

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

---



**TESI DI LAUREA**

**Esperienze di utilizzo del Reinforcement Learning nel contesto  
dell'Ingegneria dell'Automazione**

**Experiences of using Reinforcement Learning in the context of  
Automation Engineering**

Relatore

Prof. Domenico Ursino

Candidato

Alessandro Binci

---

**ANNO ACCADEMICO 2022-2023**

*Una cosa funziona  
solo se ci credi.*

## Sommario

In questa tesi si esamina l'applicazione del Reinforcement Learning (RL) in contesti industriali e robotici, con un focus particolare sulle simulazioni in ambiente MATLAB. L'indagine inizia con un'introduzione ai principi fondamentali del RL e alle sue intersezioni con il Machine Learning (ML), esplorando la storia, le metodologie e le implicazioni etiche del campo. Proseguendo, il lavoro si addentra nelle applicazioni pratiche del RL nell'Industria 4.0, dimostrando il suo potenziale nelle Smart Factory e nelle tecnologie abilitanti. Attraverso tre casi di studio (Robot Arm, Automatic Parking Valet e Walking Robot), la tesi dettaglia l'implementazione di algoritmi avanzati, come il Soft Actor Critic (SAC), il Deep Deterministic Policy Gradient (DDPG) ed il Twin-Delayed DDPG (TD3), svelando le sfide e le strategie di addestramento degli agenti intelligenti. Il documento si conclude con una discussione critica sui risultati delle simulazioni e una riflessione sui futuri sviluppi del RL, delineando il suo ruolo emergente nell'evoluzione delle industrie intelligenti e della robotica autonoma.

**Keyword:** Machine Learning, Intelligenza Artificiale, Reinforcement Learning, RL, Industria 4.0, Agente, Ambiente, Ricompensa, SAC, TD3, DDPG.

<b>Introduzione</b>	<b>1</b>
<b>1 Il Reinforcement Learning</b>	<b>4</b>
1.1 Introduzione al Reinforcement Learning . . . . .	4
1.1.1 Machine Learning . . . . .	4
1.1.2 Dati e pattern . . . . .	5
1.1.3 Tipologie di apprendimento . . . . .	5
1.1.4 Analisi delle prestazioni . . . . .	6
1.1.5 Storia del Reinforcement Learning . . . . .	7
1.2 Funzionamento del Reinforcement Learning . . . . .	7
1.2.1 Gli elementi del Reinforcement Learning . . . . .	9
1.2.2 Markov Decision Process . . . . .	10
1.3 Gli algoritmi di Reinforcement Learning . . . . .	11
1.3.1 Metodi Tabular Action-Value . . . . .	11
1.3.2 Algoritmo Q-Learning . . . . .	15
1.4 Applicazioni e casi d'uso del Reinforcement Learning . . . . .	17
1.5 Problemi e sfide del Reinforcement Learning . . . . .	18
1.6 Etica e questioni sociali del Reinforcement Learning . . . . .	20
1.7 Conclusioni . . . . .	21
<b>2 Industria 4.0</b>	<b>22</b>
2.1 Introduzione all'Industria 4.0 . . . . .	22
2.1.1 Il concetto di Smart Factory . . . . .	23
2.1.2 Tecnologie Abilitanti . . . . .	24
2.2 Intersezione tra Reinforcement Learning e Industry 4.0 . . . . .	25
2.3 Applicazione dei metodi di RL nell'Industria 4.0 . . . . .	26
2.4 Casi di successo . . . . .	27
2.5 Conclusioni . . . . .	29
<b>3 Simulazione RobotArm in ambiente MatLab</b>	<b>30</b>
3.1 Introduzione all'esperienza . . . . .	30
3.2 Soft Actor Critic (SAC): un'analisi approfondita . . . . .	32
3.2.1 Approssimatori di funzioni attore e critico . . . . .	33
3.2.2 Algoritmo di addestramento . . . . .	34
3.2.3 Metodi di aggiornamento del target . . . . .	35

---

3.3	Progettazione e configurazione del simulatore . . . . .	36
3.3.1	Definizione dell'ambiente . . . . .	36
3.3.2	Creazione dell'agente . . . . .	37
3.3.3	Addestramento dell'agente . . . . .	40
3.4	Valutazione e analisi delle esperienze di simulazione 1 . . . . .	43
3.4.1	Esperienza 1 . . . . .	43
3.4.2	Esperienza 2 . . . . .	46
3.4.3	Esperienza 3 . . . . .	48
3.5	Conclusioni . . . . .	50
<b>4</b>	<b>Simulazione Automatic Parking Valet in ambiente MatLab</b>	<b>52</b>
4.1	Introduzione all'esperienza . . . . .	52
4.2	Twin-Delayed Deep Deterministic: un'analisi approfondita . . . . .	54
4.2.1	Approssimatori di funzioni attore e critico . . . . .	56
4.2.2	Algoritmi di addestramento . . . . .	56
4.2.3	Metodi di aggiornamento del target . . . . .	57
4.3	Progettazione e configurazione del simulatore . . . . .	58
4.3.1	Creazione del parcheggio . . . . .	58
4.3.2	Moduli dei sensori . . . . .	59
4.3.3	Creazione dell'ambiente . . . . .	60
4.3.4	Creazione dell'agente . . . . .	63
4.3.5	Addestramento dell'agente . . . . .	65
4.4	Valutazione e analisi delle esperienze di simulazione 2 . . . . .	66
4.4.1	Esperienza 1 . . . . .	66
4.4.2	Esperienza 2 . . . . .	68
4.5	Conclusioni . . . . .	71
<b>5</b>	<b>Simulazione Train Biped Robot to Walk in ambiente MatLab</b>	<b>73</b>
5.1	Introduzione all'esperienza . . . . .	73
5.2	Deep Deterministic Policy Gradient: un'analisi approfondita . . . . .	76
5.2.1	Approssimatori di funzioni attore e critico . . . . .	77
5.2.2	Algoritmi di addestramento . . . . .	77
5.2.3	Metodi di aggiornamento del target . . . . .	78
5.3	Progettazione e configurazione del simulatore . . . . .	79
5.3.1	Creazione dell'ambiente . . . . .	79
5.3.2	Selezione e creazione dell'agente per l'addestramento . . . . .	79
5.3.3	Specifiche delle opzioni di addestramento e addestramento dell'agente	80
5.3.4	Confronto delle prestazioni degli agenti . . . . .	82
5.4	Valutazione e analisi delle esperienze di simulazione 3 . . . . .	84
5.4.1	Esperienza 1 . . . . .	84
5.4.2	Esperienza 2 . . . . .	86
5.5	Conclusioni . . . . .	88
<b>6</b>	<b>Discussione</b>	<b>89</b>
6.1	Discussione della simulazione con Robot Arm . . . . .	89
6.2	Discussione della simulazione con Automatic Parking Valet . . . . .	91
6.3	Discussione della simulazione con Walking Robot . . . . .	93
<b>Conclusioni</b>		<b>95</b>
<b>Appendice</b>		<b>97</b>

<b>A RobotArm Functions</b>	<b>97</b>
<b>Appendice</b>	<b>99</b>
<b>B Walking Robot Agent Functions</b>	<b>99</b>
<b>Bibliografia</b>	<b>101</b>
<b>Ringraziamenti</b>	<b>104</b>

---

## Elenco delle figure

---

1.1	Schema di funzionamento tra ambiente e agente nel Reinforcement Learning	8
1.2	Esempio di <i>finite MDP</i> con tre stati $\{S_0, S_1, S_2\}$ e due azioni $\{a_0, a_1\}$ . . . . .	12
1.3	La funzione valore e la policy interagiscono fino al raggiungimento della ottimalità e consistenza reciproca . . . . .	13
1.4	Diffrenza tra metodi DP, MC e TD . . . . .	16
1.5	Esempio in pseudocodice del funzionamento del Q-Learning . . . . .	17
2.1	Schematizzazione delle diverse rivoluzioni industriali . . . . .	23
2.2	Utilizzo del RL in base alla classe "principio" . . . . .	27
2.3	Utilizzo del RL in base alla classe "settore industriale" . . . . .	27
3.1	Rappresentazione del braccio robotico, piatto e pallina . . . . .	31
3.2	Modello Simulink del sistema collegato con il blocco agente RL . . . . .	31
3.3	Sottosistema Kinova Ball Balance . . . . .	32
3.4	Funzionamento della generazione di azioni per un agente SAC . . . . .	34
3.5	Rete neurale del critico . . . . .	38
3.6	Traiettoria della palla tramite il Ball Position scope block . . . . .	42
3.7	Animazione MATLAB della pallina sul piatto . . . . .	43
3.8	Campionamento dell'Esperienza 1 del simulatore Robot-Arm dopo 2864 episodi	44
3.9	Risultato finale dell'Esperienza 1 per il simulatore Robot-Arm . . . . .	45
3.10	Risultato finale dell'Esperienza 2 simulatore Robot-Arm . . . . .	48
3.11	Risultato finale dell'Esperienza 3 simulatore Robot-Arm . . . . .	49
4.1	Automobile nell'ambiente di simulazione MatLab e Simulink . . . . .	53
4.2	Modello Simulink del sistema dell'Auto Parking Valet . . . . .	53
4.3	Sottosistema Vehicle Mode per gestire i controllori . . . . .	54
4.4	Rappresentazione dell'ambiente di parcheggio del simulatore . . . . .	59
4.5	Campo visivo della telecamera montata sul veicolo . . . . .	60
4.6	Schema visivo del funzionamento del sensore lidar . . . . .	60
4.7	Ambiente per l'addestramento dell'agente . . . . .	61
4.8	Ambiente finale durante la simulazione . . . . .	62
4.9	Esempio di un addestramento andato a buon fine . . . . .	66
4.10	Traiettoria schematizzata dell'ego-vehicle . . . . .	67
4.11	Risultato finale dell'Esperienza 1 per il simulatore Automatic Parking Valet .	68
4.12	Risultato finale dell'Esperienza 2 per il simulatore Automatic Parking Valet .	70

5.1	Rappresentazione del robot nel magazzino . . . . .	74
5.2	Modello Simulink del sistema del Walking Robot . . . . .	74
5.3	Modello Simscape™ Multibody™ del walking robot . . . . .	75
5.4	Visuale dell'ambiente del robot bipede . . . . .	82
5.5	Confronto tra le curve di apprendimento . . . . .	83
5.6	Confronto tra le curve dell'Episode Q0 . . . . .	83
5.7	Risultato finale dell'Esperienza 1 per il simulatore Walking Biped Robot . . .	85
5.8	Risultato finale dell'Esperienza 2 per il simulatore Walking Biped Robot . . .	87

---

## Elenco delle tabelle

---

3.1	Tipo di ambiente nel quale un agente SAC può essere addestrato . . . . .	32
3.2	Tipo di attore e critico utilizzati dagli agenti SAC . . . . .	33
4.1	Tipo di ambiente nel quale un agente TD3 può essere addestrato . . . . .	55
4.2	Tipo di attore e critico utilizzati dagli agenti TD3 . . . . .	55
5.1	Tipo di ambiente nel quale un agente DDPG può essere addestrato . . . . .	76
5.2	Tipo di attore e critico utilizzati dagli agenti DDPG . . . . .	76

---

## Introduzione

---

Negli ultimi anni, concetti come Intelligenza Artificiale (IA) e Machine Learning (ML) sono diventati protagonisti nei settori lavorativi e accademici, riflettendo il loro crescente impatto sulla società moderna. Il Reinforcement Learning (RL), o apprendimento per rinforzo, in particolare, emerge come la nuova frontiera che unisce e si spinge oltre questi ambiti, fungendo da ponte tra la comprensione teorica dell'Intelligenza Artificiale e le sue applicazioni pratiche.

Possiamo definire l'IA come la tecnologia di base che consente di simulare i processi dell'intelligenza umana attraverso la creazione e l'applicazione di algoritmi integrati in un ambiente di calcolo dinamico, e possiamo trovarne sempre più applicazioni in ogni settore, dato il suo enorme sviluppo nell'ultimo decennio. Mentre, per quanto riguarda il ML, l'obiettivo principale è quello di fornire ai calcolatori l'abilità di apprendere automaticamente un comportamento sulla base di informazioni ed esempi, senza essere programmati esplicitamente per svolgere un determinato compito.

Il Reinforcement Learning si insedia a pieno titolo in questo scenario come il tessuto connettivo tra IA e ML. Il RL, rappresenta un problema di learning riguardante l'apprendimento automatico delle dinamiche di un ambiente. Più precisamente, gli algoritmi (o metodi) del Reinforcement Learning cercano di determinare come un agente debba scegliere le azioni da eseguire, dato lo stato corrente dell'ambiente nel quale è situato, con l'obiettivo di massimizzare una ricompensa totale. Gli algoritmi di RL, quindi, non si limitano a predire o classificare, ma prendono decisioni sequenziali, apprendendo attraverso l'esperienza come navigare in ambienti complessi e dinamici.

L'importanza del Reinforcement Learning si estende ben oltre le frontiere della teoria e della ricerca, andando a influenzare direttamente i progressi tecnologici e operativi in vari settori. Il RL rappresenta una svolta negli ambiti dell'Intelligenza Artificiale e del Machine Learning, poiché consente agli agenti di apprendere da un processo di trial and error, simile all'apprendimento umano, migliorando la loro performance attraverso l'interazione diretta con l'ambiente circostante. Questo metodo di apprendimento si distingue per la sua capacità di affrontare problemi complessi in cui le decisioni ottimali dipendono dalle conseguenze a lungo termine delle azioni intraprese, rendendolo ideale per applicazioni che richiedono una pianificazione strategica e una presa di decisioni autonome.

In contesti industriali, il RL può portare a significativi miglioramenti nell'efficienza operativa, nella riduzione dei costi e nell'aumento della sicurezza. Nella robotica, esso consente lo sviluppo di sistemi autonomi che possono adattarsi a situazioni impreviste, permette di migliorare l'interazione con gli umani e di operare in ambienti pericolosi o inaccessibili.

La pertinenza di uno studio approfondito sul RL, come quello presentato in questa tesi, è, dunque, evidente. In un'era in cui l'autonomia e l'adattabilità sono proprietà sempre più preziose, il Reinforcement Learning offre un quadro per lo sviluppo di agenti intelligenti capaci di prendere decisioni informate e migliorare il proprio comportamento nel tempo senza l'intervento umano. Questa tesi, con i suoi casi di studio applicati, illustra come il RL possa essere sfruttato per risolvere problemi non solo teorici ma anche pratici, fornendo soluzioni innovative in ambiti come la robotica e l'automazione industriale. La ricerca sul RL non è solo un esercizio accademico ma una necessità pratica per far fronte alle richieste di un mercato in rapido cambiamento, in cui flessibilità e intelligenza dei sistemi sono indispensabili per mantenere la competitività e per affrontare le sfide del domani.

Questa tesi si propone di trattare inizialmente, da un punto di vista puramente teorico, tutto ciò che concerne il RL, grazie ad un'analisi approfondita dei fondamenti matematici e algoritmici che sottostanno a quest'ultimo, partendo dalla sua storia ed evoluzione fino a toccare i più recenti sviluppi in questo campo. Verrà, poi, discusso l'argomento da una visione dell'Industria 4.0 con un occhio di riguardo all'impatto che il Reinforcement Learning può avere all'interno del mondo lavorativo contemporaneo, evidenziando come possa contribuire all'ottimizzazione dei processi, all'incremento dell'efficienza produttiva e alla riduzione degli sprechi.

Successivamente, si passa ad una fase sperimentale, incentrata sull'applicazione pratica del RL in tre diversi ambiti. A tal fine, è stato scelto il software MATLAB, dal quale sono stati presi dei modelli già programmati e messi disposizione dal software stesso per poi essere modificati. Tale scelta è stata dettata dalla presenza di un toolbox dedicato al RL, che fornisce strumenti avanzati per la modellazione, la simulazione e l'analisi di algoritmi di apprendimento per rinforzo, agevolando notevolmente sia lo sviluppo che la valutazione delle prestazioni degli agenti intelligenti.

Il primo simulatore riguarda un braccio robotico, il cui compito è quello di apprendere a mantenere in equilibrio una sfera su un piano, un problema classico che mette alla prova la capacità di un agente di eseguire compiti di controllo fine in un ambiente dinamico. Il secondo simulatore tratta l'addestramento di un veicolo autonomo a parcheggiare senza l'intervento umano in uno spazio limitato, sfidando il sistema a navigare con precisione e a prendere decisioni in tempo reale in un ambiente complesso e con vincoli stringenti. Il terzo esperimento si concentra su un robot bipede che, attraverso l'impiego di due differenti algoritmi di RL, verrà addestrato a camminare in modo indipendente, affrontando le sfide legate all'equilibrio e al movimento coordinato che tale compito comporta.

Ciascuna di queste esperienze è stata supportata da una solida base teorica e ha richiesto un attento lavoro di sintonizzazione e ottimizzazione degli algoritmi. Gli esperimenti sono stati strutturati in modo da testare non solo l'efficacia degli algoritmi di RL, ma anche la loro scalabilità e adattabilità a diversi tipi di problemi e ambienti. In conclusione, questa tesi non solo fornisce un contributo significativo alla comprensione teorica del Reinforcement Learning, ma estende anche la sua applicabilità pratica, dimostrando concretamente il potenziale di questa tecnologia all'avanguardia.

La presente tesi è composta da sette capitoli strutturati come di seguito specificato:

- Nel Capitolo 1 sarà introdotto il concetto di Reinforcement Learning, delineando la sua evoluzione storica e il suo contesto all'interno del più ampio campo del Machine Learning. In particolare, si esplorano le basi teoriche, come i processi decisionali di Markov, e si discute delle applicazioni e delle sfide etiche del RL.
- Nel Capitolo 2 si presenterà l'impatto del RL nell'ambito dell'Industria 4.0, esaminando il concetto di Smart Factory e come il RL possa migliorare le tecnologie abilitanti. Si analizzano, inoltre, casi di successo e si valuta l'integrazione del RL nella manutenzione predittiva e nell'ottimizzazione dei processi.

- Il Capitolo 3 si concentra sulla simulazione di un braccio robotico in MATLAB, utilizzando il metodo Soft Actor Critic (SAC) per insegnare al robot a bilanciare una palla. In particolare, si descrivono gli algoritmi, la creazione del simulatore e l'addestramento dell'agente.
- Nel Capitolo 4 si presenta la simulazione di un sistema di parcheggio automatico con il Twin-Delayed Deep Deterministic Policy Gradient (TD3). In particolare, si approfondiscono i metodi di addestramento e le sfide nello sviluppo di un agente in grado di eseguire manovre di parcheggio complesse.
- Nel Capitolo 5 viene descritto lo sviluppo di un robot bipede che apprende a camminare autonomamente attraverso l'applicazione di due diversi algoritmi di RL, il Deep Deterministic Policy Gradient (DDPG) ed il TD3, esaminando le prestazioni e i metodi di ottimizzazione degli agenti.
- Il Capitolo 6 fornisce una discussione critica dei risultati ottenuti nelle diverse simulazioni, valutando l'efficacia degli algoritmi di RL e proponendo riflessioni sulle implicazioni pratiche e teoriche degli esperimenti condotti.
- Nel Capitolo 7 verranno tratte le conclusioni e verranno delineati alcuni possibili sviluppi futuri.

# CAPITOLO 1

---

## Il Reinforcement Learning

---

*In questo capitolo si parlerà del Reinforcement Learning (RL) sotto tutti i suoi aspetti, passando da una trattazione più generale fino ad una più specifica, per capire tutto ciò che lo riguarda. Verrà introdotto inizialmente il Machine Learning per arrivare al concetto di Reinforcement Learning e al suo funzionamento. Essendo definito come un problema di learning che riguarda un apprendimento automatico, verranno anche presi in considerazione i diversi algoritmi che permettono di risolvere tale problema e anche la storia e le diverse applicazioni di esso al giorno d'oggi. Una volta fatto ciò, verranno inoltre trattate tematiche quali i problemi, le sfide, l'etica e il futuro di questa tecnica.*

### 1.1 Introduzione al Reinforcement Learning

In questo capitolo, verrà analizzato il Reinforcement Learning, delineando dapprima i principi fondamentali del Machine Learning, evidenziando la natura orientata ai dati dei vari approcci. Successivamente, si procederà con una classificazione più generale basata sulla modalità di apprendimento. I paragrafi successivi introducono il Reinforcement Learning, posizionandolo come una categoria distinta all'interno del panorama dell'apprendimento automatico, e ne illustrano gli aspetti salienti, tra cui il processo decisionale di Markov, il quale esprime formalmente l'entità *ambiente* per il Reinforcement Learning. Fatto ciò, si tratterà interamente l'argomento sotto gli aspetti evidenziati nel preambolo.

#### 1.1.1 Machine Learning

Il Machine Learning (ML) è un settore della Computer Science che mira a conferire ai calcolatori la capacità di apprendere senza una programmazione esplicita per compiti specifici. Esso esplora la progettazione di algoritmi in grado di apprendere dai dati e formulare previsioni. Attualmente, il Machine Learning è considerato uno dei pilastri fondamentali dell'Intelligenza Artificiale, poiché consente di ottimizzare e perfezionare il comportamento attraverso l'apprendimento diretto dai dati, evitando la necessità di una programmazione esplicita del comportamento e semplificando così lo sviluppo di applicazioni. Tutti gli algoritmi noti di Machine Learning sono basati sull'idea di miglioramento automatico del comportamento attraverso l'accumulo di esperienza durante il processo di addestramento. In termini più formali, un programma "si dice che *apprende* da un'esperienza E rispetto a un compito T e una misura di prestazione P, se la sua capacità di completare il compito T, misurata da P, migliora grazie all'esperienza E".

### 1.1.2 Dati e pattern

I dati rappresentano la base del Machine Learning poiché il comportamento degli algoritmi non è esplicitamente programmato, ma deriva dai *dati* stessi. Spesso viene utilizzato il termine *Pattern* per riferirsi ai dati; con tale termine si intende evidenziare una regolarità che si riscontra all'interno di un insieme di dati osservati. La disciplina che studia il riconoscimento dei pattern è la Pattern Recognition. I pattern possono essere di due tipologie, ovvero numerici o categorici. La prima rappresenta la principale tipologia di dati e si riferisce a valori associati a conteggi o caratteristiche misurabili. I pattern numerici sono tipicamente continui e soggetti a ordinamento. Sono rappresentabili come vettori nello spazio multidimensionale, detti anche feature vector, e identificano caratteristiche estratte da segnali, come immagini e suoni. I pattern categorici, invece, si riferiscono a valori associati a caratteristiche qualitative o alla presenza/assenza di una determinata caratteristica (valori booleani). I pattern categorici non possono essere semanticamente tradotti in valori numerici e sono normalmente gestiti da sistemi basati su regole o alberi di classificazione. I pattern categorici sono maggiormente utilizzati nell'ambito del Data Mining e sono spesso combinati con dati numerici. I pattern possono essere anche sequenziali o simili ad altre strutture dati complesse, come grafi e alberi (una riproduzione sonora, rappresentante la pronuncia di una parola, è ad esempio un pattern sequenziale). I pattern sequenziali sono caratterizzati dalla lunghezza variabile e sono difficili da trattare. Richiedono memoria e allineamento spazio-temporale per tener conto del passato; questo perché spesso la posizione nella sequenza e la relazione con i pattern predecessori e successori è di rilevante importanza.

### 1.1.3 Tipologie di apprendimento

Gli algoritmi di Machine Learning possono essere suddivisi in categorie mediante due diversi principi, ovvero per tipologia di apprendimento del "segnale" o per natura dell'output desiderato dal sistema di Machine Learning. Gli algoritmi che di seguito si andranno a trattare, sono quattro e sono quelli appartenenti alla categoria di apprendimento del "segnale".

- **Apprendimento Supervisionato**

L'apprendimento supervisionato è il primo di quattro modelli di ML. Negli algoritmi di apprendimento supervisionato, la macchina viene istruita con l'esempio. I modelli di apprendimento supervisionato sono strutturati in coppie di dati di "input" e "output"; questi ultimi vengono etichettati con il valore desiderato. Immaginiamo, per esempio, che l'obiettivo della macchina sia riconoscere la differenza tra una margherita e una viola. Una coppia binaria di dati di input include l'immagine di una margherita e l'immagine di una viola. L'esito desiderato di quella specifica coppia è il riconoscimento della margherita, affinché possa essere pre-identificata come risultato corretto. Tramite un algoritmo, il sistema compila nel tempo tutti questi dati di addestramento e inizia a stabilire somiglianze correlate, differenze e altri aspetti logici, fino a quando non è in grado di prevedere autonomamente le risposte alle domande "margherita o viola". Un esempio di algoritmo supervisionato sono le app di analisi del traffico.

- **Apprendimento Non Supervisionato**

L'apprendimento non supervisionato è il secondo dei quattro modelli di ML. Nei modelli di apprendimento non supervisionato non vi è una chiave di risposta. La macchina studia i dati di input (molti dei quali non sono né etichettati, né strutturati)- e inizia a identificare schemi e correlazioni utilizzando tutti i dati accessibili e pertinenti. Per certi versi, l'apprendimento non supervisionato ricalca il modello dell'osservazione del mondo da parte dell'uomo. Per raggruppare gli oggetti in categorie facciamo

appello all'intuizione e all'esperienza. Man mano che sperimentiamo un numero sempre maggiore di esempi di un oggetto, la nostra capacità di categorizzarlo e identificarlo diventa più accurata. Per una macchina, l'esperienza equivale alla quantità di dati che vengono immessi e di cui può disporre. Un esempio di algoritmo non supervisionato è il riconoscimento facciale.

- **Apprendimento Semi-Supervisionato**

L'apprendimento semi-supervisionato è il terzo dei quattro modelli di ML. Questo tipo di apprendimento si configura come una soluzione percorribile in presenza di vaste quantità di dati grezzi non strutturati. Il modello prevede l'input di ridotti volumi di dati etichettati per popolare set di dati non etichettati. In sostanza, i dati etichettati hanno il compito di imprimere una spinta iniziale al sistema, con la possibilità, poi, di migliorare notevolmente la rapidità e l'accuratezza dell'apprendimento. L'algoritmo dell'apprendimento semi-supervisionato impedisce alla macchina l'istruzione di analizzare i dati etichettati per rilevare le proprietà correlative che potrebbero essere applicate ai dati non etichettati. A tale modello si accompagnano alcuni rischi, per esempio quello di vedere replicati dal sistema gli errori presenti nei dati etichettati appresi. L'apprendimento semi-supervisionato, ad esempio, viene utilizzato nell'analisi vocale e linguistica.

- **Apprendimento per Rinforzo**

L'apprendimento per rinforzo o, per l'appunto, Reinforcement Learning, è l'ultimo dei quattro modelli di ML e verrà ampiamente trattato nei paragrafi successivi. Si può dire, però, in breve, che questo tipo di apprendimento è orientato a risolvere compiti decisionali sequenziali attraverso interazioni di tentativi ed errori con l'*ambiente*. In un compito decisionale sequenziale, un *agente* interagisce con un sistema dinamico selezionando azioni che influenzano le transizioni di stato per ottimizzare una qualche funzione di *ricompensa*. In modo più formale, in ogni passo temporale T, un agente percepisce il suo stato "S" e seleziona un'azione "A". Il sistema risponde fornendo all'agente una ricompensa numerica, che all'eventualità può essere anche zero, e commutando nello stato successivo "S''. La transizione di stato può essere determinata solo dallo stato corrente e dall'azione dell'agente, oppure può coinvolgere anche processi stocastici<sup>1</sup>.

#### 1.1.4 Analisi delle prestazioni

Un altro aspetto importante del ML è la valutazione delle prestazioni degli algoritmi. Le modalità di valutazione dipendono fortemente dalla tipologia dell'algoritmo in esame. Una possibilità consiste nell'utilizzo diretto della funzione obiettivo; in genere, però, si preferisce utilizzare misure più dirette collegate alla semantica del problema, ad esempio nel caso di problemi di classificazione. L'accuratezza di classificazione (Equazione 1.1.1), espressa in percentuale, è solitamente la misura di prestazione più significativa. Per errore di classificazione (Equazione 1.1.2), invece, si intende il complemento dell'indice di accuratezza.

$$\text{Accuracy} = \frac{\text{Pattern correttamente classificati}}{\text{Pattern classificati}} \quad (1.1.1)$$

$$\text{Error} = 100\% - \text{Accuracy} \quad (1.1.2)$$

Nel caso più semplice, si assume che il pattern da classificare appartenga a una delle classi

---

<sup>1</sup>Un processo stocastico è un processo il cui andamento futuro non può essere previsto con certezza, ma può essere descritto in modo probabilistico

note al problema di classificazione; questa tipologia prende il nome di classificazione a *closed set*. In molti casi reali, invece, i pattern da classificare possono appartenere a una delle classi note, o a nessuna di esse; in questo caso la classificazione si dice a *open set*. In classificazioni ad open set non è possibile, perciò, determinare l'accuratezza o l'errore di classificazione mediante l'utilizzo delle formule riportate perché non tutti i pattern presenti sono classificabili mediante le classi note. Per ovviare al problema esistono due possibili soluzioni. La prima aggiunge una nuova classe fittizia che rappresenta “il resto del mondo” e nel training set si etichettano gli esempi negativi con la nuova classe inserita. Una seconda soluzione si basa sull’aggiunta di una soglia di classificazione al fine di consentire al sistema di non assegnare il pattern se in nessuna delle classi la probabilità di appartenenza non supera la soglia imposta.

### 1.1.5 Storia del Reinforcement Learning

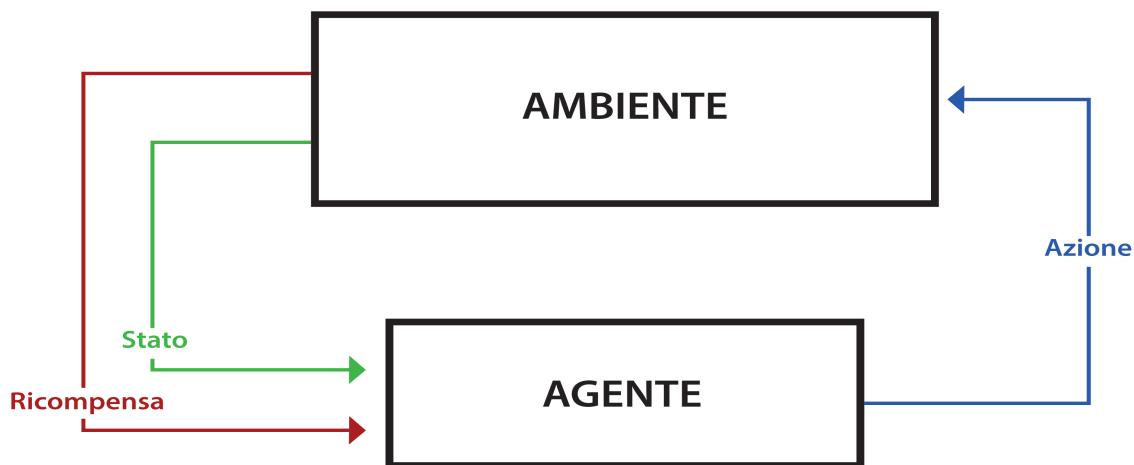
Il Reinforcement Learning (RL) ha attraversato un percorso evolutivo che rispecchia la costante sfida di addestrare algoritmi a prendere decisioni intelligenti in ambienti complessi e dinamici. Nel corso degli anni '50 la teoria del controllo ottimo di Richard Bellman ha gettato le basi concettuali per il RL fornendo un quadro matematico per risolvere problemi decisionali sequenziali. Tuttavia, solo negli anni '80, con lo sviluppo dell'apprendimento per rinforzo neurale, si è iniziato a esplorare l'applicazione di reti neurali per apprendere strategie decisionali ottimali. Gerald Tesauro ha poi introdotto l'algoritmo Q-learning nel 1989; questo ha rappresentato un passo significativo che ha permesso agli agenti di apprendere da esperienze passate attraverso una funzione di valore Q. Tale algoritmo ha avuto successi notevoli, aprendo la strada a ulteriori ricerche sulla convergenza di RL e Intelligenza Artificiale. Con l'avvento del Deep Learning negli anni 2010, il Reinforcement Learning ha vissuto una rivoluzione. Il Deep Q Network (DQN) nel 2013 ha dimostrato la capacità di apprendere da input grezzi, come pixel, portando a una spinta significativa nella capacità di affrontare problemi più complessi. Il trampolino di lancio definitivo del RL è stato nel 2016 con l'exploit di AlphaGo, sviluppato da DeepMind. AlphaGo ha dimostrato una padronanza impressionante del gioco del Go, un gioco tradizionalmente considerato difficilissimo per le macchine a causa della sua complessità. Negli anni successivi, il RL ha continuato a evolversi con un'attenzione sempre crescente per applicazioni pratiche in settori come la robotica, l'automazione, la finanza e l'assistenza sanitaria. Tecnologie come il Reinforcement Learning basate su un modello e l'apprendimento di rinforzo multi-agente stanno contribuendo a superare le sfide e a rendere il RL più adattabile a scenari reali.

Tuttora si parla, comunque, di un argomento in pieno sviluppo, del quale si studiano di giorno in giorno nuove applicazioni ed implementazioni per migliorarlo ed ampliarlo sempre di più.

## 1.2 Funzionamento del Reinforcement Learning

Il Reinforcement Learning (RL), o Apprendimento per Rinforzo, come già sopra riportato, è una tecnica di Machine Learning in cui un *agente* impara a svolgere un'attività tramite ripetute interazioni di tipo “trial-and-error” (eseguite per tentativi ed errori) con un *ambiente* dinamico. Questo approccio all'apprendimento consente all'agente di adottare una serie di decisioni in grado di massimizzare un parametro di *ricompensa* per l'attività, senza essere esplicitamente programmato per tale operazione e senza l'intervento dell'uomo. Questo, in altre parole, significa che al sistema non viene detto quale azione è meglio scegliere, al contrario di quanto accade nella maggior parte delle tipologie di ML, il sistema dovrà scoprire, tramite prove iterate, quali sono le azioni che permettono di ricevere la ricompensa più elevata. Ci sono dei casi più complicati, nei quali le azioni scelte possono influenzare il

tempo di completamento dell'intero processo, perché influenzano le ricompense non solo dell'iterazione corrente ma anche di quelle successive. Queste due caratteristiche, cioè la ricerca mediante trial-and-error e il ritardo nell'ottenimento di ricompense, sono quelle più importanti che distinguono il Reinforcement Learning dalle altre tipologie di Learning. Il Reinforcement Learning non rappresenta soltanto un approccio all'apprendimento, ma definisce un problema di apprendimento più ampio. Qualsiasi metodo idoneo a risolvere questo problema è considerato un metodo di Reinforcement Learning. L'idea fondamentale è quella di catturare gli elementi chiave di un problema del mondo reale, facendo interagire un agente in grado di apprendere con l'ambiente che rappresenta il problema, al fine di raggiungere un obiettivo. Per raggiungere tale scopo è essenziale che l'agente interagisca con l'ambiente e ne osservi lo stato in ogni istante. L'agente deve essere in grado di eseguire azioni che influenzino l'ambiente, modificandone il suo stato. Inoltre, esso deve avere uno o più obiettivi legati allo stato dell'ambiente (Figura 1.1). La formulazione del problema di apprendimento si basa su questi tre elementi: *osservazione, azione e obiettivo*.



**Figura 1.1:** Schema di funzionamento tra ambiente e agente nel Reinforcement Learning

Si può, quindi, dire che il RL si differenzia dalle altre tipologie di apprendimento riportate precedentemente, proprio perché in ambienti nuovi questo tipo di algoritmo funziona meglio degli altri. Una delle sfide affrontate dal Reinforcement Learning e non dalle altre tipologie di learning è il bilanciamento tra l'esplorazione di nuove situazioni e lo sfruttamento delle informazioni già apprese. Infatti, per ottenere una ricompensa elevata, un agente dovrà scegliere le azioni utilizzate nel passato, che hanno permesso ad esso di produrre una buona ricompensa. Il problema è che per scoprire tali azioni l'agente deve scegliere di eseguire azioni che non ha mai provato prima. Quindi, l'agente dovrà sfruttare quello che già conosce in modo da massimizzare la ricompensa finale; però, allo stesso tempo, dovrà esplorare l'ambiente in modo da scegliere azioni migliori nelle esecuzioni future. Il dilemma, a questo punto, è che né l'esplorazione, né lo sfruttamento dell'esperienza, se adottate singolarmente, consentono di completare il compito senza fallire. L'agente, quindi, dovrà provare diverse azioni differenti e progressivamente scegliere quelle che sono apparse come migliori. In un ambiente stocastico, ogni azione dovrà essere provata più volte per ottenere una stima reale della ricompensa prevista. Un altro aspetto fondamentale del Reinforcement Learning è la sua considerazione dell'intero problema dell'interazione tra agente e ambiente, evitando di concentrarsi su sotto-problemi specifici. Quindi tutti gli agenti di RL che sono forniti di un obiettivo esplicito sono in grado di osservare l'ambiente e possono scegliere quale azione intraprendere per influenzare quest'ultimo. Si assume, inoltre, dall'inizio che l'agente dovrà interagire con l'ambiente nonostante la notevole incertezza nella scelta delle azioni.

### 1.2.1 Gli elementi del Reinforcement Learning

Come già anticipato, oltre all'agente e all'ambiente, devono essere definiti altri quattro elementi che caratterizzano il RL: una *policy*, una *reward function*, una *value function* ed un *modello* per l'ambiente.

La *Policy* definisce il comportamento che avrà l'agente ad un certo istante durante la fase di apprendimento. Per policy si intende la relazione tra gli stati osservati dell'ambiente e le azioni da scegliere quando l'agente si trova in tali stati. A volte, la policy può essere una semplice funzione o una tabella di ricerca, mentre in altri casi può comportare un calcolo più complesso, come, ad esempio, un processo di ricerca. In ogni caso, la policy costituisce il nucleo dell'agente, in quanto da sola è sufficiente per determinarne il comportamento.

Per quanto riguarda la *Reward Function*, o funzione di ricompensa, essa stabilisce l'obiettivo o il fine del sistema. Questa funzione assegna a ogni coppia stato-azione (o, più precisamente, a ogni azione intrapresa in uno stato specifico) un singolo valore numerico denominato "reward" che indica intrinsecamente quanto sia conveniente eseguire una particolare azione in uno stato specifico. L'obiettivo dell'agente è massimizzare la somma totale dei reward ricevuti nel corso dell'intero periodo. La funzione di ricompensa determina la qualità degli eventi per l'agente. Le ricompense ottenute nelle coppie stato-azione rappresentano per l'agente le caratteristiche immediate del problema affrontato. Pertanto, l'agente non ha il compito di modificare la funzione di ricompensa, ma può utilizzarla per adattare la sua politica comportamentale. Ad esempio, se un'azione selezionata dalla politica è seguita da una bassa ricompensa, la politica potrebbe essere modificata in modo da scegliere azioni diverse in situazioni simili in futuro.

La *Value Function*, o funzione di valore, delinea ciò che è vantaggioso a lungo termine. Il valore di uno stato rappresenta la somma complessiva delle ricompense che l'agente prevede di accumulare nel futuro, partendo da tale stato. Mentre la ricompensa indica il desiderio immediato di raggiungere uno stato dell'ambiente, il valore esprime, invece, il desiderio a lungo termine, considerando non solo lo stato attuale, ma anche tutti i possibili stati successivi e le ricompense che possono essere ottenute raggiungendoli. A titolo di esempio, uno stato potrebbe garantire una ricompensa costantemente bassa, ma, al contempo, consentire l'accesso a stati altrimenti inaccessibili, che offrono ricompense più elevate. Invece, i valori riflettono un giudizio più raffinato e prospettico su come l'agente sarà soddisfatto o insoddisfatto in base allo stato specifico dell'ambiente. I reward, quindi, rappresentano una ricompensa primaria, mentre i valori rappresentano una predizione della ricompensa totale o secondaria. Senza i reward, i valori non esisterebbero, e l'unico scopo di stimare i valori è quello di ottenere ricompense totali più elevate. Tuttavia, le decisioni saranno prese sulla base dei valori stimati; questo perché l'obiettivo dell'agente è quello di massimizzare la ricompensa totale e non le ricompense immediate. Sfortunatamente determinare i valori è più complicato che determinare i reward; questo perché i secondi vengono forniti dall'ambiente direttamente all'agente mentre i primi devono essere stimati continuamente mediante le sequenze di osservazioni dell'agente. Proprio per questo motivo il componente più importante della maggior parte degli algoritmi di Reinforcement Learning è il metodo per la stima efficiente dei valori. La maggior parte dei metodi di RL è, quindi, strutturata attorno alla stima della funzione valore.

L'ultimo elemento in alcuni sistemi di RL è rappresentato dal *modello* dell'ambiente. Con il termine modello si fa riferimento a un'entità capace di simulare il comportamento dell'ambiente. Ad esempio, una volta forniti uno stato e un'azione, il modello è in grado di prevedere il risultato del prossimo stato e la ricompensa associata. Questi modelli vengono impiegati nel processo di pianificazione, intesa come qualsiasi strategia decisionale basata su situazioni future potenziali, prima ancora che queste si verifichino effettivamente.

### 1.2.2 Markov Decision Process

Come è stato precedentemente detto, l’agente, in fase di apprendimento, basa le proprie decisioni sullo stato percepito dell’ambiente; per definire questa proprietà, si utilizza una proprietà chiamata *Proprietà di Markov*. Assumendo che stati e reward siano finiti, per semplicità di formule matematiche, allora possiamo definire queste ultime in modo più semplice.

In generale possiamo dire che lo stato  $S$  dell’ambiente, all’istante  $t + 1$ , dopo l’esecuzione dell’azione  $A_t$  all’istante  $t$ , è definibile mediante la *distribuzione di probabilità*, definita nell’Equazione 1.2.1 che vale:

$$P\{R_{t+1} = r, S_{t+1} = s' \mid S_0, A_0, R_1, \dots, S_{t-1}, A_{t-1}, R_t, S_t, A_t\} \quad (1.2.1)$$

Questa proprietà deve valere per ogni valore di  $r$  ed  $s'$  e per ogni possibile sequenza di eventi passati:  $S_0, A_0, R_1, \dots, S_{t-1}, A_{t-1}, R_t, S_t, A_t$ . Se un ambiente può assumere uno stato da un insieme finito con la proprietà di Markov, la risposta dell’ambiente all’istante  $t + 1$  dipende esclusivamente dallo stato e dall’azione al tempo  $t$ . In tale circostanza, le dinamiche dell’ambiente possono essere descritte utilizzando la distribuzione di probabilità, definita nell’Equazione 1.2.2:

$$P\{R_{t+1} = r, S_{t+1} = s' \mid S_t, A_t\} \quad \forall r, s', S_t, A_t. \quad (1.2.2)$$

Quindi si può dire che un ambiente godrà della proprietà di Markov, se e solo se l’equazione 1.2.1 è uguale all’equazione 1.2.2 per ogni  $r, s'$ , e ogni possibile valore assumibile dagli eventi passati  $S_0, \dots, A_t$ .

