Politecnico di Torino

POLITECNICO DI TORINO

Corso di Laurea Magistrale in
INGEGNERIA MECCANICA
A.a. 2023/2024
Sessione di Laurea LUGLIO 2024

Pianificazione di traiettorie adattive di un robot collaborativo con tecniche di Reinforcement Learning

Relatore:
Prof. Stefano Paolo Pastorelli
Correlatori:
Ing. Michele Polito
Ing. Valerio Cornagliotto

Candidato:
Stefano Altopiedi

ABSTRACT

La robotica collaborativa, introdotta con l'Industria 4.0, trova la sua massima espressione con l'avvento dell'Industria 5.0, la cui filosofia è basata sulla stretta collaborazione uomo-robot. I cobot offrono numerosi benefici, migliorando la produttività, garantendo una maggiore sicurezza sul posto di lavoro ed una riduzione della fatica fisica degli operatori. Un aspetto fondamentale per questi dispositivi è rappresentato dalla pianificazione della traiettoria, che assicura un movimento sicuro, efficiente e preciso, all'interno di ambienti complessi. Ciò fa della pianificazione della traiettoria un campo della robotica di grande interesse, in continuo studio e sviluppo, con introduzione di tecniche innovative, anche grazie al recente apporto dell'intelligenza artificiale.

Lo scopo dell'elaborato consiste nell'analisi delle possibilità e dei limiti del Reinforcement Learning, applicato per la realizzazione di un pianificatore di traiettoria per robot antropomorfi che si adatti alle condizioni dinamiche di lavoro. In particolare, si vuole realizzare un pianificatore che segua una primitiva di movimento imposta, ma dalla quale possa discostarsi, sia in termini di percorso che di traiettoria, per adattarsi all'ambiente in cui si trova, ottimizzando l'efficienza di movimento pur garantendo la sicurezza dell'operatore. Si è scelto di utilizzare il Reinforcement Learning per il suo paradigma di ottimizzazione, con potenzialità nella risoluzione di problemi di pianificazione complessi, tenendo conto anche delle esigenze espresse dal singolo operatore. Questo approccio sfrutta interazioni di tipo trial-and-error per generare una Intelligenza Artificiale che ottimizzi il comportamento desiderato del pianificatore, rappresentato dalla funzione di reward.

La prima parte dell'elaborato tratta il collegamento necessario per la comunicazione tra il cobot UR3 e l'ambiente di programmazione Matlab/Simulink, realizzato mediante l'utilizzo di Toolbox annessi ed il software ROS. Si è poi verificato l'avvenuto collegamento con l'esecuzione di prove di pianificazione di traiettoria, per poi eseguire un'analisi dei feedback ricevuti dal robot.
La seconda parte della tesi ha riguardato l'implementazione dell'architettura del Reinforcement Learning. Sono stati progettati più Environment tramite la costruzione in Simulink di modelli rappresentativi del robot e del suo ambiente di lavoro. Si è costruito un primo Environment ad alta complessità che considerasse una situazione molto generale come la presenza di un operatore e l'eventualità di collisioni interne ed esterne. In seguito, per permettere un allenamento progressivo, si è analizzata una condizione in cui il robot dovesse apprendere unicamente una traiettoria specifica, per poi inserire gradualmente fattori relativi a situazioni di maggiore complessità. Si è passati poi alla fase di Training, con la definizione delle fasi di allenamento e dei relativi parametri. Infine, è stato eseguito lo studio dei risultati, con riflessioni riguardanti le potenzialità e le limitazioni dell'applicazione del Reinforcement Learning nell'ambito della pianificazione di traiettoria.

