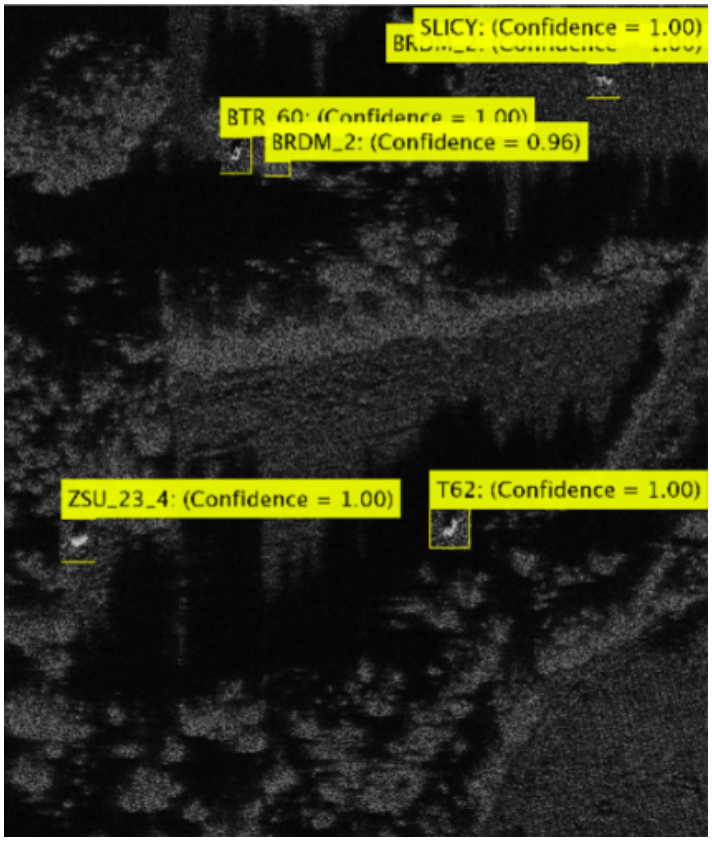

Figura 4.14: Risultato del test dell'esperimento

- F_P è il numero di falsi positivi, che si hanno quando il rilevatore individua un bersaglio quando non è presente.
- F_N è il numero di falsi negativi, che si hanno quando il detector non riesce a rilevare un bersaglio quando è presente.

Se un detector presenta precision pari ad 1 è considerato efficace nel rilevamento dei bersagli. Viceversa, se un detector presenta recall pari ad 1, significa che è in grado di evitare falsi rilevamenti. L'esperimento condotto ha determinato il grafico, mostrato in Figura 4.15. Diamo, in seguito, una descrizione di esso.

Il grafico rappresenta le curve di precisione e recall per ciascun tipo di bersaglio. Ogni curva è identificata da un colore univoco:

- Curva blu per *SLICY*. L'*average precision* pari a 0.7 indica una performance discreta, con margini di miglioramento.
- Curva gialla per *BRDM-2*. Presenta un *average precision* pari a 0.908, che indica una performance complessivamente buona.
- Curva verde per *T62*. Ha un *average precision* pari a 0.98, che è un valore molto alto e indica una performance eccellente.
- Curva viola per *ZSU-23/4*. L'*average precision* di 0.969 è molto alta ed indica un'elevata precisione.
- Curva rossa per *BTR-60*. Presenta una *average precision* di 0.936, suggerendo un buon equilibrio tra recall e precision.

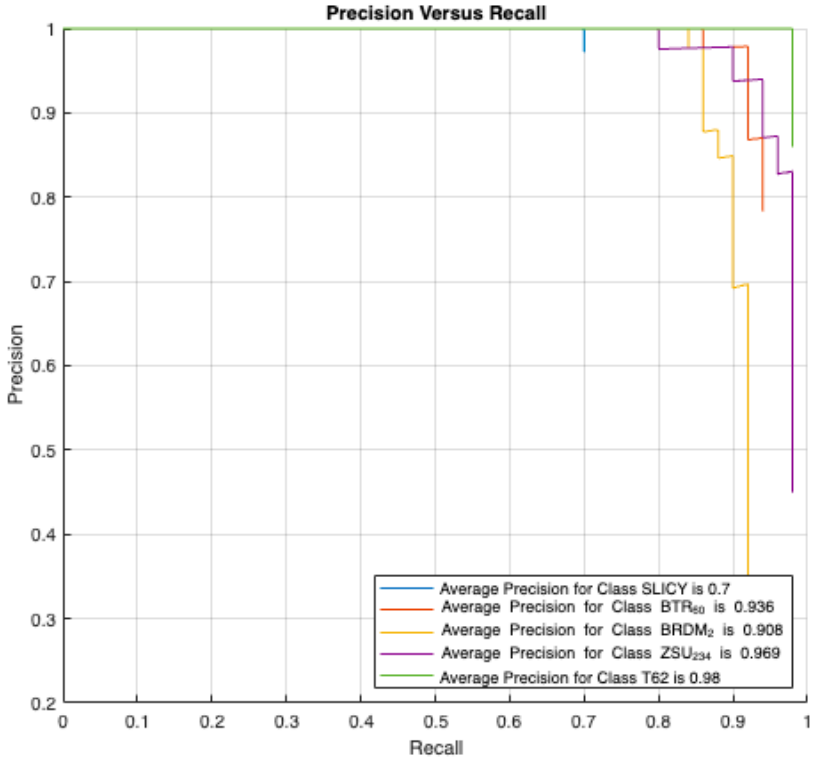

Figura 4.15: Grafico PR relativo all'esperimento

Con l'addestramento vengono generati parametri identici a quelli ottenuti dal simulatore *Object Detection Using SSD Deep Learning*.