Se un ambiente gode della proprietà di Markov, allora grazie all’equazione 1.2.2 è possibile predire il prossimo stato e il prossimo reward conoscendo soltanto la coppia stato-azione dell’istante precedente. Per questo motivo è possibile considerare l’equazione 1.2.2 come la base per la scelta delle azioni da intraprendere. Possiamo, quindi, dire che la migliore policy che basa la scelta delle azioni considerando la proprietà di Markov è buona quanto la migliore policy che basa la scelta delle azioni considerando l’intera sequenza di eventi passati.

Anche quando l’ambiente non gode della proprietà di Markov è, comunque, appropriato pensare allo stato nel RL come un’approssimazione di uno stato di Markov. La proprietà di Markov è importante nel Reinforcement Learning perché tutti i metodi legati all’apprendimento per rinforzo basano le proprie scelte assumendo che i valori forniti dall’ambiente siano in funzione di solo e soltanto lo stato corrente e dell’azione intrapresa all’istante precedente. Un’istanza di Reinforcement Learning che soddisfa la Proprietà di Markov è chiamata *Processo Decisionale di Markov*, o Markov Decision Process (MDP). Se l’insieme degli stati e l’insieme delle azioni sono finiti, l’MDP è detto *finite Markov Decision Process*, ed è definito come segue: è un insieme di processi che forniscono una struttura matematica per la modellizzazione del processo decisionale in situazioni in cui i risultati sono in parte casuali e in parte sotto il controllo di un decisore. Gli MDP sono utili per lo studio di una vasta gamma di problemi di ottimizzazione, risolti con la programmazione dinamica e il Reinforcement Learning. Noti fin dal 1950, essi sono utilizzati in una vasta area di discipline in cui il processo di presa di decisione avviene in un intorno dinamico. Tra queste discipline abbiamo la robotica, l’automazione e la produzione industriale.

Assumendo che stati e reward siano finiti, per semplicità di formule matematiche, si possono definire i diversi elementi che definiscono l’MDP, questi sono:

- uno spazio degli stati  $S$ ;
- uno spazio delle azioni  $A = U_s \in_s A(s)$  che possono essere intraprese in funzione dello stato;

- le probabilità di transizione  $P_a(S, S') : S \times A \times S \mapsto \mathbb{R}$ , che definiscono le dinamiche one-step dell'ambiente, ovvero la probabilità che, dati uno stato  $s$  e un'azione  $a$  al tempo  $t$ , si raggiunga il possibile stato successivo  $s' : P_a(S, S') = Pr(S_{t+1} = s' | s_t = s, a_t = a)$ ;
- il *valore atteso* della ricompensa  $R_a(s, s') : S \times A \times S \mapsto \mathbb{R}$ : dato uno stato  $s$  ed un'azione  $a$ , se si passa allo stato  $s'$  si ottiene una ricompensa pari a:  
 $R_a(s, s') = R(s'|s, a) = E(r_{t+1}|s_t = s, a_t = a, s_{t+1} = s')$ , dove  $E$  rappresenta il valore atteso (o previsione);
- $\gamma \in [0, 1]$  è il fattore di *sconto* (o discount), che rappresenta l'importante differenza tra le ricompense future (future reward) e le ricompense presenti (present reward).

Si può dire che il problema centrale di un MDP è quello di identificare quale sia la migliore azione da eseguire in un dato stato, in modo da ottenere il massimo valore possibile di una funzione cumulativa della ricompensa. La funzione che, per ogni stato  $s \in S$ , identifica l'azione  $a \in A$  da applicare, è chiamata "politica" (*policy*) stazionaria, ed è definita come  $\pi = S \mapsto A$ . Tipicamente questa funzione che valuta la ricompensa è il valore atteso di una somma con sconto su un orizzonte potenzialmente infinito, definita nell'Equazione 1.2.3:

$$\sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1}) \quad (1.2.3)$$

dove  $a_t = \pi(s_t)$  sono le azioni dettate dalla policy  $\pi$ , e  $\gamma$  è il fattore di sconto  $\in \{0, 1\}$ .

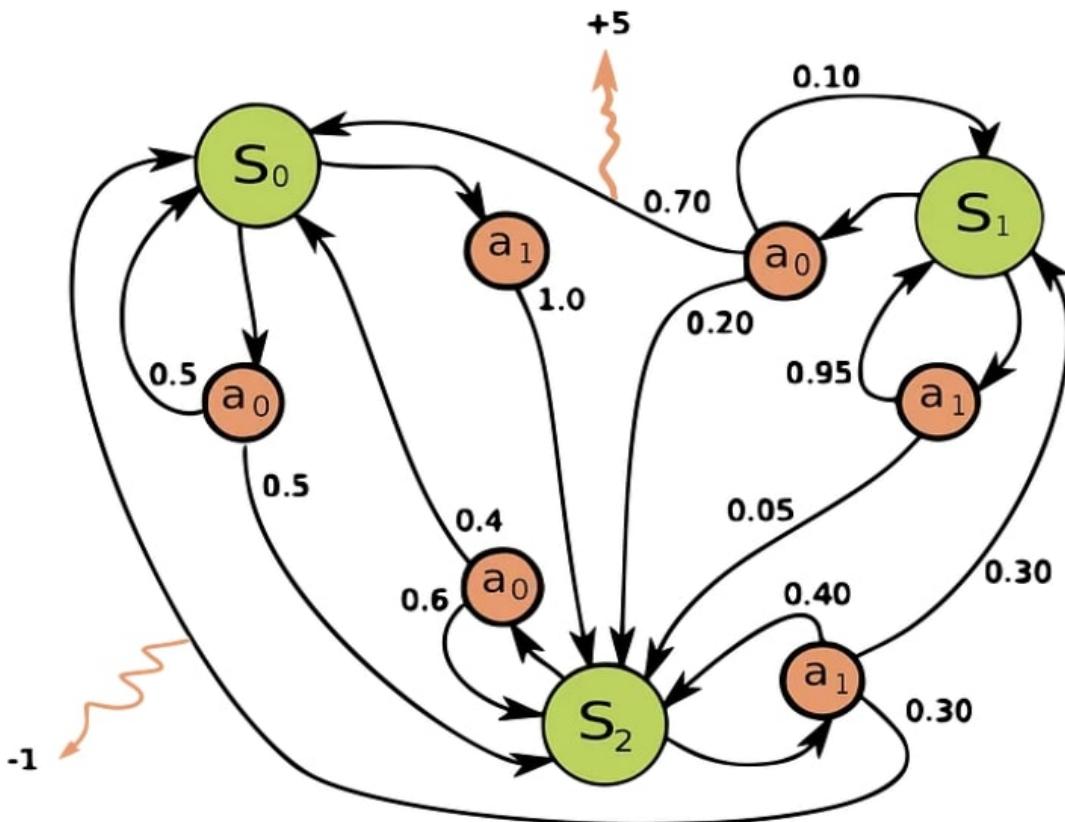
Nella Figura 1.2 si può vedere un'immagine di esempio nella quale viene illustrato un processo decisionale di Markov composto da un insieme finito di stati  $S_i$  e un insieme finito di azioni  $a_j$ . Per ogni stato è possibile eseguire una delle possibili azioni; in questo caso la scelta è solo tra due azioni, ovvero  $a_0$  o  $a_1$ . In figura è possibile osservare come l'esecuzione di un'azione in un dato stato è seguita da una o al più  $i$  transizioni, con  $i = 3$  dell'MDP in esame. Ad ogni transizione in figura è associato un valore rappresentativo  $P(s_{i+1}|s_i, a_j)$ . Trattandosi di una probabilità, la somma delle probabilità delle transizioni uscenti da una qualsiasi coppia  $S_i, a_k$  deve essere pari a 1. Le transizioni possono avere, inoltre, un secondo valore associato, il quale rappresenta  $r(s_i, a_j, s_{i+1})$ .

## 1.3 Gli algoritmi di Reinforcement Learning

Esistono diversi algoritmi per risolvere i problemi di Reinforcement Learning, ma, alla base di ogni algoritmo, è possibile individuare le idee in comune a tutti. Comparando i vari approcci all'apprendimento della funzione valore emerge la presenza di diverse metodologie. Tuttavia, è possibile notare che tutti questi metodi condividono un concetto fondamentale chiamato *Generalized Policy Iteration* o *GPI*. La GPI rappresenta un approccio iterativo finalizzato all'approssimazione delle funzioni di policy e valore. Nello specifico, la funzione valore viene modificata iterativamente per approssimare quella associata alla policy corrente, mentre la policy stessa viene migliorata ripetutamente in base alla funzione valore attuale. Quindi, si può dire che la GPI è la logica iterativa che accomuna tutti gli algoritmi (o metodi).

### 1.3.1 Metodi Tabular Action-Value

In questo sottoparagrafo sono introdotte le idee che accomunano i diversi tipi di algoritmi del RL, basati su una stima della Value Function tramite la creazione della tabella Action-Value; questa è definita come una struttura dati utilizzata per memorizzare e organizzare le stime della funzione valore per tutte le possibili coppie di stati e azioni in un ambiente di RL. In pratica, la tabella Action-Value contiene valori Q associati a ciascuna combinazione



**Figura 1.2:** Esempio di *finite MDP* con tre stati  $\{S_0, S_1, S_2\}$  e due azioni  $\{a_0, a_1\}$

di stato  $s$  e azione  $a$ . Durante il processo di apprendimento, gli agenti di RL aggiornano iterativamente i valori nella tabella in base alle esperienze acquisite nell'ambiente. Questo processo consente all'agente di imparare quale azione eseguire in uno stato specifico per massimizzare le ricompense cumulative nel lungo termine. È, inoltre, importante notare che l'uso di tabelle Action-Value può essere pratico solo in ambienti con uno spazio di stati e azioni discreti e relativamente piccolo, poiché la tabella cresce in modo esponenziale con le dimensioni di questi spazi. In ambienti più complessi è spesso necessario utilizzare approcci approssimativi, come i metodi basati su funzioni di approssimazione o reti neurali, per gestire spazi di stato e azioni più estesi.

Viene quindi prima descritta la logica iterativa in comune a tutti i metodi, che prende il nome di Generalized Policy Iteration, come già sopra riportato. In seguito vengono sommariamente descritte le tre famiglie di metodi definendone pregi e difetti. Fatto ciò, nel paragrafo successivo, viene descritto uno degli algoritmi più usati nel RL, chiamato *Q-Learning*, che alla base ha i metodi che verranno riportati in questo sottoparagrafo.

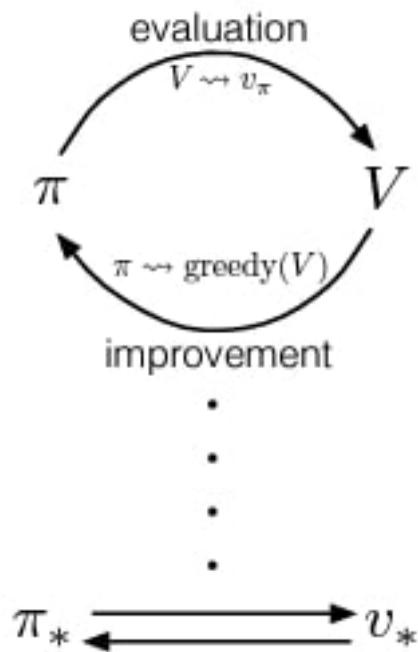
### Generalized Policy Iteration

La Generalized Policy Iteration consiste in due processi simultanei e interagenti: uno rende la funzione valore  $v$  coerente con la policy  $\pi$  attuale, ottenendo così  $v_\pi$  (processo detto *Policy Evaluation*), mentre l'altro rende la policy "greedy"<sup>2</sup> rispetto alla funzione valore attuale,

<sup>2</sup>Gli algoritmi greedy (o "avidii") sono algoritmi che cercano, attraverso una serie di scelte locali ottimali, di ottenere una soluzione globalmente ottimale.

$\pi \rightarrow greedy(v)$  (processo detto *Policy Improvement*). Nella GPI questi due processi si alternano, completando ciascuno prima che l'altro inizi, tuttavia, questo non è necessario, esistono, infatti, delle varianti in cui i processi terminano parzialmente prima di far partire quelli successivi. Finché entrambi i processi continuano ad aggiornare tutti gli stati, il risultato finale è tipicamente lo stesso: convergenza alla funzione valore ottima  $v_*$  e ad una policy ottimale  $\pi_*$ . Utilizziamo il termine Generalized Policy Iteration per riferirci all'idea generale di permettere l'interazione tra i processi di Policy Evaluation e Policy Improvement, indipendentemente dalla granularità<sup>3</sup> e da altri dettagli dei due processi. Quasi tutti i metodi di apprendimento per rinforzo possono essere ben descritti come GPI.

In altre parole, tutti hanno politiche e funzioni valore identificabili, con la politica che viene sempre migliorata rispetto alla funzione valore e la funzione valore che viene sempre guidata verso la politica per il calcolo di una nuova  $v$ . Se sia il processo di valutazione che quello di miglioramento si stabilizzano, cioè non producono più cambiamenti, allora la funzione valore e la politica devono essere ottimali; questo perché la Value Function si stabilizza solo quando è coerente con la Policy corrente e la Policy si stabilizza solo quando è corretta rispetto alla Value Function corrente; pertanto, entrambi i processi si stabilizzano solo quando si è trovata una politica che sia giusta rispetto alla sua funzione di valutazione. I processi di valutazione e di miglioramento in GPI possono essere visti come in competizione e in cooperazione. La competizione consiste nel fatto che le due politiche si muovono in direzioni opposte. Nel lungo periodo, tuttavia, questi due processi interagiscono per trovare una soluzione congiunta: la funzione di valore ottimale e una politica ottimale (Figura 1.3).



**Figura 1.3:** La funzione valore e la policy interagiscono fino al raggiungimento della ottimalità e consistenza reciproca

<sup>3</sup>Con il termine "granularità" ci si riferisce al livello di dettaglio o alla dimensione degli elementi costituenti in un sistema

## Dynamic Programming

La *programmazione dinamica* (Dynamic Programming o DP) è una tecnica di risoluzione di problemi che coinvolge la suddivisione di un problema in sottoproblemi più piccoli e la risoluzione di ciascun sottoproblema solo una volta, memorizzando le soluzioni per evitarne il calcolo ripetuto. Questa tecnica è particolarmente utile per risolvere problemi che possono essere decomposti in sottostrutture sovrapposte e che presentano una proprietà chiamata "ottimalità di sottostruttura". La programmazione dinamica è spesso utilizzata per risolvere problemi di ottimizzazione, dove l'obiettivo è trovare la soluzione migliore in un insieme di soluzioni possibili. L'approccio è chiamato "dinamico" perché coinvolge la costruzione di una tabella (o memorization table) che viene popolata durante la risoluzione dei sottoproblemi.

Un esempio classico di programmazione dinamica è l'algoritmo di ricerca della sequenza più lunga comune (LCS - Longest Common Subsequence), che viene utilizzato in problemi di confronto di sequenze, come la modifica di testo o la biologia computazionale. È importante notare che la programmazione dinamica non è da confondere con il concetto di allocazione dinamica della memoria. Tipico di molti linguaggi di programmazione dinamica nell'ambito algoritmico è una strategia algoritmica specifica.

I metodi DP sono applicabili solo se è presente un modello perfetto dell'ambiente, il quale deve equivalere ad un processo decisionale di Markov. Proprio per questo motivo, gli algoritmi DP sono di poca utilità nel Reinforcement Learning, sia per la loro assunzione di un perfetto modello dell'ambiente, che per l'alta e costosa computazione; tuttavia è comunque opportuno citarli perché rappresentano la base teorica dell'apprendimento per rinforzo. Infatti, tutti i metodi di RL cercano di ottenere lo stesso obiettivo dei metodi DP, solo con costo computazionale inferiore e senza l'assunzione di un modello perfetto dell'ambiente.

## Metodi Monte Carlo

I *Metodi Monte Carlo*, o MC Methods, sono algoritmi di apprendimento per stimare le funzioni valore e scoprire le politiche ottimali. In questo caso non si presuppone una conoscenza completa dell'ambiente. I metodi Monte Carlo richiedono solo l'*esperienza* dell'agente, cioè sequenze campione di stati, azioni e ricompense derivanti dall'interazione reale o simulata con un ambiente. Anche se non c'è una conoscenza preliminare della dinamica dell'ambiente, questi possono, comunque, consentire di ottenere un comportamento ottimale. Si può dire che l'apprendimento dall'esperienza simulata è più potente poiché, sebbene sia necessario un modello, esso deve generare solo transizioni campione e non la distribuzione completa della probabilità delle transizioni possibili, che è richiesta dalla programmazione dinamica (DP). In molti casi, è facile generare esperienze campionate conformi alle distribuzioni di probabilità desiderate, ma è impossibile ottenere le distribuzioni in forma esplicita.

Per garantire la disponibilità di rendimenti campionari ben definiti, si può definire il metodo Monte Carlo solo per compiti in diversi episodi. Ovvero, come per la GPI, ipotizziamo che l'esperienza sia suddivisa in episodi e che tutti gli episodi finiscano per terminare, indipendentemente dalle azioni selezionate. Solo al completamento di un episodio si modificano le stime dei valori (value function) e le politiche (policy). A differenza dei metodi DP, che calcolano i valori per ogni stato, i metodi Monte Carlo calcolano i valori per ogni coppia stato-azione; questo perché, in assenza di un modello, i soli valori di stato non sono sufficienti per decidere quale azione è meglio eseguire in un determinato stato. Per questo motivo, nei metodi Monte Carlo si cerca di ottenere la funzione valore  $q_*(s, a)$ . Il processo di valutazione per i valori di stato-azione è basato sulla stima di  $q_\pi(s, a)$ , ovvero l'Expected Return ottenuto a partire dallo stato  $s$ , scegliendo l'azione  $a$ , seguendo la policy  $\pi$ .

Un grande vantaggio dei metodi Monte Carlo, rispetto ai metodi DP, è la caratteristica di focalizzarsi sulle stime di un più piccolo sotto-insieme di stati assumibili dall'ambiente.

Quindi, una regione di speciale interesse può essere accuratamente valutata senza il bisogno di valutare accuratamente l'intero insieme degli stati.

### Temporal Difference Method

Se si dovesse identificare un'idea centrale e innovativa per l'apprendimento per rinforzo, questa sarebbe, senza dubbio, l'apprendimento mediante metodi Temporal Difference (TD). L'apprendimento TD è una combinazione tra le idee alla base dei metodi Monte Carlo e della Dynamic Programming (DP). Infatti, i metodi TD apprendono direttamente dall'esperienza in assenza di un modello delle dinamiche dell'ambiente, idea alla base dei metodi Monte Carlo. Inoltre, come i metodi DP, i metodi TD aggiornano le proprie stime basando il calcolo in parte su stime passate (quindi, eseguono *bootstrap*); infatti l'affinità tra TD, DP e i metodi di Monte Carlo è un tema ricorrente nella teoria del Reinforcement Learning. Come già verificato per le altre famiglie di metodi, anche i metodi TD seguono l'idea definita dalla GPI per l'iterazione di policy, variandone la granularità.

Grazie alla combinazione delle caratteristiche dei metodi Monte Carlo e DP, i metodi TD presentano vantaggi rispetto ad entrambi. Rispetto ai metodi basati su programmazione dinamica, i metodi TD non richiedono un modello dell'ambiente in grado di stimare la distribuzione di probabilità dei reward e degli stati seguenti, ciò ne aumenta notevolmente le applicazioni. Rispetto ai metodi Monte Carlo, il vantaggio è dato dal fatto che i metodi TD sono realizzati in modo da seguire appieno l'esperienza acquisita incrementalmente con un apprendimento on-line, che stima i nuovi valori delle coppie stato-azione, passo dopo passo. Al passo  $t + 1$ , sono già in grado di aggiornare il valore dello stato  $S_t$ , osservando il reward  $R_{t+1}$  e utilizzando il valore del nuovo stato visitato  $V(S_{t+1})$ . Nel caso più semplice e generale, la logica di aggiornamento dei valori è definita mediante la formula definita nell'Equazione 1.3.1:

$$V(S_t) = V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \quad (1.3.1)$$

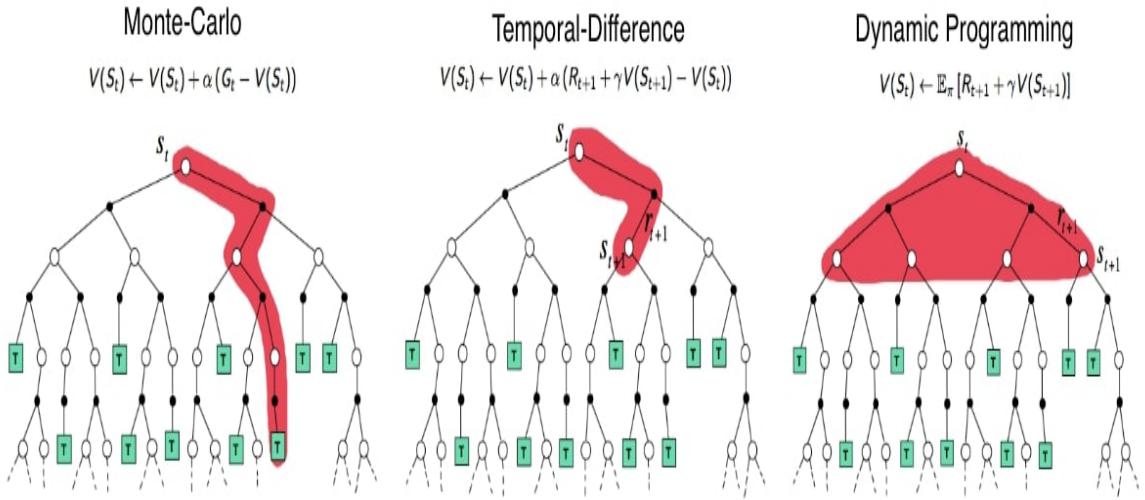
In essa il parametro  $\alpha$ , compreso tra 0 e 1, rappresenta l'indice di aggiornamento del valore di stato  $S_t$ , rispetto alla nuova stima. Dall'equazione 1.3.1 si nota che il calcolo del valore di uno stato dipende dal reward ottenuto allo step  $t + 1$  e dal valore del nuovo stato  $S_{t+1}$ , ridotto dal parametro di discount  $\gamma$  e sottratto dallo stato stesso attuale.

Con i metodi Monte Carlo, invece, occorre aspettare la fine di un episodio prima di apportare le modifiche ai valori degli stati. In certe circostanze in cui gli episodi hanno una durata elevata, l'apprendimento episodio per episodio di MC allunga notevolmente i tempi. Inoltre, l'uso del bootstrap per la stima dei nuovi valori permette ai metodi TD di ottenere performance migliori rispetto ai metodi MC. I metodi TD si differenziano in due classi: on-policy e off-policy. Si può dire che, ad esempio, il metodo *Sarsa* on-policy, mentre il metodo *Q-Learning* è off-policy. Quest'ultimo metodo sarà, anche, il tipo di algoritmo trattato nel prossimo sottoparagrafo dato che è uno dei tipi di metodi più utilizzati e viene preso come esempio per capire meglio come funziona un algoritmo per risolvere il problema del RL. Naturalmente i due sopra citati non sono gli unici metodi che si possono implementare, anzi ne esistono molti altri che si possono utilizzare per risolvere il problema del RL.

Nella Figura 1.4 vengono illustrate le principali differenze tra le tre famiglie di metodi di Reinforcement Learning.

### 1.3.2 Algoritmo Q-Learning

Q-Learning è un tipo di algoritmo di apprendimento Temporal Difference. Questo permette ad un agente di apprendere il comportamento ottimale in un processo decisionale



**Figura 1.4:** Diffrenza tra metodi DP, MC e TD

di Markov. Q-Learning, come Sarsa, stima la funzione valore  $Q(s, a)$  in modo incrementale, aggiornando ad ogni step dell’ambiente il valore della coppia stato-azione, seguendo la logica di aggiornamento della formula generale di stima dei valori per i metodi TD riportata nell’equazione 1.3.1. Come già anticipato, Q-Learning, a differenza di Sarsa, ha caratteristiche off-Policy, ovvero, mentre il miglioramento della policy avviene in funzione dei valori stimati da  $Q(s, a)$ , la funzione valore aggiorna le stime seguendo una policy secondaria strettamente *greedy*. Quindi, dato uno stato, l’azione scelta è sempre quella che massimizza il valore  $\max_a Q(s, a)$ . La policy  $\pi$  ha, comunque, un importante ruolo nella stima dei valori perché mediante essa vengono determinate le coppie stato-azione da visitare e aggiornare.

La formula che rappresenta le modalità di stima delle coppie stato-azione nel metodo Q-Learning è la seguente, definita nell’Equazione 1.3.2:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \quad (1.3.2)$$

Come nell’equazione 1.3.1, questa formula presenta due parametri:  $\gamma$  è il parametro di sconto, compreso tra 0 e 1, il quale permette di dare meno peso a reward ottenuti nel futuro, mentre  $\alpha$  rappresenta il tasso di apprendimento e, come  $\gamma$ , può assumere valori compresi tra 0 e 1. Ponendo  $\alpha=1$  l’aggiornamento del valore di una coppia stato-azione sovrascrive la stima passata, sostituendola con la nuova stima, mentre con  $\alpha=0.5$  il valore aggiornato della coppia stato-azione risulta pari alla media fra la nuova stima e la stima precedente.

Q-Learning utilizza una policy di tipo  $\epsilon$ -greedy. Come succede nel metodo Sarsa,  $\epsilon$  può essere scelta in modo da decrementare l’esplorazione linearmente rispetto agli episodi portati a termine, ad esempio ponendo  $\epsilon = \frac{1}{epoch}$  (dove “epoch” rappresenta un singolo ciclo completo di addestramento dell’agente all’interno dell’ambiente). Si può dire che, grazie alla natura off-policy di Q-Learning, l’analisi dell’algoritmo risulta più semplice perché assume un comportamento indipendente dalla policy di scelta.

Nel caso più semplice, Q-Learning si serve di una tabella per memorizzare ogni coppia stato-azione. Ad ogni step, l’agente osserva lo stato corrente dell’ambiente e servendosi della policy  $\pi$ , seleziona ed esegue l’azione. Eseguendo l’azione l’agente ottiene il reward  $R_{t+1}$  e il nuovo stato  $S_{t+1}$ . A questo punto l’agente è in grado di calcolare  $Q(S_t, A_t)$  aggiornandone la stima.

Nella Figura 1.5 viene riportato l’algoritmo di Q-Learning in pseudocodice.

```
Q-learning (off-policy TD control) for estimating  $\pi \approx \pi_*$ 

Algorithm parameters: step size  $\alpha \in (0, 1]$ , small  $\varepsilon > 0$ 
Initialize  $Q(s, a)$ , for all  $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$ , arbitrarily except that  $Q(\text{terminal}, \cdot) = 0$ 
Loop for each episode:
  Initialize  $S$ 
  Loop for each step of episode:
    Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)
    Take action  $A$ , observe  $R, S'$ 
     $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$ 
     $S \leftarrow S'$ 
  until  $S$  is terminal
```

**Figura 1.5:** Esempio in pseudocodice del funzionamento del Q-Learning

È necessario, però, notare che i metodi e gli algoritmi che sono stati riportati sono solo una parte di tutti quelli finora conosciuti; la scelta è stata fatta per portare degli esempi e far capire con tipi di algoritmi più semplici il funzionamento alla base del Reinforcement Learning (nella parte sperimentale, i tipi di algoritmi utilizzati dai simulatori verranno brevemente trattati nel capitolo apposito).

## 1.4 Applicazioni e casi d’uso del Reinforcement Learning

Le applicazioni del Reinforcement Learning al giorno d’oggi sono molteplici e riguardano diversi settori nel mondo del lavoro, dello studio e della ricerca. Per quanto il RL sia un concetto ancora in fase di studio da molti anni (come riportato nel sottoparagrafo riguardante la sua storia), esso è comunque entrato nelle vite di tutti i giorni di moltissime persone di tutto il mondo. Esistono, ad esempio, lavori per i quali l’apprendimento per rinforzo è diventato indispensabile per garantirne il corretto svolgimento. Dato che il RL è un paradigma generale di apprendimento, previsione e decisione, allora può essere utile per una certa applicazione o settore se può essere considerato o trasformato in un problema decisionale sequenziale e se è possibile costruire stati, azioni e ricompense. Concettualmente si intende che, fornendo certi dati, questi dovranno poi essere manipolati e trasformati proprio nel problema di RL da risolvere con il tipo di algoritmo opportuno.

Verranno, di seguito, specificati diversi settori lavorativi e ambiti di studio/ricerca nei quali il Reinforcement Learning è applicato; per ciascun ambito verranno forniti degli esempi specifici:

- *Robotica e Controllo Autonomo*

Per quanto riguarda la robotica ed il controllo autonomo, il RL può essere utilizzato nell’ambito dell’*automazione industriale* (riguardo quest’area di lavoro se ne parlerà più approfonditamente nel secondo capitolo), per i *veicoli autonomi* o per l’addestramento di certi tipi di *robot autonomi*. Per esempio l’addestramento per rinforzo può essere

usato per ottimizzare i processi di produzione, addestrando gli agenti a gestire compiti sempre più complessi, oppure si potrebbe utilizzare il Reinforcement Learning per insegnare ai veicoli autonomi a guidare in modo sicuro, navigare in ambienti complessi e rispondere a scenari imprevisti.

- *Finanza*

L'utilizzo del RL nell'ambito della finanza è sempre più ampio, attualmente viene utilizzato prevalentemente in settori come il *trading algoritmico* e la *gestione del portafoglio azionario*. Riguardo il primo, il Reinforcement Learning è applicato per sviluppare strategie di trading dinamiche basate su analisi di mercato in tempo reale, migliorando le decisioni di investimento. Mentre, per quanto riguarda il secondo, l'apprendimento per rinforzo è utilizzato per ottimizzare la distribuzione degli investimenti con l'obiettivo di massimizzare il rendimento e minimizzare il rischio.

- *Sistemi di gestione del traffico*

Il Reinforcement Learning è anche ampiamente utilizzato e si sta diffondendo sempre più nei sistemi di gestione del traffico. Infatti esso può essere implementato per il *controllo del traffico*, utilizzando i suoi algoritmi per sincronizzare i semafori e ottimizzare il flusso del traffico, oppure anche per la *logistica del trasporto*, pianificando percorsi ottimali per i mezzi e migliorando l'efficienza della consegna.

- *Assistenza Sanitaria*

Il Reinforcement Learning può essere utilizzato anche in ambito medico. Alcuni esempi sono la *terapia personalizzata*, nella quale viene impiegato il RL per adattare i piani di trattamento ai pazienti in base alla loro risposta individuale, migliorando l'efficacia delle cure, oppure per la *gestione delle risorse sanitarie*, pianificando il servizio del personale e delle attrezzature per riuscire ad avere sempre la disponibilità quando necessaria.

- *Educazione*

Un altro esempio di applicazione del Reinforcement Learning che vale la pena citare, è nell'ambito educativo. Esso può essere, infatti, utilizzato per una *personalizzazione dell'apprendimento* o per *simulazioni didattiche*. Per quanto riguarda il primo caso d'uso si può applicare il RL per adattare i contenuti e le modalità di apprendimento in base alle esigenze individuali degli studenti, migliorando, così, l'efficacia dell'istruzione; mentre, nel secondo caso, l'apprendimento per rinforzo può essere impiegato per creare simulazioni interattive che consentono agli studenti di acquisire competenze pratiche in modo virtuale.

Esistono molti altri ambiti in cui si fa uso del Reinforcement Learning, quali l'*apprendimento automatico di giochi e sport*, dove gli agenti vengono allenati per ottimizzare tattiche e strategie nelle simulazioni per giochi di squadra e non, oppure per *sistemi di raccomandazione*, grazie ai quali, nelle piattaforme di streaming o nei siti di e-commerce, vengono personalizzate le scelte in base al comportamento dell'utente. Il Reinforcement Learning viene anche utilizzato nel campo dei *videogiochi*, nei quali gli agenti del RL vengono addestrati per giocare autonomamente contro giocatori umani e migliorarsi ad ogni partita (i classici esempi sono gli scacchi, giochi Atari o AlphaGo).

## 1.5 Problemi e sfide del Reinforcement Learning

Nonostante gli enormi progressi, il Reinforcement Learning deve affrontare ancora molti *problemi* e superare sempre nuove *sfide* al giorno d'oggi; infatti, essendo un campo in continua crescita ed espansione, ad ogni nuova scoperta si affiancano dei nuovi problemi da risolvere,

mettendo sempre più alla prova questo tipo di apprendimento nell'ambito del Machine Learning e dell'AI. Di seguito verranno esposti alcuni esempi di problemi e sfide che sta affrontando il Reinforcement Learning al giorno d'oggi:

- *Apprendimento da pochi dati*

Molte applicazioni del RL richiedono un grande numero di iterazioni per apprendere comportamenti utili alla risoluzione del problema. Ridurre, quindi, la dipendenza da grandi quantità di dati per l'addestramento rimane una sfida tuttora attuale, specialmente in contesti in cui la raccolta di dati è costosa o pericolosa.

- *Stabilità nell'addestramento di Reti Neurali Profonde*

L'addestramento di reti neurali profonde in ambienti RL può essere instabile e soggetto a problemi come l'instabilità del gradiente della funzione. Ciò può rendere difficile l'ottenimento di modelli affidabili e generalizzabili.

- *Esplorazione Efficente*

L'esplorazione di nuovi stati e azioni è un aspetto critico nel RL. Gli algoritmi devono essere bilanciati tra l'esplorazione per scoprire nuove strategie e lo sfruttamento delle conoscenze esistenti per massimizzare le ricompense. Questo è sempre stato uno dei problemi del RL; naturalmente, con l'evoluzione di nuovi algoritmi, il problema viene sempre meno, ma rimane, comunque, una delle sfide principali da risolvere per l'apprendimento per rinforzo.

- *Generalizzazione a contesti nuovi*

Far sì che gli agenti RL generalizzino (cioè si adattino) efficacemente da un ambiente di addestramento a situazioni nuove e diverse è un'altra sfida. La mancanza di generalizzazione può portare a comportamenti inaspettati o inefficienti.

- *Trasferimento di Conoscenza (Transfer Learning)*

Il trasferimento di conoscenza da un compito o dominio a un altro è ancora una sfida aperta nel RL. Ottenere modelli che possano beneficiare dell'esperienza appresa in un contesto per migliorare le prestazioni in un altro è una priorità che, con il tempo, sta migliorando sempre più, ma rimane uno dei problemi più attuali nel contesto del Reinforcement Learning.

- *Dimensionalità dei Dataset*

Con l'aumentare delle dimensioni degli spazi di stato e azione, gli algoritmi devono confrontarsi con la cosiddetta "maledizione della dimensionalità". Con questo termine si intende una serie di problematiche che emergono quando si lavora con *dataset ad alta dimensionalità*, il che può rendere l'addestramento più difficile e richiedere risorse computazionali più elevate.

- *Eticità e Bias*

Possono emergere problemi di *eticità* e *bias*<sup>4</sup> durante l'addestramento di agenti RL, specialmente se i dati di addestramento incorporano pregiudizi o disuguaglianze presenti nella società.

Naturalmente non tutti gli algoritmi e modelli del RL sono influenzati da questo tipo di problematiche, e non tutti dovranno affrontare le stesse sfide. Infatti, di fronte ad un certo tipo di problema, un dato modello di RL risponderà in modo computazionalmente diverso da un altro, anche se l'idea alla base è sempre la stessa.

<sup>4</sup>Con il termine "bias" si intende un errore dovuto ad assunzioni errate nel processo di apprendimento automatico.

## 1.6 Etica e questioni sociali del Reinforcement Learning

Un obiettivo ambizioso dell'apprendimento per rinforzo è quello di creare agenti che si comportino in modo etico. La capacità di rispettare le norme morali umane amplierebbe notevolmente il contesto in cui gli agenti autonomi potrebbero essere impiegati in modo pratico e sicuro. Ciò non è, però, un argomento facile da trattare; questo perché gli algoritmi di RL possono avere impatti significativi sulla società, sull'individuo e sull'ambiente in cui vengono applicati.

Riuscire, quindi, a far coesistere l'etica del comportamento di un agente di Reinforcement Learning con l'applicazione di tale agente, nel contesto della vita di tutti i giorni è una sfida tutt'altro che semplice. Uno dei modi con i quali si può procedere è quello di addestrare gli agenti etici premiando il comportamento corretto in base a una specifica teoria morale (ad esempio, l'utilitarismo); permane, tuttavia, un diffuso disaccordo sulla natura della moralità. Quindi, riconoscendo tale disaccordo, recenti lavori di filosofia morale propongono che il comportamento etico richieda di agire nell'ambito dell'incertezza morale, ossia di tenere conto, quando si agisce, che la propria credenza è divisa tra diverse teorie etiche plausibili. Naturalmente questo è solo un esempio di come si potrebbe procedere per agire su questo concetto così delicato. La corretta considerazione degli aspetti etici è essenziale per garantire che tali sistemi siano sviluppati e implementati in modo responsabile, evitando potenziali conseguenze negative.

Esistono alcuni aspetti chiave dell'etica nel Reinforcement Learning, come, per esempio, quelli di seguito riportati. Concetti come la *trasparenza e spiegabilità* sono importanti; infatti la mancanza di trasparenza può rendere difficile identificare e correggere discriminazioni ingiuste. Sviluppare modelli che siano trasparenti e spiegabili è essenziale per comprendere come vengono prese le decisioni e mitigare il rischio di discriminazione.

Anche l'*equità nel trattamento* è un aspetto importante; gli algoritmi di RL devono essere progettati in modo da garantire un trattamento equo per tutti gli utenti, evitando discriminazioni basate su caratteristiche sensibili, come la razza, il genere o l'età.

Un altro concetto chiave è la *responsabilità nel prendere le decisioni autonome*, poiché, data la natura autonoma degli agenti di RL, è importante definire chiaramente la responsabilità quando si verificano decisioni impreviste o indesiderate.

Anche la *gestione dei dati sensibili* ha la sua grande importanza; infatti, poiché l'addestramento di modelli di RL può richiedere l'uso di dati sensibili degli utenti, è cruciale proteggere la privacy delle informazioni personali, implementando misure di sicurezza e politiche che rispettino la riservatezza degli individui.

Anche aspetti come *valori sociali e norme culturali* hanno la loro rilevanza nell'etica del RL e devono essere considerati per implementare un buon algoritmo etico; per fare ciò, i modelli di Reinforcement Learning devono essere sensibili ai valori sociali e alle norme culturali delle diverse comunità con i quali devono essere a contatto; oltre a ciò devono essere in grado di conoscere e rispettare la diversità culturale per evitare soluzioni eticamente inaccettabili.

Oltre a tutti i concetti appena visti, per implementare degli algoritmi etici, bisogna prendere in considerazione anche fattori come l'*impatto socio-economico* e le *conseguenze a lungo termine*; ciò al fine di prevenire effetti collaterali indesiderati sulla vita dell'uomo e monitorare costantemente gli impatti delle decisioni degli algoritmi stessi.

Si può, quindi, affermare in sintesi, che l'etica nel Reinforcement Learning richiede una valutazione olistica delle implicazioni etiche legate a ogni fase del ciclo di vita degli algoritmi; dalla progettazione all'addestramento fino all'implementazione pratica. L'adozione di principi etici solidi contribuisce, quindi, a garantire che il RL sia sviluppato in modo responsabile, nel rispetto dei valori umani e della giustizia sociale. Bisogna, tuttavia, evidenziare che, come sopra riportato, questa rimane una sfida ancora aperta sotto molti punti di vista.

## 1.7 Conclusioni

In conclusione, l’analisi approfondita condotta in questo capitolo ha evidenziato l’importanza e la complessità del Reinforcement Learning (RL) nell’ambito dell’Intelligenza Artificiale e del Machine Learning. Dagli argomenti trattati, emergono chiaramente le sue potenzialità nella risoluzione di problemi complessi e nella realizzazione di sistemi intelligenti, in grado di apprendere e migliorare le proprie performance nel tempo. Si può, anche, dire che l’adattabilità dei modelli di RL alle varie situazioni, la capacità di apprendere da interazioni con l’ambiente e la possibilità di ottimizzare decisioni in tempo reale ne fanno una tecnologia chiave per una vasta gamma di settori, dall’industria alla robotica, dall’economia alla salute. Tuttavia, è importante sottolineare che, nonostante i notevoli progressi raggiunti, vi sono ancora sfide aperte e aree di miglioramento. La gestione dell’esplorazione, l’efficienza computazionale e la scalabilità a problemi di dimensioni maggiori rimangono argomenti di ricerca critici che richiedono attenzione continua.

Guardando al futuro, ci sono diverse direzioni promettenti per l’evoluzione del Reinforcement Learning. Innanzitutto, l’integrazione di tecniche di apprendimento profondo (Deep Reinforcement Learning) potrebbe portare a ulteriori avanzamenti, consentendo ai modelli di apprendere rappresentazioni più complesse e astratte delle informazioni. Inoltre, l’applicazione del RL a settori emergenti come l’*Edge Computing* e l’*Internet of Things* (IoT) apre nuove opportunità per la creazione di sistemi intelligenti e autonomi.

In conclusione, il Reinforcement Learning, come si è visto, rappresenta un argomento con potenziale di crescita enorme, nonostante gli enormi progressi già raggiunti nel campo dell’Intelligenza Artificiale e del Machine Learning, e il suo impatto continuo su diverse industrie è destinato a crescere. Il percorso futuro richiederà un impegno costante nella ricerca e nello sviluppo, con l’obiettivo di superare le sfide attuali e sfruttare appieno il potenziale di questa tecnologia innovativa.