Sommario

Indice delle figure

Indice delle Tabelle

1. INTRODUZIONE

1.1 La robotica collaborativa

La robotica collaborativa, in inglese Human-Robot Collaboration (HRC), è un approccio ingegneristico che ha rivoluzionato il settore industriale e produttivo combinando i vantaggi dell'automazione con le competenze trasversali dei lavoratori umani. Il presupposto è che esseri umani e macchine lavorino insieme in spazi condivisi, non necessariamente in modo continuo/sincrono, garantendo un'interazione sicura all'interno dello spazio di lavoro del robot. La robotica collaborativa ha lo scopo di assistere gli esseri umani nell'esecuzione di compiti, riducendo al minimo il lavoro fisico e sfruttando le capacità cognitive umane per migliorare l'efficienza e la qualità del lavoro [1] [2]. Per garantire questa collaborazione è quindi fondamentale percepire l'ambiente. Vista l'interazione con l'uomo è necessario prevedere le intenzioni dell'operatore per garantire una coesistenza e una collaborazione sicura ed efficace tra uomo e robot. Questi aspetti permettono ai robot di adattarsi dinamicamente alle azioni umane, migliorando così la produttività e la sicurezza nel luogo di lavoro [7].

La robotica collaborativa è un elemento dell'Automazione Industriale, in modo specifico dell'Automazione Robotica. L'automazione industriale si riferisce a sistemi o processi che automatizzano il funzionamento di uno o più macchinari. Questa disciplina è nata nel 18° secolo con la Rivoluzione Industriale, con l'obiettivo di sollevare l'uomo da azioni produttive ripetitive, usuranti ed abitudinarie; Da allora, l'automazione industriale ha avuto una crescita esponenziale per campi di applicazione e di performance tecnologiche. Ha integrato nuovi obiettivi, coperto un maggior numero di funzioni e contribuito all'ottimizzazione delle operazioni e alla velocità di produzione, entrambi fattori cruciali per la produttività di un dato processo. L'automazione ha dato inizio al consumismo di massa, alla standardizzazione e al miglioramento della qualità dei prodotti, garantendo una migliore qualità del lavoro umano, con riduzione dei costi grazie a maggiori ritmi produttivi ed un minor consumo di risorse [4].

L'Automazione Robotica, diramazione dell'Automazione industriale, prevede l'utilizzo di robot autonomi o semiautonomi che, in connessione con altri macchinari, automatizzano i processi produttivi per i diversi task [5]. All'inizio della terza rivoluzione industriale, la robotica prevedeva gli esseri umani e i robot operare separatamente, con i robot che talvolta sostituivano completamente la mano d'opera umana [4]. In risposta a questa dinamica, si è sviluppato il concetto di robotica collaborativa, diffusa in larga scala con l'introduzione dei robot collaborativi, noti come Cobot. Questi rappresentano l'ultimo gradino evolutivo della robotica industriale. Grazie agli avanzati sistemi di sicurezza integrati in questi dispositivi è stato possibile condividere spazio e mansioni con gli esseri umani [5]. Questi tipi di concetti sono stati introdotti nel paradigma dell'Industria 4.0 si caratterizza per l'applicazione avanzata delle tecnologie digitali e dell'Internet of Things (IoT), dove i

dispositivi e i sistemi sono interconnessi ed in grado di scambiare dati in tempo reale. Grazie l'utilizzo dei Cobot, le fabbriche intelligenti possono operare in modo più efficiente, flessibile e personalizzato. L'Industria 4.0 rappresenta una trasformazione radicale nei modelli di produzione e nelle pratiche aziendali [3].

I campi di applicazione in cui si utilizza l'approccio della robotica collaborativa sono numerosi e in continuo aumento, con notevole ottimizzazione dei processi ed un netto guadagno produttivo. Alcuni dei principali ambiti sono [8]:

- Aerospaziale;
- Settore automobilistico;
- Fabbricazione e lavorazione dei metalli;
- Produzione elettronica;
- Produzione di alimenti e bevande;
- Laboratorio e ricerca;
- Produzione di beni durevoli;
- Stampaggio materie plastiche;
- Mobili e attrezzature;