Questi parametri sono fondamentali per valutare le prestazioni del modello e monitorare l'avanzamento dell'addestramento.

L'ultimo output, quindi, è una tabella contente valori relativi ad alcuni parametri, come numero di epoche, numero di iterazioni, tempo di addestramento, valori di mini-batch accuracy e valori di Mini-Bacth loss. Per completare l'addestramento sono state necessarie 10 epoche, circa 6300 iterazioni ed una durata temporale di quasi 40 ore.

I valori di Mini-Batch accuracy e loss descrivono l'andamento dell'esperimento.

Per una visualizzazione più gradevole, abbiamo trasformato la tabella in due grafici attraverso l'utilizzo delle libreria di visualizzazione *matplotlib* in Python.

Nel primo grafico in Figura 4.16 è rappresentato l'andamento dei valori relativi al parametro Mini-Batch loss in funzione del numero di iterazioni.

Cerchiamo di fornire una breve descrizione dell'andamento del grafico:

- *0-1000 iterazioni:* nella prima fase, il modello sta ancora apprendendo le basi per il rilevamento dei bersagli. La Mini-Batch loss è elevata e, per questo, il modello commette molti errori.
- *1000-2000 iterazioni:* in questa seconda fase, il modello inizia ad apprendere le caratteristiche dei bersagli. La Mini-Batch loss inizia a diminuire, ma è ancora relativamente elevata.
- *2000-3000 iterazioni:* nella terza fase, la mini batch loss diminuisce ulteriormente.
- *3000-6000 iterazioni:* in questa fase conclusiva, la Mini-Batch loss scende al di sotto di valori pari a 1.5 e si mantiene stabile attorno a questi. Poco prima dell'iterazione 5000 la Mini-Batch loss tocca il valore minimo.

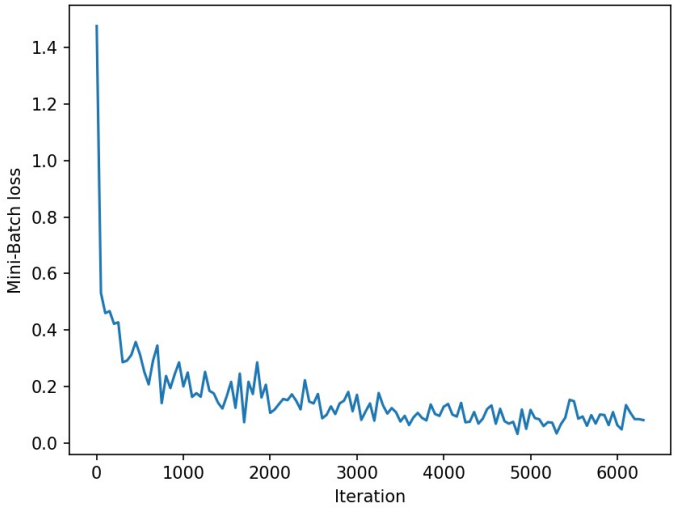

Figura 4.16: Grafico Iteration-Mini-Batch loss

La decrescita dei valori di tale parametro, con l'avanzare dell'addestramento, indica che il modello diviene sempre più preciso.

Il secondo grafico, mostrato in Figura 4.17, invece, descrive l'andamento dei valori relativi al parametro Mini-Batch accuracy in funzione del numero di iterazioni.

Analizzando l'andamento del grafico possiamo notare vari aspetti. Sicuramente, la Mini-Batch accuracy mostra un aumento progressivo nel corso delle iterazioni, indicando che il modello sta migliorando nell'effettuare previsioni corrette.

Nella prima fase, si può notare un miglioramento improvviso nella precisione, fenomeno comune durante l'addestramento. Nella fase finale, la precisione sembra stabilizzarsi o migliorare solo in modo marginale.

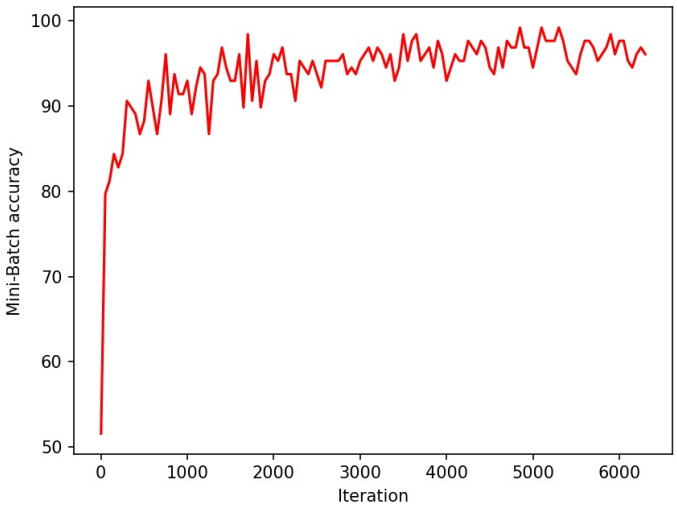

Figura 4.17: Grafico Iteration-Mini-Batch accuracy

Concludiamo affermando che l'aumento dei valori di questo parametro durante l'addestramento indica che il modello sta apprendendo dal dataset e, per questo, sta diventando più preciso nelle sue previsioni.