# CAPITOLO 2

---

## Industria 4.0

---

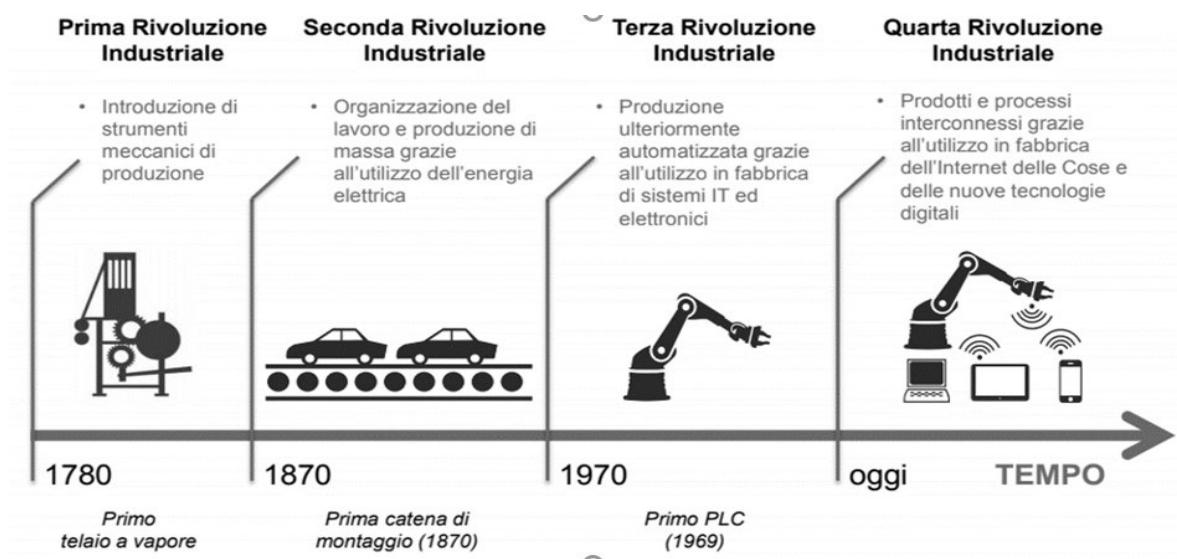
*L'evoluzione continua del panorama industriale ha dato vita a una rivoluzione senza precedenti, nota come Industria 4.0, un paradigma in cui la digitalizzazione, l'interconnessione e l'Intelligenza Artificiale convergono per ridefinire radicalmente i processi produttivi. Nel corso dell'ultimo decennio, il Reinforcement Learning (RL) si è affermato come una forza motrice nella trasformazione digitale, offrendo nuove prospettive e soluzioni avanzate in diversi settori, inclusa l'Industria 4.0. Questo capitolo si propone, infatti, di esplorare le applicazioni del Reinforcement Learning nell'ambito di tale contesto. Dopo una breve rassegna dei concetti fondamentali riguardanti l'Industria 4.0, si delineerà un quadro dettagliato dell'intersezione tra quest'ultima e il Reinforcement Learning, evidenziando come il RL possa contribuire in modo significativo a migliorare l'efficienza, la flessibilità e la gestione dei processi industriali. Si analizzeranno dei casi di successo, riguardanti progetti specifici che hanno adottato il Reinforcement Learning, evidenziando i risultati ottenuti. Guardando al futuro, il capitolo si concluderà con una panoramica delle prospettive emergenti e degli sviluppi tecnologici nel campo del Reinforcement Learning nell'Industria 4.0, offrendo considerazioni conclusive che in merito alle implicazioni di questa convergenza per il futuro dell'industria e della società nel suo complesso.*

### 2.1 Introduzione all'Industria 4.0

L'Industria 4.0 è un processo che scaturisce dalla quarta rivoluzione industriale e che sta portando alla produzione industriale del tutto automatizzata e interconnessa. Infatti, questo concetto è basato sull'inserimento di alcune nuove tecnologie produttive per migliorare le condizioni di lavoro, creare nuovi modelli di business, aumentare la produttività degli impianti e migliorare la qualità dei prodotti.

Le nuove tecnologie digitali hanno un impatto profondo nell'ambito di quattro direttive di sviluppo. La prima riguarda l'utilizzo dei dati, la potenza di calcolo e la connettività, e si declina in Big Data, Open Data, Internet of Things e Cloud Computing, per la centralizzazione delle informazioni e la loro conservazione. La seconda è quella degli analytics, concetto che si basa sul fatto che, una volta raccolti i dati, bisogna ricavarne il valore. Oggi, infatti, solo l'1% dei dati raccolti viene utilizzato dalle imprese, che potrebbero, invece, ottenere vantaggi grazie all'utilizzo del Machine Learning che consente alle macchine di perfezionare la loro resa imparando dai dati via via raccolti e analizzati (come, per esempio, nel caso del Reinforcement Learning). La terza direttrice di sviluppo è l'interazione tra uomo e macchina, che coinvolge le interfacce "touch", sempre più diffuse, e la realtà aumentata. Infine c'è tutto il settore che si occupa del passaggio dal digitale al "reale" e che comprende la manifattura additiva, la stampa 3D, la robotica, le comunicazioni, le interazioni machine-to-machine e le nuove tecnologie per immagazzinare e utilizzare l'energia in modo mirato, razionalizzando i costi e ottimizzando le prestazioni.

L'espressione "Industry 4.0" è stata usata per la prima volta alla Fiera di Hannover nel 2011 in Germania. A ottobre 2012 un gruppo di lavoro dedicato all'Industria 4.0, presieduto da Siegfried Dais della multinazionale di ingegneria ed elettronica Robert Bosch GmbH, e da Henning Kagermann della AcaTech (Accademia tedesca delle Scienze e dell'Ingegneria), presentò al governo federale tedesco una serie di raccomandazioni per la sua implementazione. L'8 aprile 2013, all'annuale Fiera di Hannover, fu diffuso il report finale del gruppo di lavoro. Finora, come si può vedere in Figura 2.1, le rivoluzioni industriali nel mondo occidentale sono state tre, la prima nel 1784 con la nascita della macchina a vapore, la seconda nel 1870 con l'inizio della produzione di massa e la terza nel 1970 con la nascita dell'informatica, dalla quale è scaturita l'era digitale destinata ad incrementare i livelli di automazione avvalendosi di sistemi elettronici. La data d'inizio della quarta rivoluzione industriale non è ancora stabilita, probabilmente perché è tuttora in corso e solo a posteriori sarà possibile indicarne l'atto fondante.



**Figura 2.1:** Schematizzazione delle diverse rivoluzioni industriali

### 2.1.1 Il concetto di Smart Factory

L'Industria 4.0 passa per il concetto di *Smart Factory* che si compone di tre parti. La prima è la *Smart Production*, della quale fanno parte tutte quelle nuove tecnologie produttive che creano collaborazione tra tutti gli elementi presenti nella produzione, ovvero collaborazione tra operatore, macchine e strumenti. La seconda parte, che viene chiamata *Smart Service*, è formata da tutte quelle infrastrutture informatiche e tecniche che permettono di integrare i sistemi, ma anche da tutte le strutture che permettono, in modo collaborativo, di integrare le aziende tra loro. L'ultima parte è la *Smart Energy*, che si occupa di avere un occhio attento sui consumi energetici delle prime due parti, creando sistemi più performanti e riducendo gli sprechi di energia secondo i paradigmi tipici dell'energia sostenibile.

L'elemento centrale dell'Industria 4.0 (che fa parte del concetto dello Smart Service) sono i *sistemi Cyber-Fisici* (Cyber Physical Systems, o CPS), ovvero sistemi fisici collegati con i sistemi informatici, con i quali possono interagire in modo continuo, ma che possono anche interagire e collaborare con altri sistemi CPS. Questo tipo di sistema è composto da elementi fisici dotati ciascuno di capacità computazionale e riunisce strettamente le cosiddette "tre C": capacità computazionale, comunicazione e capacità di controllo. Queste strutture artificiali di calcolo e comunicazione formano un sistema distribuito che interagisce direttamente e

dinamicamente con il mondo reale che le circonda, permettendo, in questo modo, di integrarsi perfettamente con il concetto di Industria 4.0 e di tutto ciò che lo riguarda.

### 2.1.2 Tecnologie Abilitanti

Da studi recenti emerge che la quarta rivoluzione industriale è incentrata sull’adozione di alcune tecnologie definite *tecnologie abilitanti*. Alcune di queste erano già presenti da prima, ma non hanno mai sfondato il muro della divisione tra ricerca applicata e sistemi di produzione veri e propri; oggi, invece, grazie all’interconnessione e alla collaborazione tra sistemi, le tecnologie abilitanti sono diventate di interesse per l’intero settore manifatturiero. Questo tipo di tecnologie possono essere definite come fondamentali, poiché aumentano il valore della catena del sistema produttivo e hanno la capacità di innovare i processi, i prodotti e i servizi in tutti i settori economici dell’attività umana. Parlare, quindi, di queste tecnologie, implica, di fatto, parlare dell’utilizzo dell’Intelligenza Artificiale applicata agli oggetti e creare, pertanto, un collegamento ed una collaborazione tra la realtà fisica e quella virtuale.

Nel contesto di questa trasformazione, alcune tecnologie emergono come pilastri centrali, svolgendo un ruolo chiave nella definizione del nuovo volto dell’industria moderna. Tali tecnologie sono di seguito definite:

- *Intelligenza Artificiale e Machine Learning*. L’apprendimento automatico (incluso il Reinforcement Learning) costituisce il cuore dell’Intelligenza Artificiale nell’Industria 4.0. La capacità di apprendere da dati e adattarsi dinamicamente alle variazioni dell’ambiente si traduce in un miglioramento significativo dell’efficienza operativa e della capacità predittiva.
- *Internet of Things*. L’IoT rappresenta una colonna portante dell’Industria 4.0, consentendo la connessione e la comunicazione tra dispositivi e sistemi. Nel contesto industriale, esso si traduce in una vasta rete di sensori e attuatori che rilevano, monitorano e scambiano dati in tempo reale. L’integrazione di queste informazioni con i diversi macchinari offre possibilità uniche di ottimizzazione e automazione dei processi industriali.
- *Edge Computing and Fog Computing*. La distribuzione del calcolo a livello periferico (Edge Computing) e la gestione dei dati in prossimità dei dispositivi (Fog Computing) sono essenziali per ridurre la latenza e migliorare l’efficienza. Queste tecnologie trovano applicazione, quindi, nell’Industria 4.0 con un impatto fondamentale, poiché permettono la migliore gestione possibile dei dati che verranno, poi, utilizzati dalle diverse macchine.
- *Stampa 3D e Additive Manufacturing*. La stampa 3D e l’Additive Manufacturing stanno rivoluzionando i processi di produzione, consentendo la creazione di componenti complessi in modo più efficiente. Queste tecnologie, quindi, permettono un’ottimizzazione dei parametri di produzione, consentendo una personalizzazione più approfondita e una maggiore efficienza nell’industria manifatturiera.
- *Blockchain e Sicurezza Digitale*. La Blockchain, con la sua architettura sicura e decentralizzata, trova applicazione nel garantire la sicurezza dei dati e delle transazioni nell’Industria 4.0. La sicurezza digitale, garantita dalla Blockchain, è, quindi, anch’essa una tecnologia molto importante al giorno d’oggi, poiché consente alle varie industrie una sicurezza riguardante i dati e garantisce riservatezza sulle informazioni delle singole industrie.

Naturalmente, queste tecnologie abilitanti sopra riportate non sono le uniche esistenti al giorno d'oggi, ma solo quelle attualmente più importanti e più utilizzate nell'ambito dell'Industry 4.0.

## 2.2 Intersezione tra Reinforcement Learning e Industry 4.0

La rivoluzione industriale in corso, incarnata dall'Industria 4.0, si caratterizza per l'integrazione sinergica di tecnologie avanzate e approcci innovativi. In questo senso, il Reinforcement Learning può essere considerato il punto di convergenza di IA e ML (tecnologie abilitanti) con l'Industria 4.0. Queste due forze propulsive si intersecano per plasmare il futuro dell'ambiente produttivo. In questo sottocapitolo vengono descritte le interazioni dinamiche e le opportunità emergenti che derivano dalla fusione di Intelligenza Artificiale e produzione industriale avanzata.

Poiché l'Industria 4.0 non è soltanto una trasformazione tecnologica, ma anche un cambiamento fondamentale nelle modalità di gestione, produzione e presa di decisioni, il Reinforcement Learning, con la sua capacità di apprendimento autonomo e adattamento continuo, si posiziona come concetto chiave in questo panorama. Si possono identificare, infatti, delle aree cruciali in cui l'implementazione del RL offre un valore significativo.

Uno dei principali punti di convergenza risiede nell'adattabilità dei processi produttivi. L'Industria 4.0, infatti, promuove la flessibilità e la personalizzazione della produzione per rispondere alle mutevoli esigenze del mercato. Il RL si inserisce naturalmente in questo contesto, consentendo ai sistemi di apprendere dai dati in tempo reale e ottimizzare dinamicamente le strategie operative; ciò consente una rapida risposta alle variazioni nella domanda di mercato, alle nuove specifiche del prodotto e alle condizioni ambientali mutevoli, garantendo una produzione efficiente e su misura. Un'altra area chiave è rappresentata dall'ottimizzazione delle catene di approvvigionamento. Il RL può essere impiegato per gestire in modo intelligente la logistica e la distribuzione, adattando le strategie di approvvigionamento sulla base di fattori dinamici, come la domanda del mercato, le condizioni meteorologiche e la disponibilità delle risorse; tale approccio contribuisce a minimizzare i costi operativi, ridurre gli sprechi e migliorare l'efficienza complessiva delle catene di approvvigionamento. L'efficienza energetica è anch'essa un obiettivo cruciale nell'Industria 4.0. Il Reinforcement Learning offre metodologie avanzate per la gestione ottimizzata dell'energia e delle risorse; infatti, attraverso l'apprendimento continuo, nei processi di produzione, i sistemi di RL possono ottimizzare automaticamente i parametri operativi per ridurre il consumo energetico senza compromettere le prestazioni. Ciò contribuisce non solo a migliorare la sostenibilità, ma anche a ridurre i costi associati all'utilizzo delle risorse.

La convergenza tra RL e Industria 4.0 si manifesta anche nella realizzazione di sistemi in grado di prendere decisioni autonome. L'apprendimento per rinforzo consente ai dispositivi e ai sistemi di apprendere dalle esperienze passate, adattandosi a scenari complessi e dinamici senza la necessità di intervento umano costante. Questo aspetto è probabilmente il più importante tra quelli citati, proprio perché permette ai macchinari di essere completamente (o quasi) autonomi; questo, poiché, essendo implementati dagli algoritmi del Reinforcement Learning, per natura del RL stesso, imparano autonomamente le azioni da intraprendere per ottenere il risultato migliore possibile. Ad esempio, i robot autonomi nelle catene di montaggio possono apprendere a gestire variazioni nei processi produttivi senza necessità di programmazione manuale. Questa autonomia decisionale non solo migliora l'efficienza operativa, ma consente anche una risposta rapida alle dinamiche del mercato e alle mutevoli condizioni dell'ambiente produttivo.

In sintesi, la convergenza tra RL e Industria 4.0 si manifesta attraverso una serie di impatti significativi. Dall'adattabilità dei processi produttivi all'ottimizzazione delle catene di ap-

provvigionamento, dalla gestione efficiente delle risorse alla presa di decisioni autonome dei sistemi, questa sinergia rappresenta un passo cruciale verso un futuro industriale intelligente, flessibile e sostenibile.

## 2.3 Applicazione dei metodi di RL nell’Industria 4.0

I metodi di Reinforcement Learning (RL) possono risolvere con successo problemi di ottimizzazione complessi. Le loro applicazioni nel campo dell’Industria 4.0 possono variare da caso a caso, infatti, esistono dei metodi che sono più adatti per un certo tipo di applicazione, al contrario di altri, e viceversa. Per dimostrare ciò, in uno studio (Angelos Angelopoulos [2020]), si è notato che, dividendo i diversi metodi di RL e classificando le loro applicazioni per *“principio”* e *“settore industriale”*, si può capire quale tipo di metodo è meglio usare in una determinata situazione o applicazione pratica.

Per esempio, in Figura 2.2 si può notare, utilizzando i dati inerenti alla classificazione per *“principio”*, come nelle classi di predizione, previsione e stima, i metodi di approssimazione della funzione valore e i processi decisionali di Markov (MDP) sono sovrarappresentati. Questo ci permette di concludere che i metodi complessi sono meno numerosi, il che è perfettamente in linea con l’obiettivo di comprendere meglio il comportamento dell’ambiente senza forti obiettivi di ottimizzazione. Nelle classi di rilevamento, riconoscimento, prevenzione e protezione, invece, i metodi Policy Gradient sono sovrarappresentati, mentre gli MDP sono sottorappresentati. Ciò dimostra che i ricercatori sono più interessati a modelli complessi con prestazioni predittive più elevate che a soluzioni di base. Nelle classi di valutazione, stima e allocazione, assegnazione e gestione delle risorse, i metodi multi-agente sono maggiormente in evidenza, il che ci dice che questo campo è propenso a distribuire i compiti a strumenti di livello inferiore invece che a processi di dati centralizzati.

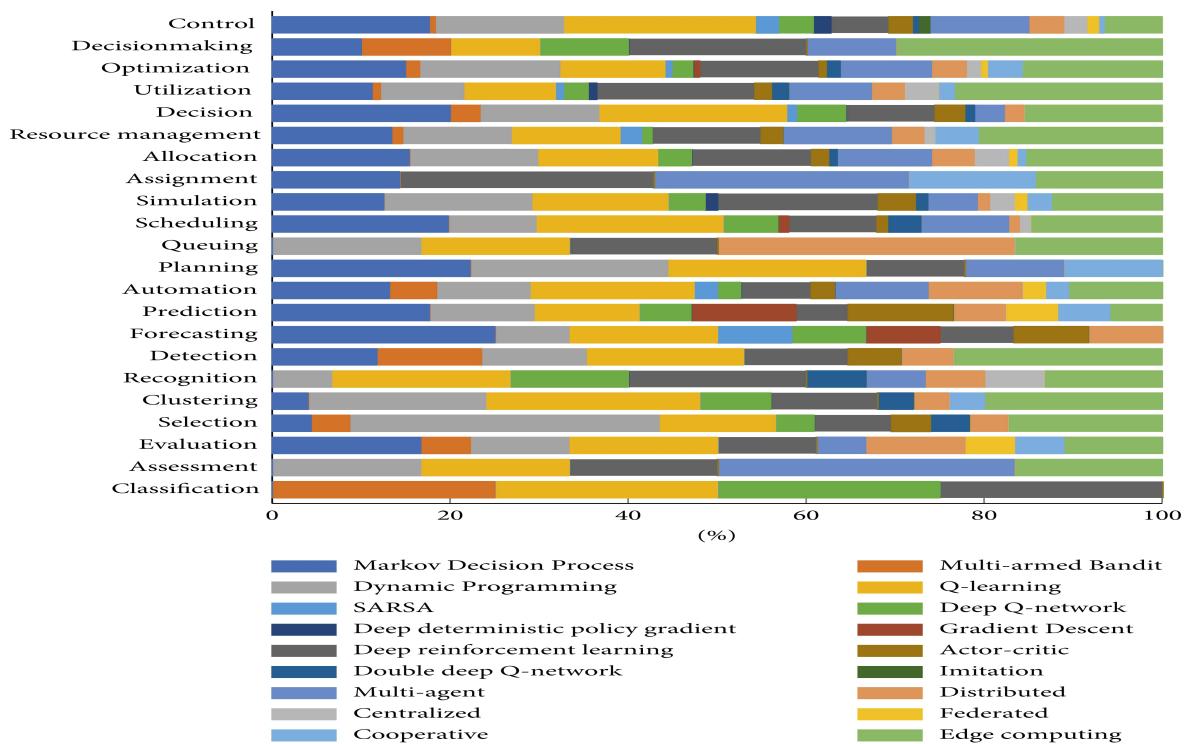
Questo, quindi, ci permette inizialmente di riuscire a capire quali tipi di algoritmi di RL è meglio utilizzare in base al principio che si vorrà implementare per far svolgere un certo tipo di compito ad un macchinario, un robot o ad un agente utilizzando il Reinforcement Learning.

Per quanto riguarda i dati che figurano nella classificazione per settore industriale, come si può vedere in Figura 2.3, si può notare come nella classe dell’energia (solare ed elettrica), le applicazioni dei metodi di Q-Learning sono sovrarappresentate, mentre i metodi di base e i metodi di Policy Gradient sono sottorappresentati. Nelle classi delle telecomunicazioni, della comunicazione, del networking, di internet, del 5G, del Wi-Fi e della telefonia mobile, invece, i metodi a Policy Gradient sono sovrarappresentati e c’è una forte attenzione alle applicazioni dell’edge computing. Mentre, ad esempio, le applicazioni dei processi decisionali di Markov, sono evidenziati nelle classi del wireless, della radio, delle antenne e dei segnali. Allo stesso modo, nelle classi dei veicoli (terrestri o aerei senza pilota), dei droni e dei velivoli, le applicazioni del processo decisionale di Markov sono sovrarappresentate, insieme ai metodi di policy gradient, mentre le soluzioni multi-agente sono meno utilizzate.

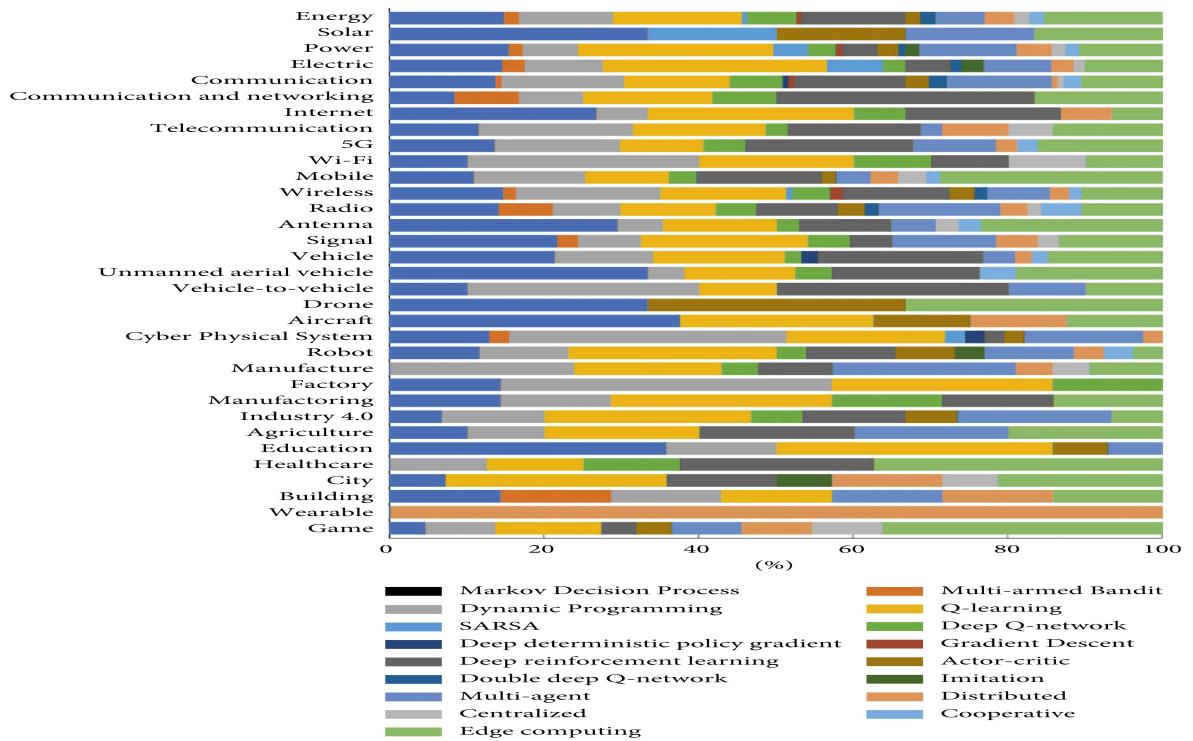
Confrontando i dati presenti in Figura 2.3 con quelli presenti nella Figura 2.2, il campo di applicazione di un certo tipo di metodo rispetto ad un altro si restringe, riuscendo ad ottenere, così, un risultato sempre migliore e specifico ad ogni caso.

Ciò è importante, perché sapere in anticipo quale tipo di algoritmo è meglio applicare in ogni contesto permette, anzitutto, di ottimizzare i tempi di studio iniziale in una data circostanza, e, inoltre, di assicurarsi la migliore esecuzione di un compito di un eventuale macchinario, robot o agente.

In conclusione, si può affermare che, grazie a questo studio, è possibile riuscire ad applicare i migliori metodi ad ogni specifico caso appartenente all’Industria 4.0, per ottenere, in questo modo, il miglior rendimento in ogni situazione.



**Figura 2.2:** Utilizzo del RL in base alla classe "principio"



**Figura 2.3:** Utilizzo del RL in base alla classe "settore industriale"

## 2.4 Casi di successo

In questa sezione, esamineremo concretamente il Reinforcement Learning nell'ambito dell'Industria 4.0, attraverso l'analisi di casi di successo. Con degli esempi reali, verrà illustrato

come l'applicazione di algoritmi di RL abbia contribuito in modo tangibile e innovativo alla realizzazione di progetti industriali avanzati. Verranno, quindi, approfonditi specifici scenari in cui il Reinforcement Learning ha dimostrato di essere una risorsa strategica, fornendo risultati positivi e spunti per futuri sviluppi.

Un primo caso che si può prendere in considerazione è nell'ambito della vendita al dettaglio. Il Reinforcement Learning ha un potenziale immenso in questo settore, in quanto consente ai rivenditori di comprendere a fondo le preferenze dei clienti, di ottimizzare prezzi e promozioni e di offrire esperienze personalizzate che favoriscono la fidelizzazione e la crescita. Infatti il RL viene utilizzato da molte aziende; una di queste è, per esempio, *Amazon*; infatti, essa utilizza algoritmi di RL per ottimizzare dinamicamente i prezzi dei suoi prodotti. L'agente RL impara dal comportamento dei clienti, dai prezzi dei concorrenti e dalle condizioni di mercato a regolare i prezzi in tempo reale, massimizzando i ricavi e mantenendo la competitività. Un'altra azienda specializzata nel settore che utilizza algoritmi di Reinforcement Learning per le vendite è, ad esempio, l'azienda britannica *Tesco*. Essa utilizza il RL per la pianificazione degli assortimenti. Gli agenti RL imparano dai dati di vendita, dalle preferenze dei clienti e dalle tendenze del mercato per ottimizzare gli assortimenti dei prodotti, assicurando che i prodotti giusti siano disponibili nei negozi giusti, migliorando la soddisfazione dei clienti e le vendite.

Il Reinforcement Learning è anche utilizzato nella catena di distribuzione (o supply chain). Esso consente ai professionisti della supply chain di trasformare le loro attività, sfruttando le intuizioni basate sui dati per ottimizzare l'inventario, migliorare la reattività e offrire esperienze ai clienti. Delle aziende che utilizzano il Reinforcement Learning, sono, ad esempio, *UPS* e *Proximus* (un'azienda di comunicazioni Belga). La prima utilizza algoritmi di RL per ottimizzare i percorsi di consegna, infatti, gli agenti RL imparano dai dati sul traffico in tempo reale, dai volumi dei colli e dalle finestre temporali dei clienti per adattare dinamicamente i piani di percorso, riducendo il consumo di carburante e migliorando l'efficienza delle consegne; la seconda utilizza il RL per la selezione e la negoziazione dei fornitori. Gli agenti RL imparano dai dati sulle prestazioni dei fornitori, dai modelli di prezzo e dalle condizioni contrattuali per ottimizzare la selezione dei fornitori e negoziare accordi favorevoli.

Altri casi si possono trovare nell'ambito dell'assistenza sanitaria; infatti, il *Massachusetts General Hospital*, utilizza il RL per ottimizzare il dosaggio personalizzato di farmaci anticoagulanti per i pazienti. L'agente RL apprende dai dati del paziente per raccomandare dosi personalizzate, riducendo il rischio di eventi avversi e migliorando i risultati del trattamento. Un altro caso riguardante lo stesso ambito è il *sistema chirurgico da Vinci*, ampiamente utilizzato negli interventi di chirurgia robotica assistita. L'agente RL apprende dalle dimostrazioni di chirurghi esperti per assistere questi ultimi nell'esecuzione di interventi mini-invasivi con maggiore precisione e destrezza.

Si può trovare applicazione di algoritmi di RL anche in industrie specializzate nella robotica. Ad esempio, *Siemens* ha implementato algoritmi di RL per attività di assemblaggio robotizzato nel settore manifatturiero. Gli agenti RL imparano ad afferrare e manipolare gli oggetti, a eseguire operazioni di assemblaggio e ad adattarsi alle variazioni di posizione e orientamento degli oggetti, migliorando l'efficienza e la flessibilità delle linee di assemblaggio robotiche. Un altro caso di utilizzo del Reinforcement Learning in quest'ambito è *NVIDIA*; infatti, la piattaforma NVIDIA Jetson AGX Xavier impiega il RL per il controllo autonomo del volo dei droni; gli agenti RL imparano a navigare e a eseguire manovre complesse in ambienti dinamici, come evitare gli ostacoli e la pianificazione ottimale del percorso di volo.

Chiudendo questo sottocapitolo è essenziale sottolineare che gli esempi di successo del Reinforcement Learning nell'Industria 4.0 sopra riportati rappresentano solo la punta dell'iceberg. Questi casi evidenziano il potenziale trasformativo degli algoritmi di RL in contesti

industriali e non; tuttavia, è importante riconoscere che le applicazioni si estendono ben oltre quanto qui esposto. Settori come l'industria automobilistica, la finanza, l'agricoltura, l'energia e molti altri stanno abbracciando attivamente il Reinforcement Learning per ottimizzare processi, prendere decisioni più informate e innovare in modo significativo. L'Industria 4.0 si presenta, così, come un terreno fertile per l'applicazione del RL, promettendo ulteriori sviluppi e successi nelle diverse aree dell'attività umana.

## 2.5 Conclusioni

Questo capitolo riguardante l'intersezione tra Reinforcement Learning (RL) e Industria 4.0 è servito ad esplorare l'integrazione delle tecnologie avanzate e la trasformazione dei processi industriali. Dall'introduzione ai concetti fondamentali dell'Industria 4.0, all'esplorazione approfondita delle applicazioni pratiche del RL sono state attraversate diverse tappe di questo percorso di evoluzione industriale. I casi di successo analizzati hanno fornito una finestra sulle potenzialità concrete di questa sinergia, dimostrando come il RL stia giocando un ruolo sempre più rilevante nello sviluppo di sistemi produttivi intelligenti e flessibili. Si è visto che l'Industria 4.0 rappresenta molto più di una semplice evoluzione. È una rivoluzione tecnologica che ridefinisce radicalmente il modo in cui le aziende producono, gestiscono e innovano. L'integrazione di tecnologie avanzate, come l'Internet of Things (IoT), l'Intelligenza Artificiale (IA) e il Reinforcement Learning alimenta una trasformazione profonda nei processi produttivi e nelle dinamiche decisionali. L'intersezione tra RL e Industria 4.0 è parte cruciale di questa rivoluzione. Il RL emerge come un pilastro fondamentale per l'autonomia, l'adattabilità e l'ottimizzazione dei sistemi industriali. La sua capacità di apprendere autonomamente dalle esperienze passate, di adattarsi a scenari complessi e di prendere decisioni in tempo reale lo posiziona al centro di una nuova era di produzione intelligente. Gli esempi di successo esaminati offrono un'istantanea delle applicazioni tangibili del RL in diversi contesti industriali. Dai processi produttivi personalizzati alla gestione delle catene di approvvigionamento fino all'ottimizzazione dei processi di assistenza sanitaria, ogni caso sottolinea come il RL stia influenzando positivamente l'efficienza, la flessibilità e la sostenibilità nelle operazioni industriali.

Il futuro dell'Industria 4.0, guidato dal Reinforcement Learning, si delinea attraverso varie prospettive stimolanti. L'integrazione sempre più stretta con l'Intelligenza Artificiale e l'Internet of Things promette sistemi ancora più intelligenti e interconnessi. La sfida della scalabilità, la garanzia della sicurezza e l'adattamento a nuove sfide emergenti rimangono al centro delle prospettive future. In sintesi, l'Industria 4.0 guidata dal Reinforcement Learning sta costruendo un futuro intelligente e dinamico. Questa trasformazione non è solo tecnologica ma porta con sé cambiamenti culturali, organizzativi ed economici. L'adozione diffusa del RL non è solo un segno di evoluzione tecnologica, ma anche di un cambiamento nella mentalità e nell'approccio verso la gestione delle operazioni industriali.

Per concludere, l'Industria 4.0 alimentata dal Reinforcement Learning è un concetto in continua evoluzione, adattamento, apprendimento e innovazione. L'apertura a nuovi orizzonti, la sfida di frontiere precedentemente inaccessibili e la creazione di un ecosistema industriale più intelligente sono parte fondamentale di questa nuova era.

# CAPITOLO 3

---

## Simulazione RobotArm in ambiente MatLab

---

*Il presente capitolo esplora l'implementazione e la simulazione di un braccio robotico nell'ambiente di sviluppo MATLAB e Simulink, focalizzandosi sull'applicazione dell'algoritmo di Reinforcement Learning Soft Actor Critic (SAC) utilizzato per implementare l'agente. L'obiettivo principale è fornire una comprensione dettagliata dell'esperienza di simulazione, introducendo il contesto, spiegando l'algoritmo chiave utilizzato e presentando le scelte di progettazione. Verranno, poi, effettuati degli esperimenti di simulazione che saranno successivamente discussi. Concludendo il capitolo, verranno offerte riflessioni critiche sulle esperienze di simulazione e sull'efficacia dell'algoritmo SAC nell'ambiente MATLAB in questo tipo di contesto.*

### 3.1 Introduzione all'esperienza

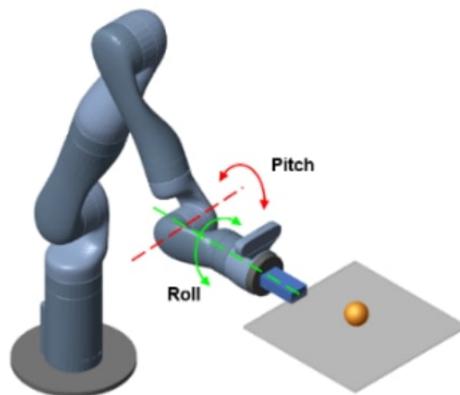
Nella continua ricerca di soluzioni avanzate per il controllo di sistemi robotici e automatici, la simulazione gioca un ruolo cruciale per testare e validare algoritmi di apprendimento automatico.

In questo capitolo verrà analizzata un'esperienza di simulazione in ambiente MATLAB e Simulink riguardante un agente implementato con l'algoritmo di Reinforcement Learning (RL) Soft Actor Critic (SAC), addestrato a controllare un braccio robotico per un compito di bilanciamento di una pallina. L'obiettivo di questa simulazione è quello di addestrare, tramite l'algoritmo di RL sopra riportato, l'agente a far permanere in equilibrio una pallina sopra un piano mantenuto da un braccio robotico. Quest'ultimo è un robot Kinova Gen3, un manipolatore a sette gradi di libertà (Seven Degree Of Freedom, o 7DOF). Il braccio ha il compito di bilanciare una pallina da ping pong al centro di una superficie piana (piatto) collegata alla pinza del robot. Soltanto gli ultimi due giunti sono attivati e contribuiscono al movimento sugli assi di beccheggio e rollio, come mostrato in Figura 3.1. I restanti giunti sono fissi e non contribuiscono al movimento.

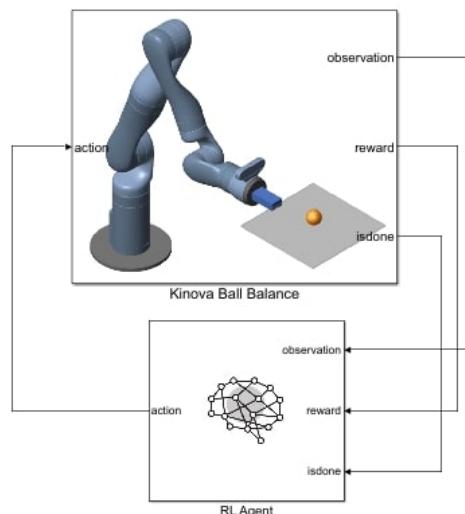
Aprendo il modello Simulink si può visualizzare il sistema. Il modello contiene un sottosistema Kinova Ball Balance collegato a un blocco agente RL. L'agente applica un'azione al sottosistema robot e riceve i segnali di osservazione, ricompensa e completamento risultanti (Figura 3.2). Il modello si può aprire con il comando:

```
open_system("rlKinovaBallBalance")
```

Come si può notare, il modello rispetta perfettamente la composizione di un modello di RL (Figura 1.1), nel quale l'agente si relaziona con l'ambiente attraverso delle azioni iterate, mentre l'ambiente, attraverso uno stato ed una ricompensa, si interfaccia con l'agente stesso, dal quale apprende e ottiene informazioni.



**Figura 3.1:** Rappresentazione del braccio robotico, piatto e pallina



**Figura 3.2:** Modello Simulink del sistema collegato con il blocco agente RL

In Figura 3.3 si può notare come è stato impostato il sottosistema del Kinova Ball Balance. Esso può essere aperto tramite il comando

```
open_system("rlKinovaBallBalance/Kinova Ball Balance")
```

In questo modello si possono notare diversi elementi, in particolare:

- i componenti fisici del sistema (manipolatore, sfera e piastra) sono modellati utilizzando i componenti Simscape™ Multibody™;
- il piatto è vincolato all’effettore<sup>1</sup> finale del manipolatore;
- la pallina ha sei gradi di libertà e può muoversi liberamente nello spazio;
- le forze di contatto tra la sfera e il piatto sono modellate utilizzando il blocco Spatial Contact Force;
- gli ingressi di controllo al manipolatore sono i segnali di coppia per i giunti azionati.

---

<sup>1</sup>Un effettore è un dispositivo periferico che si attacca al polso di un robot, consentendo a quest’ultimo di interagire per portare avanti il suo compito.

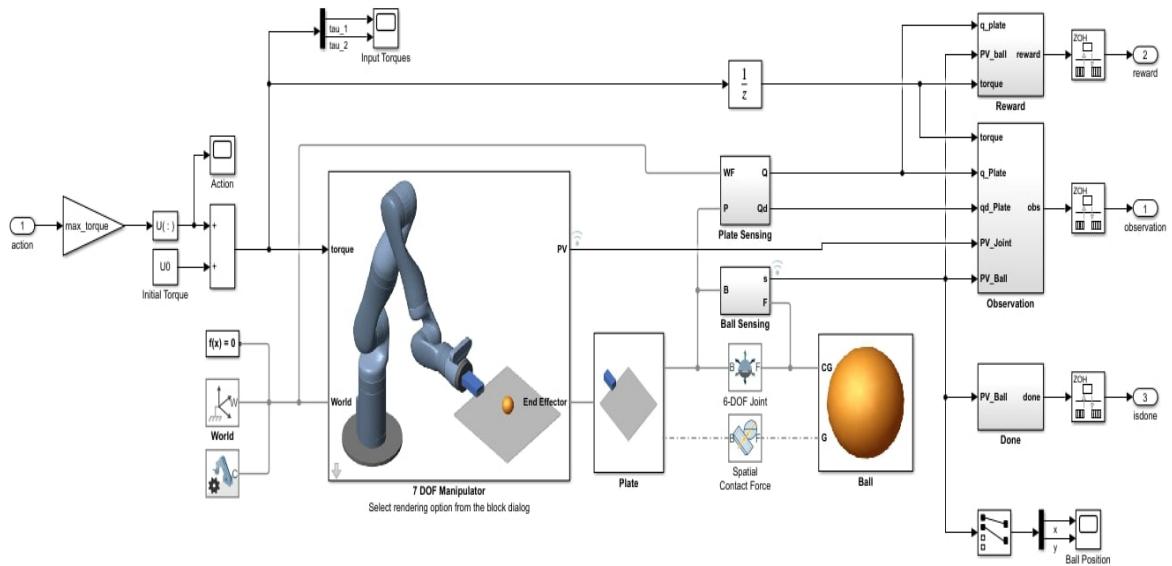


Figura 3.3: Sottosistema Kinova Ball Balance

### 3.2 Soft Actor Critic (SAC): un’analisi approfondita

L’algoritmo Soft Actor Critic (SAC) è un metodo di Reinforcement Learning *model-free*<sup>2</sup>, *off-policy* e con *actor-critic*<sup>3</sup>. L’algoritmo SAC calcola una politica ottimale che massimizza sia la ricompensa attesa a lungo termine sia l’entropia della politica. Quest’ultima è una misura dell’incertezza della politica in base allo stato. Un valore di entropia più alto promuove una maggiore esplorazione. La massimizzazione sia della ricompensa cumulativa attesa a lungo termine sia dell’entropia aiuta a bilanciare lo sfruttamento e l’esplorazione dell’ambiente. L’implementazione dell’agente SAC, in questo specifico caso, utilizza due critici della *Q-Value Function*, che impediscono la sovrastima della funzione valore. Altre implementazioni dell’algoritmo SAC utilizzano un ulteriore critico della funzione valore.

Gli agenti SAC possono essere addestrati in ambienti con gli spazi di osservazione e azione presenti in Tabella 3.1.

Observation Space	Action Space
Discrete or continuous	Continuous

Tabella 3.1: Tipo di ambiente nel quale un agente SAC può essere addestrato

Inoltre, essi utilizzano l’attore e il critico presenti in Tabella 3.2.

Durante l’addestramento, un agente SAC esegue le seguenti attività:

- aggiorna le proprietà dell’attore e del critico a intervalli regolari durante l’apprendimento;

<sup>2</sup>Un metodo di Reinforcement Learning è considerato "model-free" quando non richiede la costruzione o l’utilizzo di un modello esplicito dell’ambiente in cui l’agente opera.

<sup>3</sup>L’approccio Actor-Critic è un paradigma di Reinforcement Learning che combina elementi di due approcci principali: l’approccio basato sulla policy (attore) e l’approccio basato sulla valutazione dei valori (critico). Questa combinazione mira a sfruttare i vantaggi di entrambi, fornendo una strategia più stabile ed efficiente per l’addestramento degli agenti di RL.

Critics	Actor
Funzione di valore $Q$ critica $Q(S, A)$ , creata con <code>rlQValueFunction</code>	Attore di politica stocastica $\pi(S)$ , creato utilizzando <code>rlContinuousGaussianActor</code>

**Tabella 3.2:** Tipo di attore e critico utilizzati dagli agenti SAC

- stima la media e la deviazione standard di una distribuzione di probabilità gaussiana per lo spazio d’azione continuo, quindi seleziona casualmente le azioni in base alla distribuzione;
- aggiorna un termine di peso dell’entropia che bilancia il rendimento atteso e l’entropia della politica;
- memorizza le esperienze passate utilizzando un buffer circolare di esperienze. L’agente aggiorna l’attore e il critico utilizzando un mini-batch di esperienze campionate casualmente dal buffer.

Se l’opzione `UseExplorationPolicy` dell’agente è impostata a `false`, viene sempre utilizzata l’azione con la massima probabilità. Di conseguenza, l’agente simulato e la politica generata si comportano in modo deterministico. Se `UseExplorationPolicy` è impostato a `true`, l’agente seleziona le sue azioni campionando la sua distribuzione di probabilità. Di conseguenza, la politica è stocastica e l’agente esplora il suo spazio di osservazione. Questa opzione influisce solo sulla simulazione e sulla distribuzione ma non sull’addestramento.