Le prospettive per il prossimo futuro sono racchiuse nell'Industria 5.0, in cui ci si focalizza sulla creazione di un ambiente lavorativo in cui gli esseri umani, i robot e l'intelligenza artificiale (AI) collaborano in modo sinergico tale da massimizzare il valore aggiunto ed il benessere dell'individuo. L'obiettivo dell'Industria 5.0 è la creazione di un sistema produttivo centrato sull'uomo che coordina e programma le funzioni di automazione con la supervisione e l'aiuto delle AI. Grazie alle recenti innovazioni dell'intelligenza artificiale, si sta cercando di estendere il loro utilizzo ad una vasta gamma di applicazioni industriali, consentendo una maggiore automazione ed ottimizzazione dei processi [4]. Si implementa il binomio uomo-robot che operando insieme beneficiano mutuamente dell'azione coordinata di entrambi. La robotica collaborativa è pienamente in linea con questo concetto, per una produzione ed industria più sostenibile, resiliente ed incentrata sul ruolo dell'uomo, in accordo con la doppia transizione verde e digitale. La tecnologia è il mezzo per migliorare la qualità del lavoro umano e facilitare la personalizzazione anche della produzione su larga scala [7].

1.2 Livelli di collaborazione

Per garantire questa collaborazione uomo-robot è quindi fondamentale percepire l'ambiente e prevedere le intenzioni dell'operatore per garantire una coesistenza e una collaborazione sicura ed efficace tra uomo e robot. Questi aspetti permettono ai robot di adattarsi dinamicamente alle azioni umane, migliorando così la produttività e la sicurezza nel luogo di lavoro [7].

Si analizzano i diversi livelli di collaborazione per meglio comprendere il significato di robotica collaborativa. Questi livelli si distinguono in base allo spazio condiviso e al tipo di interazione tra uomo e robot [6]:

- Nessuna coesistenza: si ha separazione fisica tra uomo e robot, con riferimento alle tradizionali celle robotiche recintate;

- Coesistenza: l'essere umano lavora in uno spazio parzialmente o completamente condiviso con il robot, senza obiettivi condivisi: consente la condivisione dello spazio di lavoro fisico ma le attività uomo-robot sono slegate;

- Cooperazione: umano e robot lavorano con un obiettivo comune in uno spazio parzialmente o completamente condiviso: è simile al concetto di Coesistenza, la differenza sta nell'avere lo stesso obiettivo;

- Collaborazione: umano e robot lavorano simultaneamente su un oggetto condiviso in uno spazio condiviso.

La condivisione dello spazio di lavoro significa che l'essere umano e il robot possono operare e muoversi nello stesso spazio fisico. Se il compito del lavoratore dipende dalle attività del robot, l'interazione tra i due passa ad un livello più alto di collaborazione e fattori quali il timing, la leggibilità del movimento e la consapevolezza congiunta della situazione acquisiscono rilevanza significativa.

I livelli sopra elencati possono essere ulteriormente suddivisi in sottolivelli basati su parametri di raffinazione. Questi includono la condivisione dello spazio di lavoro, l'attività del robot in presenza di un essere umano, il tipo di sforzo congiunto ed il contatto fisico:

-La collaborazione può essere limitata spazialmente: uomo entra nello spazio di lavoro ed il robot si ferma o temporalmente, uomo e robot si alternano nell'eseguire attività nello spazio condiviso portando ad un lavoro sequenziale;

- quando l'uomo è presente, il robot può essere fermato o rallentato, muoversi al di fuori dello spazio condiviso, può essere guidato a mano o eseguire attività simultanee con l'essere umano;

-sforzo congiunto se si lavora su un obiettivo o un oggetto condiviso utilizzando processi diversi o condivisi;

-il contatto fisico può essere escluso, consentito, possibile o richiesto.

2. STATO DELL'ARTE

Si entra nello specifico della robotica collaborativa con l'utilizzo dei Cobot. Il nostro lavoro è concentrato sullo studio e l'applicazione della pianificazione di traiettoria su un cobot a 6 gradi di libertà dell'Universal Robot, UR3 CB-series.

2.1 Robot Collaborativi

La robotica collaborativa prende forma tramite i Cobot, una tra le tecnologie più avanzate nel campo della robotica. I Cobot, o robot collaborativi, sono progettati per operare in sicurezza insieme agli esseri umani grazie a dispositivi avanzati, come sensori, che consentono loro di interrompere immediatamente il loro lavoro e attivare una modalità di sicurezza quando rilevano la presenza di persone nel loro spazio operativo. Al contrario, i robot tradizionali continuerebbero a operare senza considerare l'interazione umana, risultando una minaccia per l'incolumità dell'operatore [9]. I Cobot sono robot antropomorfi dotati di movimenti su sei assi, progettati per soddisfare criteri di sicurezza e compattezza, e per garantire una notevole flessibilità operativa. Grazie a queste caratteristiche, i Cobot possono adattarsi a diverse applicazioni industriali e produttive, migliorando l'efficienza e la sicurezza nei processi di lavoro [10].