CAPITOLO 5

Discussioni in merito alle esperienze effettuate

In questo capitolo, discuteremo delle esperienze vissute nel contesto dei simulatori di guida e di volo. Tramite esso vogliamo fornire un'analisi dettagliata dei punti di forza, delle criticità e delle osservazioni personali rilevate in entrambi gli ambiti.

5.1 Discussione dei simulatori di guida

Come già affrontato nel Capitolo 4, il mondo dei simulatori di guida rappresenta un passo fondamentale nell'evoluzione delle tecnologie automobilistiche. In questa sezione, esaminiamo in dettaglio le esperienze vissute con i simulatori di guida, focalizzandoci sui punti di forza e sulle criticità di ognuno di essi.

- *Automatic Parking Valet with Unreal Engine Simulation*

  A differenza di tutti gli altri simulatori di Reinforcement Learning, questo si distingue per essere l'unico ad offrire la possibilità, attraverso una piccola grafica, di visualizzare l'addestramento in modo diretto. Esso non si limita a mostrare soltanto l'andamento dell'addestramento attraverso un grafico ma integra anche un'interessante rappresentazione visiva: in concomitanza con il grafico, viene presentata un'immagine dell'auto che tenta di effettuare la manovra di parcheggio.

  – *Punti di forza*

    Questo simulatore, durante la mia personale esperienza, ha rivelato una buona quantità di vantaggi che derivano dal suo utilizzo.

    * *Reinforcement Learning:* il simulatore è un ambiente di Reinforcement Learning avanzato, progettato per consentire all'agente di apprendere in maniera dinamica dagli episodi di addestramento. Questa caratteristica permette all'agente di adattarsi in tempo reale alle variazioni dell'ambiente, determinando un miglioramento costante delle sue abilità. Essendo un simulatore di Reinforcement Learning permette la visualizzazione diretta dell'addestramento dell'agente attraverso un grafico.
    * *Visualizzazione in Tempo Reale:* come precedentemente menzionato nell'introduzione al simulatore, ritengo che la possibilità di visualizzare l'automobile durante il processo di addestramento aggiunga un elemento visivo che arricchisce notevolmente l'esperienza rendendola molto dinamica e meno noiosa.

Questa caratteristica rende l'interazione con il simulatore più coinvolgente, offrendo una comprensione visiva diretta di come l'automobile affronta le sfide legate al parcheggio. Attraverso questo elemento visivo, è stato possibile ottenere una comprensione più approfondita dei valori che poi venivano rappresentati sul grafico relativo all'addestramento. Ad esempio, se l'auto entrava correttamente nel parcheggio ma commetteva errori nella fase conclusiva, ciò determinava un punteggio di episode reward elevato. D'altra parte, se l'auto non riusciva neppure ad entrare nel parcheggio vuoto, collidendo con un'altra auto parcheggiata, si verificava un punteggio più basso di episode reward. Questa associazione diretta tra l'azione dell'auto durante la simulazione e i valori di reward ha contribuito a migliorare l'esperienza e renderla più comprensibile.

  * *Possibilità di modifiche*: il simulatore ha agevolato la mia esperienza, consentendomi di apportare modifiche al codice, in particolare alle reti neurali, in modo incredibilmente semplice. Questa flessibilità mi ha permesso di condurre esperimenti diversificati, in modo tale da poter effettuare dei confronti tra i risultati ottenuti dai vari esperimenti. È importante sottolineare che tale facilità di modifica del codice non è stata riscontrata in tutti i simulatori che ho precedentemente utilizzato.

- *Criticità*
  * *Durata degli esperimenti:* a causa delle limitazioni di un hardware non particolarmente potente, le simulazioni hanno richiesto un tempo di elaborazione considerevole, stimato in circa 10 ore per completare entrambi gli esperimenti. Questa prolungata durata delle simulazioni ha rappresentato una sfida.
  * *Limite di modifiche:* è corretto menzionare il limite riscontrato nella possibilità di modificare l'ambiente in cui l'agente agisce. Ad esempio, infatti, nonostante la mia volontà, non è stato possibile apportare modifiche a livello di condizioni meteorologiche o di altri parametri ambientali. Tutto ciò, dal mio punto di vista, rappresenta una limitazione nella possibilità di testare l'agente in scenari più diversificati.

- *Train RL Agent for Lane Keeping Assist with Constraint Enforcement*

  Questo simulatore, durante gli esperimenti effettuati con esso, ha mostrato lati positivi ma anche lati negativi; analizziamoli.