### 3.2.1 Approssimatori di funzioni attore e critico

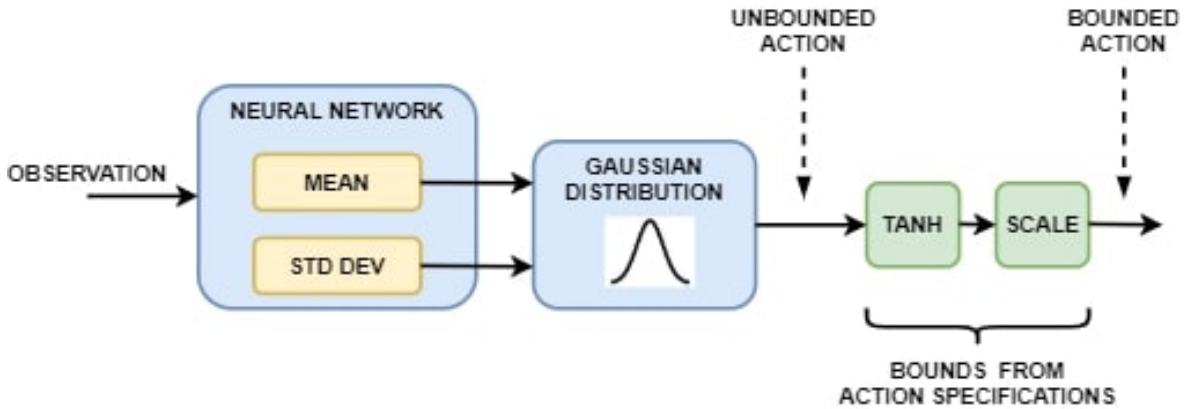
Per stimare la politica e la funzione valore, un agente SAC mantiene i seguenti approssimatori di funzione:

- *Attore stocastico*  $\pi(A|S; \theta)$  - L’attore, con il parametro  $\theta$ , fornisce la media e la deviazione standard della probabilità gaussiana condizionale di intraprendere ogni azione continua  $A$  quando si trova nello stato  $S$ .
- *Uno (o due) critici di valore*  $Q_k(S, A; \phi_k)$  - I critici, ciascuno con parametri  $\phi_k$ , prendono in ingresso l’osservazione  $S$  e l’azione  $A$  e restituiscono l’aspettativa corrispondente della Value Function, che comprende sia la ricompensa a lungo termine sia l’entropia.
- *Uno (o due) target critici*  $Q_{tk}(S, A; \phi_{tk})$  - Per migliorare la stabilità dell’ottimizzazione, l’agente impone periodicamente i parametri dei target critici  $\phi_{tk}$  sugli ultimi valori dei parametri dei critici corrispondenti. Il numero di target critici corrisponde al numero di critici.

Quando si utilizzano due critici,  $Q_1(S, A; \phi_1)$  e  $Q_2(S, A; \phi_2)$ , ciascuno può avere una struttura diversa. Quando i critici hanno la stessa struttura, devono avere valori di parametri iniziali differenti. Ogni critico  $Q_k(S, A; \phi_{tk})$  e il corrispondente critico target  $Q_{tk}(S, A; \phi_{tk})$  devono avere la stessa struttura e parametrizzazione. Durante l’addestramento, l’agente sintonizza i valori dei parametri in  $\theta$ . Al termine dell’addestramento, i parametri rimangono al loro valore sintonizzato e l’approssimatore della funzione attore addestrato viene memorizzato in  $\pi(A|S)$ .

### Generazione di azioni

L’attore di un agente SAC genera uscite con media e deviazione standard. Per selezionare un’azione, l’attore sceglie prima in modo casuale un’azione non vincolata da una distribuzione gaussiana con i suddetti parametri. Durante l’addestramento, l’agente SAC utilizza la distribuzione di probabilità non vincolata per calcolare l’entropia della politica per l’osservazione data. Se lo spazio delle azioni dell’agente SAC è delimitato, l’attore genera azioni delimitate applicando le operazioni di *tanh* e *scaling* alle azioni non delimitate (Figura 3.4).



**Figura 3.4:** Funzionamento della generazione di azioni per un agente SAC

### 3.2.2 Algoritmo di addestramento

Gli agenti SAC utilizzano il seguente algoritmo di addestramento, in cui aggiornano periodicamente i loro modelli di attore e critico e il peso dell’entropia. In tale algoritmo,  $K = 2$  è il numero di critici e  $k$  è l’indice dei critici.

L’algoritmo è così definito:

- Inizializzare ogni critico  $Q_k(S, A; \phi_k)$  con valori casuali dei parametri  $\phi_k$  e inizializzare ogni critico target con gli stessi valori casuali dei parametri:  $\phi_{tk} = \phi_k$ .
- Inizializzare l’attore  $\pi(S; \theta)$  con i valori casuali dei parametri  $\theta$ .
- Eseguire una sequenza di azioni seguendo la politica casuale iniziale in  $\pi(S)$ . Per ogni azione, memorizzare l’esperienza nel buffer dell’esperienza.
- Per ogni passo temporale dell’addestramento:
  1. Per l’osservazione corrente  $S$ , selezionare l’azione  $A$  utilizzando la politica in  $\pi(S; \theta)$ .
  2. Eseguire l’azione  $A$ . Osservare la ricompensa  $R$  e la successiva osservazione  $S'$ .
  3. Memorizzare l’esperienza  $(S, A, R, S')$  nel buffer dell’esperienza.
  4. Campionare un mini-batch casuale di  $M$  esperienze  $(S_i, A_i, R_i, S'_i)$  dal buffer delle esperienze.
  5. Ad ogni passo temporale  $Dc$ , aggiornare i parametri di ogni critico, minimizzando la perdita  $L_k$  su tutte le esperienze campionate. La perdita  $L_k$  è definita nell’Equazione 3.2.1

$$L_k = \frac{1}{2M} \sum_{i=1}^M (y_i - Q_k(S_i, A_i; \phi_k))^2 \quad (3.2.1)$$

Se  $S'_i$  è uno stato terminale, l’obiettivo della funzione valore  $y_i$  è uguale alla ricompensa dell’esperienza  $R_i$ . Altrimenti, l’obiettivo della funzione valore è la somma di  $R_i$ , della ricompensa futura minima attualizzata dei critici e dell’entropia ponderata. La Value Function  $y_i$  è definita nell’Equazione 3.2.2

$$y_i = R_i + \gamma * \min_k(Q_{tk}(S'_i, A'_i; \phi_{tk})) - \alpha \ln \pi(S'_i; \theta) \quad (3.2.2)$$

In tale equazione,  $A'_i$  è l’azione vincolata derivata dall’output non vincolato dell’attore  $\pi(S'_i)$ ,  $\gamma$  è il fattore di sconto e  $-\alpha \ln \pi(S, \theta)$  è l’entropia ponderata della politica per l’output vincolato dell’attore quando si trova nello stato  $S$ , con  $\alpha$  che è il peso della perdita di entropia.

6. Ad ogni passo temporale  $S_A$ , aggiornare i parametri dell’attore minimizzando la funzione obiettivo presente nell’Equazione 3.2.3.

$$J_\pi = \frac{1}{M} \sum_{i=1}^M (-\min_k(Q_k(S_i, A_i; \phi_k)) + \alpha \ln \pi(S_i; \theta)) \quad (3.2.3)$$

7. Ad ogni passo temporale  $D_A$ , aggiornare anche il peso dell’entropia minimizzando la funzione di perdita presente nell’Equazione 3.2.4.

$$L_\alpha = \frac{1}{M} \sum_{i=1}^M (-\alpha \ln \pi(S_i; \theta) - \alpha \xi) \quad (3.2.4)$$

Dove  $\xi$  è l’entropia target.

8. Ad ogni passo  $D_T$ , aggiornare i criteri di destinazione in base al metodo di aggiornamento della destinazione.
9. Ripetere i passaggi da 4 a 8 per  $N_G$  volte, dove  $N_G$  è il numero di gradienti.

### 3.2.3 Metodi di aggiornamento del target

Gli agenti SAC aggiornano i parametri critici del proprio target utilizzando uno dei seguenti metodi di aggiornamento:

- *Smoothing* - aggiorna i parametri critici target a ogni passo temporale utilizzando il fattore di smoothing  $\tau$ . La funzione alla base di questo metodo è definita nell’Equazione 3.2.5.

$$\phi_{tk} = \tau \phi_k + (1 - \tau) \phi_{tk} \quad (3.2.5)$$

- *Periodic* - aggiorna periodicamente i parametri critici del target senza il fattore di smoothing. La funzione corrispondente è definita nell’Equazione 3.2.6.

$$\phi_{tk} = \phi_k \quad (3.2.6)$$

- *Periodic smoothing* - aggiorna periodicamente i parametri di destinazione con il fattore di smoothing.

### 3.3 Progettazione e configurazione del simulatore

In questa sezione verranno trattate la progettazione e la configurazione del simulatore stesso; in particolare, verranno prese in considerazione la definizione dell’ambiente e la creazione dell’agente; oltre a ciò verrà trattata l’implementazione dell’addestramento di quest’ultimo. Tutto ciò verrà fatto prendendo in considerazione l’ambiente di sviluppo citato nella Sezione 3.1, quindi MATLAB e Simulink.

#### 3.3.1 Definizione dell’ambiente

Per addestrare un agente di Reinforcement Learning è necessario definire l’ambiente con cui interagirà.

Per l’ambiente di bilanciamento della palla vengono prese in considerazione le seguenti implementazioni:

- Le osservazioni sono rappresentate da un vettore di 22 elementi che contiene informazioni sulle posizioni (seno e coseno degli angoli dei giunti) e sulle velocità (derivate degli angoli dei giunti) dei due giunti azionati, sulle posizioni (distanze  $x$  e  $y$  dal centro del piatto) e sulle velocità (derivate  $x$  e  $y$ ) della sfera, sull’orientamento (quaternioni<sup>4</sup>) e sulle velocità (derivate dei quaternioni) del piatto, sulle coppie dei giunti dell’ultimo passo temporale, sul raggio della sfera e sulla massa.
- Le azioni sono valori di coppia normalizzati dei giunti.
- Il tempo di campionamento è  $T_s = 0,01_s$  e il tempo di simulazione è  $T_f = 10_s$ .
- La simulazione termina quando la palla cade dal piatto.
- La ricompensa  $r_t$  al passo temporale  $t$  è data dall’Equazione 3.3.1.

$$r_t = r_{ball} + r_{plate} + r_{action} \quad (3.3.1)$$

In essa:

1.  $r_{ball} = e^{-0.001(x^2+y^2)}$
2.  $r_{plate} = -0.1(\phi^2 + \theta^2 + \psi^2)$
3.  $r_{action} = -0.05(\tau_1^2 + \tau_2^2)$

Qui,  $r_{ball}$  è una ricompensa per l’avvicinamento della palla al centro del piatto,  $r_{plate}$  è una penalità per l’orientamento del piatto e  $r_{action}$  è una penalità per lo sforzo di controllo.  $\phi$ ,  $\theta$  e  $\psi$  sono i rispettivi angoli di rollio, beccheggio e imbardata del piatto, espressi in radianti.  $\tau_1$  e  $\tau_2$  sono le coppie dei giunti.

Vengono, poi, create le specifiche di osservazione e azione per l’ambiente, utilizzando spazi di osservazione e azione continui.

```
numObs = 22;    % Number of dimension of the observation space
numAct = 2;    % Number of dimension of the action space

obsInfo = rlNumericSpec([numObs 1]);
```

---

<sup>4</sup>Un quaternione è una struttura matematica che estende i numeri complessi. Mentre i numeri complessi sono rappresentati da due componenti (parte reale e parte immaginaria), i quaternioni sono rappresentati da quattro componenti.

```
actInfo = rlNumericSpec([numAct 1]);
actInfo.LowerLimit = -1;
actInfo.UpperLimit = 1;
```

Definiamo l’interfaccia dell’ambiente Simulink utilizzando le specifiche di osservazione e azione.

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

A questo punto, specifichiamo una funzione di reset per l’ambiente utilizzando il parametro `ResetFcn`.

```
env.ResetFcn = @kinovaResetFcn;
```

Questa funzione (fornita in Appendice A) inizializza in modo casuale le posizioni iniziali  $x$  e  $y$  della palla rispetto al centro del piatto. Per un addestramento più robusto, è possibile randomizzare anche altri parametri all’interno della funzione di reset, come la massa e il raggio della pallina.

Verranno specificati, anche, il tempo di campionamento  $T_s$  e il tempo di simulazione  $T_f$ .

```
Ts = 0.01;
Tf = 10;
```

### 3.3.2 Creazione dell’agente

L’agente in questo esempio è un agente Soft Actor Critic (SAC). Come già ampiamente discusso nella sezione precedente, gli agenti SAC utilizzano uno o due approssimatori di funzioni Q-value parametrizzate per stimare il valore della politica. Un critico di funzione Q-value riceve in ingresso l’osservazione corrente e un’azione, e restituisce in uscita un singolo scalare.

L’agente SAC di questo esempio utilizza due critici. Per modellare le funzioni Q-value parametrizzate all’interno dei critici, si utilizzerà una rete neurale con due strati di ingresso (uno per il canale di osservazione, specificato da `obsInfo`, e l’altro per il canale di azione, specificato da `actInfo`) e uno strato di uscita (che restituisce il valore scalare).

Bisogna definire ogni percorso di rete come una matrice di oggetti "layer" (livello). Assegneremo, poi, i nomi ai livelli di ingresso e di uscita di ciascun percorso. Questi nomi consentono di collegare i percorsi e, successivamente, di associare esplicitamente i livelli di ingresso e di uscita della rete al canale ambientale appropriato.

```
% Set the random seed for reproducibility.
rng(0)

% Define the network layers.
criticNet = [
    featureInputLayer(numObs,Name="obsInLyr")
    fullyConnectedLayer(128)
    concatenationLayer(1,2,Name="concat")
    reluLayer
    fullyConnectedLayer(64)
```

```

reluLayer
fullyConnectedLayer(32)
reluLayer
fullyConnectedLayer(1,Name="QValueOutLyr")
];

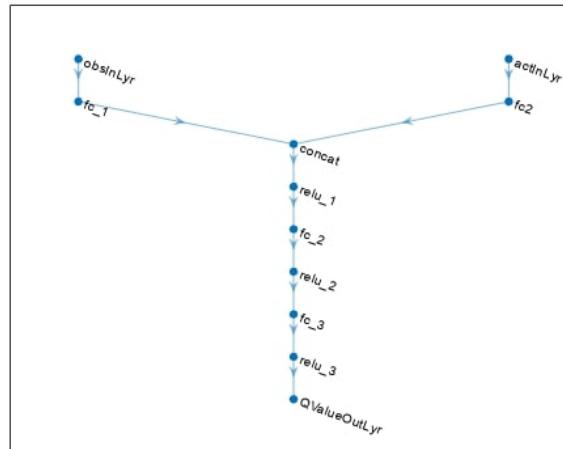
actionPath = [
    featureInputLayer(numAct,Name="actInLyr")
    fullyConnectedLayer(128,Name="fc2")
];

% Connect the layers.
criticNet = layerGraph(criticNet);
criticNet = addLayers(criticNet, actionPath);
criticNet = connectLayers(criticNet,"fc2","concat/in2");

```

C'è anche la possibilità di "plottare" la rete neurale del critico (Figura 3.5).

```
plot(criticNet)
```



**Figura 3.5:** Rete neurale del critico

Quando si utilizzano due critici, un agente SAC richiede che essi abbiano due parametri iniziali diversi. Verranno, quindi, creati e inizializzati due oggetti `dlnetwork`.

```

criticDLnet = dlnetwork(criticNet,'Initialize',false);
criticDLnet1 = initialize(criticDLnet);
criticDLnet2 = initialize(criticDLnet);

```

Successivamente, verranno, anche, create le funzioni critiche utilizzando `rlQValueFunction`.

```

critic1 = rlQValueFunction(criticDLnet1,obsInfo,actInfo, ...
    ObservationInputNames="obsInLyr");
critic2 = rlQValueFunction(criticDLnet2,obsInfo,actInfo, ...
    ObservationInputNames="obsInLyr");

```

Gli agenti SAC utilizzano una politica stocastica parametrizzata su uno spazio di azione continuo, che viene implementata da un attore continuo gaussiano. Questo attore riceve in ingresso un'osservazione e restituisce in uscita un'azione casuale campionata da una distribuzione di probabilità gaussiana.

Per approssimare i valori medi e le deviazioni standard della distribuzione gaussiana, è necessario utilizzare una rete neurale con due livelli di uscita, ciascuno dei quali ha tanti elementi quante sono le dimensioni dello spazio delle azioni. Uno strato di uscita deve restituire un vettore contenente i valori medi per ogni dimensione dell'azione. L'altro strato di uscita deve restituire un vettore contenente la deviazione standard per ogni dimensione dell'azione. Poiché le deviazioni standard devono essere non negative, è necessario utilizzare un livello softplus o ReLU<sup>5</sup> per imporre la non negatività. L'agente SAC legge automaticamente l'intervallo di azione dalle proprietà UpperLimit e LowerLimit di actInfo (utilizzate per creare l'attore), quindi scala internamente la distribuzione e delimita l'azione. Pertanto, non è necessario aggiungere un tanhLayer<sup>6</sup> come ultimo livello non lineare nel percorso di uscita medio.

Verrà, quindi, definito ogni percorso di rete come una matrice di oggetti layer e si assegneranno i nomi ai layer di ingresso e di uscita di ogni percorso.

```
% Create the actor network layers.
commonPath = [
    featureInputLayer(numObs, Name="obsInLyr")
    fullyConnectedLayer(128)
    reluLayer
    fullyConnectedLayer(64)
    reluLayer(Name="comPathOutLyr")
];

meanPath = [
    fullyConnectedLayer(32, Name="meanFC")
    reluLayer
    fullyConnectedLayer(numAct, Name="meanOutLyr")
];

stdPath = [
    fullyConnectedLayer(numAct, Name="stdFC")
    reluLayer
    softplusLayer(Name="stdOutLyr")
];

% Connect the layers.
actorNetwork = layerGraph(commonPath);
actorNetwork = addLayers(actorNetwork, meanPath);
actorNetwork = addLayers(actorNetwork, stdPath);
actorNetwork = connectLayers(actorNetwork, "comPathOutLyr", "meanFC/in");
actorNetwork = connectLayers(actorNetwork, "comPathOutLyr", "stdFC/in");
```

A questo punto, si creerà la funzione dell'attore utilizzando `rlContinuousGaussianActor`.

---

<sup>5</sup>In MATLAB, i termini "softplus" e "ReLU" si riferiscono a funzioni di attivazione comunemente utilizzate nelle reti neurali, utilizzati per introdurre la non linearità.

<sup>6</sup>In MATLAB, il "tanhLayer" rappresenta un layer di tangente iperbolica (*tanh*) utilizzato nelle reti neurali per introdurre la non linearità.

```

actordlnet = dlnetwork(actorNetwork);
actor = rlContinuousGaussianActor(actordlnet, obsInfo, actInfo, ...
    ObservationInputNames="obsInLyr", ...
    ActionMeanOutputNames="meanOutLyr", ...
    ActionStandardDeviationOutputNames="stdOutLyr");

```

In questo esempio, l'agente SAC si allena a partire da un buffer di esperienza di capacità massima  $1e6$  ( $10^6$ ), selezionando in modo casuale mini-lotti di dimensione 128. Il fatto che il fattore di sconto sia 0,99, cioè molto vicino a 1, significa che l'agente tiene maggiormente in considerazione le ricompense a lungo termine (un fattore di sconto più vicino a 0, invece, farebbe pesare di meno le ricompense future, favorendo quindi quelle a breve termine).

Si specificano, anche, gli iperparametri dell'agente per l'addestramento.

```

agentOpts = rlsACAgentOptions( ...
    SampleTime=Ts, ...
    TargetSmoothFactor=1e-3, ...
    ExperienceBufferLength=1e6, ...
    MiniBatchSize=128, ...
    NumWarmStartSteps=1000, ...
    DiscountFactor=0.99);

```

Per questo esempio, le reti neurali attore e critico vengono aggiornate utilizzando l'algoritmo Adam<sup>7</sup> con un tasso di apprendimento di  $1e-4$  ( $10^{-4}$ ) e una soglia di gradiente di 1. Si specificano, quindi, i parametri dell'ottimizzatore.

```

agentOpts.ActorOptimizerOptions.Algorithm = "adam";
agentOpts.ActorOptimizerOptions.LearnRate = 1e-4;
agentOpts.ActorOptimizerOptions.GradientThreshold = 1;

for ct = 1:2
    agentOpts.CriticOptimizerOptions(ct).Algorithm = "adam";
    agentOpts.CriticOptimizerOptions(ct).LearnRate = 1e-4;
    agentOpts.CriticOptimizerOptions(ct).GradientThreshold = 1;
end

```

Infine, viene creato l'agente SAC.

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

### 3.3.3 Addestramento dell'agente

Per addestrare l'agente, bisogna specificare inizialmente le opzioni di addestramento utilizzando `rlTrainingOptions`. Per questo esempio, sono state utilizzate le seguenti opzioni:

- eseguire ogni allenamento per un massimo di 5000 episodi, con ogni episodio della durata massima di  $\lceil \frac{T_f}{T_s} \rceil$  time step;
- interrompere l'addestramento quando l'agente riceve una ricompensa cumulativa media superiore a 550 su 100 episodi consecutivi;

---

<sup>7</sup>L'algoritmo Adam (Adaptive Moment Estimation) è un algoritmo di ottimizzazione utilizzato per aggiornare iterativamente i pesi di una rete neurale durante il processo di apprendimento.

- se si desidera accelerare la formazione, impostare l'opzione `UseParallel` (addestramento in parallelo) su `true`.

```
trainOpts = rlTrainingOptions(...
    MaxEpisodes=6000, ...
    MaxStepsPerEpisode=floor(Tf/Ts), ...
    ScoreAveragingWindowLength=100, ...
    Plots="training-progress", ...
    StopTrainingCriteria="AverageReward", ...
    StopTrainingValue=675, ...
    UseParallel=false);
```

Si può, anche, specificare un flag di visualizzazione `doViz` che mostra l'animazione della pallina in un'animazione MATLAB. Per l'addestramento in parallelo, si disabilitano tutte le visualizzazioni.

```
if trainOpts.UseParallel
    % Disable visualization in Simscape Mechanics Explorer
    set_param(mdl, SimMechanicsOpenEditorOnUpdate="off");
    set_param(mdl+"/Kinova Ball Balance/7 DOF Manipulator", ...
        "VChoice", "None");
    % Disable animation in MATLAB figure
    doViz = false;
    save_system(mdl);
else
    % Enable visualization in Simscape Mechanics Explorer
    set_param(mdl, SimMechanicsOpenEditorOnUpdate="on");
    % Enable animation in MATLAB figure
    doViz = true;
end
```

È, anche, possibile registrare i dati di addestramento su disco utilizzando la funzione `rlDataLogger`. In questo esempio, si registrano l'esperienza dell'episodio e le perdite delle funzioni attore e critico. Le funzioni `logAgentLearnData` e `logEpisodeData` sono definite in Appendice A.

```
logger = rlDataLogger();
logger.AgentLearnFinishedFcn = @logAgentLearnData;
logger.EpisodeFinishedFcn = @(data) logEpisodeData(data, doViz);
```

Si addestrerà l'agente utilizzando la funzione `train`. Tale addestramento è un processo intensivo dal punto di vista computazionale e richiede diversi minuti per essere completato. Per risparmiare tempo durante l'esecuzione di questo esempio, si potrebbe caricare un agente preaddestrato impostando `doTraining` su `false`. Per addestrare l'agente da soli, è necessario impostare `doTraining` su `true`.

```
doTraining = false;
if doTraining
    trainResult = train(agent, env, trainOpts, Logger=logger);
else
    load("kinovaBallBalanceAgent.mat")
end
```

### Simulazione dell'agente addestrato

Per iniziare la simulazione dell'agente addestrato, si dovranno apportare ulteriori specificazioni. Innanzitutto, verrà impostata la posizione iniziale della palla rispetto al centro del piatto. Per randomizzare la posizione iniziale della palla durante la simulazione si deve impostare il flag `UserSpecifiedConditions` a `false`.

```
userSpecifiedConditions = true;
if userSpecifiedConditions
    ball.x0 = 0.10;
    ball.y0 = -0.10;
    env.ResetFcn = [];
else
    env.ResetFcn = @kinovaResetFcn;
end
```

A questo punto, verrà creato un oggetto "opzioni di simulazione" per configurare la simulazione. L'agente sarà simulato per un massimo di  $[(\frac{T_f}{T_s})]$  passi per ogni episodio di simulazione:

```
simOpts = rlSimulationOptions (MaxSteps=floor(Tf/Ts));
```

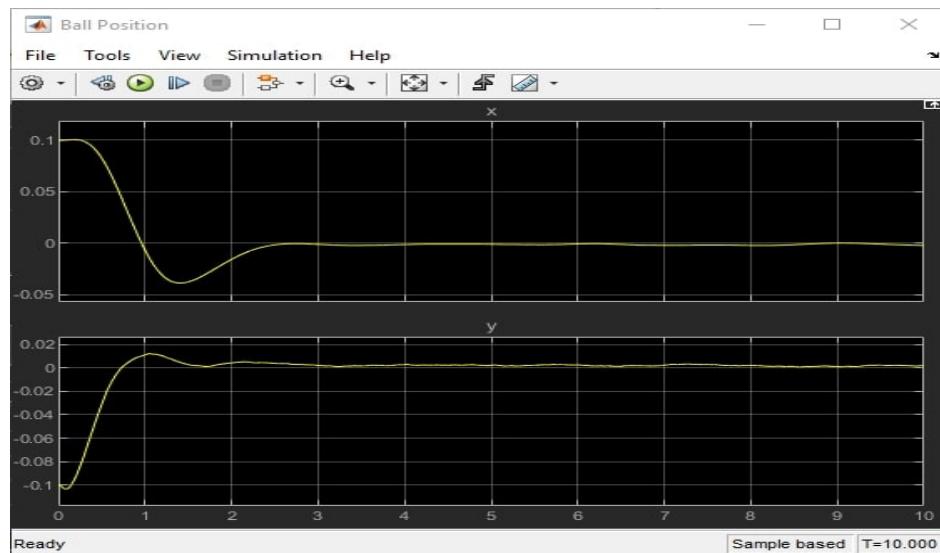
Bisogna, quindi, abilitare la visualizzazione durante la simulazione.

```
set_param(mdl, 'SimMechanicsOpenEditorOnUpdate','on');
doViz = true;
```

Infine, verrà simulato l'agente:

```
experiences = sim(agent,env,simOpts);
```

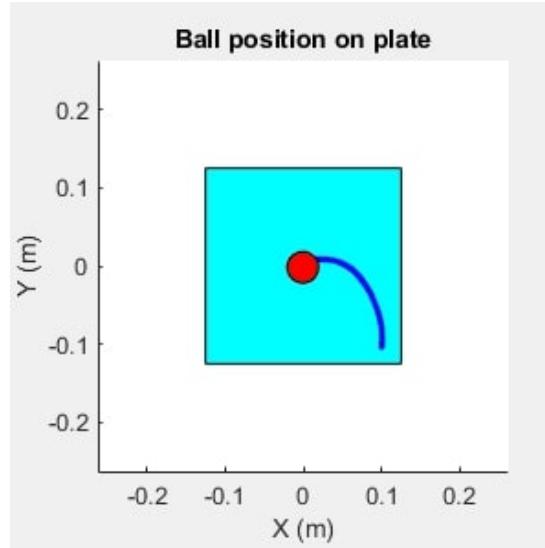
Durante la simulazione è possibile, anche, visualizzare la traiettoria della palla utilizzando il Ball Position scope block, come possiamo notare in Figura 3.6.



**Figura 3.6:** Traiettoria della palla tramite il Ball Position scope block

In alternativa, è possibile vedere l'animazione della palla nel piatto con un animazione MATLAB (Figura 3.7), implementando il seguente codice:

```
fig = animatedPath(experiences);
```



**Figura 3.7:** Animazione MATLAB della pallina sul piatto

## 3.4 Valutazione e analisi delle esperienze di simulazione 1

Prima di iniziare la discussione sull’analisi delle esperienze, è bene definire degli elementi comuni a tutte le simulazioni e ai loro relativi esperimenti. Tali elementi sono i seguenti:

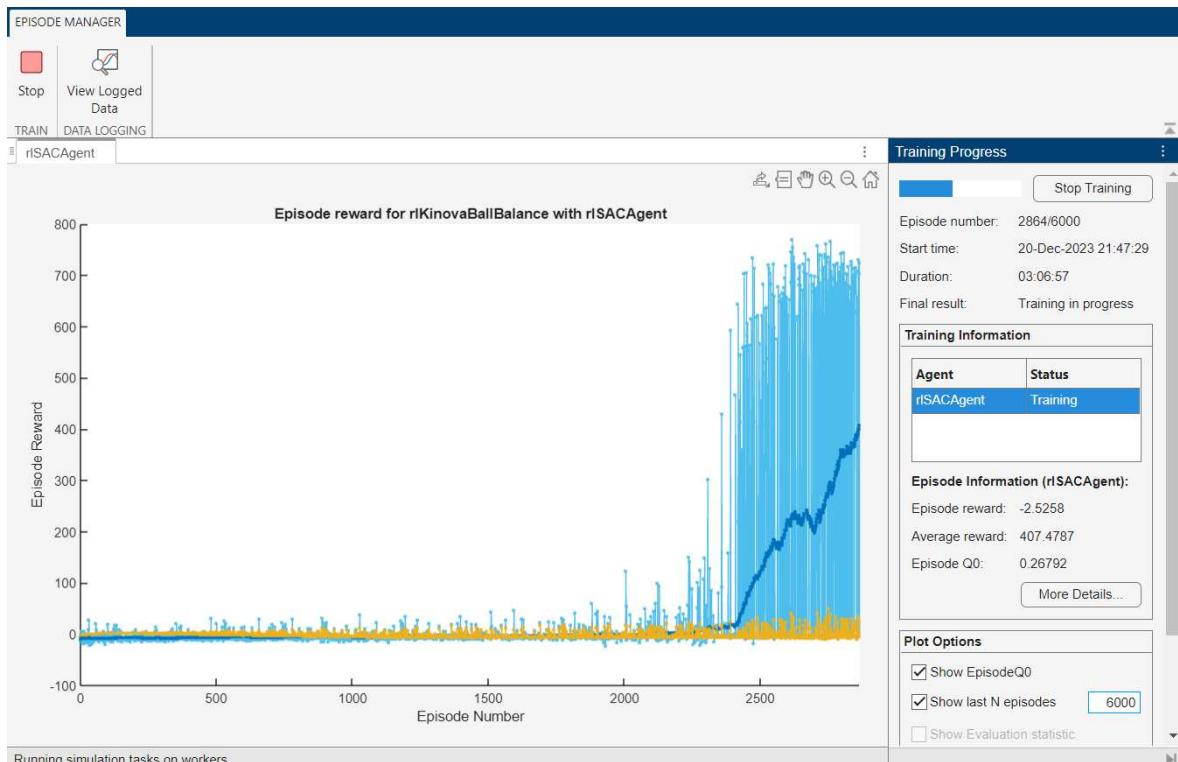
- *Average Reward* (o Ricompensa Media): è una misura statistica che rappresenta la media di tutte le ricompense ottenute da un agente durante l’intero processo di apprendimento. La ricompensa media fornisce una valutazione complessiva delle prestazioni dell’agente nel corso del tempo.
- *Episode Reward* (o Ricompensa Periodica): è una misura della somma totale delle ricompense ottenute da un agente durante l’intera durata di un singolo episodio. Da episodio a episodio ci possono essere variazioni significative di questo parametro.
- *Episode Q0*: 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.
- *Episodio*: rappresenta ogni singolo tentativo, 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.

Detto ciò, è doveroso anche dire che le esperienze che si valuteranno, per questo tipo di simulazione hanno in comune l’obiettivo di raggiungere un valore di Average Reward maggiore di 550 su 100 episodi consecutivi (come riportato nella sezione precedente), entro i 6000 episodi prefissati con i quali andrà ad interagire.

### 3.4.1 Esperienza 1

La prima esperienza viene effettuata sulla base del codice precedentemente riportato, quindi senza effettuare nessuna modifica ad esso, i risultati ottenuti sono come di seguito specificato.

Inizialmente, è stato preso un esempio campione, come si può notare in Figura 3.8, dove si possono vedere i dati inseriti in un grafico fino all'episodio 2864. Da tale esempio, si nota come non è ancora stato raggiunto l'obiettivo prefissato dell'average reward.



**Figura 3.8:** Campionamento dell'Esperienza 1 del simulatore Robot-Arm dopo 2864 episodi

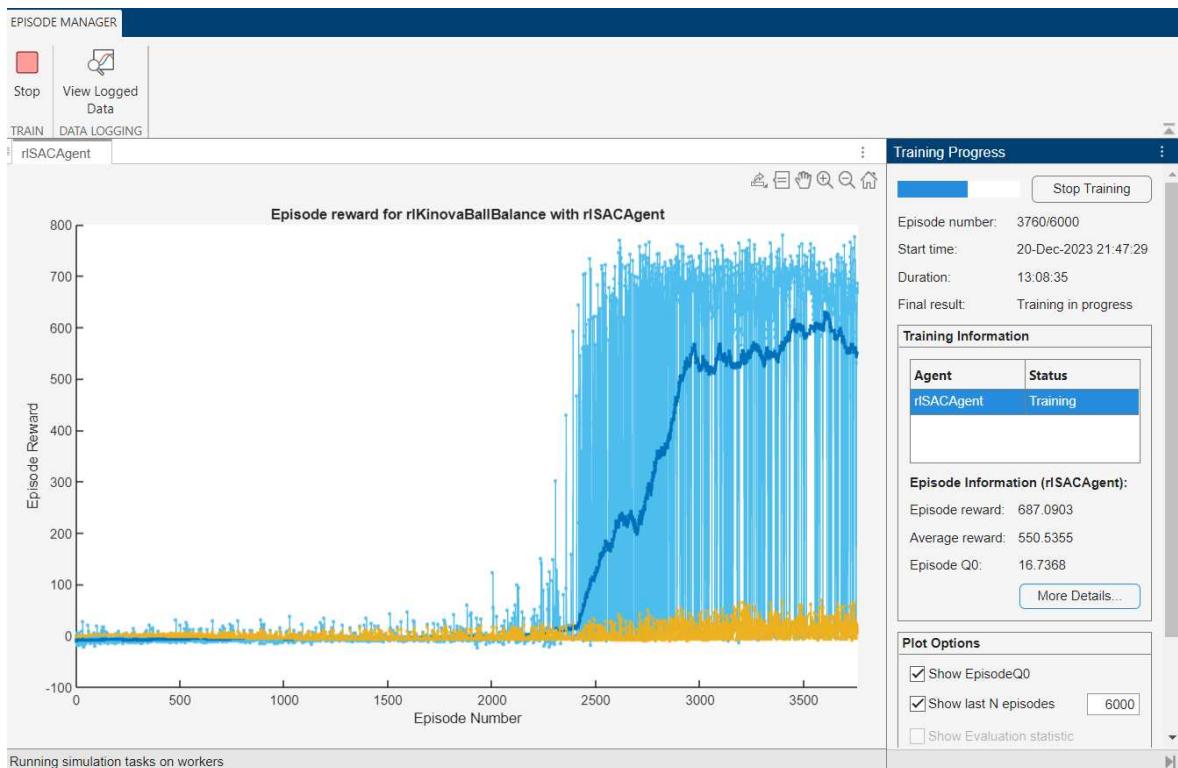
La curva dell'*episode Q0* (in giallo) è rimasta generalmente stabile dall'inizio fino al campionamento corrente, oscillando tra valori che variano da -20 a +80 circa; questo significa che l'agente, prendendo in considerazione le azioni passate ad ogni episodio, non ha mai considerato un'azione molto vantaggiosa rispetto ad altre, rimanendo sempre in un range ristretto di incertezza, anche se, da come si vede, acquisendo sempre più dati, questa curva risulta leggermente crescente verso la fine di tale campionamento. Da ciò si evince che esso sta acquisendo consapevolezza; di conseguenza, ritiene vantaggioso seguire un certo tipo di strada rispetto ad un'altra.

Riguardo la curva dell'*episode reward* (in azzurro), si può notare come, fino all'episodio 2300 (circa), essa rimane stabile anche se con qualche picco (tra i -30 e i +190), mentre, da quel punto in poi, inizia a crescere in maniera quasi esponenziale, arrivando a toccare picchi di reward di quasi +800. Questo significa che, inizialmente, l'agente sta sperimentando nuove strategie e, ad ogni episodio, sta apprendendo in base alle ricompense ottenute. Nella parte finale di tale campionamento, l'agente ha capito qual'è la strategia ottimale da seguire per arrivare al risultato prefissato e ha scelto autonomamente di vertere il suo comportamento verso essa, essendo tale strategia quella che faceva ottenere ad esso dei reward elevati.

Per quanto riguarda la curva dell'*average reward* (in blu), notiamo come abbia una tendenza generale alla stabilità, perlomeno fino all'intorno degli episodi 2300, cioè fino a dove inizia a crescere anche la curva dell'*episode reward* (ciò è interessante perché non sempre c'è una condizione causa-effetto, cioè non sempre al crescere della curva dell'*episode reward* si avrà una conseguente crescita dell'*average reward*), rimanendo comunque con una tendenza leggermente crescente. Dal range di episodi sopra riportato in poi, la curva ha un netto miglioramento, fino ad arrivare (perlomeno per questo campionamento) ad una ricompensa

media di +407, come si può notare in Figura 3.8. Questo notevole aumento dei valori della curva è un chiaro segnale che l’addestramento sta andando verso la giusta direzione.

Dopo aver esaminato questi risultati preliminari ottenuti durante questo campionamento iniziale, si possono analizzare i dati finali, presenti in Figura 3.9. È interessante notare come, per arrivare al risultato sperato, il tempo di simulazione è durato di più di 13 ore e ci sono voluti 3760 tentativi. Questo ci fa capire quanto, attraverso il RL, il tempo per addestrare un agente può arrivare ad essere lungo ed il numero di episodi che esso deve eseguire potrà essere elevato, prima di arrivare ad un risultato quanto più simile a quello che poi dovrà essere applicato nella realtà.



**Figura 3.9:** Risultato finale dell’Esperienza 1 per il simulatore Robot-Arm

A questo punto, si possono andare ad analizzare approssimativamente, le diverse fasi relative all’*average reward*. Nella fase iniziale (0-500), la curva presenta dei valori prettamente negativi, nell’intorno del -5, il che significa che le decisioni iniziali dell’agente non portano verso il risultato sperato. Dopo i primi 500 tentativi si può iniziare a notare un piccolo cambio di tendenza e si iniziano a vedere i primi valori di reward positivi; da qui in poi, infatti, si nota una stabilizzazione nell’intorno dello zero. La curva inizierà, successivamente, a crescere significativamente nell’intorno dell’episodio 2300, nel quale, come si nota in figura 3.9, si hanno anche valori di ricompensa di +100, o maggiori. Da questo punto in poi la curva dell’average reward cresce quasi esponenzialmente, anche se in qualche punto (per esempio nell’intorno dell’episodio 2700) riceve dei valori di reward minori degli episodi antecedenti, ma comunque ampiamente positivi. Durante la fase finale, si può notare una grande alternanza di valori di ricompense, anche se con tendenza sempre positiva. L’addestramento si ferma in automatico quando ha raggiunto un valore medio pari a 550 nell’arco degli ultimi 100 episodi, come si può notare sia graficamente che numericamente in Figura 3.9.

Oltre all’andamento della curva precedente è interessante notare l’andamento della curva *episode reward*. Inizialmente, quest’ultima mostra valori sia bassi che alti dai -20 ai +50; ciò indica prestazioni inizialmente non ottimali da parte dell’agente. C’è una svolta

notevole che si verifica a partire intorno agli episodi 2300, quando iniziano ad apparire picchi positivi con valori intorno al 150, che rappresentano il raggiungimento di obiettivi positivi da parte dell’agente, il che indica un miglioramento nelle sue prestazioni. Nonostante questi miglioramenti è interessante notare che continuano ad esserci picchi negativi, anche se di poco minori allo zero, fino all’episodio 2500 circa. Questi picchi possono indicare situazioni in cui l’agente non riesce a raggiungere gli obiettivi o compie azioni non ottimali, che portano, appunto, ad un valore della ricompensa negativo.

Per quanto riguarda, invece, l’andamento della curva relativa all’*episode Q0*, si può dire che essa ha un andamento costante positivo; infatti, si può notare come, all’aumentare degli episodi, aumenti anche il reward atteso dall’agente stesso; ciò rappresenta un chiaro segnale del fatto che esso stesse seguendo la strada giusta per arrivare all’obiettivo atteso.

Avendo analizzato i dati ottenuti dall’Esperienza 1, di seguito, analizzeremo i dati acquisiti dalla seconda esperienza con questo simulatore.

### 3.4.2 Esperienza 2

Per l’esecuzione dell’Esperienza 2 sono state apportate delle modifiche al codice di base del simulatore. Successivamente, vengono analizzate tali modifiche.

La prima modifica effettuata riguarda gli iperparametri. Nel contesto dell’addestramento di modelli di Machine Learning e Reinforcement Learning, essi sono parametri esterni al modello stesso che devono essere impostati prima dell’inizio del processo di addestramento. A differenza dei parametri del modello, che vengono appresi durante il training, gli iperparametri sono fissati prima dell’addestramento e influenzano il comportamento globale del modello.

Tra questi, sono stati modificati l’`ExperienceBufferLength`, il `MiniBatchSize` ed il `DiscountFactor`.

Il primo, che rappresenta la dimensione massima del buffer di esperienza e permette di memorizzare le esperienze passate dell’agente durante l’addestramento, è stato impostato da un valore iniziale di `1e6` ad un valore pari a `1e5`. Questo implicherà che la dimensione del buffer delle esperienze dell’agente diminuirà drasticamente.

Il secondo, invece, che indica il numero di esempi utilizzati in ogni iterazione dell’algoritmo di ottimizzazione durante l’addestramento di una rete neurale, è stato inizializzato a 200, invece che a 128. Ciò significa, di fatto, una velocità maggiore di addestramento, una variabilità nell’aggiornamento dei pesi ed una memoria richiesta maggiore.

Il terzo, invece, che controlla quanto il modello considera le future ricompense, è stato impostato ad un valore di 0.1 invece del valore precedente pari a 0.99. Riguardo quest’ultimo, ad esempio, un valore più basso suggerisce che l’agente attribuisce meno peso alle ricompense future; ciò potrebbe essere adeguato in situazioni in cui le decisioni a breve termine sono più importanti; viceversa, un valore più alto suggerirà di dare più peso alle ricompense future, dando maggiore importanza alle decisioni a lungo termine.

```
agentOpts = rlSACAgentOptions( ...
    SampleTime=Ts, ...
    TargetSmoothFactor=1e-3, ...
    ExperienceBufferLength=1e5, ...
    MiniBatchSize=200, ...
    NumWarmStartSteps=1000, ...
    DiscountFactor=0.1);
```

La seconda modifica effettuata riguarda il tasso di apprendimento, ovvero il `LearnRate`. Questa è una delle decisioni più importanti riguardo l’addestramento di un modello di RL.