La normativa ISO/TS 15066 [12] introduce le linee guida per l'identificazione generale dei pericoli e la valutazione del rischio relativi a sistemi di collaborazione uomo-robot e i requisiti per le applicazioni dei sistemi robotici collaborativi. Qui si distinguono sono riportate quattro differenti funzioni necessarie per garantire la collaborazione [12][13]:

-Stop monitorato di sicurezza: il movimento del robot viene interrotto se l'operatore entra nell'area di lavoro collaborativa;

-Guida manuale: l'operatore utilizza un dispositivo manuale, o direttamente muove il braccio del robot per completare l'operazione;

-Monitoraggio della distanza e della velocità: velocità di movimento del robot che varia dinamicamente a seconda della distanza tra operatore e robot; al di sotto della distanza minima di protezione il robot si ferma;

-Limitazione potenza e forza: funzioni di sicurezza arrestano il robot quando il livello massimo di forza o di potenza vengono raggiunte, in modo da ottenere uno spazio di lavoro collaborativo completamente condiviso e la possibilità di collisioni involontarie e imprevedibili con l'operatore senza danni per quest'ultimo.

I vantaggi dei Cobot rispetto ai robot tradizionali sono molteplici: dalla struttura alle performance fino al benessere dell'operatore. I robot tradizionali risultano essere performanti solo su grandi volumi produttivi, assicurando velocità e ripetibilità su grandi lotti dello stesso prodotto. Per quanto riguarda la sicurezza, i robot industriali vengono posizionati in zone di lavoro separate dall'operatore tramite l'ausilio di barriere di protezione

che circondano il robot a tutela dell'operatore. [10] I Cobot sono piccoli, leggeri e compatti tali da essere spostati agevolmente nel layout. Sono versatili e flessibili dal punto di vista dell'utilizzo, apprendono velocemente e sono facilmente programmabili [14], capaci di lavorare a molteplici operazioni per la produzione di prodotti eterogenei. [10] Sono coerenti e precisi, garantiscono ripetibilità nelle azioni, ottenendo la stessa qualità dei processi ed aumento della produzione. Sostituiscono gli operatori su azioni monotone e pericolose evitando infortuni dovuti ad azioni eseguite di frequente (RSI, Repetitive Strain Injury) garantendo qualità del lavoro e permettendo all'operatore di concentrarsi su compiti più creativi così contribuendo al suo sviluppo personale [9].

2.2 Pianificazione di traiettoria

La pianificazione della traiettoria è un problema fondamentale nella robotica industriale. Consiste nel determinare una sequenza di movimenti, traiettoria, che il robot deve seguire per svolgere un compito specifico evitando ostacoli e rispettando vari vincoli di tipo cinematici e dinamici, dovuti alla geometria del robot e all'ambiente circostante. La pianificazione della traiettoria può essere eseguita in spazio giunti o in spazio operativo:

- La pianificazione della traiettoria in spazio giunti determina l'evoluzione dei valori degli angoli per ciascun giunto nel tempo, rappresentato dalla funzione q(t). Questa pianificazione risulta essere più semplici ma porta a movimenti indefiniti dell'end effector nello spazio cartesiano;

- Con la pianificazione della traiettoria in spazio operativo si determina l'evoluzione del movimento dell'end-effector del robot nello spazio in modo che segua un determinato percorso. La traiettoria è rappresentata da una sequenza di posizioni ed orientazioni, posa, nello spazio cartesiano. Lo svantaggio della pianificazione in spazio operativo è la necessità della risoluzione di un problema di cinematica inversa per tradurre le posizioni cartesiane nelle configurazioni di giunti corrispondenti, ciò implica un costo computazionale maggiore e possibili problemi di singolarità [16].