  - *Criticità*
    * *Durata degli esperimenti:* anche in questo caso gli esperimenti hanno richiesto una quantità di tempo molto elevata.
    * *Mancanza di alcuni output:* in seguito alla conclusione dell'esperienza con tale simulatore, ho individuato un ulteriore simulatore pressocchè identico che produceva una quantità di output maggiori. Per ragioni di brevità questo simulatore non è stato inserito.
    * *Mancanza di visulizzazione in tempo reale:* diversamente dal primo simulatore, questo non offre la possibilità di visualizzare l'addestramento tramite una piccola grafica; infatti, si limita a fornire come output soltanto il grafico relativo all'addestramento. Questa mancanza di una rappresentazione visiva più dinamica ha reso l'esperienza meno coinvolgente e più difficile da interpretare.
  - *Vantaggi*

    * *Reinforcement Learning:* analogamente al primo simulatore, esso presenta un vantaggio relativo all'esperienza di utilizzo, in quanto permette di visualizzare i risultati dell'addestramento dell'agente in tempo reale attraverso un grafico. Questa caratteristica introduce un risultato visivo che coinvolge molto, poichè permette agli utenti di monitorare in tempo reale il comportamento dell'agente, le ricompense ottenute e le previsioni relative alle sue future decisioni. La visualizzazione dei risultati prodotti dall'addestramento rende l'esperienza più interattiva e fornisce anche una comprensione diretta delle decisioni che l'agente effettua nel corso del tempo per mantenere il veicolo all'interno della corsia.
    * *Possibilità di modifiche:* questo simulatore si è rivelato essere flessibile nella possibilità di apportare una notevole quantità di modifiche al codice, non solo alla struttura della rete neurale, ma anche ai parametri del veicolo. In particolare, ho avuto la possibilità di personalizzare i dati in input, introducendo del rumore gaussiano. Ciò mi ha permesso di addestrare l'agente in condizioni più realistiche e sfidanti consentendomi di analizzare scenari in cui non è riuscito a raggiungere l'obiettivo. Questa variazione ha consentito di determinare un esperimento "diverso" dagli altri.

- *Train DDPG Agent for Adaptive Cruise Control*

  Dal mio punto di vista, questo simulatore è stato estremamente coinvolgente durante l'esperienza di utilizzo. Una sua caratteristica notevole è rappresentata dai tre grafici in output, che forniscono una visione dettagliata del comportamento dell'agente addestrato.

  La presenza di questi grafici permette di comprendere in modo approfondito come l'agente regola la velocità dell'auto ego per evitare il crash con l'auto leader in funzione della distanza di sicurezza. Questa rappresentazione visiva contribuisce a una comprensione più chiara e intuitiva delle decisioni adottate dall'agente.

  – *Punti di forza*
    * *Reinforcement Learning:* analogamente ai primi due simulatori, anche questo offre la possibilità di visualizzare l'avanzamento dell'addestramento dell'agente attraverso un grafico. Questa funzionalità rappresenta uno strumento prezioso per valutare in tempo reale le prestazioni e l'evoluzione delle capacità dell'agente con il passare degli episodi.
    * *Visualizzazione degli strati di rete:* a differenza degli altri simulatori, questo, come descritto nella sua implementazione, offre la possibilità, tramite il codice di default, di visualizzare dettagliatamente i vari strati che compongono la rete neurale e illustra chiaramente come questi sono interconnessi tra loro.
    * *Possibilità di effettuare modifiche:* similmente agli altri, il simulatore ha dimostrato una notevole facilità nel apportare modifiche alle caratteristiche delle automobili e ai parametri delle reti neurali, permettendomi di eseguire due esperimenti da confrontare.
    * *Simulazione:* come precedentemente menzionato nella parte introduttiva, questo simulatore offre la possibilità di simulare l'agente addestrato, consentendo la visualizzazione dei parametri relativi all'accelerazione, alla velocità iniziale e obiettivo dell'auto ego, alla velocità dell'auto leader, alla distanza relativa e di sicurezza all'interno di tre grafici dedicati. Attraverso questa simulazione, è possibile osservare come i valori di questi parametri cambiano all'interno dei grafici in risposta a varie condizioni a cui è sottoposto l'agente.

    * *Velocità di addestramento:* con questo simulatore è stato possibile condurre gli esperimenti in un periodo di tempo leggermente inferiore rispetto ai simulatori correlati alla guida autonoma.
  - *Criticità*
    * *Mancanza di visulizzazione in tempo reale:* anche questo, nella sezione dedicata all'addestramento, non include l'implementazione di alcuna visualizzazione dinamica. Avrei gradito che esso offrisse una visualizzazione più interattiva, ad esempio mostrando le due auto che si seguono a distanza relativa e che, eventualmente, potrebbero collidere, determinando così il termine dell'episodio.

- *Object Detection Using SSD Deep Learning*

  Diversamente dagli altri simulatori, questo è orientato verso il Deep Learning, piuttosto che verso il Reinforcement Learning.