Infatti, il tasso di apprendimento influisce su quanto velocemente il modello si adatta ai dati di addestramento. Se esso è troppo alto, il modello può oscillare intorno al minimo globale, o addirittura divergere. Se è troppo basso, esso può richiedere molto tempo o il modello può convergere a un minimo locale. In questo caso, per non incorrere negli errori appena riportati, si è deciso di far variare il tasso di apprendimento da un valore iniziale pari a 1e-4 ad un valore di 1e-2. Questo significa che, dal valore iniziale (che è quello che viene utilizzato più spesso in questi tipi di simulazioni), il quale implica che gli aggiornamenti dei pesi sono più piccoli e l'algoritmo si adatta più lentamente ai dati di addestramento apprendendo lentamente e con maggiore precisione i dettagli dei dati, si passa ad un valore maggiore, il quale implica aggiornamenti dei pesi più grandi, e quindi un adattamento più rapido ai dati di addestramento.

```
agentOpts.ActorOptimizerOptions.Algorithm = "adam";
agentOpts.ActorOptimizerOptions.LearnRate = 1e-2;
agentOpts.ActorOptimizerOptions.GradientThreshold = 1;

for ct = 1:2
    agentOpts.CriticOptimizerOptions(ct).Algorithm = "adam";
    agentOpts.CriticOptimizerOptions(ct).LearnRate = 1e-2;
    agentOpts.CriticOptimizerOptions(ct).GradientThreshold = 1;
end
```

Una volta introdotte le modifiche, si possono analizzare i dati per l'Esperienza 2, riportati in Figura 3.10. Si può notare da subito che la seconda esperienza ha prodotto risultati insoddisfacenti, poiché, anche se sono stati sfruttati tutti i 6000 tentativi messi a disposizione, l'addestramento non è arrivato all'obiettivo prefissato, anzi ha prodotto solo risultati negativi, anche se si è concluso in un quarto del tempo dell'esperienza precedente.

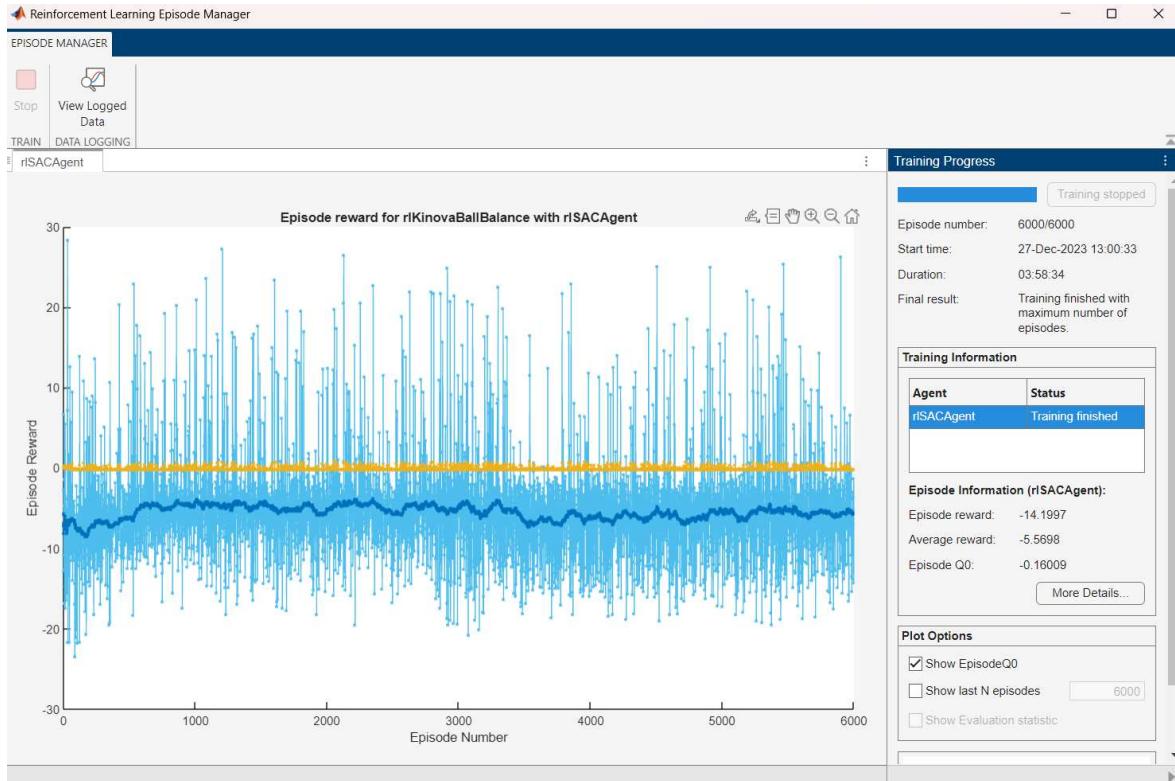
Come nel caso precedente, si possono analizzare le diverse fasi dell'andamento delle curve.

L'andamento della curva *average reward* è pressoché costantemente negativo. Infatti, dalla Figura 3.10, si può notare che, al contrario dell'esperienza precedente, dove da un certo intorno di episodi in poi l'andamento passa da essere negativo ad essere positivo, in questo caso esso rimane costante e sempre sotto lo zero, variando in un intervallo tra -8 e -5. Si può, quindi, dire che non c'è mai un vero e proprio punto di svolta durante tutto il processo di addestramento, portando, così, ad un risultato non ottimale; il valore medio finale, infatti, è pari a circa -5, molto lontano dal valore atteso.

Riguardo l'andamento della curva *episode reward*, invece, si può dire che, per quanto sia costante durante tutto l'addestramento, esso presenta un'alternanza di picchi positivi e negativi durante il processo nella sua costanza. Questi ultimi variano da un valore di circa -20 fino ad un valore di circa +28. Il fatto che non ci siano mai stati dei picchi di ricompensa, è segno che l'agente non è riuscito a lavorare in maniera ottimale e che qualcosa è andato inevitabilmente storto.

L'andamento della curva *episode Q0* è, in questo caso, costante ma non crescente. Infatti, anche quest'ultimo rimane sempre nell'intorno del valore di ricompensa 0 e non presenta picchi, né positivi né negativi, il che significa che il processo non si è mai aspettato, per tutta la durata del processo, valori di ricompensa elevati in nessun episodio, come poi è successo.

Avendo ottenuto dei risultati così scadenti con questa esperienza, si è deciso di effettuare un'ulteriore esperienza (cambiando nuovamente i parametri), per completezza di ricerca e per verificare con certezza sotto quali condizioni questo tipo di simulazione, con questo specifico tipo di implementazione, possa dare i migliori risultati. Alla fine del capitolo verranno fornite delle conclusioni basate sui risultati ottenuti dalle tre esperienze.



**Figura 3.10:** Risultato finale dell’Esperienza 2 simulatore Robot-Arm

### 3.4.3 Esperienza 3

Come sopra specificato, per l’esecuzione dell’Esperienza 3, sono state apportate nuovamente delle modifiche al codice iniziale del simulatore. Di seguito, vengono analizzate tali modifiche.

Si è deciso inizialmente di effettuare delle modifiche agli stessi iperparametri precedenti per cercare di capire se il problema riscontrato dall’Esperienza 2 fosse effettivamente dovuto al cambio dei valori di questi ultimi.

Per quanto riguarda l’`ExperienceBufferLength`, è stato modificato il valore precedente da `1e5` a `1e7`, così da avere un buffer di esperienza maggiore, aumentato di un fattore 10, e permettere, in questo modo, all’agente di verificare in un intervallo più ampio di esperienze passate e trovare tra più possibili scelte quella che si adattasse meglio a quella corrente.

Il valore del `MiniBatchSize` è stato portato a 164, facendo, così, una media tra il valore della prima esperienza (128) e quello della seconda (200). Questo farà sì che gli esempi di addestramento ad ogni iterazione diminuiscano rispetto alla seconda esperienza, ma ciò può essere una buona cosa; infatti, un `MiniBatchSize` più piccolo introduce una maggiore varietà e un maggior rumore stocastico nei campioni di addestramento in ogni iterazione. Ciò può aiutare a evitare che il modello si adatti eccessivamente (*overfitting*) ai dati di addestramento.

Si è deciso, anche, di modificare il `DiscountFactor`, come per l’iperparametro precedente, facendo una media tra quello della prima esperienza (0.99) e quello della seconda (0.1); pertanto, esso, è stato impostato a 0.5. In questo modo, l’agente dovrà dare più importanza alle scelte a breve termine, senza, però, tralasciare quelle lungo termine. Facendo così, esso avrà, quindi, più libertà di scelta e tenderà ad adattarsi meglio (in generale).

```
agentOpts = rlsACAgentOptions( ...
    SampleTime=Ts, ...
    TargetSmoothFactor=1e-3, ...)
```

```
ExperienceBufferLength=1e7, ...
MiniBatchSize=164, ...
NumWarmStartSteps=1000, ...
DiscountFactor=0.5);
```

Per quanto riguarda le seconde modifiche effettuate, anche in questo caso è stato cambiato il valore del `LearnRate`, ovvero il tasso di addestramento. Il valore scelto è pari alla media tra i valori della prima esperienza ( $1e-4$ ) e quelli della seconda ( $1e-2$ ), ovvero  $1e-3$ . Ciò farà sì che l'algoritmo avrà una capacità di adattamento media ai dati di addestramento rispetto ai precedenti due casi, ma anche aggiornamenti dei pesi più piccoli rispetto alla seconda esperienza.

```
agentOpts.ActorOptimizerOptions.Algorithm = "adam";
agentOpts.ActorOptimizerOptions.LearnRate = 1e-3;
agentOpts.ActorOptimizerOptions.GradientThreshold = 1;

for ct = 1:2
    agentOpts.CriticOptimizerOptions(ct).Algorithm = "adam";
    agentOpts.CriticOptimizerOptions(ct).LearnRate = 1e-3;
    agentOpts.CriticOptimizerOptions(ct).GradientThreshold = 1;
end
```

Introdotte le nuove modifiche, si possono analizzare i dati per l'Esperienza 3. Essi sono riportati in Figura 3.11.



**Figura 3.11:** Risultato finale dell'Esperienza 3 simulatore Robot-Arm

Come si può notare da subito, in Figura 3.11 i risultati sono stati insoddisfacenti, poiché, anche in questo caso, sono stati sfruttati tutti i 6000 tentativi messi a disposizione, ma l'addestramento non è arrivato all'obiettivo prefissato, anzi ha prodotto solo risultati negativi.

È interessante notare come l'esperienza si è conclusa in circa un terzo del tempo rispetto alla prima esperienza.

A questo punto, possiamo analizzare le diverse curve e il loro andamento. In realtà, si può dire che l'andamento delle 3 curve è pressoché identico a quello dell'esperienza precedente, con l'unica differenza che, per quanto riguarda la curva dell'*average reward*, si nota (sia graficamente che analiticamente) un leggero miglioramento del risultato. Infatti, come si può leggere in Figura 3.11, il risultato finale dell'esperimento, per quanto non sia minimamente paragonabile con quello atteso, è leggermente migliorato rispetto a quello dell'Esperienza 2, la quale era terminata con un valore pari a circa -7, al contrario di questa esperienza, terminata con un valore di *average reward* pari a circa -4.

Le curve di *episode Q0* e di *episode reward*, come sopra riportato, e come si può notare confrontando la Figura 3.11 con la Figura 3.10, sono quasi identiche. Ciò, naturalmente, significa che non ci sono stati miglioramenti, anche avendo modificato il codice.

Nella prossima sezione verranno stilate delle conclusioni sulla base dei dati ottenuti e delle considerazioni fatte per le tre differenti esperienze.

## 3.5 Conclusioni

In conclusione, questo capitolo ha offerto una panoramica completa dell'esperienza di simulazione, focalizzandosi su diversi aspetti chiave. L'introduzione all'esperienza ha fornito un contesto iniziale, delineando gli obiettivi e le motivazioni alla base della simulazione. Successivamente, l'analisi approfondita del modello Soft Actor Critic ha fornito una comprensione dettagliata dell'algoritmo di Reinforcement Learning utilizzato in questa simulazione. La sezione sulla progettazione e configurazione del simulatore ha esplorato le scelte di progettazione implementate nel codice, evidenziando l'implementazione della simulazione stessa.

Infine, nella sezione di valutazione e analisi delle esperienze di simulazione, sono state esaminate tre esperienze specifiche, ciascuna avente lo stesso codice, ma con parametri diversi e modificati opportunamente, per valutare le prestazioni e le dinamiche apprese dall'agente nei tre differenti casi presi in esame.

Da essi si può evincere che l'Esperienza 1, cioè quella in cui il codice iniziale non è stato modificato, è stata l'unica che ha portato a dei risultati concreti. Con molta probabilità, ciò è dovuto al fatto che, basandosi sui dati ottenuti dalle Esperienze 2 e 3, con questo tipo di implementazione e con il modello Soft Actor Critic (SAC), l'agente riesce ad apprendere meglio con l'iperparametro *DiscountFactor* elevato, come nel primo caso preso in esame (0.99). Quindi, significa, che, per questo tipo di esperienza, l'agente deve dare molta importanza alle ricompense future e alle decisioni a lungo termine. Si è arrivati a questa conclusione poiché, nelle Esperienze 2 e 3, i *DiscountFactor* erano stati inizializzati, rispettivamente, a 0.1 e 0.5; questi valori non permettono di dare molta importanza alle decisioni a lungo termine. Nell'Esperienza 3, dove è stato incrementato il valore del *DiscountFactor* rispetto all'Esperienza 2, c'è stato un miglioramento, anche se non netto (probabilmente colpa, anche, dell'incompatibilità degli altri valori che sono stati modificati).

Un altro dato interessante da considerare è, però, il *LearnRate*. Infatti, dai dati ottenuti, si deduce che, aumentando il tasso di apprendimento, si diminuisce notevolmente il tempo di addestramento. Si può dire però che, oltre a questo fatto, con molta probabilità, aumentando tale, l'agente ha meno tempo di "ragionare" sull'addestramento stesso; le sue azioni sono, così, più casuali e meno mirate all'obiettivo specifico, portando, al fallimento dell'addestramento.

Per quanto riguarda il *MiniBatchSize* e l'*ExperienceBufferSize*, si può dire che sono due concetti molto legati tra loro per definizione stessa. Naturalmente una lunghezza del buffer troppo piccola rispetto alla dimensione degli esempi di addestramento, farà sì di

incorrere in errore, perché, naturalmente, il MiniBatchSize non saprebbe dove cercare gli episodi passati da cui l'agente deve imparare. Se, invece, avessimo un ExperienceBufferLength troppo grande rispetto al MiniBatchSize, quest'ultimo sarebbe inutile, perché avremmo un buffer con molti dati, ma troppo pochi tra essi verrebbero presi in considerazione ad ogni iterazione, avendo una memoria piccola dove inserirli. Quindi, è importante che questi due iperparametri vadano di pari passo per permettere un ottimale funzionamento di questo tipo di simulazione. Probabilmente, le Esperienze 2 e 3 non sono andate nel verso giusto anche per i motivi sopra riportati, ovvero per la disparità di grandezza tra questi due importanti parametri.

Si può concludere, quindi, che questo tipo di simulazione "RobotArm in ambiente MatLab", ha bisogno di un'implementazione nella quale il DiscountFactor gioca un ruolo fondamentale, e deve essere settato ad un valore elevato per avere dei risultati ottimali. Il LearnRate non dovrà essere troppo elevato, per permettere all'agente di non incorrere in errori causati da una velocità troppo alta, per quanto, così facendo, l'addestramento subirà un estremo rallentamento. Infine, gli iperparametri MiniBatchSize e ExperienceBufferLength dovranno essere correttamente bilanciati tra loro. Tutto questo dovrà, naturalmente, essere integrato con il codice specificato nel terzo paragrafo del suddetto capitolo.

# CAPITOLO 4

---

## Simulazione Automatic Parking Valet in ambiente MatLab

---

*Il presente capitolo esplora l'implementazione e la simulazione di un'automobile, che attraverso il controllo predittivo del modello adattivo (MPC) ed il Reinforcement Learning (RL) nell'ambiente di sviluppo MATLAB e Simulink, viene addestrata a parcheggiare autonomamente. Ci si focalizzerà, anche, sull'applicazione dell'algoritmo di Reinforcement Learning Twin-Delayed Deep Deterministic (TD3) utilizzato per implementare l'agente. L'obiettivo principale è fornire una comprensione dettagliata dell'esperienza di simulazione, introducendo il contesto, spiegando l'algoritmo chiave utilizzato e presentando le scelte di progettazione. Verranno, poi, effettuati degli esperimenti di simulazione che saranno successivamente discussi. Concludendo il capitolo, verranno presentate riflessioni critiche sulle esperienze di simulazione e sull'efficacia dell'algoritmo TD3 nell'ambiente MATLAB in questo tipo di contesto.*

### 4.1 Introduzione all'esperienza

In questo capitolo verrà analizzata un'esperienza di simulazione in ambiente MATLAB e Simulink riguardante un agente implementato con l'algoritmo di Reinforcement Learning (RL) Twin-Delayed Deep Deterministic (TD3), addestrato a controllare un'automobile in un parcheggio (Figura 4.1). L'obiettivo di questa simulazione è quello di addestrare l'agente, tramite l'algoritmo di RL sopra riportato, a parcheggiare autonomamente l'automobile in un unico parcheggio vuoto, essendo gli altri tutti già occupati da altre autovetture.

Questo simulatore, oltre a ciò, si avvale del funzionamento di un sistema ibrido di parcheggio automatico, che combina il controllo predittivo del modello adattivo (MPC) con il Reinforcement Learning (RL). Inoltre, il percorso del veicolo viene visualizzato utilizzando l'ambiente di simulazione Unreal Engine®.

L'algoritmo di controllo esegue una serie di manovre rilevando ed evitando gli ostacoli in spazi ristretti. Per completare la manovra di parcheggio, passa da un controllore MPC adattivo a un agente di RL. Il controllore MPC adattivo muove il veicolo a velocità costante lungo un percorso di riferimento, alla ricerca di un posto auto libero. Quando viene trovato un posto libero, l'agente TD3 prende il controllo ed esegue una manovra di parcheggio pre-addestrata. Entrambi i controllori dispongono di una conoscenza preliminare dell'ambiente (il parcheggio), compresa la posizione dei posti vuoti e dei veicoli parcheggiati.

Aprendo il modello Simulink si può visualizzare il sistema. Il modello contiene cinque diversi blocchi di sottosistemi, ciascuno applicato per eseguire un certo compito (Figura 4.2). Il modello si può aprire con il comando:

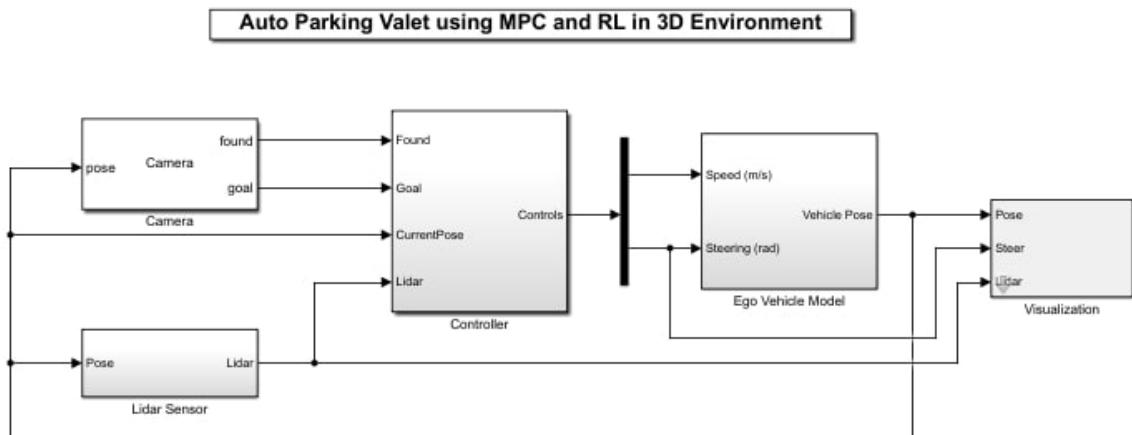
```
mdl = "rlAutoParkingValet3D";
open_system(mdl)
```



**Figura 4.1:** Automobile nell’ambiente di simulazione MatLab e Simulink

Bisognerà, dapprima, caricare i parametri dell’auto parking valet tramite il seguente comando:

```
autoParkingValetParams3D
```



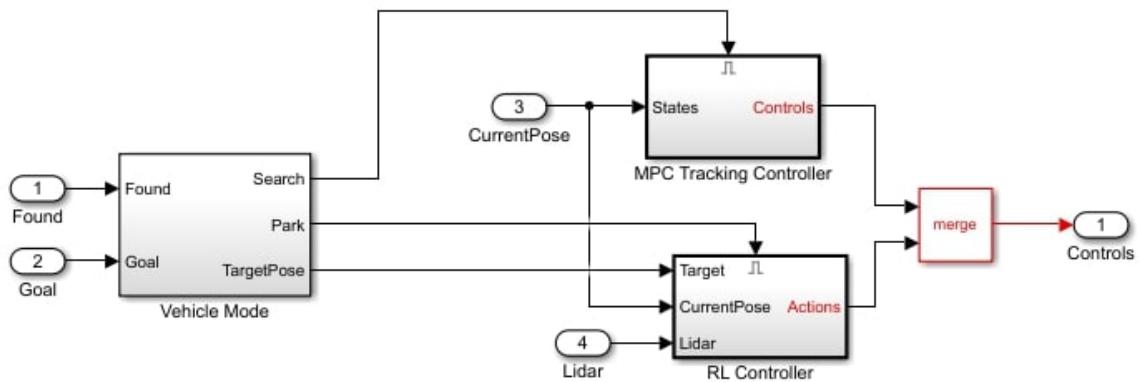
**Figura 4.2:** Modello Simulink del sistema dell’Auto Parking Valet

In questo modello si può notare che:

- Il veicolo è modellato nel sottosistema Ego Vehicle Model. La cinematica è rappresentata da un modello di cinematica di bicicletta<sup>1</sup> a binario singolo con due segnali di ingresso, ovvero la velocità del veicolo  $v$  ( $m/s$ ) e l’angolo di sterzata  $\delta$  (in radianti).
- I blocchi di agenti MPC e RL adattivi si trovano, rispettivamente, nei sottosistemi MPC Tracking Controller e RL Controller.

<sup>1</sup>Il modello di cinematica di bicicletta è un tipo di modello spesso utilizzato nell’ambito della dinamica dei veicoli per semplificare l’analisi del movimento del veicolo.

- La commutazione di modalità tra i controllori è gestita dal sottosistema *Vehicle Mode*, che emette segnali di ricerca e di parcheggio (Figura 4.3). Inizialmente, il veicolo è in modalità di ricerca e il controllore MPC adattivo segue il percorso di riferimento. Quando viene trovato un posto libero, il segnale Park attiva l’agente RL per eseguire la manovra di parcheggio.
- Il Sottosistema di visualizzazione gestisce l’animazione dell’ambiente.



**Figura 4.3:** Sottosistema Vehicle Mode per gestire i controllori

### Controllore MPC

È necessario, prima di continuare, definire brevemente cos’è un controllore MPC. Il Model Predictive Controller è un tipo di controllore utilizzato nei sistemi di controllo automatico. L’MPC è una strategia avanzata di controllo che utilizza un modello matematico del sistema da controllare per prendere decisioni ottimali, facendo previsioni sul comportamento futuro del sistema; queste previsioni vengono fatte su un orizzonte temporale futuro. L’MPC richiede, quindi, un modello matematico accurato del sistema che si desidera controllare. Questo modello è, solitamente, una rappresentazione matematica delle dinamiche del sistema e delle relazioni tra le variabili di input e di output. Il Model Predictive Controller è spesso utilizzato in applicazioni dove è necessario prendere decisioni di controllo ottimali considerando la dinamica del sistema e le condizioni operative attuali. Può essere impiegato in vari settori, ad esempio nel controllo industriale, per i veicoli autonomi (come nel nostro caso), per l’automazione di processo, etc.

Per utilizzare questo tipo di controllore, bisognerà creare l’oggetto MPC adattivo per il tracciamento della traiettoria di riferimento utilizzando il seguente script:

```
createMPCForParking3D
```

## 4.2 Twin-Delayed Deep Deterministic: un’analisi approfondita

L’algoritmo Twin-Delayed Deep Deterministic Policy Gradient (TD3) è un metodo di apprendimento per rinforzo model-free, online e off-policy. Un agente TD3 è un agente di Reinforcement Learning critico dell’attore, che cerca una politica ottimale che massimizzi la ricompensa cumulativa attesa a lungo termine.

L’algoritmo TD3 è un’estensione dell’algoritmo Deep Deterministic Policy Gradient (DDPG). Gli agenti DDPG possono sovrastimare le funzioni di valore, il che può produrre politiche subottimali. Per ridurre la sovrastima della funzione di valore, l’algoritmo TD3 prevede le seguenti modifiche dell’algoritmo DDPG:

- Un agente TD3 apprende due Q-value function e utilizza la stima della funzione di valore minimo durante gli aggiornamenti della politica.
- Un agente TD3 aggiorna la politica e gli obiettivi con una frequenza minore rispetto alle funzioni Q.
- Quando aggiorna la politica, un agente TD3 aggiunge del rumore all’azione target, rendendo meno probabile che la politica sfrutti azioni con stime di valore Q elevate.

È possibile utilizzare un agente TD3 per implementare uno dei seguenti algoritmi di addestramento, a seconda del numero di critici specificato:

- TD3 - Addestra l’agente con due funzioni di valore Q. Questo algoritmo implementa tutte e tre le modifiche precedenti.
- DDPG ritardato - Addestra l’agente con una singola funzione di valore Q. Questo algoritmo addestra un agente DDPG con l’attenuazione della politica di target e aggiornamenti ritardati della politica e del target.

Gli agenti TD3 possono essere addestrati in ambienti con gli spazi di osservazione e azione presenti in Tabella 4.1.

Observation Space	Action Space
Continuous or discrete	Continuous

**Tabella 4.1:** Tipo di ambiente nel quale un agente TD3 può essere addestrato

Inoltre, gli agenti TD3 utilizzano gli attori e i critici presenti in Tabella 4.2.

Critics	Actor
Uno o più critici di funzione $Q(S, A)$ , creati dall’utente con <code>rlQValueFunction</code>	Attore di politica deterministica $\pi(S)$ , creato dall’utente utilizzando <code>rlContinuousDeterministicActor</code>

**Tabella 4.2:** Tipo di attore e critico utilizzati dagli agenti TD3

Durante l’addestramento, un agente TD3:

- Aggiorna le proprietà dell’attore e del critico a ogni passo temporale durante l’apprendimento.
- Memorizza le esperienze passate utilizzando un buffer circolare di esperienze. L’agente aggiorna l’attore e il critico utilizzando un mini-batch di esperienze campionate a caso dal buffer.
- Perturba l’azione scelta dalla politica utilizzando un modello di rumore stocastico a ogni passo di addestramento.

### 4.2.1 Approssimatori di funzioni attore e critico

Per stimare la politica e la funzione valore, un agente TD3 mantiene i seguenti approssimatori di funzione:

- *Attore deterministico*  $\pi(S; \theta)$  - L’attore, con il parametro  $\theta$ , riceve l’osservazione  $S$  e restituisce l’azione corrispondente che massimizza la ricompensa a lungo termine.
- *Attore target*  $\pi_t(S; \theta_t)$  - Per migliorare la stabilità dell’ottimizzazione, l’agente aggiorna periodicamente i parametri dell’attore target  $\theta_t$  utilizzando i valori più recenti dei parametri dell’attore.
- *Uno (o due) critici di valore*  $Q_k(S, A; \phi_k)$  - I critici, ciascuno con parametri diversi  $\phi_k$ , ricevono in ingresso l’osservazione  $S$  e l’azione  $A$ , restituendo la corrispondente aspettativa della ricompensa a lungo termine.
- *Uno (o due) critici target*  $Q_{tk}(S, A; \phi_{tk})$  - Per migliorare la stabilità dell’ottimizzazione, l’agente aggiorna periodicamente i parametri dei critici target  $\phi_{tk}$  utilizzando gli ultimi valori dei parametri dei critici corrispondenti. Il numero di critici target corrisponde al numero di critici.

Sia  $\pi(S; \theta)$  che  $\pi_t(S; \theta_t)$  hanno la stessa struttura e parametrizzazione. Inoltre, per ogni critico, anche  $Q_k(S, A; \phi_k)$  e  $Q_{tk}(S, A; \phi_{tk})$  hanno la stessa struttura e parametrizzazione.

Quando si utilizzano due critici,  $Q_1(S, A; \phi_1)$  e  $Q_2(S, A; \phi_2)$ , ogni critico può avere una struttura diversa, anche se TD3 funziona meglio quando i critici hanno la stessa struttura. In quest’ultimo caso, però, i critici devono avere valori di parametri iniziali diversi.

Durante l’addestramento, l’agente sintonizza i valori dei parametri in  $\theta$ . Al termine dell’addestramento, i parametri rimangono al loro valore sintonizzato e l’approssimatore della funzione attore addestrato viene memorizzato in  $\pi(S)$ .

### 4.2.2 Algoritmi di addestramento

Gli agenti TD3 utilizzano un algoritmo di addestramento, in cui aggiornano i loro modelli di attore e di critico ad ogni passo temporale. In tale algoritmo,  $K = 2$  è il numero di critici e  $k$  è l’indice dei critici.

L’algoritmo è così definito:

- Inizializzare ogni critico  $Q_k(S, A; \phi_k)$  con i valori casuali dei parametri  $\phi_k$  e inizializzare ogni critico target  $Q_{tk}$  con gli stessi valori casuali dei parametri:  $\phi_{tk} = \phi_k$ .
- Inizializzare l’attore  $\pi(S; \theta)$  con i valori casuali dei parametri  $\theta$ , e inizializzare l’attore target con gli stessi valori dei parametri:  $\theta_t = \theta$ .
- Per ogni passo temporale dell’addestramento:
  1. Per l’osservazione corrente  $S$ , selezionare l’azione  $A = \pi(S; \theta) + N$ , dove  $N$  è il rumore stocastico del modello del rumore.
  2. Eseguire l’azione  $A$ . Osservare la ricompensa  $R$  e la successiva osservazione  $S'$ .
  3. Memorizzare l’esperienza  $(S, A, R, S')$  nel buffer dell’esperienza.
  4. Campionare un mini-batch casuale di  $M$  esperienze  $(S_i, A_i, R_i, S'_i)$  dal buffer delle esperienze.

5. Se  $S'_i$  è uno stato terminale, impostare il target della value-function  $y_i$  su  $R_i$ . Altrimenti, impostarlo come definito nell’Equazione 4.2.1.

$$y_i = R_i + \gamma * \min_k(Q_{tk}(S'_i, \text{clip}(\pi_t(S'_i, \theta_t) + \epsilon); \phi_{tk})) \quad (4.2.1)$$

Come si evince dalla formula, la funzione valore target è la somma della ricompensa dell’esperienza  $R_i$  e della ricompensa futura minima scontata dei critici. Per calcolare la ricompensa cumulativa, l’agente calcola prima un’azione successiva passando l’osservazione successiva  $S'_i$  dall’esperienza campionata all’attore target. Quindi, l’agente aggiunge il rumore  $\epsilon$  all’azione calcolata e ritaglia l’azione in base ai limiti superiore e inferiore del rumore. L’agente trova le ricompense cumulative passando l’azione successiva ai critici target. Il fattore di sconto è definito con  $\gamma$ . Se si specifica un valore pari ad  $N$  per il numero di passi per andare avanti, il rendimento a  $N$  passi (che somma le ricompense degli  $N$  passi successivi e il valore stimato scontato dello stato che ha causato la ricompensa  $N$ -esima) viene utilizzato per calcolare il target  $y_i$ .

6. A ogni passo di addestramento, aggiornare i parametri di ogni critico minimizzando la perdita  $L_k$  (definita nell’Equazione 4.2.2) su tutte le esperienze campionate.

$$L_k = \frac{1}{2M} \sum_{i=1}^M (y_i - Q_k(S_i, A_i; \phi_k))^2 \quad (4.2.2)$$

7. Ogni  $D_1$  passi, aggiornare i parametri dell’attore utilizzando il gradiente di politica (definito nell’Equazione 4.2.3), campionato per massimizzare la ricompensa scontata prevista.

$$\nabla_\theta J \approx \frac{1}{M} \sum_{i=1}^M G_{ai} G_{\pi i} \quad (4.2.3)$$

Nella precedente equazione possiamo trovare:

- $G_{ai} = \nabla_A \min_k(Q_k(S_i, A_i, \phi))$  dove  $A = \pi(S_i, \theta)$ .
- $G_{\pi i} = \nabla_\theta \pi(S_i, \theta)$ .

Qui,  $G_{ai}$  è il gradiente dell’uscita minima del critico rispetto all’azione calcolata dalla rete degli attori, mentre  $G_{\pi i}$  è il gradiente dell’uscita dell’attore rispetto ai parametri dell’attore. Entrambi i gradienti sono valutati per l’osservazione  $S_i$ .

8. Ogni  $D_2$  passi, aggiornare l’attore e i critici di destinazione, a seconda del metodo di aggiornamento della destinazione.

Per semplicità, gli aggiornamenti dell’attore e del critico in questo algoritmo mostrano un aggiornamento del gradiente utilizzando la discesa stocastica del gradiente di base.

### 4.2.3 Metodi di aggiornamento del target

Gli agenti TD3 aggiornano l’attore di destinazione e i parametri critici utilizzando uno dei seguenti metodi di aggiornamento del target:

- *Smoothing* - aggiorna i parametri target a ogni passo temporale utilizzando il fattore di smoothing  $\tau$ . Le funzioni alla base di questo metodo sono definite nelle Equazioni 4.2.4 e 4.2.5.

$$\phi_{tk} = \tau\phi_k + (1 - \tau)\phi_{tk} \text{ (parametri dei critici)} \quad (4.2.4)$$

$$\theta_t = \tau\theta + (1 - \tau)\theta_t \text{ (parametri degli attori)} \quad (4.2.5)$$

- *Periodic* - aggiorna periodicamente i parametri target senza il fattore di smoothing (TargetSmoothFactor = 1). Le funzioni alla base di questo metodo sono definite nell'Equazione 4.2.6.

$$\phi_{tk} = \phi_k \text{ e } \theta_t = \theta \quad (4.2.6)$$

- *Periodic Smoothing* - aggiorna periodicamente i parametri target con il fattore di smoothing.

## 4.3 Progettazione e configurazione del simulatore

In questa sezione verranno trattate la progettazione e la configurazione del simulatore stesso; in particolare, verranno prese in considerazione la struttura del parcheggio, il funzionamento dei sensori, la definizione dell'ambiente e la creazione dell'agente; oltre a ciò, verrà trattata l'implementazione dell'addestramento di quest'ultimo. Tutto ciò verrà fatto prendendo in considerazione l'ambiente di sviluppo citato nella Sezione 4.1, quindi MATLAB e Simulink.

### 4.3.1 Creazione del parcheggio

L'ambiente del parcheggio utilizzato in questo esempio è una sottosezione della scena Large Parking Lot (appartenente al toolbox Automated Driving di MatLab). Il parcheggio è rappresentato dall'oggetto `ParkingLotEnvironment`, che memorizza informazioni sull'*ego-vehicle*, sui posti auto vuoti e sugli ostacoli statici (auto parcheggiate e confini). Ogni posto auto ha un indice univoco e un indicatore luminoso verde (libero) o rosso (occupato). I veicoli parcheggiati sono rappresentati in nero, mentre gli altri confini sono delineati in verde. Tutto ciò si può vedere in Figura 4.4.

Si specifica un tempo di campionamento  $T_s$  per i controllori e un tempo di simulazione  $T_f$  (entrambi in secondi).

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

Bisognerà creare, poi, una traiettoria di riferimento per l'*ego-vehicle* utilizzando la funzione di aiuto `createReferenceTrajectory` inclusa in questo esempio. La traiettoria di riferimento parte dall'angolo sud-est del parcheggio e termina a ovest, come indicato dalla linea rosa tratteggiata.

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

Verrà creato un oggetto `ParkingLotEnvironment` con un posto libero all'indice 32 e il percorso di riferimento specificato `xRef`.

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



**Figura 4.4:** Rappresentazione dell’ambiente di parcheggio del simulatore

Si specifica una posizione iniziale  $(X_0, Y_0, \theta_0)$  per l’ego-vehicle. Le unità di  $X_0$  e  $Y_0$  sono espresse in metri e  $\theta_0$  è in radianti.

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

Si calcola, poi, la posizione di destinazione del veicolo utilizzando la funzione `createTargetPose`. La posizione di destinazione corrisponde alla posizione in `freeSpotIndex`.

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

### 4.3.2 Moduli dei sensori

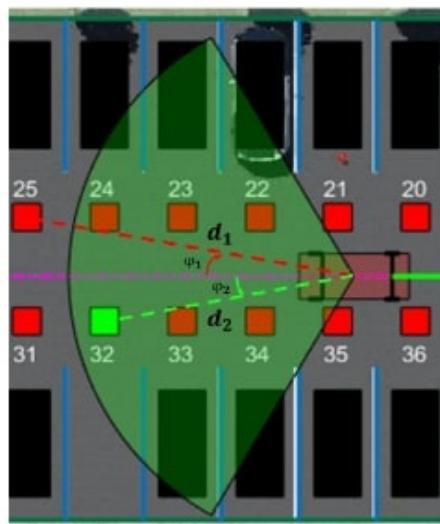
L’algoritmo di parcheggio utilizza approssimazioni geometriche di una telecamera e di un sensore lidar per raccogliere informazioni dall’ambiente del parcheggio.

#### Camera

In questo esempio, il campo visivo di una telecamera montata sul veicolo è rappresentato dall’area ombreggiata in verde in Figura 4.5. La telecamera ha un campo visivo  $\phi$  delimitato da  $\pm \frac{\pi}{3}$  radianti e una profondità massima di misurazione  $d_{max}$  di 10m. Mentre l’ego-vehicle avanza, il modulo della telecamera rileva i posti auto all’interno del campo visivo e determina se un posto è libero oppure occupato. Per semplicità, questa azione è implementata utilizzando relazioni geometriche tra le posizioni dei punti e la posizione corrente del veicolo. Un posto auto si trova nel campo visivo della telecamera se  $d_i \leq d_{max}$  e  $\phi_{min} \leq \phi_i \leq \phi_{max}$ , dove  $d_i$  è la distanza dal posto auto e  $\phi_i$  è l’angolo rispetto al punto di parcheggio.

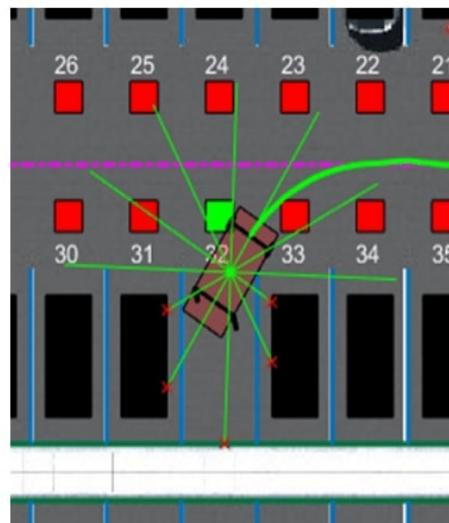
#### Lidar

In questo esempio, il sensore lidar è modellato utilizzando segmenti di linea radiali che emergono dal centro geometrico del veicolo (Figura 4.6). Le distanze dagli ostacoli vengono misurate lungo questi segmenti di linea. La distanza massima misurabile dal lidar lungo ogni



**Figura 4.5:** Campo visivo della telecamera montata sul veicolo

segmento di linea è di 6m. L'agente del Reinforcement Learning utilizza queste letture per determinare la vicinanza del veicolo ad altre automobili nell'ambiente.



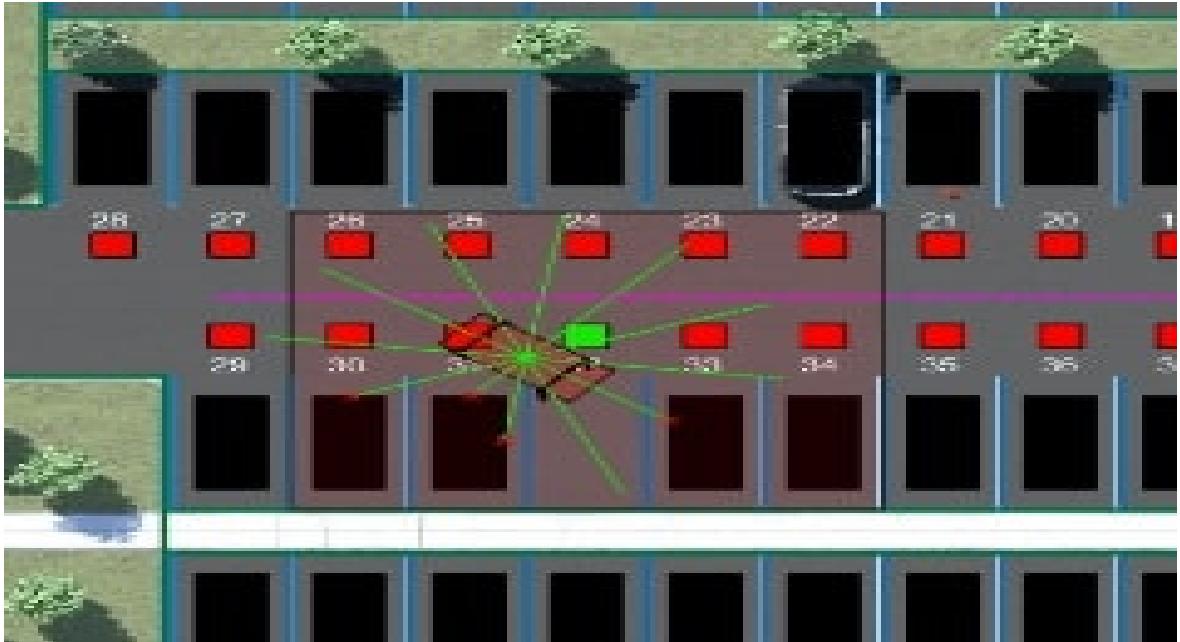
**Figura 4.6:** Schema visivo del funzionamento del sensore lidar

### 4.3.3 Creazione dell'ambiente

L'ambiente per l'addestramento è la regione ombreggiata in rosso nella Figura 4.7. Grazie alla simmetria del parcheggio, l'addestramento all'interno di questa regione è sufficiente affinché la politica si adatti ad altre regioni dopo l'applicazione delle trasformazioni delle coordinate alle osservazioni. Inoltre, la limitazione dell'addestramento a questa regione riduce significativamente la durata dell'addestramento rispetto all'addestramento sull'intero parcheggio.