La tradizionale pianificazione della traiettoria per robot ad elevato numero di gradi di libertà ha subito una rivoluzione significativa nella pianificazione del movimento con lo sviluppo di algoritmi basati sul campionamento [17]. Tra questi algoritmi si includono il Dynamic Motion Primitives, il Rapidly Exploring Random Trees (RRT), il Reinforcement learning:

- DMP (Dynamic Motion Primitives) è un approccio di apprendimento automatico, machine learning, in cui si applica il concetto di pianificazione del movimento

tramite apprendimento per dimostrazione “learning from demostration”. Si basa sull’idea che ogni movimento può essere scomposto in elementi costitutivi del movimento primitivo. Le simulazioni dimostrano che l’allenamento tramite questo metodo garantiscono una precisione superiore al 95% nel riprodurre i movimenti appresi per la pianificazione della traiettoria. Inoltre, sempre tramite simulazioni si dimostra che il pianificatore di movimento basato su DMP ha la versatilità necessaria per pianificare con successo i movimenti da diverse condizioni iniziali e diverse velocità rispetto ai dati del set di allenamento. Il DMP, tramite dati acquisiti durante il movimento dimostrativo, genera equazioni differenziali non lineari stabili, le cui risoluzioni rappresentano la traiettoria da eseguire. Tramite queste equazioni differenziali il DMP può generalizzare il movimento a condizioni iniziali e finali diverse. Per una comprensione migliore del metodo viene riportato l’esempio di applicazione del paper analizzato “A study case of Dynamic Motion Primitives as a motion planning methos to automate the work of forestry cranes” in cui si utilizza il DMP per l’automazione dei movimenti delle gru forestali. Nello studio si considera una gru di avanzamento dotata di sensori per la registrazione dei dati relativi a movimenti per portare i tronchi dalla zona di prelievo al camion e riporli in questo. Per l’acquisizione dei dati questi movimenti sono comandati manualmente da operatore tramite joystick. Con il DMP si otterrà l’esecuzione del movimento a velocità diverse, con punti iniziali e finali diversi, portando all’automatizzazione del compito [17];

- Il MoMP (Mixture of Motors Primitives) è il miglioramento dell’algoritmo DMP. Utilizza un nuovo modello primitivo di movimento per ottenere la stessa direzione della traiettoria appresa utilizzando meno dati per costruire la base informativa della primitiva di movimento originale. Sviluppa una strategia di ponderazione ottimale per apprendere nuove primitive di movimento e definire la traiettoria di movimento. Questo nuovo metodo garantisce un’elevata precisione di pianificazione della traiettoria con maggiore esattezza nel movimento pianificato. Sulla base del comportamento viene creata una libreria di informazioni sui movimenti primitivi. Vengono assegnati i pesi corrispondenti delle primitive di movimento in base alla somiglianza tra i segnali esterni e le informazioni presenti nella libreria. Nel modello migliorato la DMP è utilizzata per controllare le informazioni rilevanti sul segnale e generare traiettorie per guidare l’acquisizione delle traiettorie effettive. Viene aggiunta una funzione di forzatura legata al tempo, rendendo il DMP non lineare [18];

- Il Rapidly-exploring Random Tree (RRT) [19] è un algoritmo per la pianificazione del percorso con struttura ad albero basato su un campionamento casuale uniforme. L’algoritmo parte da un albero costituito da un solo nodo, corrispondente al punto di partenza, per poi passare alla fase di campionamento casuale con espansione verso l’esterno ad ogni iterazione tramite la generazione di un punto casuale all’interno