  - *Punti di forza*
    * *Differenza con gli altri simulatori:* come evidenziato nell'introduzione, il simulatore in questione è l'unico tra i simulatori di guida autonoma che si basa sul Deep Learning piuttosto che sul Reinforcement Learning. Questa caratteristica ha rappresentato un vantaggio significativo nella mia esperienza, poichè, utilizzando questo simulatore, ho avuto l'opportunità di acquisire dimestichezza con concetti chiave del mondo del Deep Learning, come le epoche e altri parametri, che fino a quel momento non avevo incontrato. Esso mi ha permesso, quindi, di diversificare la mia esperienza.
    * *Possibilità di modifiche:* esso mi ha offerto la possibilità di personalizzare l'esperienza consentendomi di aggiungere una foto, individuata sul browser, per effettuare la simulazione del detector addestrato. La possibilità di aggiungere tale foto per la simulazione ha reso l'esperienza con questo simulatore più coinvolgente e meno statica. Tutto ciò mi ha consentito di analizzare come il detector risponde a immagini differenti da quelle standard, aggiungendo un tocco di realismo.
    * *Output nuovi:* a differenza degli altri simulatori dedicati alla guida autonoma, questo ha prodotto nuovi output mai incontrati precedentemente. Il primo si è presentato sotto forma di una tabella contenente parametri strettamente connessi al mondo del Deep Learning, che ho cercato di interpretare. Tale output si presenta come un elemento distintivo, poiché, nei simulatori precedenti, non ho mai ottenuto in output informazioni di questo genere. Il secondo si è presentato sotto forma di grafico relativo alla precisione e al recall. Questo output fornisce una valutazione generale e dettagliata delle capacità del detector addestrato, contribuendo a rendere più comprensibile l'esperienza.
  - *Criticità*
    * *Black box:* questo non ha fornito la possibilità di visualizzare in alcun modo l'addestramento del detector, ad esempio attraverso grafici come i precedenti simulatori. Essendo un simulatore orientato al Deep Learning, l'addestramento del detector è considerato come una "black box". Questo aspetto rappresenta una limitazione nella comprensione dell'esperienza.
    * *Difficoltà nelle modifiche:* durante la mia esperienza con il simulatore, ho cercato invano di apportare delle modifiche al codice originale, ed in particolare alla rete neurale utilizzata. Ho cercato di modificare la rete ResNet-50 di base con

una rete ResNet-101. Nonostante avessi rispettato tutti i passaggi per apportare tale modifica, ho riscontrato degli errori nel codice che, personalmente, non sembravano essere dovuti a problemi di implementazione da parte mia. Questo ha reso più complessa l'esperienza con il simulatore.

* *Difficoltà di esecuzione:* oltre alla rete neurale ho cercato di modificare l'architettura, con l'obiettivo di passare da *SSD* a *YOLOv2* o *YOLOv4*. Tuttavia, a causa delle limitate prestazioni dell'hardware, ho riscontrato problemi di crash ripetuti durante questi tentativi, poichè architetture più complesse potrebbero richiedere risorse di calcolo più elevate. A causa di queste difficoltà incontrate, non sono completamente soddisfatto dell'esperienza con questo simulatore.
* *Più epoche del previsto:* per ottenere risultati più accurati durante l'addestramento del detector, ho dovuto aumentare il numero di epoche. In questo modo, sono riuscito a risolvere il problema in cui attorno a un singolo veicolo venivano generate più bounding box e label del necessario.
* *Rilevazione limitata:* l'ultimo svantaggio di questo simulatore è dovuto al fatto che è progettato per identificare solo veicoli. Di conseguenza, nella mia immagine finale, né un ciclista né i segnali stradali venivano individuati. Dal mio punto di vista, un'esperienza più completa si sarebbe avuta se il detector fosse stato più completo.

5.2 Discussione dei simulatori di volo

In questa sezione, come fatto nella Sezione 5.1, esaminiamo in dettaglio le esperienze vissute con i simulatori di volo, focalizzandoci sui vantaggi e sulle sfide di ognuno di essi.

- *Simulate, Detect, and Track Anomalies in a Landing Approach*

  Questo simulatore è stato quello che ha suscitato in me un coinvolgimento particolarmente profondo. Attraverso tale esperienza, sono stato in grado di acquisire una comprensione più approfondita di alcune regole fondamentali nelle fasi di atterraggio.

  – *Punti di forza*
    * *Concetto di traiettoria corretta:* come precedentemente accennato nell'introduzione, tramite questo simulatore ho compreso al meglio il concetto di traiettoria di un aereo durante la fase di atterraggio.
    * *Visualizzazione in tempo reale:* l'esperienza avuta con questo simulatore ha suscitato il mio interesse in modo significativo grazie ad una stimolante esperienza visiva. Attraverso una grafica, il simulatore offriva una visualizzazione in tempo reale degli aerei durante l'avvicinamento alla pista per la fase di atterraggio. In modo chiaro, venivano segnalate in tempo reale le traiettorie anomale e quelle sicure.
    * *Tabella relativa alle traiettorie:* il simulatore offre come output una tabella in cui vengono evidenziate le traiettorie segnalate come anomale, verificandone l'attendibilità, e quelle segnalate come corrette. La fusione di tale tabella alla rappresentazione grafica precedentemente citata rende l'esperienza facilmente comprensibile.
    * *Possibilità di effettuare modifiche:* anche con questo simulatore, ho avuto l'opportunità di apportare modifiche al codice di base. In particolare, ho potuto inserire un nuovo waypoint e una nuova perturbazione, dando vita a un esperimento totalmente diverso dal primo effettuato. Ciò ha contribuito a differenziare e arricchire ulteriormente l'esperienza complessiva.

  - *Svantaggi*
    * *Mancanza di informazioni:* dal mio punto di vista, sarebbe stata un'aggiunta significativa se all'interno della rappresentazione grafica fossero stati inclusi dati come la velocità e l'altitudine del veivolo. Queste informazioni avrebbero reso maggiormente comprensibile il motivo per cui una traiettoria veniva considerata anomala.

- *Simulate and Track En-Route Aircraft in Earth-Centered Scenarios*

  Questo simulatore ha contribuito a un'esperienza positiva fornendomi l'opportunità di acquisire conoscenze sull'individuazione e la definizione della traccia di un velivolo.