Per questo ambiente valgono le seguenti specifiche:

- L'area di addestramento è uno spazio di 13,625 m x 12,34 m con il punto di riferimento al centro orizzontale.



**Figura 4.7:** Ambiente per l’addestramento dell’agente

- Le osservazioni sono gli errori di posizione  $X_e$  e  $Y_e$  dell’ego-vehicle rispetto alla posizione del bersaglio, del seno e del coseno dell’angolo di rotta reale  $\theta$  e delle letture del sensore lidar.
- La velocità del veicolo durante il parcheggio è costante e pari a  $2m/s$ .
- I segnali di azione sono angoli di sterzata discreti che variano tra  $\pm \frac{\pi}{4}$  radianti in passi di  $0,2618$  radianti o  $15$  gradi.
- Il veicolo viene considerato parcheggiato se gli errori rispetto alla posizione del bersaglio rientrano nelle tolleranze specificate di  $\pm 0,75m$  (posizione) e  $\pm 10$  gradi (orientamento).
- L’episodio termina se il veicolo esce dai confini della regione di addestramento, se si scontra con un ostacolo o se parcheggia con successo.
- La ricompensa  $r_t$  fornita al tempo  $t$ , è definita nell’Equazione 4.3.1

$$r_t = 2e^{-(0.05X_e^2 + 0.04Y_e^2)} + 0.5e^{-40\theta_e^2} - 0.05\delta^2 + 100f_t - 50g_t \quad (4.3.1)$$

In questa equazione,  $X_e$ ,  $Y_e$  e  $\theta_e$  sono gli errori di posizione e di angolo di direzione del veicolo rispetto alla posizione target, mentre  $\delta$  è l’angolo di sterzata.  $f_t$  (0 o 1) indica se il veicolo ha parcheggiato e  $g_t$  (0 o 1) indica se il veicolo si è scontrato con un ostacolo o se ha lasciato la regione di addestramento al tempo  $t$ .

Le trasformazioni delle coordinate sulle osservazioni della posizione del veicolo  $(X, Y, \theta)$  per le diverse posizioni del parcheggio sono le seguenti:

- Posti auto 1-14:  $\bar{X} = X$ ,  $\bar{Y} = Y + 20$ ,  $\bar{\theta} = \theta$ ;
- Posti auto 15-28:  $\bar{X} = 41 - X$ ,  $\bar{Y} = -64.485 - Y$ ,  $\bar{\theta} = \theta - \pi$ ;
- Posti auto 29-37: nessuna trasformazione;

- Posti auto 38-46:  $\bar{X} = 41 - X$ ,  $\bar{Y} = -84.48 - Y$ ,  $\bar{\theta} = \theta - \pi$ ;

Si devono creare, poi, 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";
```

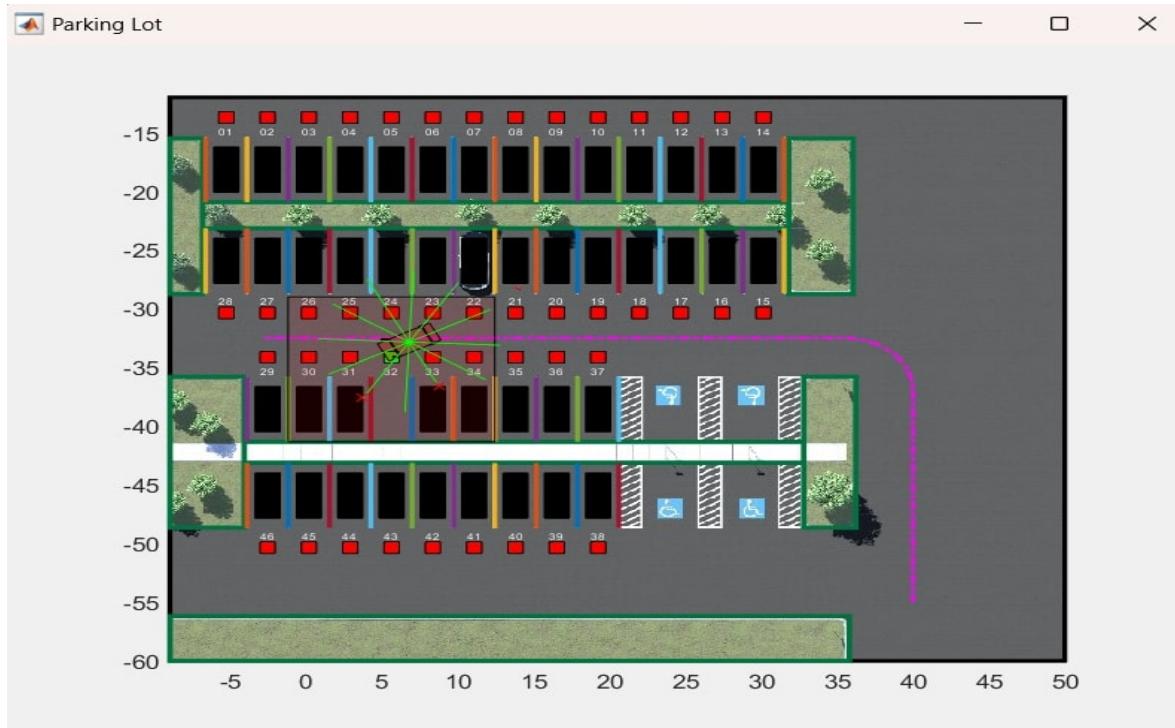
È anche, necessario creare l'interfaccia dell'ambiente Simulink, specificando il percorso del blocco Agente RL.

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

Infine, si dovrà specificare una funzione di reset per l'addestramento. La funzione `autoParkingValetResetFcn` reimposta la posa iniziale del veicolo ego su valori casuali all'inizio di ogni episodio.

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

Una volta che l'ambiente, il parcheggio ed i sensori sono stati creati, il risultato finale, durante la simulazione, sarà quello presente in Figura 4.8.



**Figura 4.8:** Ambiente finale durante la simulazione

#### 4.3.4 Creazione dell'agente

L'agente in questo esempio è un agente Twin-Delayed Deep Deterministic Policy Gradient (TD3). Gli agenti TD3 si basano su oggetti attori e critici approssimatori per apprendere la politica ottimale come ampiamente specificato nella Sezione 4.2.

Le reti di attori e critici sono inizializzate in modo casuale. Per garantire la riproducibilità, è necessario inizializzare il "seed" del generatore casuale.

```
rng(0)
```

Gli agenti TD3, come già sappiamo, utilizzano due approssimatori della funzione Q-value parametrizzata per stimare il valore della politica. Per modellare la funzione di valore Q parametrizzata all'interno di entrambi i critici, si utilizza una rete neurale con 16 ingressi e un'uscita. L'uscita della rete critica è la funzione di valore stato-azione per l'esecuzione di una determinata azione a partire da una determinata osservazione.

Bisogna, quindi, definire ogni percorso di rete come una matrice di oggetti layer e assegnare i nomi ai livelli di ingresso e di uscita di ciascun percorso. Questi nomi consentono di collegare i percorsi e, successivamente, di associare esplicitamente i livelli di ingresso e di uscita della rete al canale ambientale appropriato.

```
% 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")
];

% Convert to layergraph object and connect layers
criticNet = layerGraph(mainPath);
criticNet = addLayers(criticNet, actionPath);
criticNet = connectLayers(criticNet, "fc2", "concat/in2");
```

Convertiamo, quindi, l'oggetto `criticNet` in un oggetto `dlnetwork` e visualizziamo il numero di parametri apprendibili.

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

Creiamo, a questo punto, i critici utilizzando `criticdlnet`, le specifiche di osservazione e azione dell'ambiente e i nomi dei livelli di input della rete da collegare ai canali di osservazione e azione dell'ambiente.

```

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

```

È necessario creare anche la rete neurale per l'attore. L'uscita della rete attore è l'angolo di sterzata.

```

anet = [
    featureInputLayer(nObs)
    fullyConnectedLayer(128)
    reluLayer
    fullyConnectedLayer(128)
    reluLayer
    fullyConnectedLayer(nAct)
    tanhLayer
];

```

```
actorNet = layerGraph(anet);
```

Convertiamo, quindi, l'oggetto `actorNet` in un oggetto `dlnetwork` e visualizziamo il numero di parametri apprendibili.

```

actordlnet = dlnetwork(actorNet);
summary(actordlnet)

```

A questo punto, creiamo l'oggetto attore per l'agente TD3.

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

Specifichiamo, anche, le opzioni dell'agente e creiamo l'agente TD3.

```

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

```

Impostiamo, poi, le opzioni di rumore per l'esplorazione.

```

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

```

Per questo esempio, impostiamo i tassi di apprendimento dell'attore e del critico a  $1e - 3$  e  $2e - 3$ , rispettivamente. Impostiamo, anche, un fattore di soglia del gradiente pari a 1 per limitare i gradienti durante l'addestramento.

Quindi, specifichiamo le opzioni di addestramento per l'attore.

```

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

```

E anche quelle per il critico.

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

Infine, creiamo l'agente usando l'attore, i critici e gli oggetti opzioni dell'agente.

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

### 4.3.5 Addestramento dell'agente

Per addestrare l'agente è necessario specificare le opzioni di addestramento utilizzando la funzione `rlTrainingOptions`. L'agente viene addestrato per un massimo di 10000 episodi e ogni episodio dura al massimo 200 passi temporali. L'addestramento termina quando viene raggiunto il numero massimo di episodi o la ricompensa media su 200 episodi raggiunge un valore maggiore o uguale a 120.

```
trainOpts = rlTrainingOptions(...
    MaxEpisodes=10000, ...
    MaxStepsPerEpisode=200, ...
    ScoreAveragingWindowLength=200, ...
    Plots="training-progress", ...
    StopTrainingCriteria="AverageReward", ...
    StopTrainingValue=120);
```

L'agente verrà addestrato utilizzando la funzione `train`. L'addestramento completo di questo agente è un processo intensivo dal punto di vista computazionale che può richiedere diverse ore per essere completato. Per risparmiare tempo durante l'esecuzione di questo esempio, è possibile caricare un agente preaddestrato impostando `doTraining` su `false`. Per addestrare l'agente da soli, è necessario impostare `doTraining` su `true`.

```
doTraining = false;
if doTraining
    trainingResult = train(agent,env,trainOpts);
else
    load("rlAutoParkingValetAgent.mat","agent");
end
```

### Simulazione di un agente addestrato

Per vedere se l'addestramento dell'agente è andato a buon fine, si deve simulare il modello e osservare la manovra di parcheggio (Figura 4.9).

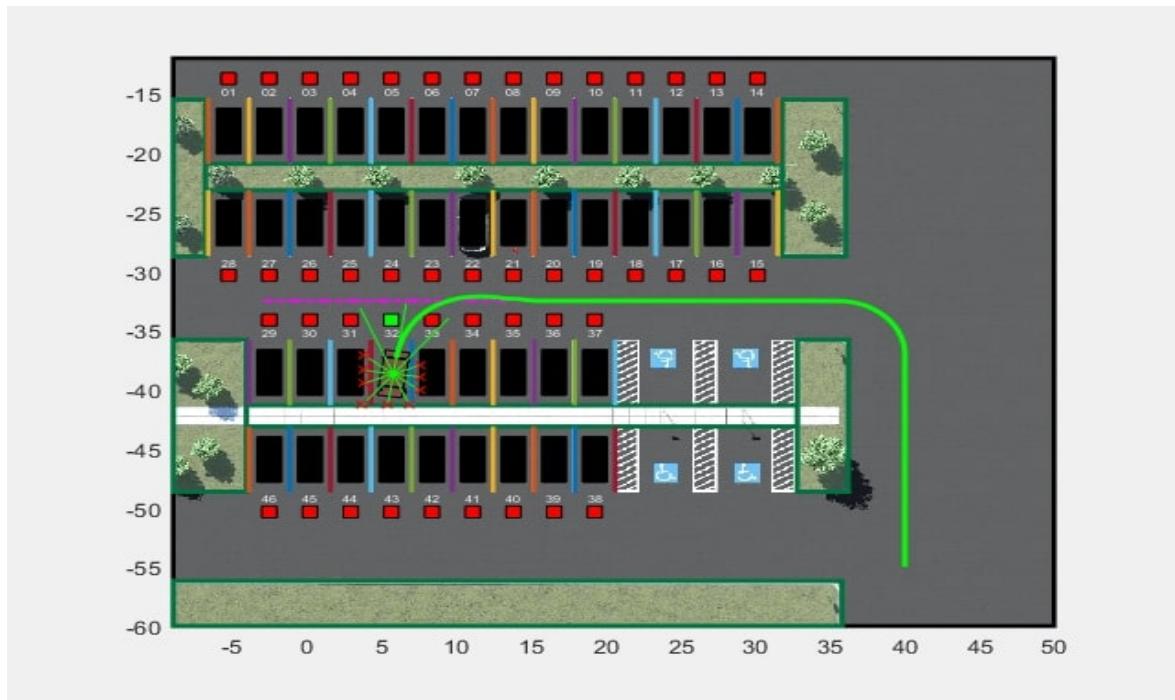
```
sim(mdl);
```

Come già accennato, il veicolo segue il percorso di riferimento utilizzando il controllore MPC prima di passare il controllo all'agente RL quando viene rilevato il punto di destinazione. Una volta che l'agente TD3 prende il controllo, il veicolo completa la manovra di parcheggio.

Per visualizzare la traiettoria (Figura 4.10), apriamo l'Ego Vehicle Pose scope.

```
open_system(mdl + "/Ego Vehicle Model/Ego Vehicle Pose")
```

È anche possibile parcheggiare il veicolo in un posto differente da quello preimpostato.



**Figura 4.9:** Esempio di un addestramento andato a buon fine

```
freeSpotIndex = 20;
sim(mdl)
```

Ciò permette di addestrare l'agente a parcheggiare l'ego-vehicle non sempre nello stesso posto, e quindi ad avere una panoramica più ampia di tutto il parcheggio e di essere pronti a situazioni differenti, qualora queste dovessero verificarsi.

## 4.4 Valutazione e analisi delle esperienze di simulazione 2

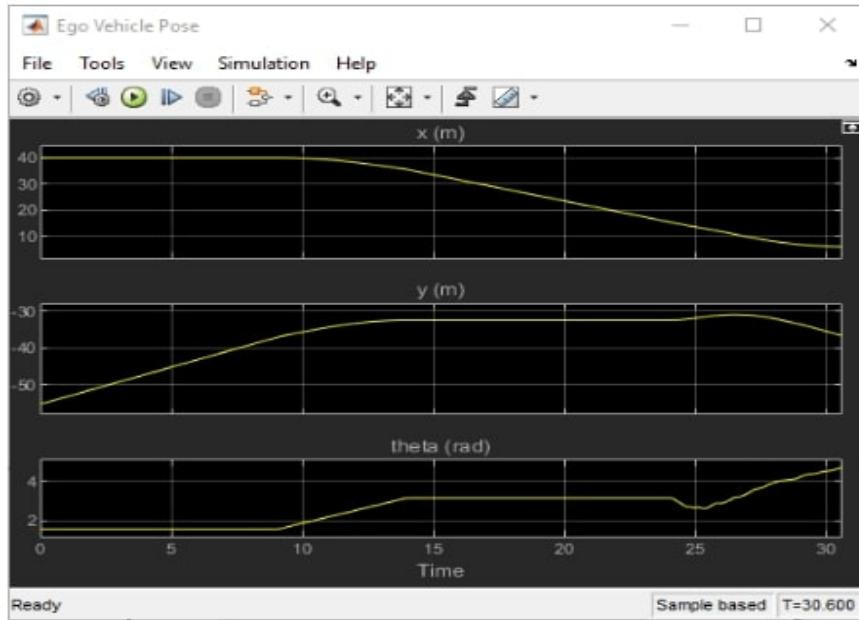
Prima di iniziare la discussione sull'analisi delle esperienze, è bene ripetere che gli elementi che valuteremo, relativi agli esperimenti, sono gli stessi di quelli visti nella Sezione 3.4. Tali elementi sono:

- *Average Reward* (o Ricompensa Media);
- *Episode Reward* (o Ricompensa Periodica);
- *Episode Q0*;

Detto ciò, è doveroso anche dire che le esperienze che si valuteranno per questo tipo di simulazione hanno in comune l'obiettivo di raggiungere un valore di Average Reward maggiore o uguale a 120 su 200 episodi consecutivi (come riportato nella sezione precedente), entro i 10000 episodi prefissati con i quali si interagirà.

### 4.4.1 Esperienza 1

La prima esperienza viene effettuata sulla base del codice precedentemente riportato, quindi senza effettuare nessuna modifica ad esso. Analizziamo i dati finali relativi a questa prima esperienza, presenti in Figura 4.11. È interessante notare come, per arrivare al risultato

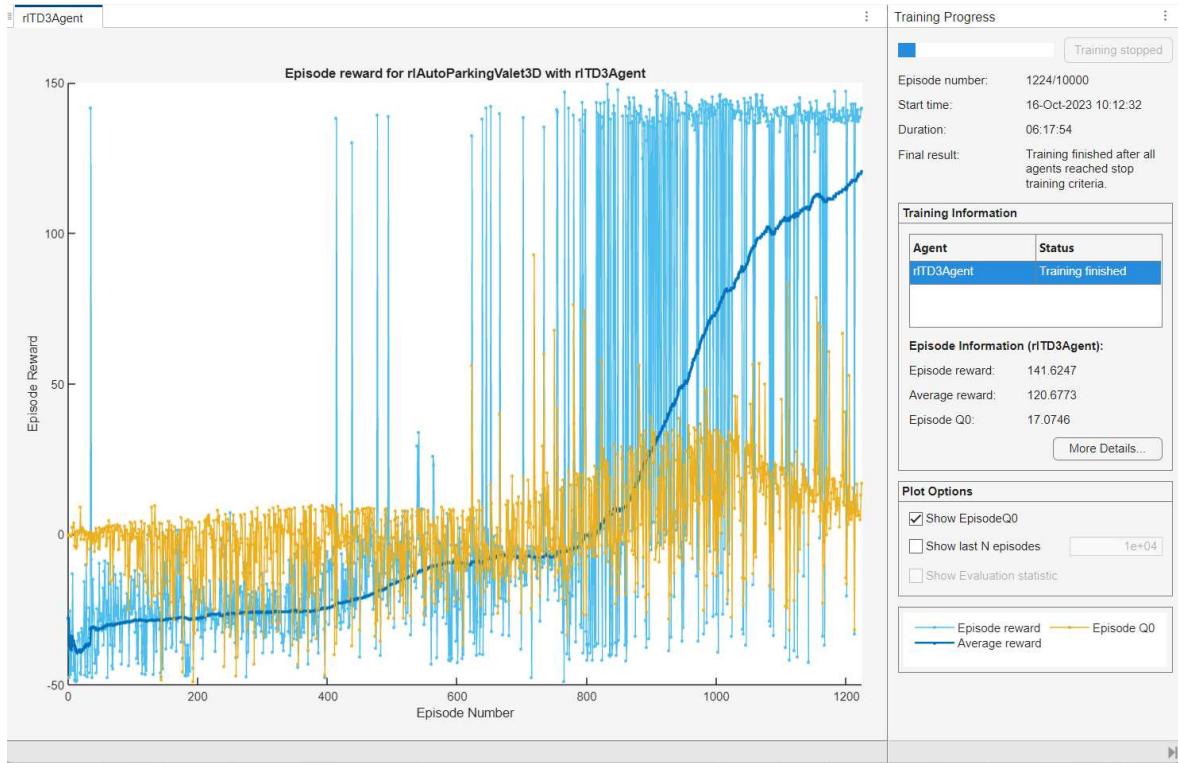


**Figura 4.10:** Traiettoria schematizzata dell’ego-vehicle

sperato, il tempo di simulazione è durato poco più di 6 ore e ci sono voluti 1224 tentativi sui 10000 inizialmente impostati.

A questo punto, si possono analizzare approssimativamente le diverse fasi relative alla curva dell’*average reward*. Si ha una fase iniziale (che considera gli episodi 0-200 circa), in cui la curva è molto minore di zero, il che suggerisce che l’agente sta iniziando ad apprendere e la sua performance media è abbastanza bassa. A questo punto, si nota che tra l’episodio 200 e l’episodio 600 circa, la curva inizia a salire gradualmente, fino ad arrivare ad un valore medio di zero, indicando che l’agente sta migliorando e sta ottenendo ricompense medie maggiori; questa può essere considerata come una fase di apprendimento da parte dell’agente stesso. Tra l’episodio 600 e l’episodio 1000 (circa) si ha una fase di consolidamento dei risultati. La curva mostra, infatti, un aumento più marcato, quasi esponenziale, riflettendo una performance media dell’agente che migliora in modo più consistente, arrivando anche a valori medi pari a +90; questo fa capire che quest’ultimo è sulla giusta via per arrivare al risultato sperato. Nell’ultima fase, che va dall’episodio 1000 al 1224, la curva si stabilizza definitivamente, indicando che l’agente ha raggiunto un livello di performance che non varia molto con ulteriori episodi, arrivando, infatti, al risultato sperato. La cosa interessante è che l’agente è arrivato ad ottenere il valore atteso in un tempo relativamente corto per una simulazione di RL, che generalmente completa il processo in molto più tempo (basti pensare ai risultati dell’Esperienza 1 del simulatore Robot-Arm definito nel Capitolo 3); oltre a ciò, il numero di tentativi utilizzati è poco più di un decimo di quelli totali impostati inizialmente.

Per quanto riguarda le diverse fasi relative alla curva dell’*Episode Reward*, si può dire che essa ha un andamento costantemente positivo. Infatti, dall’episodio 0 al 200, si ha una fase di instabilità e variabilità e la curva mostra una grande varietà di picchi e valli, con molte ricompense anche negative, il che indica che l’agente commette molti errori o sta esplorando ampiamente. Si ha poi, una fase di apprendimento graduale tra gli episodi 200-600 (circa), nella quale i picchi diventano meno frequenti e la curva mostra una tendenza generale all’aumento, riflettendo un miglioramento graduale nelle ricompense per episodio. Dopo quest’ultima, la curva sale notevolmente, e si può notare un progresso accelerato dall’episodio 600 al 1000, nella quale le ricompense per episodio diventano sempre più positive e meno variabili, indicando che l’agente sta apprendendo strategie di successo. Infine, nell’ultima fase



**Figura 4.11:** Risultato finale dell’Esperienza 1 per il simulatore Automatic Parking Valet

(episodi 1000-1224), si hanno alte performance da parte dell’agente, e la curva si stabilizza ad alti livelli di ricompensa per episodio, suggerendo che l’agente ha appreso come ottenere ricompense elevate in modo consistente, permettendo di arrivare, così, al risultato sperato.

Oltre all’andamento delle precedenti curve, è interessante notare anche l’andamento della curva *episode Q0*. Inizialmente, la curva mostra una grande varietà di valori, la metà dei quali sono negativi, il che potrebbe indicare incertezza o esplorazione da parte dell’agente. Da qui in poi, però, la curva inizia a mostrare un trend ascendente, suggerendo che l’agente sta iniziando a valutare meglio il potenziale dei suoi stati iniziali e sta apprendendo dai vari errori commessi. Dall’episodio 600 circa in poi, la curva sale più rapidamente, indicando che l’agente sta migliorando la sua stima dei valori Q0 e diventa più sicuro nelle sue decisioni iniziali, ottenendo picchi di valori sempre più elevati. Verso la fine, si nota che la curva sembra appiattirsi, suggerendo che l’agente ha raggiunto una valutazione stabile e ottimale degli stati iniziali, il che coincide con il raggiungimento dell’obiettivo.

Una volta analizzati i dati ottenuti dall’Esperienza 1, passiamo, ora, ad analizzare i dati acquisiti dalla seconda esperienza per questo simulatore.

#### 4.4.2 Esperienza 2

Per l’esecuzione dell’Esperienza 2 sono state apportate delle modifiche al codice di base del simulatore. Successivamente, vengono analizzate tali modifiche.

La prima modifica effettuata riguarda la complessità della rete neurale utilizzata per l’agente; infatti, viene aumentato il numero di neuroni, passando da 128 a 192 (tale numero viene, quindi, moltiplicato per un fattore di 1,5) in entrambi gli strati fully connected. Questa cosa viene fatta sia per l’attore che per il critico, come di seguito riportato.

```
mainPath = [
    featureInputLayer (nObs, Name="StateInLyr")
```

```

fullyConnectedLayer(256)
concatenationLayer(1,2,Name="concat")
reluLayer
fullyConnectedLayer(256)
reluLayer
fullyConnectedLayer(1,Name="CriticOutLyr")
];
criticNet = layerGraph(mainPath);

anet = [
    featureInputLayer(nObs)
    fullyConnectedLayer(256)
    reluLayer
    fullyConnectedLayer(256)
    reluLayer
    fullyConnectedLayer(nAct)
    tanhLayer
];
actorNet = layerGraph(anet);

```

Aumentare la complessità di una rete neurale nel Machine Learning significa, di fatto, rendere il modello più capace di apprendere e rappresentare relazioni più sofisticate nei dati, permettendo all'agente di essere, abusando del termine, "più intelligente". C'è da dire, però, che aumentare la complessità del modello porta con sé il rischio di sovradattamento, dove il modello impara perfettamente i dati di addestramento ma fallisce nel generalizzare su dati non visti. Pertanto, è essenziale bilanciare la complessità con la capacità del modello di generalizzare, spesso attraverso delle tecniche di valutazione del modello.

La successiva modifica è stata apportata al `MiniBatchSize` (il quale è stato già trattato nel Capitolo 3), che è stato portato da un valore iniziale di 128 ad un valore pari a 192 (anche in questo caso viene moltiplicato di un fattore pari a 1,5).

```

agentOpts = rlTD3AgentOptions(SampleTime=Ts, ...
    DiscountFactor=0.99, ...
    ExperienceBufferLength=1e6, ...
    MiniBatchSize=192);

```

Le modifiche sotto riportate, invece, sono state effettuate ai parametri di esplorazione. Questi ultimi si riferiscono al comportamento di un agente che sceglie azioni che potrebbero non avere il massimo valore stimato di ricompensa a breve termine, ma che possono portare a nuove informazioni o scoperte sull'ambiente. L'esplorazione è cruciale nel Reinforcement Learning perché gli agenti devono scoprire quali azioni portano alle migliori ricompense attraverso la sperimentazione. Naturalmente, come tutto il resto, questi parametri dovranno essere bilanciati tra di loro e con altri per non incorrere in errori o malfunzionamenti.

```

agentOpts.ExplorationModel.StandardDeviation = 0.3;
agentOpts.ExplorationModel.StandardDeviationDecayRate = 1e-5;
agentOpts.ExplorationModel.StandardDeviationMin = 0.03;

```

Per non andare incontro a situazioni inaspettate, si è deciso di effettuare modifiche moderate a questi ultimi. Infatti, la `StandardDeviation` è passata da 0.1 a 03, la `Standard`

`DeviationDecayRate` è passata da un valore pari a `1e-4` ad un valore di `1e-5`, mentre la `StandardDeviationMin` è stata modificata da `0.01` a `0.03`.

Le ultime modifiche apportate riguardano le opzioni di addestramento per l'attore e per i critici. In particolare, vengono modificati i `LearnRate` di entrambi, portati tutti ad un valore pari a `3e-3`; oltre a questi, anche i parametri del `GradientThreshold` del critico e dell'attore sono stati cambiati; quest'ultimo è un parametro che serve a limitare l'entità della modifica applicata ai pesi della rete neurale durante un'iterazione di addestramento. L'ultimo parametro che viene modificato è il `L2RegularizationFactor` dell'attore, che è un fattore utilizzato per controllare l'intensità della regolarizzazione  $L^2$  applicata ai pesi della rete neurale durante l'addestramento.

```
agentOpts.ActorOptimizerOptions.LearnRate = 3e-3;
agentOpts.ActorOptimizerOptions.GradientThreshold = 1,5;
agentOpts.ActorOptimizerOptions.L2RegularizationFactor = 3e-3;

agentOpts.CriticOptimizerOptions(1).LearnRate = 3e-3;
agentOpts.CriticOptimizerOptions(2).LearnRate = 3e-3;
agentOpts.CriticOptimizerOptions(1).GradientThreshold = 1,5;
agentOpts.CriticOptimizerOptions(2).GradientThreshold = 1,5;
```

Introdotte le nuove modifiche, si possono analizzare i dati per l'Esperienza 3, riportati in Figura 4.12.



**Figura 4.12:** Risultato finale dell'Esperienza 2 per il simulatore Automatic Parking Valet

Si può notare da subito che l'esperienza è finita con esito positivo, con 2136 episodi sui 10000 prestabiliti. Inoltre, la durata totale della simulazione è di 3 ore e 24 minuti, un tempo

<sup>2</sup>La regolarizzazione  $L^2$ , anche conosciuta come regolarizzazione di Tikhonov o weight decay, è una tecnica utilizzata per prevenire il sovraccarico del modello ai dati di addestramento.

minore rispetto alla prima esperienza. Come nel caso precedente, si possono analizzare le diverse curve e le loro fasi.

L’andamento della curva *average reward* è mediamente crescente; infatti, si ha una prima fase (episodi 0-400) in cui c’è una prevalenza di bassi valori medi di ricompensa, indicando prestazioni iniziali scarse dell’agente, anche se, comunque, queste crescono costantemente, seppur lentamente. Si ha, poi, una fase di crescita, che va dall’episodio 400 all’episodio 800 circa, in cui siamo in presenza di un aumento costante nella ricompensa media, riflettendo il miglioramento progressivo delle prestazioni. Dopo di che, si nota un’ulteriore fase di apprendimento, fino all’episodio 1200 circa. La curva, infatti, mostra un trend al rialzo più marcato, segnalando che l’agente sta apprendendo efficacemente. Superata questa fase, si nota una fase di stabilizzazione del rendimento dall’episodio 1200 all’episodio 1600; qui la crescita della ricompensa media rallenta e inizia a stabilizzarsi, suggerendo che l’agente sta raggiungendo un rendimento costante. L’ultima fase, con la quale si conclude anche l’esperienza, è una fase di mantenimento in cui la curva della ricompensa media si mantiene alta e relativamente piatta, indicando che l’agente ha raggiunto e mantiene un livello elevato di prestazioni, fino ad arrivare al valore medio desiderato.

Per quanto riguarda la curva dell’*episode reward*, si può dire che si mantiene costante per quasi tutta la simulazione, tranne all’inizio, con un’alternanza di picchi positivi e negativi. Si può notare, infatti, una fase iniziale, che arriva fino all’intorno dell’episodio 400, dove si possono osservare variazioni marcate e presenza di ricompense negative, suggerendo un alto livello di esplorazione e instabilità nelle prestazioni da parte dell’agente. Dopo questa fase iniziale, l’agente inizia ad apprendere velocemente e si ha, quindi, una riduzione della variabilità e un aumento generale delle ricompense; queste, per ogni episodio, mostrano una tendenza al rialzo più stabile con meno variabilità, riflettendo un miglioramento nelle strategie dell’agente. Dall’episodio 1600 fino alla fine, c’è una fase di raffinamento da parte dell’agente; infatti, la curva si mantiene su valori elevati con variazioni minori, suggerendo che si sono raggiunti livelli di performance ottimali.

L’ultima curva, ossia l’*episode Q0*, ha una andamento simile a quello dell’ultima trattata, quindi ha una fase iniziale più variabile e con valori più bassi, per poi rimanere prettamente costante per tutto il resto della simulazione. Nello specifico, fino all’episodio 600 circa, ci troviamo in una fase di valutazione iniziale, nella quale possiamo trovare alta variabilità e potenziali valutazioni negative o basse, indicando incertezza nelle stime di valore iniziale, anche se con un aumento graduale e costante del valore Q0, il che suggerisce che l’agente sta, mano a mano, migliorando la sua capacità di valutare gli stati iniziali. Dall’episodio 600 fino all’episodio 1600 si ha una fase di stabilizzazione e convergenza verso valori maggiori; infatti, la curva diventa meno variabile e mostra una tendenza al rialzo, indicando valutazioni più consistenti, fino ad una semi-stabilizzazione del valore Q0. Dall’episodio 1600 fino alla fine, si nota un’ultima fase di sostentimento, nella quale si possono trovare dei valori Q0 elevati e stabili, suggerendo che l’agente mantiene stime affidabili degli stati iniziali, ed è più sicuro di ogni azione che andrà ad eseguire.

Una volta raccolti e trattati i dati finali, si può dire, quindi, che ci sono stati dei cambiamenti rispetto alla prima esperienza, sia per quanto riguarda il tempo di completamento, sia per il numero dei tentativi effettuati, ma anche per quello che concerne l’andamento delle tre diverse curve. Nel prossimo paragrafo verranno trattate delle conclusioni sulla base dei dati ottenuti e delle considerazioni fatte per le due differenti esperienze.

## 4.5 Conclusioni

In conclusione, questo capitolo ha offerto una panoramica completa dell’esperienza di simulazione, focalizzandosi su diversi aspetti chiave. L’introduzione all’esperienza ha fornito

un contesto iniziale, delineando gli obiettivi e le motivazioni alla base della simulazione. Successivamente, l’analisi approfondita del modello Twin-Delayed Deep Deterministic (TD3) ha fornito una comprensione dettagliata dell’algoritmo di Reinforcement Learning utilizzato nel contesto in questa simulazione. La sezione sulla progettazione e configurazione del simulatore ha esplorato le scelte di progettazione implementate nel codice, evidenziando l’implementazione della simulazione stessa.

Infine, nella sezione di valutazione e analisi delle esperienze di simulazione, sono state esaminate due esperienze specifiche, ognuna avente lo stesso codice, ma con parametri diversi e modificati opportunamente, per valutare le prestazioni e le dinamiche apprese dall’agente nei due differenti casi presi in esame.

Da questi ultimi, si può evincere che entrambe le esperienze hanno avuto esito positivo, seppure i dati ottenuti da esse sono stati leggermente differenti, come si può vedere dai grafici stessi se messi a confronto.

Si può, inoltre, dire che, per l’Esperienza 2, con un’architettura della rete più complessa (`fullyConnectedLayer`), tassi di apprendimento più elevati (`LearnRate`), una maggiore soglia di gradiente (`GradientThreshold`), e una politica di esplorazione più ampia (`ExplorationMode1`), l’esplorazione e l’adattamento a complesse relazioni nei dati erano prioritari rispetto all’Esperienza 1. Tuttavia, queste impostazioni hanno aumentato anche il rischio di sovradattamento. Inoltre, un `MiniBatchSize` più grande (Esperienza 2) ha portato ad una stima del gradiente più stabile, anche se, naturalmente, ha richiesto più memoria.

Da tutto ciò, si può, quindi, dire che l’Esperienza 2 è stata più efficiente in termini di tempistica, ma peggiore in termini di numero di episodi. Questo è sicuramente dovuto alla differenza di complessità della rete neurale e alla dimensione del mini-batch, che hanno permesso di catturare relazioni più complesse tra i dati. Inoltre, una soglia di gradiente più alta ha permesso variazioni maggiori nei pesi durante l’addestramento, il che ha fatto sì, insieme alle politiche di esplorazione, che nella seconda esperienza il grafico avesse quell’andamento costante in più punti rispetto al primo.

Infine, si può dire che tra queste due esperienze non c’è n’è stata una migliore rispetto all’altra, poiché una compensa l’altra sotto diversi aspetti. La prima adotta un metodo più conservativo, che potrebbe essere scelto per garantire una convergenza più stabile e per ridurre il rischio di sovradattamento, anche se ciò potrebbe comportare una velocità di apprendimento più lenta o una minore capacità di catturare relazioni complesse nei dati. La seconda, al contrario, adotta un metodo che esplora molto più del primo, ma con il rischio di una convergenza meno stabile, anche se con tempi di apprendimento più veloci. Sarà, quindi, a discrezione di chi utilizzerà questo tipo di agente sotto queste determinate condizioni, decidere su quale delle due esperienze basarsi, in base alle proprie esigenze.

# CAPITOLO 5

---

## Simulazione Train Biped Robot to Walk in ambiente MatLab

---

*Il presente capitolo esplora l'implementazione e la simulazione di un robot bipede che, attraverso il Reinforcement Learning (RL) nell'ambiente di sviluppo MATLAB e Simulink, viene addestrato ad imparare a camminare in maniera autonoma. Ci si focalizzerà, anche, sull'applicazione dell'algoritmo di Reinforcement Learning Deep Deterministic Policy Gradient (DDPG), utilizzato per implementare l'agente. L'obiettivo principale è fornire una comprensione dettagliata dell'esperienza di simulazione, introducendo il contesto, spiegando l'algoritmo chiave utilizzato e presentando le scelte di progettazione. Verranno, poi, effettuati degli esperimenti di simulazione che saranno successivamente discussi. Concludendo il capitolo, verranno presentate riflessioni critiche sulle esperienze di simulazione e sull'efficacia dell'algoritmo DDPG nell'ambiente MATLAB in questo tipo di contesto.*

### 5.1 Introduzione all'esperienza

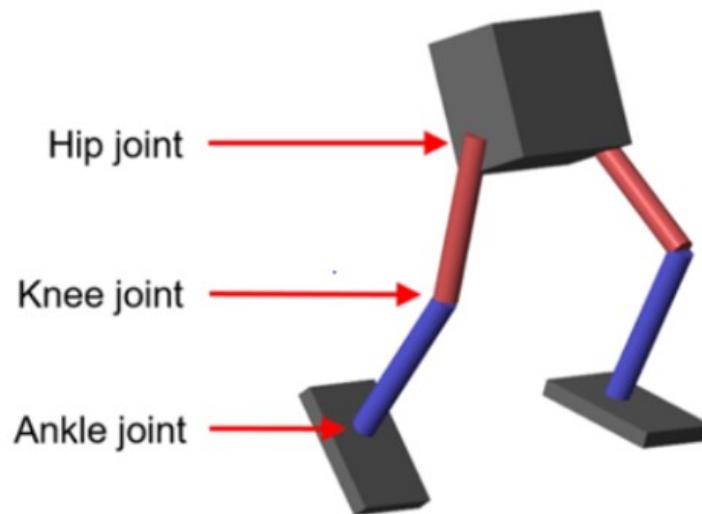
In questo capitolo verrà analizzata un'esperienza di simulazione in ambiente MATLAB e Simulink riguardante un agente, implementato tramite gli algoritmi di Reinforcement Learning (RL) Deep Deterministic Policy Gradient (DDPG) e TD3 (già presentato e trattato nel Capitolo 3), addestrato a controllare un robot autonomo. L'obiettivo di questa simulazione è quello di addestrare l'agente, tramite gli algoritmi di RL sopra riportati, per insegnare ad un robot bipede a camminare in linea retta con il minimo sforzo di controllo. Con questa simulazione si confronteranno tra loro le prestazioni di questi agenti addestrati. Il robot di questo esempio è modellato in Simscape™ Multibody™, presente in Figura 5.1.

Ai fini del confronto, si addestreranno entrambi gli agenti nell'ambiente del robot bipede con gli stessi parametri del modello. Inoltre, gli agenti saranno configurati in modo che abbiano in comune le seguenti impostazioni:

- strategia delle condizioni iniziali del robot bipede;
- struttura di rete dell'attore e del critico;
- opzioni per le rappresentazioni dell'attore e del critico;
- opzioni di addestramento (tempo di campionamento, fattore di sconto, dimensione del mini-batch, lunghezza del buffer di esperienza, rumore di esplorazione).

Bisognerà, dapprima, caricare i parametri del modello tramite il seguente comando:

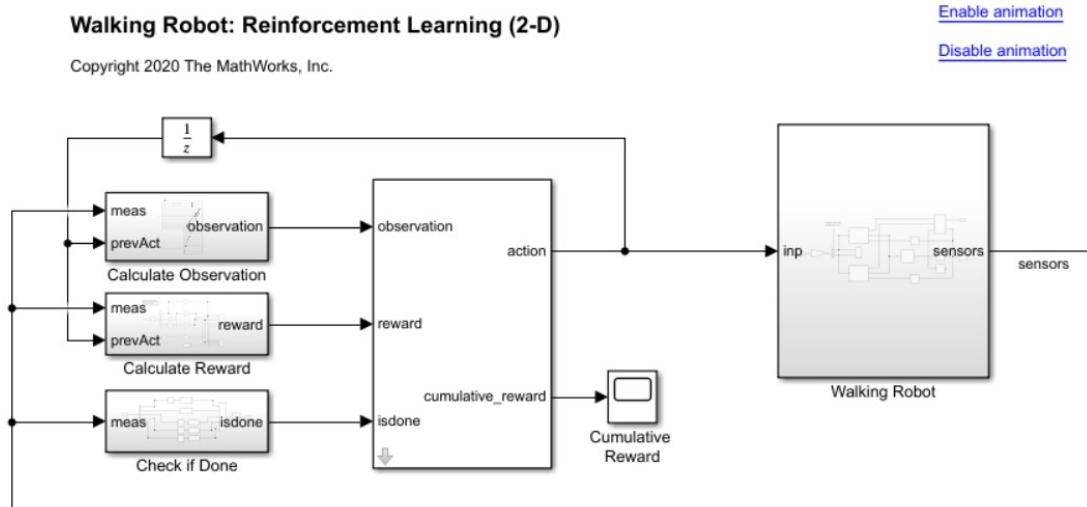
```
robotParametersRL
```



**Figura 5.1:** Rappresentazione del robot nel magazzino

Aprendo il modello Simulink si può visualizzare il sistema. Il modello contiene diversi blocchi di sottosistemi, ciascuno applicato per eseguire un certo compito (Figura 5.2). Il modello si può aprire con il comando:

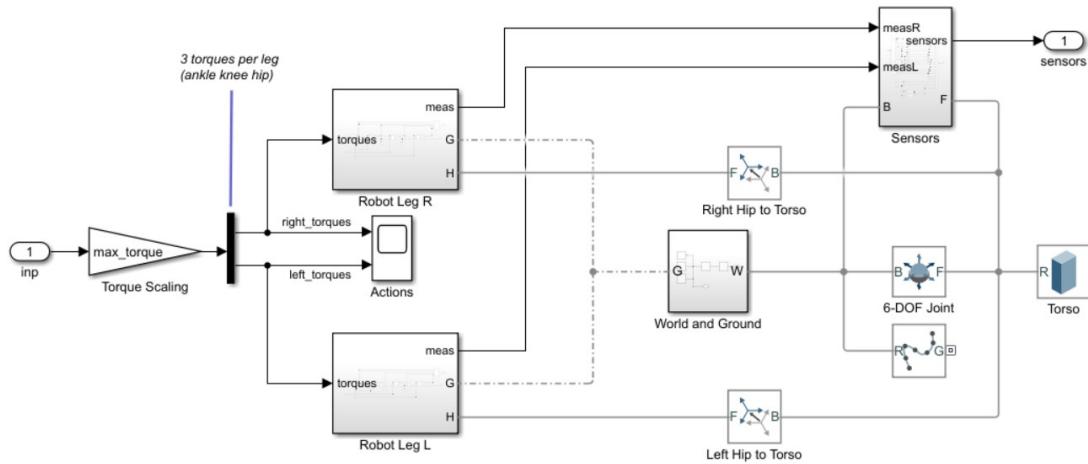
```
mdl = "rlWalkingBipedRobot";
open_system(mdl)
```



**Figura 5.2:** Modello Simulink del sistema del Walking Robot

Il robot è modellato con Simscape™ Multibody™, come si può vedere in Figura 5.3. Per questo modello si ha:

- Nella posizione neutra a 0 rad, entrambe le gambe sono diritte e le caviglie sono piatte.
- Il contatto con il piede viene modellato utilizzando il blocco *Spatial Contact Force* (che si trova all’interno del modello Simscape Multibody).



**Figura 5.3:** Modello Simscape™ Multibody™ del walking robot

- L’agente può controllare 3 articolazioni individuali (caviglia, ginocchio eanca) su entrambe le gambe del robot applicando segnali di coppia compresi tra -3 e 3 N · m. I segnali d’azione effettivamente calcolati sono normalizzati tra -1 e 1.

L’ambiente fornisce all’agente le seguenti osservazioni:

- Traslazione Y (laterale) e Z (verticale) del centro di massa del torso del robot. La traslazione in direzione Z è normalizzata a un intervallo simile a quello delle altre osservazioni.
- Velocità di traslazione X (in avanti), Y (laterale) e Z (verticale).
- Angoli di imbardata, beccheggio e rollio del torso.
- Velocità angolari di imbardata, beccheggio e rollio del torso.
- Posizioni angolari e velocità delle tre articolazioni (caviglia, ginocchio,anca) di entrambe le gambe.
- Valori di azione del passo temporale precedente.

L’episodio termina se si verifica una delle seguenti condizioni:

- Il centro di massa del torso del robot è inferiore a 0,1 m in direzione Z (il robot cade) o superiore a 1 m in direzione Y (il robot si sposta troppo lateralmente).
- Il valore assoluto di rollio, beccheggio o imbardata è superiore a 0,7854 rad.

La funzione di ricompensa  $r_t$  che viene fornita a ogni passo temporale è quella mostrata nell’Equazione 5.1.1.

$$r_t = v_x - 3y^2 - 50z^2 + 25 \frac{T_s}{T_f} - 0.02 \sum_i (u_{t-1}^i)^2 \quad (5.1.1)$$

In quest’ultima, possiamo notare che:

- $v_x$  è la velocità di traslazione in direzione X (in avanti verso l’obiettivo) del robot.

- $y$  è lo spostamento laterale del robot rispetto alla traiettoria rettilinea di destinazione.
- $\hat{z}$  è lo spostamento verticale normalizzato del centro di massa del robot.
- $u_{t-1}^i$  è la coppia del giunto  $i$  dal passo temporale precedente.
- $T_s$  è il tempo di campionamento dell’ambiente.
- $T_f$  è il tempo di simulazione finale dell’ambiente.

La Funzione 5.1.1 di ricompensa incoraggia l’agente, in primis, ad andare avanti, fornendo una ricompensa positiva per una velocità di avanzamento positiva, ma anche ad evitare la conclusione dell’episodio, fornendo una ricompensa costante ( $25 \frac{T_s}{T_f}$ ) a ogni passo temporale. Gli altri termini della funzione di ricompensa sono penalità per cambiamenti sostanziali nelle traslazioni laterali e verticali e per l’uso di sforzi di controllo eccessivi.

Nella prossima sezione, verrà trattato unicamente l’algoritmo di RL Deep Deterministic Policy Gradient (DDPG), dato che l’altro algoritmo su cui si basa questa simulazione (il TD3), è già stato ampiamente trattato nel Capitolo 3.

## 5.2 Deep Deterministic Policy Gradient: un’analisi approfondita

L’algoritmo Deep Deterministic Policy Gradient (DDPG) è un metodo di apprendimento per rinforzo model-free, online e off-policy. Un agente DDPG è un agente di Reinforcement Learning critico dell’attore che cerca una politica ottimale che massimizzi la ricompensa cumulativa attesa a lungo termine.

Gli agenti DDPG possono essere addestrati in ambienti con spazi di osservazione e azione presenti in Tabella 5.1.

Observation Space	Action Space
Continuous or Discrete	Continuous

**Tabella 5.1:** Tipo di ambiente nel quale un agente DDPG può essere addestrato

Inoltre, gli agenti DDPG utilizzano l’attore e il critico presenti in Tabella 5.2.

Critic	Actor
Un critico di funzione $Q(S, A)$ , creati dall’utente con <code>r1QValueFunction</code>	Attore di politica deterministica $\pi(S)$ , creato dall’utente utilizzando <code>r1ContinuousDeterministicActor</code>

**Tabella 5.2:** Tipo di attore e critico utilizzati dagli agenti DDPG

Durante l’addestramento, un agente DDPG:

- Aggiorna le proprietà dell’attore e del critico a ogni passo temporale durante l’apprendimento.
- Memorizza le esperienze passate utilizzando un buffer circolare di esperienze. L’agente aggiorna l’attore e il critico utilizzando un mini-batch di esperienze campionate a caso dal buffer.
- Perturba l’azione scelta dalla politica utilizzando un modello di rumore stocastico a ogni passo di addestramento.

### 5.2.1 Approssimatori di funzioni attore e critico

Per stimare la politica e la funzione valore, un agente DDPG mantiene quattro approssimatori di funzione:

- *Attore*  $\pi(S; \theta)$  - L’attore, con parametri  $\theta$ , prende l’osservazione  $S$  e restituisce l’azione corrispondente che massimizza la ricompensa a lungo termine.
- *Attore target*  $\pi_t(S; \theta_t)$  - Per migliorare la stabilità dell’ottimizzazione, l’agente aggiorna periodicamente i parametri dell’attore target  $\theta_t$  utilizzando i valori più recenti dei parametri dell’attore.
- *Critico*  $Q(S; A; \phi)$  - Il critico, con parametri  $\phi$ , riceve in ingresso l’osservazione  $S$  e l’azione  $A$  e restituisce la corrispondente aspettativa della ricompensa a lungo termine.
- *Critico target*  $Q_t(S, A; \phi_t)$  - Per migliorare la stabilità dell’ottimizzazione, l’agente aggiorna periodicamente i parametri dei critici target  $\phi_t$  utilizzando i valori più recenti dei parametri dei critici.

Sia  $Q(S; A; \phi)$  che  $Q_t(S, A; \phi_t)$  hanno la stessa struttura e parametrizzazione, lo stesso vale per  $\pi(S; \theta)$  e  $\pi_t(S; \theta_t)$ .

Durante l’addestramento, l’agente sintonizza i valori dei parametri in  $\theta$ . Al termine dell’addestramento, i parametri rimangono al loro valore sintonizzato e l’approssimatore della funzione attore addestrato viene memorizzato in  $\pi(S)$ .

### 5.2.2 Algoritmi di addestramento

Gli agenti DDPG utilizzano il seguente algoritmo di addestramento, in cui aggiornano i loro modelli di attore e di critico a ogni passo temporale. Per configurare l’algoritmo di addestramento, si devono specificare le opzioni utilizzando l’oggetto `r1DDPGAgentOptions`.

L’algoritmo è così definito:

- Inizializzare il critico  $Q(S, A; \phi)$  con i valori casuali dei parametri  $\phi$  e inizializzare ogni critico target  $\phi_t$  con gli stessi valori casuali dei parametri:  $\phi_t = \phi$ .
- Inizializzare l’attore  $\pi(S; \theta)$  con i valori casuali dei parametri  $\theta$ , e inizializzare i parametri dell’attore target  $\theta_t$  con gli stessi valori:  $\theta_t = \theta$ .
- Per ogni passo temporale di addestramento:
  1. Per l’osservazione corrente  $S$ , selezionare l’azione  $A = \pi(S; \theta) + N$ , dove  $N$  è il rumore stocastico del modello di rumore.
  2. Eseguire l’azione  $A$ . Osservare la ricompensa  $R$  e la successiva osservazione  $S'$ .
  3. Memorizzare l’esperienza  $(S, A, R, S')$  nel buffer dell’esperienza.
  4. Campionare un mini-batch casuale di  $M$  esperienze  $(S_i, A_i, R_i, S'_i)$  dal buffer delle esperienze.
  5. Se  $S'_i$  è uno stato terminale, impostare il target della funzione valore  $y_i$  su  $R_i$ . Altrimenti, impostarlo come definito nell’Equazione 5.2.1

$$y_i = R_i + \gamma Q_t(S'_i, \pi_t(S'_i; \theta_t); \phi_t) \quad (5.2.1)$$

La funzione valore target è la somma della ricompensa dell’esperienza  $R_i$  e della ricompensa futura scontata. Per calcolare la ricompensa cumulativa, l’agente calcola

prima un’azione successiva passando l’osservazione successiva  $S'_i$  dall’esperienza campionata all’attore target. Quindi, trova la ricompensa cumulativa passando l’azione successiva al critico target. Il fattore di sconto è definito con  $\gamma$ .

Se si specifica un valore pari a  $N$  per il numero di passi per andare avanti, il rendimento a  $N$  passi (che somma le ricompense degli  $N$  passi successivi e il valore stimato scontato dello stato che ha causato la ricompensa  $N$ -esima) viene utilizzato per calcolare il target  $y_i$ .

6. Si aggiornano i parametri dei critici minimizzando la perdita  $L$  (definita nell’Equazione 5.2.2) su tutte le esperienze campionate.

$$L = \frac{1}{2M} \sum_{i=1}^M (y_i - Q(S_i, A_i; \phi))^2 \quad (5.2.2)$$

7. Si aggiornano i parametri dell’attore utilizzando il gradiente di politica (definito nell’Equazione 5.2.3), campionato per massimizzare la ricompensa scontata prevista.

$$\nabla_\theta J \approx \frac{1}{M} \sum_{i=1}^M G_{ai} G_{\pi i} \quad (5.2.3)$$

Nella precedente equazione possiamo trovare:

- $G_{ai} = \nabla_A Q(S_i, A_i, \phi)$  dove  $A = \pi(S_i, \theta)$ .
- $G_{\pi i} = \nabla_\theta \pi(S_i, \theta)$ .

Qui,  $G_{ai}$  è il gradiente dell’output del critico rispetto all’azione calcolata dalla rete degli attori, mentre  $G_{\pi i}$  è il gradiente dell’output dell’attore rispetto ai parametri di quest’ultimo. Entrambi i gradienti sono valutati per l’osservazione  $S_i$ .

8. Si aggiornano l’attore di destinazione e i parametri critici in base al metodo di aggiornamento della destinazione.

Per semplicità, gli aggiornamenti dell’attore e del critico in questo algoritmo mostrano un aggiornamento del gradiente utilizzando la discesa stocastica del gradiente di base.

### 5.2.3 Metodi di aggiornamento del target

Gli agenti DDPG aggiornano i parametri critici e dell’attore target utilizzando uno dei seguenti metodi di aggiornamento del target:

- *Smoothing* - aggiorna i parametri target a ogni passo temporale utilizzando il fattore di smoothing  $\tau$ . Le funzioni alla base di questo metodo sono definite nelle Equazioni 5.2.4 e 5.2.5.

$$\phi_t = \tau\phi + (1 - \tau)\phi_t \text{ (parametri dei critici)} \quad (5.2.4)$$

$$\theta_t = \tau\theta + (1 - \tau)\theta_t \text{ (parametri degli attori)} \quad (5.2.5)$$

- *Periodic* - aggiorna periodicamente i parametri del target senza il fattore di smoothing (`TargetSmoothFactor = 1`).
- *Periodic Smoothing* - Aggiorna periodicamente i parametri di destinazione con il fattore di smoothing.

## 5.3 Progettazione e configurazione del simulatore

In questa sezione verranno trattate la progettazione e la configurazione del simulatore stesso; in particolare, verranno prese in considerazione la definizione dell’ambiente e la creazione dell’agente; oltre a ciò verrà trattata l’implementazione dell’addestramento di quest’ultimo. Tutto ciò verrà fatto prendendo in considerazione l’ambiente di sviluppo citato nella Sezione 5.1, quindi MATLAB e Simulink.

### 5.3.1 Creazione dell’ambiente

Creare, inizialmente, la specifica di osservazione.