  - *Criticità*
    * *Difficoltà di interazione con la mappa:* come svantaggio, è da notare che l'output del simulatore consisteva in una mappa che rappresentava un velivolo in movimento; tuttavia, l'interazione con questa mappa risultava fortemente problematica poiché era difficile capire come zoomare o spostarsi. Inoltre, la visualizzazione della mappa andava a scatti e presentava una buona quantità di bug, rendendo l'esperienza complessiva meno intuitiva e poco user-friendly.
    * *Limitazione delle modifiche:* a differenza di altri simulatori, quest'ultimo non mi ha offerto la possibilità di apportare modifiche significative al codice, permettendomi di determinare un esperimento unico che non ho potuto confrontare.
  - *Punti di forza*
    * *Output:* un punto di forza del simulatore è rappresentato dai molteplici output che esso fornisce. Il simulatore mi ha fornito informazioni dettagliate sulle traiettorie generate da radar, ADS-B e la combinazione di entrambi, consentendomi di approfondire la comprensione di esse. Successivamente, ho potuto visualizzare, grazie a delle opportune finestre, le differenze tra le traiettorie derivate da queste fonti, e confrontarle con la traiettoria reale.
      In aggiunta, il simulatore mi ha concesso l'opportunità di esaminare, comprendere e visualizzare in modo dettagliato il funzionamento delle stazioni di controllo. L'attenzione è stata posta sulle aree geografiche che queste stazioni coprono e sul concetto fondamentale del cono del silenzio.
    * *Grafico OSPA:* attraverso l'ultimo output, il grafico OSPA, ho potuto analizzare e confrontare le traiettorie generate dalle diverse fonti (radar, radar + ADS-B, ADS-B). Grazie ad un attenta analisi ho potuto valutare le prestazioni dei diversi sensori identificando la precisione di ciascuno nella determinazione delle traiettorie.
    * *Velocità di simulazione:* a differenza di altri simulatori, l'esecuzione di questo è stata davvero veloce, dal momento che la sua durata è stata di circa un'ora.

- *Automatic Target Recognition (ATR) in SAR Images*

  Questo simulatore è molto simile al simulatore di guida autonoma *Object Detection Using SSD Deep Learning*, dal momento che anche esso si concentra sulla rilevazione di oggetti.

  - *Punti di forza*

    * *Grafico precision-recall:* come nel simulatore di guida autonoma, anche in questo caso viene prodotto come output il grafico relativo alla precisione media. Nella descrizione del simulatore, vengono fornite le formule con cui viene ottenuto il grafico PR.
    * *Possibilità di generare grafici:* un output del simulatore, espresso attraverso una tabella contenente valori di specifici parametri, ha costituito la base per la creazione di due grafici. Questi ultimi sono stati fondamentali per l'analisi e la valutazione dell'andamento dell'esperimento. La rappresentazione grafica dei dati mi ha fornito la possibilità di comprendere in maniera più semplice le variazioni dei parametri.
    * *Rilevamento Multiclasse:* mentre il simulatore di guida autonoma citato nell'introduzione si concentra principalmente sulla rilevazione di automobili, questo si distingue per la sua capacità di individuare una vasta gamma di bersagli appartenenti a diverse categorie. Ciò lo caratterizza rendendolo un rilevatore multiclasse.

- *Criticità*
    * *Dataset pesante:* il dataset impiegato per l'addestramento è grande, con una dimensione di 1.6 GB. Questa considerevole dimensione indica la presenza di un gran numero di dati, fornendo al modello un'ampia varietà di esempi da cui apprendere. Il peso del dataset, da una parte, contribuisce a garantire che il modello possa sviluppare una comprensione robusta dei dati, ma, d'altro canto, incide sulla velocità di esecuzione.
    * *Tempo di esecuzione:* il simulatore in questione ha richiesto un notevole impiego di tempo per l'esecuzione, richiedendo approssimativamente 40 ore.
    * *Black box:* in questo simulatore, il processo di addestramento si svolge in modo simile ad una "black box", in cui i dettagli interni non sono accessibili all'utente.

Conclusioni

La presente tesi è iniziata con una panoramica dettagliata sull'Intelligenza Artificiale (IA), fornendo una visione teorica e pratica di questa tecnologia. Inizialmente, è stata presentata un'ampia introduzione sull'IA che ci ha consentito di evidenziare cosa sia realmente tale disciplina. Successivamente è stata descritta la storia dell'IA, offrendo una cronologia accurata degli eventi. In seguito, sono stati esplorati a fondo i vantaggi e gli svantaggi dell'IA, e sono stati descritti i vari campi in cui essa trova applicazione, riponendo particolare attenzione al settore automobilistico e aeronautico. Nel contesto automobilistico, è stato analizzato come l'IA stia trasformando la guida con l'introduzione di veicoli autonomi

Dopo aver descritto gli aspetti fondamentali dell'Intelligenza Artificiale, la tesi è proseguita introducendo due pilastri essenziali, ovvero il Deep Learning e il Reinforcement Learning. In particolare, nella tesi è stata fornita una descrizione, sono state definite le origini storiche, sono stati esaminati i vantaggi e gli svantaggi, e sono stati elencati i campi di applicazione per entrambe le discipline.

Nella sezione dedicata al Deep Learning, abbiamo esaminato in maniera dettagliata i neuroni artificiali e le reti neurali, descrivendo le diverse tipologie di reti.

Nella sezione dedicata al Reinforcement Learning, la tesi ha descritto in dettaglio i componenti chiave di questo approccio all'apprendimento automatico, con un particolare focus sul Markov Decision Process, sul valore di ritorno, sulla funzione di ritorno e sulle Equazioni di Bellman. Sempre nella sezione del Reinforcement Learning, abbiamo effettuato l'analisi dei differenti algoritmi, suddividendoli in due categorie, ovvero algoritmi model-based e algoritmi model-free.

Successivamente a questa parte iniziale prettamente teorica, la tesi ha assunto un approccio pratico. Infatti, nei capitoli successivi, l'elaborato ha integrato esperienze pratiche, riportando esperimenti condotti con simulatori di guida autonoma e di volo con Intelligenza Artificiale. Questi esperimenti sono stati descritti e analizzati nel dettaglio. Ciascun simulatore impiegato nella tesi era dedicato a una fase cruciale del rispettivo settore. In particolare, nel settore della guida autonoma si è utilizzato un simulatore specializzato sul parcheggio, uno sul mantenimento di corsia, uno sul cruise control e uno sul rilevamento di veicoli; nel settore aeronautico, è stato utilizzato un simulatore per il rilevamento di anomalie in fase di atterraggio, uno per il tracciamento del volo, ed, infine, uno per il rilevamento di bersagli a terra nell'ambito dell'aeronautica militare.

Infine, nella parte conclusiva, è stata condotta una discussione dettagliata sulle esperienze personali con i simulatori, evidenziando i principali punti di forza e le principali criticità di ognuno di essi.

In seguito alle esperienze pratiche condotte con i simulatori di guida autonoma e di volo basati sull'Intelligenza Artificiale, si aprono intriganti prospettive per ulteriori sviluppi futuri. Per quanto riguarda i simulatori relativi al mondo della guida autonoma, tra gli sviluppi futuri si potrebbero determinare agenti sempre più avanzati che presentano aspetti psicologici e decisionali sempre più vicini a quelli dell'uomo, in modo da prevenire in maniera efficace le intenzioni di un altro autista. Un ulteriore sviluppo in questo ambito potrebbe essere dato da un'interazione del simulatore con veicoli guidati da esseri umani, per valutare la sicurezza e l'efficienza dei veicoli autonomi in contesti misti.

Inoltre, l'introduzione di una simulazione più complessa definita da scenari con una maggiore varietà di interazioni veicolari e pedonali, unitamente a elementi dinamici, come temporali, nebbia e cambiamenti climatici, costrituirebbe uno sviluppo futuro, in quanto permetterebbe di creare agenti molto più sviluppati e pronti ad affrontare situazioni più realistiche.

Nel contesto dei simulatori relativi al mondo dell'aviazione, per determinare uno sviluppo verso il futuro, si potrebbero definire scenari di emergenza, come guasti ai motori, situazioni di perdita di quota e condizioni meteorologiche estreme, per migliorare la preparazione degli algoritmi di volo. Inoltre, sarebbe altamente improntata verso il futuro, la definizione di scenari in cui velivoli riescono a stabilire una comunicazione, permettendo loro di coordinare in maniera autonoma il traffico aereo. L'implementazione di un sistema di comunicazione diretta tra gli aerei potrebbe rappresentare un passo fondamentale per migliorare la gestione del traffico aereo. In conclusione, sono complessivamente soddisfatto dei risultati e delle esperienze acquisite attraverso l'utilizzo dei simulatori di guida autonoma e di volo. Tuttavia, come evidenziato, si potrebbe aspirare a simulatori più avanzati.

Bibliografia

CANTÙ, P. e TESTA, I. (2012), «Algoritmi e argomenti. La sfida dell'intelligenza artificiale», *Sistemi intelligenti*, vol. 24 (3), p. 395–414.

CHANDAN, G., JAIN, A., JAIN, H. e MOHANA (2018), «Real Time Object Detection and Tracking Using Deep Learning and OpenCV», in «2018 International Conference on Inventive Research in Computing Applications (ICIRCA)», p. 1305–1308.

FETZER, J. H. e FETZER, J. H. (1990), *What is Artificial Intelligence?*, Springer.

HAO, X., ZHANG, G. e MA, S. (2016), «Deep learning», *International Journal of Semantic Computing*, vol. 10 (03), p. 417–439.

KAELBLING, L. P., LITTMAN, M. L. e MOORE, A. W. (1996), «Reinforcement learning: A survey», *Journal of artificial intelligence research*, vol. 4, p. 237–285.

KECHAGIAS-STAMATIS, O. e AOUF, N. (2021), «Automatic Target Recognition on Synthetic Aperture Radar Imagery: A Survey», *IEEE Aerospace and Electronic Systems Magazine*, vol. 36 (3), p. 56–81.

KIRAN, B. R., SOBH, I., TALPAERT, V., MANNION, P., SALLAB, A. A. A., YOGAMANI, S. e PÉREZ, P. (2022), «Deep Reinforcement Learning for Autonomous Driving: A Survey», *IEEE Transactions on Intelligent Transportation Systems*, vol. 23 (6), p. 4909–4926.

KROGH, A. (2008), «What are artificial neural networks?», *Nature biotechnology*, vol. 26 (2), p. 195–197.

KULIDA, E. e LEBEDEV, V. (2020), «About the Use of Artificial Intelligence Methods in Aviation», in «2020 13th International Conference "Management of large-scale system development" (MLSD)», p. 1–5.