```
numObs = 29;
obsInfo = rlNumericSpec([numObs 1]);
obsInfo.Name = "observations";
```

Creare, successivamente, la specifica dell’azione.

```
numAct = 6;
actInfo = rlNumericSpec([numAct 1],LowerLimit=-1,UpperLimit=1);
actInfo.Name = "foot_torque";
```

Creare, infine, l’interfaccia ambientale per il modello del walking robot.

```
blk = mdl + "/RL Agent";
env = rlSimulinkEnv(mdl,blk,obsInfo,actInfo);
env.ResetFcn = @(in) walkerResetFcn(in, ...
    upper_leg_length/100, ...
    lower_leg_length/100, ...
    h/100);
```

### 5.3.2 Selezione e creazione dell’agente per l’addestramento

Questa simulazione offre la possibilità di addestrare il robot utilizzando un agente DDPG o TD3. Per simulare il robot con l’agente scelto, è necessario impostare il flag `AgentSelection` di conseguenza.

```
AgentSelection = "DDPG";
switch AgentSelection
    case "DDPG"
        agent = createDDPGAgent(numObs,obsInfo,numAct,actInfo,Ts);
    case "TD3"
        agent = createTD3Agent(numObs,obsInfo,numAct,actInfo,Ts);
    otherwise
        disp("Assign AgentSelection to DDPG or TD3")
end
```

Le funzioni helper `createDDPGAgent` e `createTD3Agent` eseguono le seguenti azioni:

- Creare le reti di attori e critici.
- Specificare le opzioni per attori e critici.

- Creare gli attori e i critici utilizzando le reti e le opzioni create in precedenza.
- Configurare le opzioni specifiche dell'agente.
- Creare l'agente.

### Agente DDPG

Come già specificato, gli agenti DDPG utilizzano una politica deterministica parametrizzata su spazi di azione continui, che viene appresa da un attore deterministico continuo, e un approssimatore della funzione Q-value parametrizzato per stimare il valore della politica. Si possono utilizzare le reti neurali per modellare sia la politica che la funzione Q-value.

### Agente TD3

Il critico di un agente DDPG può sopravvalutare il Q-value. Poiché l'agente utilizza il Q-value stesso per aggiornare la sua politica (grazie all'attore), la politica risultante può essere subottimale e può accumulare errori di addestramento che possono portare a un comportamento divergente. Quindi, come già specificato nella Sezione 4.2, l'algoritmo TD3 è un'estensione del DDPG con miglioramenti che lo rendono più robusto, impedendo la sovrastima dei valori Q.

Detto ciò, quindi, si possono riassumere brevemente questi miglioramenti dell'algoritmo TD3 rispetto ad all'algoritmo DDPG.

- *Due reti critiche* - gli agenti TD3 apprendono due reti critiche in modo indipendente e utilizzano la stima della funzione di valore minimo per aggiornare l'attore (politica). In questo modo si evita l'accumulo di errori nei passaggi successivi e la sovrastima dei valori Q.
- *Aggiunta di rumore alla politica di destinazione* - l'aggiunta di rumore tagliato alle funzioni di valore attenua i valori della funzione Q su azioni simili. In questo modo si evita di apprendere un picco errato di valori stimati dal rumore.
- *Aggiornamenti ritardati della politica e dell'obiettivo* - per un agente TD3, ritardare l'aggiornamento della rete di attori consente alla funzione Q di ridurre l'errore (avvicinandosi all'obiettivo richiesto) prima di aggiornare la politica. In questo modo si evita la varianza nelle stime dei valori e si ottiene un aggiornamento della politica di qualità superiore.

È importante ripetere che la struttura delle reti di attori e critici utilizzate per questo agente sono le stesse utilizzate per l'agente DDPG.

#### 5.3.3 Specifica delle opzioni di addestramento e addestramento dell'agente

Anche le opzioni di addestramento per gli agenti DDPG e TD3 sono le stesse, cioè quelle di seguito specificate:

- Eseguire ogni sessione di allenamento per 2000 episodi, con ogni episodio della durata massima di `maxSteps`.
- Visualizzare l'avanzamento dell'allenamento nella finestra di dialogo Episode Manager (impostare l'opzione `Plots`) e disabilitare la visualizzazione da riga di comando (impostare l'opzione `Verbose`).

- Terminare l’addestramento solo quando si raggiunge il numero massimo di episodi (`maxEpisodes`). Ciò consente di confrontare le curve di apprendimento di più agenti durante l’intera sessione di allenamento.

```
maxEpisodes = 2000;
maxSteps = floor(Tf/Ts);
trainOpts = rlTrainingOptions(...%
    MaxEpisodes=maxEpisodes,...%
    MaxStepsPerEpisode=maxSteps,...%
    ScoreAveragingWindowLength=250,...%
    Verbose=false,...%
    Plots="training-progress",...%
    StopTrainingCriteria="EpisodeCount",...%
    StopTrainingValue=maxEpisodes,...%
    SaveAgentCriteria="EpisodeCount",...%
    SaveAgentValue=maxEpisodes);
```

Per addestrare l’agente in parallelo, così da velocizzare il processo, specificare le seguenti opzioni di addestramento:

- Impostare l’opzione `UseParallel` su `true`.
- Addestrare l’agente in parallelo in modo asincrono.
- Dopo ogni 32 step, ciascun lavoratore deve inviare esperienze al client del pool parallelo (cioè il processo MATLAB che avvia la formazione). Questo poiché gli agenti DDPG e TD3 richiedono che i lavoratori invino esperienze al client.

È necessario dire che, a causa della casualità dell’addestramento in parallelo, ci si può aspettare risultati diversi da quelli che si otterrebbero senza l’utilizzo di quest’ultimo.

```
trainOpts.UseParallel = true;
trainOpts.ParallelizationOptions.Mode = "async";
trainOpts.ParallelizationOptions.StepsUntilDataIsSent = 32;
trainOpts.ParallelizationOptions.DataToSendFromWorkers = ...
    ..."Experiences";
```

Si può addestrare l’agente utilizzando la funzione `train`. Questo processo è intensivo dal punto di vista computazionale e richiede diverse ore per essere completato per ogni agente. Per risparmiare tempo durante l’esecuzione delle simulazioni, è possibile caricare un agente preaddestrato impostando `doTraining` a `false`. Per addestrare l’agente autonomamente, bisogna impostare `doTraining` a `true`.

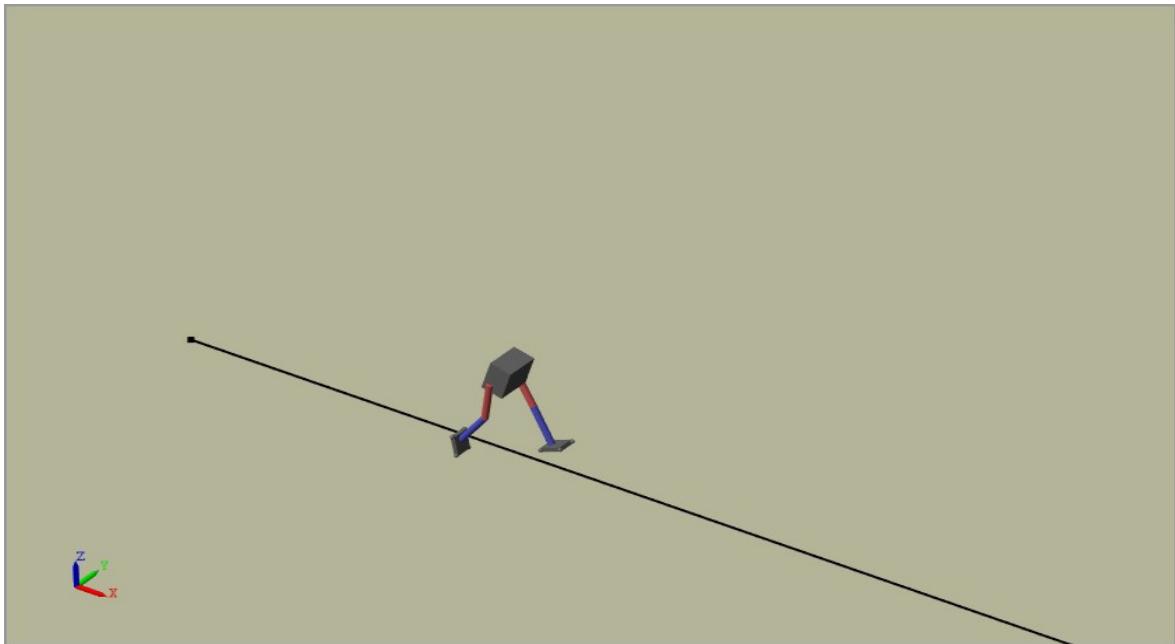
### Simulazione degli agenti addestrati

Inizialmente, bisogna impostare il seme del generatore casuale per garantire la riproducibilità.

```
rng(0)
```

Per convalidare le prestazioni dell’agente addestrato è necessario simularlo nell’ambiente del robot bipede (Figura 5.4).

```
simOptions = rlSimulationOptions(MaxSteps=maxSteps);
experience = sim(env, agent, simOptions);
```



**Figura 5.4:** Visuale dell’ambiente del robot bipede

### 5.3.4 Confronto delle prestazioni degli agenti

Per questa simulazione, poiché sono stati utilizzati due diversi agenti, è anche possibile effettuare un confronto tra questi ultimi.

Per tale confronto ogni agente è stato addestrato cinque volte utilizzando ogni volta un seme casuale diverso. A causa del rumore casuale dell’esplorazione e della casualità dell’addestramento parallelo, la curva di apprendimento per ogni esecuzione è diversa. Poiché l’addestramento degli agenti per più esecuzioni richiede diversi giorni per essere completato, per questo confronto sono stati utilizzati agenti pre-addestrati.

Per gli agenti DDPG e TD3, si può tracciare la media e la deviazione standard della ricompensa dell’episodio (Figura 5.5) e il valore Q0 dell’episodio (Figura 5.6). Il valore Q0 dell’episodio è la stima del critico della ricompensa scontata a lungo termine all’inizio di ogni episodio, data l’osservazione iniziale dell’ambiente. Per un critico ben progettato, il valore Q0 dell’episodio si avvicina alla vera ricompensa scontata a lungo termine.

Si può effettuare il confronto con il seguente comando:

```
comparePerformance ("DDPGAgent", "TD3Agent")
```

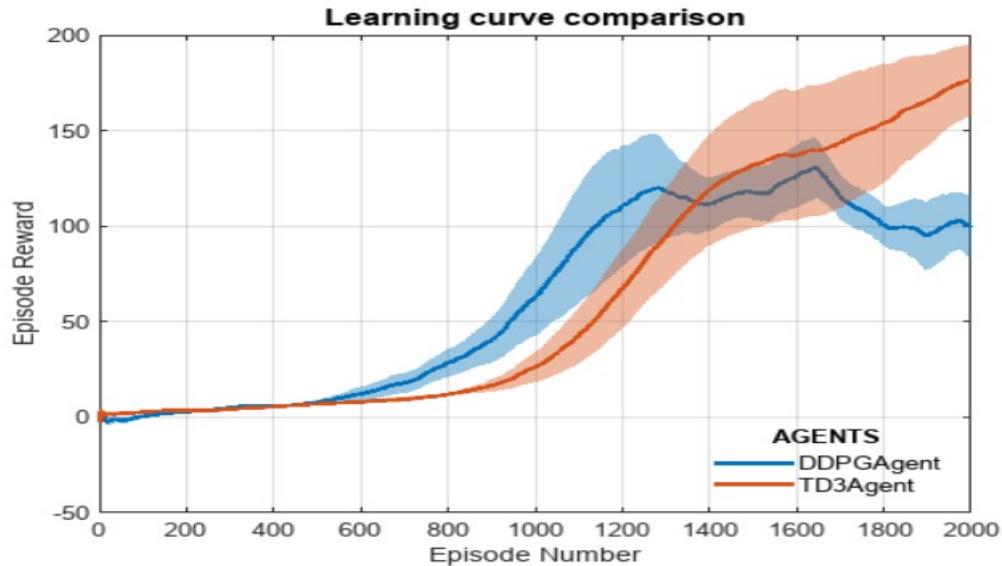
Si può, quindi, dire che, sulla base del grafico di confronto della curva di apprendimento:

- L’agente DDPG sembra apprendere più rapidamente (in media intorno all’episodio numero 600), ma raggiunge un minimo locale. TD3 inizia più lentamente, ma alla fine ottiene ricompense più alte rispetto a DDPG, in quanto evita la sovrastima dei valori Q.
- L’agente TD3 mostra un miglioramento costante nella sua curva di apprendimento, il che suggerisce una maggiore stabilità rispetto all’agente DDPG.

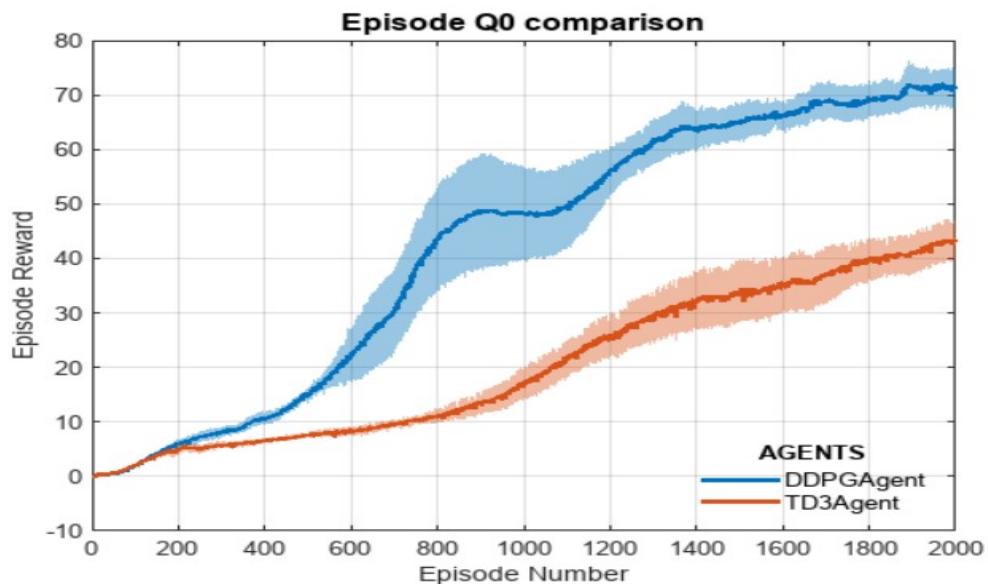
Mentre, sulla base del grafico di confronto dell’Episodio Q0, si avrà che:

- Per l’agente TD3, la stima del critico della ricompensa scontata a lungo termine (per 2000 episodi) è più bassa rispetto all’agente DDPG. Questa differenza è dovuta al fatto che

l'algoritmo TD3 adotta un approccio conservativo nell'aggiornamento degli obiettivi, utilizzando un minimo di due funzioni Q. Questo comportamento è ulteriormente rafforzato dal ritardo nell'aggiornamento degli obiettivi.



**Figura 5.5:** Confronto tra le curve di apprendimento



**Figura 5.6:** Confronto tra le curve dell'Episode Q0

- Sebbene la stima TD3 per questi 2000 episodi sia bassa, l'agente TD3 mostra un aumento costante dei valori Q0 degli episodi, a differenza dell'agente DDPG, che, al contrario, è più variabile.

In questo esempio, l'addestramento è stato interrotto a 2000 episodi. Per un periodo di addestramento più lungo, l'agente TD3, con il suo aumento costante delle stime, avrebbe mostrato una convergenza verso la vera ricompensa scontata a lungo termine.

## 5.4 Valutazione e analisi delle esperienze di simulazione 3

Prima di iniziare la discussione sull’analisi delle esperienze anche per questa simulazione, è bene ripetere che gli elementi che valuteremo, relativi agli esperimenti, sono gli stessi di quelli visti nella Sezione 3.4. Tali elementi sono:

- *Average Reward* (o Ricompensa Media);
- *Episode Reward* (o Ricompensa Periodica);
- *Episode Q0*.

Detto ciò, è doveroso anche dire che le esperienze che si valuteranno per questo specifico caso, al contrario delle Simulazioni 1 e 2, non hanno un obiettivo comune di Average Reward da raggiungere, ma l’addestramento per ogni esperienza finirà solo quando sono stati eseguiti tutti i passi pre-impostati inizialmente, cioè 2000.

Inoltre, verranno valutate due esperienze, la prima riguardante l’agente DDPG e la seconda relativa all’agente TD3, per poi mettere a confronto i risultati ottenuti dai rispettivi test.

### 5.4.1 Esperienza 1

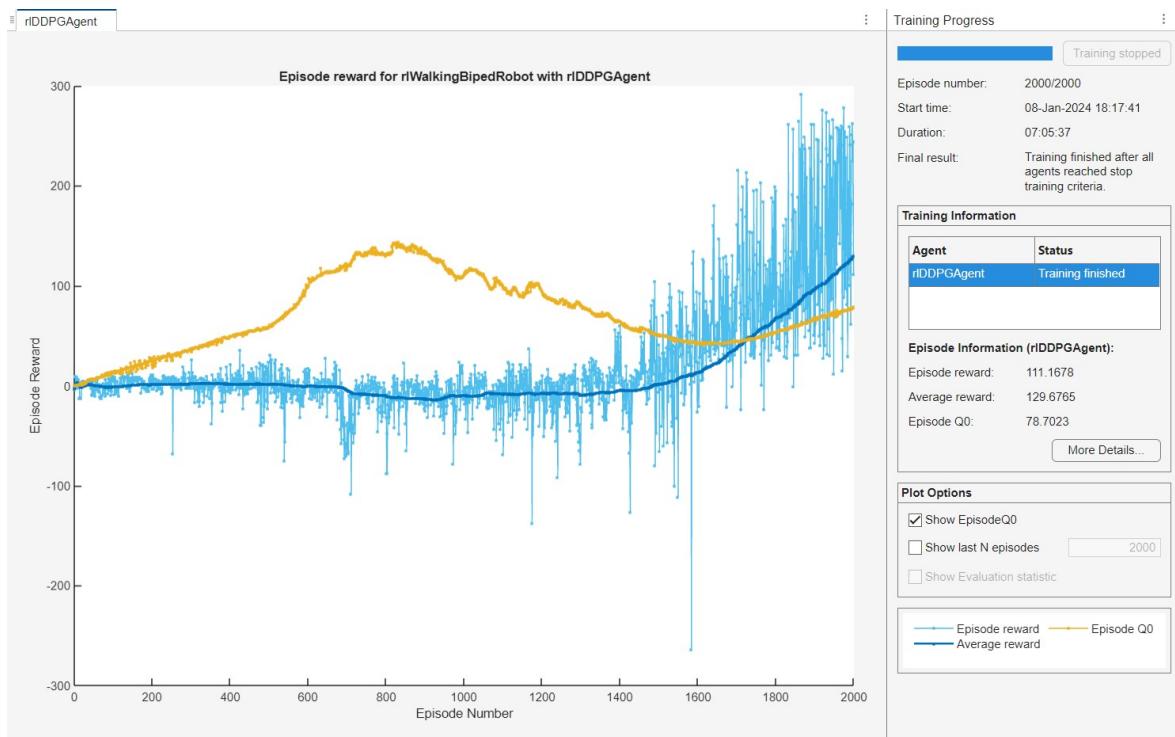
Per questa esperienza, verranno valutate le prestazioni dell’agente Deep Deterministic Policy Gradient, o DDPG, quindi, il codice rimarrà pressoché uguale; l’unica modifica che bisogna impostare è il tipo di agente che si decide di utilizzare. In questo modo, il programma saprà quale tra i due differenti agenti scegliere per effettuare la simulazione.

```
AgentSelection = "DDPG";
switch AgentSelection
    case "DDPG"
        agent = createDDPGAgent (numObs, obsInfo, numAct, actInfo, Ts);
    case "TD3"
        agent = createTD3Agent (numObs, obsInfo, numAct, actInfo, Ts);
    otherwise
        disp("Assign AgentSelection to DDPG or TD3")
end
```

Inoltre, all’interno della funzione `createDDPGAgent` si possono trovare tutte le specifiche per la creazione dell’agente in questione, definite in Appendice B.

Detto ciò, possiamo analizzare i dati finali relativi a questa prima esperienza, presenti in Figura 5.7. È interessante notare come, per arrivare al risultato finale, il tempo di simulazione è durato poco più di 7 ore. Inoltre, come già detto, la simulazione è finita soltanto quando sono stati completati tutti gli episodi, come inizialmente impostato.

A questo punto, si possono analizzare le diverse fasi relative alle tre differenti curve. La curva dell’*average reward*, come già sappiamo, mostra la media delle ricompense calcolate su una finestra scorrevole di 250 episodi, come specificato dal parametro `ScoreAveragingWindowLength`. Questa curva mostra un trend generale di miglioramento, iniziando da valori bassi e arrivando a una pendenza stabile e positiva, il che indica che l’agente ha imparato ed è migliorato nel tempo. Si possono, anche, riconoscere diverse fasi della curva; infatti, si ha una prima fase iniziale di apprendimento, fino all’episodio 500 circa, nella quale l’agente sta iniziando a imparare e ad accumulare ricompense positive. Dall’episodio 500 fino all’intorno dell’episodio 1500, si ha una fase di stabilità e linearità, il che suggerisce che l’agente ha



**Figura 5.7:** Risultato finale dell’Esperienza 1 per il simulatore Walking Biped Robot

trovato strategie efficaci che funzionano bene nell’ambiente, seppure si possa notare una leggera diminuzione della ricompensa; da qui in poi, fino alla fine della simulazione, si nota una fase di maturità; infatti, la curva tende a crescere, fino ad arrivare ad un valore di ricompensa media pari a 129.

La curva dell’*episode reward*, invece, mostra la ricompensa ottenuta per ogni episodio individuale. È evidente la grande varianza nelle ricompense individuali per episodio, il che è normale in un ambiente di Reinforcement Learning dove l’agente può esplorare diverse strategie. Nonostante la varianza, c’è un trend di miglioramento sottile, come mostrato dall’aumento della densità delle ricompense verso valori più alti man mano che il numero degli episodi aumenta. Anche in questo caso, possiamo riconoscere le diverse fasi della curva. Si ha una fase iniziale di esplorazione e instabilità; infatti, dall’inizio fino a circa l’episodio 500, la curva mostra una grande varianza, con ricompense che cambiano ampiamente da un episodio all’altro, indicando un periodo di esplorazione intensa e di apprendimento dalle diverse conseguenze delle azioni. Poi, dall’episodio 500 a circa l’episodio 1500, si nota una fase di consolidamento in cui la varianza nelle ricompense per episodio sembra diminuire leggermente, seppure con alcuni picchi negativi, suggerendo che l’agente sta consolidando le sue strategie. Dall’episodio 1500 fino alla fine, invece, si ha una fase di ottimizzazione o variabilità ambientale, il che permette una leggera diminuzione della varianza e un aumento delle ricompense, anche se con alcune di queste negative, che suggeriscono sfide nell’ambiente o nella politica dell’agente che richiedono ulteriori ottimizzazioni.

L’ultima curva, quella relativa all’*episode Q0*, rappresenta il valore Q stimato per il primo passo di ogni episodio. Si nota da subito che il valore di Q0 aumenta significativamente all’inizio e poi inizia a stabilizzarsi. Ciò indica che l’agente sta diventando più sicuro della sua stima del valore iniziale nel tempo. Il leggero calo verso la fine potrebbe indicare una certa volatilità nell’apprendimento o un possibile sovraddattamento. Per quanto riguarda le diverse fasi di questa curva, anche in questo caso, inizialmente si nota una fase di apprendimento; infatti, dall’inizio fino a circa l’episodio 200, c’è un aumento significativo, che dimostra un rapido

apprendimento iniziale dell’agente riguardo al valore dello stato iniziale. Successivamente, si ha una fase di aggiustamento, dall’episodio 200 fino all’intorno dell’episodio 1000; la curva, infatti, continua a crescere, ma con un tasso di incremento più lento, suggerendo che l’agente stia perfezionando la sua stima di Q0 basandosi sull’esperienza raccolta. Infine, dall’episodio 1000 fino a quando la simulazione non termina, la curva scende, mostra minori variazioni e tende a stabilizzarsi, indicando che l’agente ha una buona stima del valore iniziale e sta facendo aggiustamenti più fini oppure è in una fase di plateau<sup>1</sup>.

Una volta analizzati i dati ottenuti dall’Esperienza 1, passiamo, ora, ad analizzare i dati acquisiti dalla seconda esperienza per questo simulatore.

### 5.4.2 Esperienza 2

Per l’esecuzione dell’esperienza 2, verranno valutate le prestazioni dell’agente Twin Delayed Deep Deterministic Policy Gradient, o TD3; quindi, come in precedenza, il codice rimarrà uguale. L’unica modifica da fare è quella di scegliere il suddetto agente invece del precedente.