LI, Y. (2017), «Deep reinforcement learning: An overview», *arXiv preprint arXiv:1701.07274*.

PENG, J., FAN, Y., YIN, G. e JIANG, R. (2023), «Collaborative Optimization of Energy Management Strategy and Adaptive Cruise Control Based on Deep Reinforcement Learning», *IEEE Transactions on Transportation Electrification*, vol. 9 (1), p. 34–44.

SANCHEZ, B., DURAN, D., MARTINEZ, K. e VIERA, W. (2023), «Advances of Artificial Intelligence in Aeronautics», *Athenea Engineering sciences journal*, vol. 4 (12), p. 34–42.

TAGLIAVINI, A., FERRARO, D., KLODA, T., BURGIO, P. e OTHERS (2020), «An automatic scenario generator for validation of automated valet parking systems», in «VEHITS 2020-Proceedings of the 6th International Conference on Vehicle Technology and Intelligent Transport Systems», p. 489–496, SciTePress.

Siti web consultati

- IBM, What is artificial intelligence (AI)? – `https://www.ibm.com/topics/artificial-intelligence`
- Agenda Digitale, Intelligenza artificiale: cos'è, normative e applicazioni reali – `https://www.agendadigitale.eu/tag/intelligenza-artificiale/`
- Treccani, Intelligenza artificiale – `https://www.treccani.it/enciclopedia/intelligenza-artificiale_(Enciclopedia-della-Scienza-e-della-Tecnica)/`
- NetApp, CHE COS'È L'INTELLIGENZA ARTIFICIALE? – `https://www.netapp.com/it/artificial-intelligence/what-is-artificial-intelligence/`
- Council of Europe, History of Artificial Intelligence – `https://www.coe.int/en/web/artificial-intelligence/history-of-ai`
- Osservatori, Storia dell'Intelligenza Artificiale: da Turing ai giorni nostri – `https://blog.osservatori.net/it_it/storia-intelligenza-artificiale`
- Tech4Future, Il valore aggiunto dell'AI nel futuro del trasporto aereo – `https://tech4future.info/intelligenza-artificiale-aeronautica/`
- EuropAir, Il potere dell'intelligenza artificiale nell'aviazione – `https://www.europair.com/it/blog/il-potere-dellintelligenza-artificiale-nellaviazione`
- MondoAviazione, L'intelligenza artificiale per il controllo del traffico aereo – `https://mondoaviazione.com/2023/06/29/lintelligenza-artificiale-per-il-controllo-del-traffico-aereo/`
- Airbus, Artificial intelligence – `https://www.airbus.com/en/innovation/industry-4-0/artificial-intelligence`
- CORCOM, Auto a guida autonoma, intelligenza artificiale croce e delizia: piattaforme ancora acerbe –`https://www.corrierecomunicazioni.it/telco/tim-vivendi-non-molla-su-netco-vale-30-miliardi-vogliamo-dire-la-nostra-in-assemblea/`
- Economyup, Auto a guida autonoma: che cosa sono, come funzionano, le applicazioni in Italia – `https://www.economyup.it/automotive/auto-a-guida-autonoma-che-cosa-sono-come-funzionano-le-applicazioni-in-italia/`

- AI4Business, Auto a guida autonoma: il radar migliora con l'AI – https://tech4future.info/intelligenza-artificiale-aeronautica/
- Nvidia, DRIVE Chauffeur per veicoli autonomi – https://www.nvidia.com/it-it/self-driving-cars/drive-chauffeur/
- Nvidia, DRIVE Concierge – https://www.nvidia.com/en-us/self-driving-cars/drive-concierge/
- Nvidia, Introducing NVIDIA DRIVE Hyperion 9: Next-Generation Platform for Software-Defined Autonomous Vehicle Fleets – https://blogs.nvidia.com/blog/drive-hyperion-9-thor/
- Nvidia, Hardware for Self-Driving Vehicles – https://www.nvidia.com/en-us/self-driving-cars/hardware/
- IBM, Cos'è il deep learning? – https://www.ibm.com/it-it/topics/deep-learning
- MatLab, Deep Learning – https://it.mathworks.com/discovery/deep-learning.html
- AWS, Cos'è l'apprendimento approfondito? – https://aws.amazon.com/it/what-is/deep-learning/
- IBM, What are neural networks? – https://www.ibm.com/topics/neural-networks
- IBM, What are recurrent neural networks? – https://www.ibm.com/topics/recurrent-neural-networks
- IBM, What are convolutional neural networks? – https://www.ibm.com/topics/convolutional-neural-networks
- Medium, Le 4 funzioni di attivazione neuronale più usate negli Artificial Neural Network – https://medium.com/@paolopiacenti1/le-4-funzioni-di-attivazione-neuronale-pi%C3%B9-usate-negli-artificial-neural-network-41096953b85c
- MatLab, Che cos'è il Reinforcement Learning? – https://it.mathworks.com/discovery/reinforcement-learning.html
- IBM, What is reinforcement learning? – https://developer.ibm.com/learningpaths/get-started-automated-ai-for-decision-making-api/what-is-automated-ai-for-decision-making
- Artificial Intelligence Stack Exchange, What's the difference between model-free and model-based reinforcement learning? – https://ai.stackexchange.com/questions/4456/whats-the-difference-between-model-free-and-model-based-reinforcement-learning