```
AgentSelection = "TD3";
switch AgentSelection
    case "DDPG"
        agent = createDDPGAgent (numObs, obsInfo, numAct, actInfo, Ts);
    case "TD3"
        agent = createTD3Agent (numObs, obsInfo, numAct, actInfo, Ts);
    otherwise
        disp("Assign AgentSelection to DDPG or TD3")
end
```

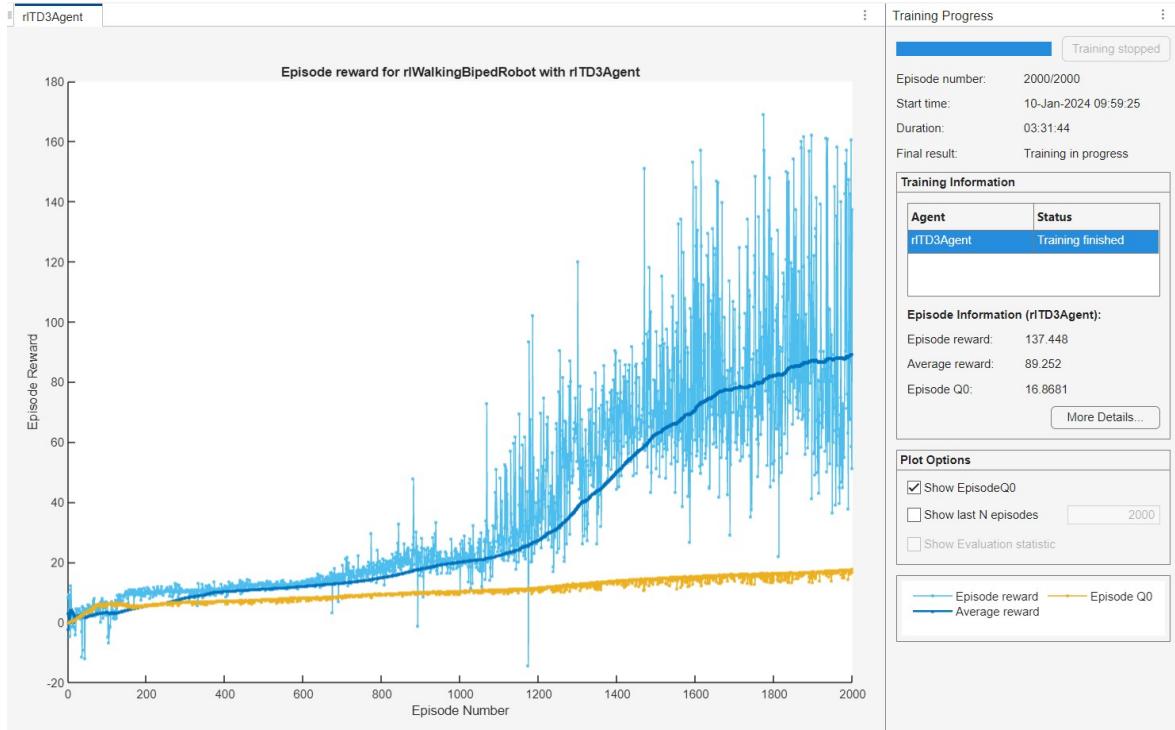
Inoltre, all’interno della funzione `createTD3Agent` si possono trovare tutte le specifiche per la creazione dell’agente in questione, definite in Appendice B.

Detto ciò, possiamo analizzare i dati finali relativi a questa prima esperienza, presenti in Figura 5.8. È interessante notare come, per arrivare al risultato finale, il tempo di simulazione è di solo 3 ore e 30 minuti, contro le 7 dell’esperienza precedente. Inoltre, come nel caso precedente, la simulazione è finita soltanto quando sono stati completati tutti gli episodi, come inizialmente impostato. Detto questo, si possono analizzare le diverse fasi delle tre differenti curve.

Per quanto riguarda la curva relativa all’*average reward*, si può dire che essa ha un andamento prevalentemente crescente; infatti, quest’ultima inizialmente è costante e poi subisce un notevole miglioramento verso la fase finale. Se si volessero dividere le diverse fasi di questa curva, allora inizialmente si è in presenza di una fase di esplorazione (fino all’episodio 500 circa) durante la quale la curva mostra un aumento graduale e con valori bassi; ciò suggerisce che l’agente sta cominciando ad apprendere dalla sua esperienza, ma non ha ancora trovato strategie consistentemente vantaggiose. Dall’episodio 500 all’episodio 1500 circa, si nota una fase di apprendimento accelerato e si osserva un’inclinazione più marcata della curva, indicando che l’agente sta migliorando in modo più efficace e sta accumulando una maggiore comprensione dell’ambiente. Da qui in poi, fino alla fine, si ha una fase di affinamento dei risultati ottenuti; infatti, la curva tende a crescere ma in modo meno marcato, suggerendo che i miglioramenti diventano meno evidenti, seppur presenti.

---

<sup>1</sup>In un contesto di apprendimento automatico e, più specificamente, di apprendimento per rinforzo, un “plateau” si riferisce a una fase durante la quale il miglioramento delle prestazioni di un modello o agente rallenta o si arresta per un certo periodo di tempo.



**Figura 5.8:** Risultato finale dell’Esperienza 2 per il simulatore Walking Biped Robot

La curva relativa all’*episode reward* ha un andamento molto simile a quella precedentemente trattata; infatti, segue la tendenza di quest’ultima durante tutta la durata della simulazione, anche se con molti picchi diversi di valori. Le fasi di questa curva si possono dividere in una fase iniziale di apprendimento, una fase di consolidamento e un’ultima di miglioramento finale. Dato che sono praticamente identiche a quelle della curva dell’average reward, non verranno nuovamente trattate.

La curva riguardante l’*episode Q0*, invece, ha un andamento diverso dal solito; infatti, è costantemente crescente, tuttavia, questo miglioramento è minimo per tutta la simulazione; si ha, quindi, una crescita graduale, anche se molto piccola, ad ogni episodio, ed il suo valore finale rimane di poco sopra allo zero. In questo caso potremmo vedere una fase iniziale di apprendimento con valori molto bassi, fino all’episodio 500 circa, nella quale la curva è piatta e bassa, indicando che l’agente sta ancora formando una stima iniziale del valore Q. Successivamente, si ha un leggero aumento fino all’intorno dell’episodio 1500 circa; in questo caso la curva sale gradualmente, anche se di poco, suggerendo che l’agente migliora la sua stima del valore Q iniziale sulla base delle ricompense che riceve. L’ultima è una fase di stabilizzazione della stima Q, nella quale la curva si stabilizza con una leggera inclinazione verso l’alto, indicando che l’agente ha una stima coerente, ma leggermente migliorata, del valore Q iniziale verso la fine dell’addestramento.

Una volta raccolti e trattati i dati finali, si può dire, quindi, che ci sono stati dei cambiamenti rispetto alla prima esperienza, sia per quanto riguarda il tempo di completamento, sia per quello che concerne l’andamento delle due diverse curve, le quali sono completamente differenti tra loro. Nella prossima sezione verranno tratte delle conclusioni sulla base dei dati ottenuti e delle considerazioni effettuate per le due differenti esperienze.

## 5.5 Conclusioni

In conclusione, anche questo capitolo ha offerto una panoramica completa dell’esperienza di simulazione, focalizzandosi su diversi aspetti chiave. L’introduzione all’esperienza ha fornito un contesto iniziale, delineando gli obiettivi e le motivazioni alla base della simulazione. Successivamente, l’analisi approfondita del modello Deep Deterministic Policy Gradient (DDPG) ha fornito una comprensione dettagliata dell’algoritmo di Reinforcement Learning utilizzato nel contesto in questa simulazione. La sezione sulla progettazione e configurazione del simulatore ha esplorato le scelte di progettazione implementate nel codice, evidenziando l’implementazione della simulazione stessa.

Infine, nella sezione di valutazione e analisi delle esperienze di simulazione, sono state esaminate due esperienze specifiche. Entrambe avevano lo stesso codice (hanno, però, sfruttato due agenti diversi per la loro esecuzione), per valutare le prestazioni e le dinamiche apprese dagli agenti nel caso preso in esame, ovvero addestrare un robot bipede a camminare autonomamente lungo una linea retta.

Da questi ultimi, si può evincere che entrambe le esperienze hanno avuto esito positivo, seppure i dati ottenuti da esse sono stati differenti, come si può vedere dai grafici stessi, se messi a confronto.

Possiamo dire, in generale, che, per quanto riguarda l’Esperienza 1, il grafico mostra una maggiore varianza rispetto a quello dell’Esperienza 2; inoltre, nel primo caso, l’average reward aumenta in maniera significativa ma mostra una flessione verso la fine, che potrebbe indicare un possibile sovrapprendimento o una difficoltà nell’adattarsi a nuove situazioni dopo aver ottimizzato la politica per quelle già incontrate. Si può anche dire che, per quanto riguarda la Figura 5.7, il valore Q0 sembra stabilizzarsi dopo un rapido aumento iniziale, suggerendo che l’agente ha effettuato una stima abbastanza coerente del valore iniziale man mano che l’addestramento progredisce.

Per quanto riguarda la Figura 5.8, invece, la varianza nella curva dell’episode reward è presente, ma sembra ridursi più rapidamente e meno drasticamente rispetto all’agente DDPG. Questo potrebbe indicare che TD3 gestisce meglio l’esplorazione, probabilmente grazie alla sua natura di algoritmo di apprendimento off-policy con una politica più regolare. Inoltre, l’average reward presenta un trend di crescita più costante e meno soggetto a declino, il che potrebbe riflettere l’efficacia delle tecniche di smoothing implementate nel TD3, come l’uso di due reti Q per mitigare la sovrastima dei valori Q. Oltre a ciò, l’episode Q0 mostra un incremento graduale, indicando un apprendimento e una stima del valore più stabili.

Confrontando i dati, quindi, si può dire che l’agente TD3 ha avuto, in generale, delle risposte migliori rispetto all’agente DDPG per questo tipo di simulazione. Ciò è, probabilmente, dovuto a diversi fattori; ad esempio, poiché TD3 è una versione migliorata del DDPG, esso introduce modifiche specifiche per ridurre il problema della sovrastima del valore Q che può essere presente nel DDPG. Oppure, ciò può essere dovuto al fatto che TD3 utilizza una doppia rete Q (Twin Q-network) e aggiorna la politica meno frequentemente rispetto a DDPG (tecnica nota come policy delay). Questo potrebbe spiegare perché l’average reward nel TD3 mostra una progressione più stabile e meno incline a flessioni.

In sintesi, le differenze nelle curve tra i due grafici possono essere attribuite alle tecniche utilizzate dai due algoritmi, alla loro reazione all’ambiente di addestramento e alla configurazione dell’addestramento stesso. TD3, essendo una versione raffinata del DDPG, sembra offrire un addestramento più stabile e prevedibile per questo tipo di simulazione; ciò si riflette nelle curve più lineari e costanti che si possono osservare nel secondo grafico rispetto al primo.

# CAPITOLO 6

---

## Discussione

---

*Nel presente capitolo si effettuerà un'analisi e una discussione dei tre distinti scenari di simulazione di Reinforcement Learning analizzati nei Capitoli 3, 4 e 5, ognuno dei quali rappresenta un'area cruciale di ricerca e applicazione nel campo dell'Intelligenza Artificiale nell'automazione. Attraverso la discussione di queste simulazioni, non solo valuteremo l'efficacia di diversi algoritmi di RL ma anche il loro impatto nel risolvere problemi specifici e complessi. In ogni simulazione, l'uso di un tipo di algoritmo diverso per programmare l'agente di RL, fornisce un terreno fertile per discussioni riguardo l'efficacia, l'adattabilità e le potenzialità degli algoritmi stessi. L'analisi si concentrerà, per ogni simulazione trattata, sull'utilizzo dei diversi algoritmi, fornendo, così, una comprensione completa delle capacità e dei limiti del RL in scenari diversificati.*

### 6.1 Discussione della simulazione con Robot Arm

In questa sezione verranno discussi i risultati ottenuti nelle tre diverse esperienze trattate nel Capitolo 3. Per analizzare le tre simulazioni, come prima cosa, esaminiamo i dettagli delle impostazioni dell'agente e i risultati, come mostrati nei grafici dei reward.

L'Esperienza 1, che ricordiamo essere stata l'unica simulazione delle tre che ha avuto esito positivo, è stata inizializzata con i seguenti parametri:

- *ExperienceBufferLength*: 1e6; è un buon compromesso tra la capacità di memorizzare un gran numero di esperienze passate senza essere sovraccaricato da vecchie informazioni che potrebbero non essere più rilevanti, permettendo all'agente di generalizzare meglio.
- *MiniBatchSize*: 128; è nella gamma tipica per il batch learning. Un minibatch di questa dimensione può favorire una convergenza stabile, consentendo una stima affidabile del gradiente.
- *DiscountFactor*: 0.99; l'agente è incentivato a considerare i reward a lungo termine, il che è fondamentale per compiti che richiedono una pianificazione prolungata, come mantenere in equilibrio una pallina su un piatto.
- *LearnRate* (sia dell'attore che del critico): 1e-4; un valore così basso suggerisce che l'agente apprende lentamente, riducendo il rischio di grandi aggiustamenti di politica che potrebbero portare a instabilità.
- *Gradient Threshold*: impostato a 1; aiuta a prevenire aggiornamenti troppo grandi, che potrebbero destabilizzare l'apprendimento.

In questa simulazione, l'alto valore del *DiscountFactor* (0.99) indica una forte considerazione per i reward futuri, il che è utile in compiti che richiedono previsione e pianificazione a lungo termine, come il mantenimento dell'equilibrio. La *ExperienceBufferLength* è abbastanza grande per permettere una vasta gamma di esperienze da cui imparare, ma non così grande da rallentare l'apprendimento o da sovraccaricare la memoria con esperienze obsolete. Un *MiniBatchSize* medio e un *LearnRate* basso per attore e critico favoriscono un apprendimento stabile e graduale. Queste impostazioni sembrano avere prodotto un agente che impara efficacemente nel tempo, come evidenziato dalla crescita costante dell'average reward nel grafico di Figura 3.9. Inoltre, suggeriscono che l'equilibrio tra exploration e exploitation<sup>1</sup> sia stato gestito efficacemente.

L'Esperienza 2 (Figura 3.10), la quale ha avuto esito insufficiente, è stata inizializzata con i seguenti parametri:

- *ExperienceBufferLength*: 1e5; l'agente potrebbe non aver avuto abbastanza esperienze passate da cui apprendere, limitando la sua capacità di generalizzare.
- *MiniBatchSize*: 200; questo valore potrebbe introdurre più varianza negli aggiornamenti dei parametri rispetto alla prima simulazione.
- *DiscountFactor*: 0.1; è un valore molto basso, ha probabilmente reso l'agente miope, focalizzandosi troppo sui reward immediati e meno su quelli futuri; ciò non è ottimale per il compito di equilibrio.
- *LearnRate* (sia dell'attore che del critico): 1e-2; potrebbe aver portato a una convergenza prematura o a oscillazioni nel processo di apprendimento.
- *Gradient Threshold*: mantenuto a 1; è utile, ma potrebbe non essere sufficiente a compensare i learning rate più alti.

Qui, un *DiscountFactor* molto basso (0.1) significa che l'agente non valorizza i reward futuri come nella prima simulazione. Questo può portare a una strategia a breve termine che non è ottimale per il compito di mantenimento dell'equilibrio. Inoltre, un *MiniBatchSize* maggiore e *LearnRate* più alti possono aver causato una convergenza prematura o instabilità nell'apprendimento, come si può notare dalla mancanza di miglioramento sostenuto nell'average reward. La combinazione di un basso discount factor e learning rate elevati potrebbe spiegare l'assenza di progressi significativi nell'apprendimento, come mostrato dall'andamento insicuro dell'episode reward e dall'average reward basso e stagnante.

L'Esperienza 3 (Figura 3.11), anch'essa risultante con esito insufficiente, è stata inizializzata con i seguenti parametri:

- *ExperienceBufferLength*: 1e7; è estremamente grande, il che potrebbe includere troppi vecchi campioni, alcuni dei quali potrebbero essere irrilevanti o controproducenti, introducendo rumore nel processo di apprendimento.
- *MiniBatchSize*: 164; è un valore insolito che non sembra giustificato da una particolare strategia e potrebbe contribuire a una stima imprecisa del gradiente.
- *DiscountFactor*: 0.5; è un valore intermedio che non sembra né troppo miope né sufficientemente lungimirante per il compito di equilibrio.

---

<sup>1</sup>Nel contesto del Reinforcement Learning (RL), il termine "exploitation" si riferisce all'approccio in cui un agente sceglie di adottare l'azione che crede massimerà il reward basandosi sulle conoscenze che ha già acquisito.

- *LearnRate* (sia dell'attore che del critico): 1e-3; è ragionevole, ma senza un buon bilanciamento degli altri parametri può non essere sufficiente.
- *Gradient Threshold*: anche in questo caso mantenuto a 1.

Il *DiscountFactor* intermedio (0.5) è una scelta bilanciata, ma senza un miglioramento consistente nell'average reward suggerisce che altri parametri non fossero ottimizzati. Anche se i *LearnRate* sono stati impostati in modo conservativo, il *MiniBatchSize* leggermente superiore e la gigantesca experience buffer potrebbero aver introdotto troppa variabilità o rumore nell'apprendimento. Quindi, l'enorme experience buffer e il discount factor intermedio, insieme a un minibatch size non ottimizzato, suggeriscono che l'agente potrebbe non essere in grado di apprendere con efficacia a causa di una strategia di apprendimento incoerente o troppo sensibile al rumore e alle variazioni nei dati di addestramento.

In conclusione, la configurazione vincente per la prima simulazione suggerisce che un bilanciamento accurato di tutti i parametri è cruciale per il successo dell'apprendimento. I parametri devono essere scelti in modo da bilanciare l'apprendimento a breve e lungo termine e per garantire che l'agente non sia né troppo rapido né troppo lento nell'aggiornare le sue politiche. La seconda e la terza simulazione sembrano soffrire di squilibri in questi aspetti, risultando in prestazioni scadenti. Le lezioni apprese qui potrebbero essere utilizzate per affinare ulteriormente l'algoritmo, magari testando variazioni più graduali dei parametri chiave o introducendo metodi di normalizzazione o regolarizzazione che potrebbero aiutare a stabilizzare il processo di apprendimento.

## 6.2 Discussione della simulazione con Automatic Parking Valet

In questa sezione verranno discussi i risultati ottenuti nelle due diverse esperienze trattate nel Capitolo 4. Per analizzare le due simulazioni, come prima cosa, esaminiamo i dettagli delle impostazioni dell'agente e i risultati come mostrati nei grafici dei reward.

L'Esperienza 1, che è terminata con esito positivo, come si può notare in Figura 4.11, è stata inizializzata con i seguenti parametri:

- *Critic Network*: due strati nascosti da 128 neuroni ciascuno.
- *Experience Buffer Length*: 1e6; è una dimensione standard per un buffer di esperienza.
- *MiniBatch Size*: 128; anche questa è una dimensione comune che bilancia l'efficienza computazionale e la stabilità dell'apprendimento.
- *Exploration Model Standard Deviation*: parte da 0.1 e si riduce fino a 0.01, indicando una politica iniziale di esplorazione moderata che diventa più conservativa nel tempo.
- *Learning Rate* (sia dell'attore che del critico): l'attore ha un learning rate di 1e-3 e il critico di 2e-3, indicando un apprendimento più cauto per l'attore e leggermente più "aggressivo" per il critico.

Dal grafico si può osservare che l'apprendimento progredisce in maniera consistente, con l'average reward che aumenta stabilmente e l'episode reward che mostra una tendenza alla crescita verso la fine del training. Ciò suggerisce che l'agente ha imparato efficacemente il compito ad esso assegnato.

L'Esperienza 2, anch'essa terminata con esito positivo, come si può notare in Figura 4.11, è stata inizializzata con i seguenti parametri:

- *Critic Network*: due strati nascosti da 256 neuroni ciascuno, quindi una capacità computazionale più elevata rispetto alla prima implementazione.
- *Experience Buffer Length*: 1e6; come nel caso precedente.
- *MiniBatch Size*: aumentato a 192, il che potrebbe permettere una stima del gradiente più precisa ma potrebbe anche aumentare la varianza.
- *Exploration Model Standard Deviation*: inizia da 0.3 e si riduce fino a 0.03, una variazione più marcata che indica una strategia di esplorazione iniziale più aggressiva.
- *Learning Rates* (sia dell'attore che del critico): entrambi i learning rate sono aumentati a 3e-3, il che potrebbe portare a un apprendimento più rapido, ma anche a una maggiore instabilità.

Nel grafico corrispondente, si noti che l'average reward si stabilizza su valori elevati, ma non si osserva lo stesso aumento costante come nella prima implementazione. Inoltre, l'episode Q0 rimane relativamente alto e stabile, suggerendo che l'agente ha una buona stima del valore delle azioni.

Guardando i due grafici, la prima implementazione sembra mostrare una curva di apprendimento più stabile e progressiva. La seconda implementazione, nonostante una rete critica più grande e tassi di apprendimento più elevati, non sembra migliorare significativamente oltre un certo punto, e l'alta varianza dell'episode reward potrebbe indicare un apprendimento meno stabile. In termini di parametri possiamo dire che:

- La dimensione del critic network più grande nella seconda implementazione non sembra tradursi in prestazioni migliori, almeno per la quantità di training mostrata.
- Un *MiniBatchSize* più grande e tassi di apprendimento più elevati potrebbero non essere stati vantaggiosi come previsto, potenzialmente a causa di una maggiore varianza nell'aggiornamento dei pesi.
- L'Exploration Model più aggressivo nella seconda implementazione potrebbe aver contribuito alla varianza nell'apprendimento, come evidenziato dalla dispersione degli episode reward.

Inoltre, dai dati ottenuti e dalle modifiche apportate, si può ipotizzare che alcuni fattori sono determinanti per le prestazioni dell'agente; questi ultimi sono di seguito riportati:

- *Dimensione della Rete e Capacità di Apprendimento*: una rete critica con più neuroni, come nella seconda implementazione, generalmente offre una maggiore capacità di apprendere rappresentazioni complesse dell'ambiente. Tuttavia, se l'ambiente di compito non è sufficientemente complesso da richiedere tale capacità, una rete più piccola può essere più efficiente, come suggerisce la prima simulazione. Questo potrebbe essere il caso del compito per questo tipo di simulazione, che, forse, non richiede la complessità aggiuntiva fornita da una rete più ampia.
- *Exploration e Exploitation*: l'esplorazione aggressiva iniziale della seconda implementazione, con una deviazione standard di esplorazione che inizia a 0.3, potrebbe aver permesso all'agente di esplorare l'ambiente più ampiamente all'inizio. Tuttavia, questa strategia non sembra aver portato a un miglioramento a lungo termine. Potrebbe darsi che, dopo una certa soglia, una maggiore esplorazione non porti a nuove informazioni utili e, piuttosto, introduca solo rumore e varianza nell'apprendimento.

- *Tasso di Apprendimento e Stabilità:* tassi di apprendimento più elevati possono accelerare l'apprendimento ma possono anche introdurre instabilità, come potrebbe essere stato il caso nella seconda implementazione. La prima implementazione con tassi di apprendimento più bassi mostra un apprendimento più stabile e progressivo, suggerendo che, per questo particolare compito, un apprendimento più lento e stabile è preferibile.

Si possono fare, inoltre, delle ipotesi sull'apprendimento e sull'ambiente, come di seguito riportato:

- *Complessità dell'Ambiente:* se l'ambiente di apprendimento del parcheggio è relativamente semplice e statico, una configurazione più semplice (come quella dell'Esperienza 1) può essere sufficiente per apprendere una strategia efficace, mentre una configurazione più complessa (come quella dell'Esperienza 2) potrebbe non offrire vantaggi significativi.
- *Varianza dell'Apprendimento:* la seconda implementazione mostra una varianza maggiore negli episode reward, che potrebbe indicare che l'agente sperimenta una strategia di apprendimento che non è ancora convergente o stabilizzata. Questo potrebbe essere il risultato dell'uso di una rete più grande, o di tassi di apprendimento più elevati, che necessitano di più tempo per convergere.
- *Convergenza del Reward:* l'incremento graduale e stabile dell'average reward nella prima implementazione suggerisce che l'agente sta convergendo verso una strategia ottimale di parcheggio. L'apparente plateau raggiunto nella seconda implementazione potrebbe indicare che l'agente ha raggiunto un punto in cui ulteriori miglioramenti sono marginali o richiedono aggiustamenti più fini nei parametri.

In conclusione, dai dati ottenuti, sembra che la prima implementazione sia più efficace per il compito di parcheggio autonomo assegnato all'agente, probabilmente a causa di un apprendimento più stabile e di una configurazione di rete che si adatta meglio alla complessità del compito stesso. La seconda implementazione non mostra miglioramenti significativi dopo un certo punto, nonostante una capacità teorica più elevata di apprendimento e una maggiore esplorazione iniziale. Sulla base di queste osservazioni, per compiti di parcheggio che possono non richiedere modelli complessi, una configurazione più semplice e stabile, con una esplorazione e un apprendimento più cauti, potrebbe essere la scelta migliore. Ulteriori sperimentazioni potrebbero includere la regolazione del MiniBatch Size e del tasso di apprendimento per ottimizzare ulteriormente la convergenza e la stabilità.

## 6.3 Discussione della simulazione con Walking Robot

Si possono analizzare e confrontare le performance delle simulazioni, avendo utilizzato i due differenti algoritmi di RL, ovvero DDPG e TD3 per gli agenti, che, però, sono stati addestrati con gli stessi parametri di simulazione e ambiente.

Il DDPG è noto per essere un buon algoritmo quando si ha a che fare con azioni in spazi continui, ma può soffrire di instabilità dovuta alla sua singola stima del Q-value. Il TD3, che, come sappiamo, è una variante del DDPG, è stato progettato per affrontare alcune di queste instabilità utilizzando una coppia di critic network per ottenere una stima del Q-value più stabile e riducendo la sovrastima dei Q-value stessi.

Si può fare, anche, un confronto riguardo alcuni fattori che sono determinanti per le prestazioni degli agenti, come di seguito riportato:

- *Stabilità dell'Apprendimento:* il TD3 mostra una maggiore stabilità nell'apprendimento, con meno varianza nell'episode reward, il che è particolarmente importante in compiti complessi come quello preso in considerazione per questo tipo di simulazione.
- *Velocità di Convergenza:* il grafico TD3 suggerisce che l'agente converge verso una politica ottimale più rapidamente rispetto al DDPG, come evidenziato dall'aumento più uniforme e rapido dell'average reward.
- *Efficienza dell'Esplorazione:* il TD3 sembra gestire meglio l'esplorazione dell'ambiente, rispetto al DDPG, il che potrebbe essere dovuto alla sua politica di esplorazione rumorosa e ai suoi meccanismi di smoothing.

Basandoci sui grafici forniti nelle Figure 5.7 e 5.8 e sulla teoria dietro i due algoritmi, il TD3 appare come l'algoritmo più efficace per l'addestramento del robot bipede. Non solo sembra apprendere più rapidamente una politica efficace, ma lo fa con una maggiore consistenza e stabilità. Questi risultati sono in linea con le aspettative teoriche e con la letteratura scientifica che, generalmente, riconosce il TD3 come un algoritmo che migliora le prestazioni e la stabilità rispetto al suo predecessore, il DDPG.

Tuttavia, è importante notare che l'efficacia dell'algoritmo può essere fortemente influenzata dal contesto specifico, inclusa la configurazione dell'ambiente di apprendimento, i parametri dell'algoritmo e le caratteristiche del compito. Inoltre, un'analisi più approfondita dovrebbe considerare non solo la performance finale, ma anche fattori come la velocità di apprendimento, la robustezza delle politiche apprese e la capacità dell'algoritmo di gestire le sfide poste da diversi scenari di test.

In conclusione, anche se entrambi gli algoritmi hanno mostrato la capacità di apprendere il compito, l'approccio TD3 sembra essere superiore in termini di stabilità e convergenza. Ciò suggerisce che tale approccio potrebbe essere una scelta migliore per le applicazioni in cui è richiesta una convergenza rapida e affidabile ed una politica di controllo efficace, come nel caso di fare camminare autonomamente un robot bipede.

---

## Conclusioni

---

La tesi presentata offre un'analisi metodica e approfondita del Reinforcement Learning (RL), una branca del Machine Learning (ML) che sta acquisendo sempre maggiore rilevanza nell'ambito dell'Intelligenza Artificiale, con applicazioni che spaziano dal settore industriale a quello robotico. Il lavoro è stato suddiviso in sei capitoli che esplorano in dettaglio gli aspetti teorici del RL, le sue implementazioni in ambito industriale e le sue applicazioni in complesse simulazioni robotiche.

Il primo capitolo è servito da introduzione al RL, delineando i suoi fondamenti e le sue relazioni con il Machine Learning. Viene presentata una panoramica delle tipologie di apprendimento e degli algoritmi più impiegati, insieme a una discussione sugli aspetti etici e sociali del loro utilizzo. Questo approccio preliminare pone le basi per una comprensione approfondita dei capitoli successivi.

Il secondo capitolo si immerge nel contesto dell'Industria 4.0, illustrando come il RL si inserisca all'interno di questo paradigma tecnologico. Vengono esplorate le sinergie tra le Smart Factory e le tecnologie abilitanti, come l'IoT e l'analisi dei big data, enfatizzando i benefici derivanti dall'applicazione del RL in questo settore.

I capitoli dal terzo al quinto rappresentano il cuore sperimentale della tesi, dove vengono presentate tre diverse simulazioni sperimentate in ambiente MATLAB e Simulink: il RobotArm, l'Automatic Parking Valet e il Walking Robot. Per ogni scenario, la tesi approfondisce l'utilizzo di specifici algoritmi di RL, come il Soft Actor Critic (SAC), il Twin-Delayed DDPG (TD3) ed il Deep Deterministic Policy Gradient (DDPG), ed illustra le metodologie di configurazione dei simulatori e l'ottimizzazione degli agenti di apprendimento. Questi capitoli offrono una visione approfondita di come gli agenti di RL vengano creati, addestrati e ottimizzati, mostrando l'efficacia delle tecniche di apprendimento automatico nel risolvere compiti specifici e complessi.

Ogni simulazione è seguita da una valutazione critica delle esperienze, fornendo analisi dettagliate delle prestazioni degli agenti e delle sfide incontrate durante l'addestramento. Queste sezioni non solo hanno messo in luce i successi ottenuti, ma anche le limitazioni e i potenziali miglioramenti, sottolineando l'importanza del processo iterativo nell'adattamento e perfezionamento delle tecniche di RL.

Il sesto ed ultimo capitolo si dedica alla discussione dei risultati ottenuti, mettendo a confronto le diverse simulazioni e valutando l'efficacia degli approcci di RL utilizzati. Qui, la tesi si distacca dai dettagli tecnici per concentrarsi sull'impatto e le implicazioni pratiche delle simulazioni.

Per quanto riguarda i possibili sviluppi futuri, la ricerca apre diverse direzioni interessanti. Un'area di sviluppo potrebbe essere l'integrazione del RL con altre forme di Intelligenza

Artificiale, come il Deep Learning (DP) e il Machine Learning, per creare agenti ancora più autonomi e adattabili. Potrebbero essere esplorati algoritmi ibridi che combinano il RL con altri metodi di apprendimento per accelerare il training e aumentare la generalizzazione delle competenze acquisite dagli agenti.

Un altro sviluppo potrebbe riguardare l'ottimizzazione degli stessi processi di simulazione. Con il miglioramento delle capacità di calcolo e l'evoluzione delle tecniche di modellazione, sarà possibile creare ambienti simulati ancora più dettagliati e realistici che potrebbero prevedere e testare il comportamento degli agenti di RL in una varietà di scenari prima impensabili, riducendo la distanza tra simulazione e realtà.

In ambito industriale, l'adozione di sistemi basati sul RL potrebbe essere ulteriormente ampliata per includere non solo le operazioni di produzione ma anche la manutenzione predittiva e la logistica interna. Le Smart Factory del futuro potrebbero impiegare robot interconnessi che apprendono e si adattano in modo collaborativo, migliorando continuamente l'efficienza operativa e la sicurezza.

Nel campo della robotica, il RL potrebbe essere applicato allo sviluppo di robot con gradi di libertà ancora maggiori, capaci di svolgere compiti di precisione o di assistere gli umani in ambienti pericolosi. La convergenza tra tecnologie indossabili, sensoristica avanzata e RL potrebbe portare alla creazione di sistemi robotici altamente interattivi e personalizzabili. I robot, quindi, potrebbero diventare compagni intelligenti in grado di apprendere dalle azioni umane e fornire assistenza proattiva, ad esempio in contesti di riabilitazione o in operazioni di soccorso, in cui sono richieste reazioni rapide e adattamenti a condizioni imprevedibili.

Infine, un concetto di vitale importanza, sul quale tutt'oggi si discute ampiamente, riguarda l'etica e la governance dell'uso degli agenti di RL. Man mano che questi sistemi diventano più autonomi e capaci, sarà fondamentale sviluppare quadri normativi e standard etici per assicurare che siano utilizzati in modo responsabile e per il bene comune.

In sintesi, mentre la tesi ha stabilito le fondamenta per l'utilizzo avanzato del RL in vari scenari simulati, i possibili sviluppi futuri suggeriscono un orizzonte espansivo per questa tecnologia. I percorsi futuri potrebbero non solo ampliare la portata del RL ma anche trasformare radicalmente il modo in cui interagiamo con le macchine e comprendiamo l'apprendimento autonomo. Con un occhio vigile sulle implicazioni etiche e un altro sull'innovazione tecnologica, il campo del Reinforcement Learning è destinato a essere un protagonista chiave nella prossima era dell'automazione intelligente e della robotica avanzata, con particolare rilevanza all'interno dell'Industria 4.0.

In conclusione, la tesi ha rappresentato un'esperienza significativa nel campo dell'Intelligenza Artificiale e del Machine Learning, in particolare, per quanto riguarda l'applicazione del Reinforcement Learning. Attraverso un'analisi dettagliata teorica e sperimentale, il lavoro ha dimostrato come gli algoritmi di RL possano essere applicati con successo per automatizzare e ottimizzare processi in scenari complessi, evidenziando il potenziale di questa tecnologia nell'evoluzione verso industrie intelligenti e robot autonomi. La ricerca lascia intravedere un futuro in cui il RL potrebbe giocare un ruolo chiave nella risoluzione di problemi avanzati, suggerendo nuove direzioni per studi futuri e applicazioni pratiche.

# APPENDICE A

---

## RobotArm Functions

---

### Environment Reset Function

```
function in = kinovaResetFcn(in)
% KinovaResetFcn is used to randomize the initial joint angles R6_q0,
% R7_q0 and the initial wrist and hand torque values.

% Ball parameters
ball.radius = 0.02;      % m
ball.mass   = 0.0027;     % kg
ball.shell   = 0.0002;     % m

% Calculate ball moment of inertia.
ball.moi = calcMOI(ball.radius,ball.shell,ball.mass);

% Initial conditions. +z is vertically upward.
% Randomize the x and y distances within the plate.

% m, initial x distance from plate center
ball.x0 = -0.125 + 0.25*rand;

% m, initial y distance from plate center
ball.y0 = -0.125 + 0.25*rand;

% m, initial z height from plate surface
ball.z0 = ball.radius;

ball.dx0 = 0;    % m/s, ball initial x velocity
ball.dy0 = 0;    % m/s, ball initial y velocity
ball.dz0 = 0;    % m/s, ball initial z velocity

% Contact friction parameters
ball.staticfriction = 0.5;
ball.dynamicfriction = 0.3;
ball.criticalvelocity = 1e-3;

% Convert coefficient of restitution to spring-damper parameters.
coeff_restitution = 0.89;
[k, c, w] = cor2SpringDamperParams(coeff_restitution,ball.mass);
ball.stiffness = k;
ball.damping = c;
ball.transitionwidth = w;

in = setVariable(in,"ball",ball);

% Randomize joint angles within a range of +/- 5 deg from the
% starting positions of the joints.
R6_q0 = deg2rad(-65) + deg2rad(-5+10*rand);
```

---

```
R7_q0 = deg2rad(-90) + deg2rad(-5+10*rand);
in = setVariable(in,"R6_q0",R6_q0);
in = setVariable(in,"R7_q0",R7_q0);

% Compute approximate initial joint torques that hold the ball,
% plate and arm at their initial configuration
g = 9.80665;
wrist_torque_0 = ...
    (-1.882 + ball.x0 * ball.mass * g) * cos(deg2rad(-65) - R6_q0);
hand_torque_0 = ...
    (0.0002349 - ball.y0 * ball.mass * g) * cos(deg2rad(-90) - R7_q0);
U0 = [wrist_torque_0 hand_torque_0];
in = setVariable(in,"U0",U0);
end
```

## Data Logging Functions

```
function dataToLog = logAgentLearnData(data)
% This function is executed after completion
% of the agent's learning subroutine

dataToLog.ActorLoss = data.ActorLoss;
dataToLog.CriticLoss = data.CriticLoss;

end

function dataToLog = logEpisodeData(data, doViz)
% This function is executed after the completion of an episode

dataToLog.Experience = data.Experience;

% Show an animation after episode completion
if doViz
    animatedPath(data.Experience);
end
```

## APPENDICE B

---

### Walking Robot Agent Functions

---

#### DDPG Agent Function

```
function agent = createDDPGAgent(numObs, obsInfo, numAct, actInfo, Ts)
% Walking Robot -- DDPG Agent Setup Script
% Copyright 2020 The MathWorks, Inc.

%% Create the actor and critic networks using the createNetworks helper function
[criticNetwork,~,actorNetwork] = createNetworks(numObs,numAct);

%% Specify options for the critic and actor representations using rlOptimizerOptions
criticOptions = rlOptimizerOptions('Optimizer','adam','LearnRate',1e-3, ...
    'GradientThreshold',1,'L2RegularizationFactor',2e-4);
actorOptions = rlOptimizerOptions('Optimizer','adam','LearnRate',1e-3, ...
    'GradientThreshold',1,'L2RegularizationFactor',1e-5);

%% Create critic and actor representations using specified networks and
%% options
critic = rlQValueFunction(criticNetwork,obsInfo,actInfo,'ObservationInputNames',...
    ...'observation','ActionInputNames','action');
actor = rlContinuousDeterministicActor(actorNetwork,obsInfo,actInfo);

%% Specify DDPG agent options
agentOptions = rlDDPGAgentOptions;
agentOptions.SampleTime = Ts;
agentOptions.DiscountFactor = 0.99;
agentOptions.MiniBatchSize = 256;
agentOptions.ExperienceBufferLength = 1e6;
agentOptions.TargetSmoothFactor = 5e-3;
agentOptions.NoiseOptions.MeanAttractionConstant = 1;
agentOptions.NoiseOptions.Variance = 0.1;
agentOptions.ActorOptimizerOptions = actorOptions;
agentOptions.CriticOptimizerOptions = criticOptions;

%% Create agent using specified actor representation, critic representation
%% and agent options
agent = rlDDPGAgent(actor,critic,agentOptions);
```

#### TD3 Agent Function

```
function agent = createTD3Agent(numObs, obsInfo, numAct, actInfo, Ts)
% Walking Robot -- TD3 Agent Setup Script
% Copyright 2020 The MathWorks, Inc.

%% Create the actor and critic networks using the createNetworks helper function
[criticNetwork1,criticNetwork2,actorNetwork] = createNetworks(numObs,numAct);
% Use of 2 Critic networks
```

---

```

%% Specify options for the critic and actor representations using rlOptimizerOptions
criticOptions = rlOptimizerOptions('Optimizer','adam','LearnRate',1e-3,...
    'GradientThreshold',1,'L2RegularizationFactor',2e-4);
actorOptions = rlOptimizerOptions('Optimizer','adam','LearnRate',1e-3,...
    'GradientThreshold',1,'L2RegularizationFactor',1e-5);

%% Create critic and actor representations using specified networks and
% options
critic1 = rlQValueFunction(criticNetwork1,obsInfo,actInfo, ...
    ...'ObservationInputNames','observation','ActionInputNames','action');
critic2 = rlQValueFunction(criticNetwork2,obsInfo,actInfo, ...
    ...'ObservationInputNames','observation','ActionInputNames','action');
actor = rlContinuousDeterministicActor(actorNetwork,obsInfo,actInfo);

%% Specify TD3 agent options
agentOptions = rlTD3AgentOptions;
agentOptions.SampleTime = Ts;
agentOptions.DiscountFactor = 0.99;
agentOptions.MiniBatchSize = 256;
agentOptions.ExperienceBufferLength = 1e6;
agentOptions.TargetSmoothFactor = 5e-3;
agentOptions.TargetPolicySmoothModel.Variance = 0.2; % target policy noise
agentOptions.TargetPolicySmoothModel.LowerLimit = -0.5;
agentOptions.TargetPolicySmoothModel.UpperLimit = 0.5;
agentOptions.ExplorationModel = rl.option.OrnsteinUhlenbeckActionNoise;
% set up OU noise as exploration noise (default is Gaussian for rlTD3AgentOptions)
agentOptions.ExplorationModel.MeanAttractionConstant = 1;
agentOptions.ExplorationModel.Variance = 0.1;
agentOptions.ActorOptimizerOptions = actorOptions;
agentOptions.CriticOptimizerOptions = criticOptions;

%% Create agent using specified actor representation, critic representations
and agent options
agent = rlTD3Agent(actor, [critic1,critic2], agentOptions);

```

---

## Bibliografia

---

- A. ASADULAEV, G. S., I. KUZNETSOV e FILCHENKOV, A. (2020), «Exploring and Exploiting Conditioning of Reinforcement Learning Agents», *IEEE Access*.
- ANGELOS ANGELOPOULOS, N. N., EMMANUEL T. MICHAILIDIS (2020), «Tackling Faults in the Industry 4.0 Era—A Survey of Machine-Learning Solutions and Key Aspects», *MDPI*. (Cited at page 26)
- COSTA, M. (2018), «Il talento capacitante in Industry 4.0», *Pensa Multimedia*.
- DAVID MORIARTY, J. G., ALAN SCHULTZ (1999), «Evolutionary Algorithms for Reinforcement Learning», *Journal of Artificial Intelligence Research*.
- ECOFFET, A. e LEHMAN, J. (2021), «Reinforcement Learning Under Moral Uncertainty», in MEILA, M. e ZHANG, T., curatori, «Proceedings of the 38th International Conference on Machine Learning», vol. 139 di *Proceedings of Machine Learning Research*, p. 2926–2936, PMLR, URL <https://proceedings.mlr.press/v139/eco021a.html>.
- FENGJIAO ZHANG, Z. L., JIE LI (2020), «A TD3-based multi-agent deep reinforcement learning method in mixed cooperation-competition environment», *Neurocomputing*.
- HA, D. (2019), «Reinforcement Learning for Improving Agent Design», *Artificial Life*.
- HEINER LASI, P. F. (2014), «Industry 4.0», *Business & Information Systems Engineering*.
- L. P. KAELBLING, A. W. M., M. L. LITTMAN (1996), «Reinforcement Learning: A Survey», *Journal of Artificial Intelligence Research*.
- LI, Y. (2019), «Reinforcement Learning Applications», *arXiv*.
- LI, Z., CHENG, X., PENG, X. B., ABBEEL, P., LEVINE, S., BERSETH, G. e SREENATH, K. (2021), «Reinforcement Learning for Robust Parameterized Locomotion Control of Bipedal Robots», in «2021 IEEE International Conference on Robotics and Automation (ICRA)», p. 2811–2817.
- MEHLIG, B. (2021), *Machine learning with neural networks*, UNIVERSITY OF GOTHENBURG.
- MITCHELL, T. M. (1997), *Machine Learning*, McGraw-Hill Science/Engineering/Math.
- SOMALVICO, M. (1987), «Intelligenza Artificiale», Rap. tecn., Politecnico di Milano.

- SUTTON, R. S. e BARTO, A. G. (2018), *Reinforcement Learning. An introduction, second edition*, The MIT Press.
- SZEPESVÁRI, C. (2022), *Algorithms for Reinforcement Learning*, Springer Nature Switzerland.
- TAMÁS KEGYES, J. A., ZOLTÁN SÜLE (2021), «The Applicability of Reinforcement Learning Methods in the Development of Industry 4.0 Applications», *Complexity*.
- TIMOTHY P. LILLICRAP, A. P., JONATHAN J. HUNT (2019), «Continuous control with deep reinforcement learning», *arXiv*.
- TOSI, G. (2015-2016), «Reinforcement Learning per robot grasping in ambiente Gym OpenAI», .
- TUOMAS, K. H., AURICK ZHOU (2019), «Soft Actor-Critic Algorithms and Application.», *Arxiv*.

### Siti web consultati

- Wikipedia –[www.wikipedia.org](http://www.wikipedia.org)
- SAP.com –<https://www.sap.com/italy/products/artificial-intelligence/what-is-machine-learning.html>
- MathWorks.com –<https://it.mathworks.com/>
- Robotic Sea Bass –<https://roboticseabass.com/2020/08/02/an-intuitive-guide-to-reinforcement-learning/>
- GitHub.com –<https://aarl-ieee-nitk.github.io/reinforcement-learning,/value-based-learning,/bootstrapped-learning,/sampled-learning/2019/12/19/Temporal-Difference-Learning.html>
- EconomyUP.it –<https://www.economyup.it/innovazione/cos-e-1-industria-40-e-perche-e-importante-saperla-affrontare/>
- OdinSchool –<https://www.odinschool.com/blog/top-100-reinforcement-learning-real-life-examples-and-its-challenges>
- MathWorks –<https://it.mathworks.com/help/reinforcement-learning/ug/sac-agents.html>
- MathWorks –<https://it.mathworks.com/help/reinforcement-learning/ug/train-sac-agent-for-ball-balance-control.html>
- Arxiv –<https://arxiv.org/abs/1812.05905>
- MathWorks –<https://it.mathworks.com/help/reinforcement-learning/ug/td3-agents.html>
- MathWorks –<https://it.mathworks.com/help/reinforcement-learning/ug/automatic-parking-valet-with-mpc-and-unreal-engine.html>

- **Wikipedia** – [https://en.wikipedia.org/wiki/Model\\_predictive\\_control](https://en.wikipedia.org/wiki/Model_predictive_control)
- **Medium** – <https://medium.com/analytics-vidhya/regularization-understanding-l1-and-l2-regularization-for-deep-learning-a7b9e4a409bf>
- **MathWorks** – <https://it.mathworks.com/help/reinforcement-learning/ug/train-biped-robot-to-walk-using-reinforcement-learning-agents.html>
- **MathWorks** – <https://it.mathworks.com/help/reinforcement-learning/ug/ddpg-agents.html>
- **AmazonWebServer** – <https://aws.amazon.com/it/what-is/hyperparameter-tuning/>

---

## Ringraziamenti

---

Questi tre anni di studio ed i sacrifici fatti, oltre che a me stesso, sono dedicati a tutte le persone importanti nella mia vita, a chi è ancora presente, a chi c'è stato per un breve periodo e a chi sarà per sempre al mio fianco.

Vorrei ringraziare, innanzitutto, la mia famiglia, che, come ha sempre fatto (e come sono sicuro che sempre farà), mi ha sostenuto durante tutto il mio percorso di studi. Ringrazio i miei genitori, Ferdinando e Debora, che non mi hanno mai fatto pesare un esame andato male, facendomi accettare le sconfitte, che, come sempre, accadono nella vita. Li ringrazio per avermi fatto capire che un fallimento è solo un modo per migliorare la volta successiva, che l'importante nella vita è provare e non fermarsi al "se avessi fatto", ma, soprattutto, li ringrazio per avermi fatto seguire sempre la mia strada, convinti che qualsiasi scelta avessi preso sarebbe stata quella giusta. Ringrazio mia sorella, Sofia, che, per quanto è spesso lontana, è allo stesso tempo sempre presente.

Vorrei ringraziare i miei amici, quelli con cui esco da una vita, con i quali ci si aiuta nei momenti difficili; in particolare Leonardo, con la sua genialità e sregolatezza, sempre pronto a darmi una mano quando studiando non capivo qualcosa. Parlando di amici, vorrei ringraziare anche il mio fedele collega e amico Federico, compagno di studio e non, senza il quale l'università sarebbe stata molto più difficile e pesante; solo a pensare alle cavolate dette durante le sessioni di studio ancora mi viene da ridere.

Vorrei ringraziare, inoltre, la mia ragazza Aurora, sempre presente al mio fianco da più di un anno a questa parte, con la quale ho condiviso gioie e dolori, sia universitari che non.

Infine, in ambito universitario, vorrei ringraziare inizialmente il Professore Ursino, sempre gentile, cordiale, disponibile e puntuale, per avermi concesso la possibilità di scrivere questa tesi, alla quale mi sono molto appassionato. Ringrazio anche Alex, che mi ha aiutato in questi anni a destreggiarmi tra tutte le pratiche burocratiche dell'università. Per finire, vorrei anche ringraziare tutte le persone e gli amici che ho conosciuto in questi anni di università, alle quali penserò sempre con affetto.

Grazie a tutti.