dello spazio di ricerca. Una volta generato il punto casuale, l'algoritmo cerca nell'albero il nodo esistente più vicino al punto generato casualmente. Ad ogni iterazione si ripetono i passaggi di campionamento, selezione del nodo più vicino e generazione del nuovo nodo. Il ciclo continua fino a quando la distanza tra il nuovo punto ed il punto obiettivo rientra in un intervallo di tolleranza. Se questo algoritmo viene applicato in un ambiente non strutturato l'efficienza del calcolo sarà ridotta. È poco efficace quando viene applicato in ambienti non strutturati, l'efficienza del calcolo può ridursi a causa di campionamenti casuali inefficaci con generazione di punti casuali con esplorazioni non produttive, può inoltre espandersi in direzioni non ottimali, aumentando il tempo di calcolo [19]. Si analizza il caso applicativo del paper "A Path Planning Strategy of Wearable Manipulators with Target Pointing End Effectors" [20] in cui l'algoritmo RRT viene utilizzato per la pianificazione del percorso dei manipolatori indossabili dotati di end effector di puntamento del bersaglio. Essendo un problema ad alta dimensione viene scomposto in più gruppi a bassa dimensione con uno o più segmenti del manipolatore, con la pianificazione che parte dal primo gruppo e procede a turno attraverso tutti i gruppi. L'RRT parte determinando le posizioni del punto iniziale e di destinazione, inserendo il punto iniziale come primo punto del set di punti del percorso. Dopodiché, l'algoritmo genererà un punto casuale nello spazio di configurazione del percorso individuando il punto già esistente nel percorso che è più vicino al nuovo punto casuale, ed in assenza di collisioni, si collegheranno i due punti rendendo il punto generato casualmente parte del percorso. Il processo di generazione e verifica continuerà a ripetersi fino a quando la distanza tra il nuovo punto generato ed il punto di destinazione non diventa sufficientemente piccola;

- RRT migliorato, consiste in un miglioramento del RRT tradizionale poiché prevede la costruzione di più alberi simultaneamente invece di uno che si espande dal punto di partenza. Gli alberi divergono in contemporanea verso l'esterno generando ad ogni iterazione un punto casuale per albero. L'espansione degli alberi continua in simultanea, mantenendo il ciclo di generazione di nuovi nodi e verificando le collisioni con ostacoli, fino a quando gli alberi si uniscono completando il percorso desiderato. Con questo algoritmo si ha un miglioramento dell'efficienza in ambienti non strutturati, riduzione del tempo di calcolo ed aumento della probabilità di trovare percorsi ottimali in ambienti complessi, evitando inefficienze del RRT tradizionale rappresentate da espansioni in direzioni non produttive con nodi radice ben scelti, inclusi punti intermedi [19];

- Reinforcement Learning (RL) è una tecnica di apprendimento automatico basato sull'approccio tentativi ed errori, e quindi sull'esperienza. In questo metodo un Agent, che rappresenta il dispositivo di interesse, impara a compiere azioni in un

ambiente con l'obiettivo di massimizzare una ricompensa cumulativa. Ogni volta che l'Agent prende una decisione e compie un'azione l'Environment in cui è presente cambia e l'Agent riceve feedback sottoforma di un valore numerico positivo o negativo indice della qualità della sua scelta, indicando se l'Agent si è avvicinato o meno al suo obiettivo. L'Agent ad ogni iterazione amplia la sua conoscenza dell'ambiente e delle possibili decisioni individuando quella migliore [21].

Altra soluzione è l'utilizzo degli algoritmi di calcolo evolutivi, Evolutionary Computation (EC), per risolvere compiti parziali in una determinata area. Sono ispirati a fenomeni naturali e di biologia e vengono utilizzati in vari settori della robotica industriale con focalizzazione sulla pianificazione del movimento [22]. Algoritmi evolutivi sono:

- Particle Swarm Optimization (PSO), metodo ispirato al comportamento di foraggiamento di uno stormo di uccelli o di un banco di pesci, progettato per aumentare l'efficienza energetica. L'algoritmo PSO funziona come una popolazione di particelle che lavorano insieme per trovare iterativamente la soluzione ottimale relativa alla configurazione o posizione nello spazio di ricerca che massimizza o minimizza la funzione obiettivo. Le particelle adattano le proprie posizioni in uno spazio di ricerca multidimensionale, basandosi sulla propria esperienza e sulla conoscenza collettiva dello sciame, con la generazione di traiettorie fluide ed efficienti. Ogni particella si muove in modo casuale portando con sé informazioni relative alle posizioni che ha sperimentato tra le quali individua la migliore trovata fino a quel momento comunicandola al resto dello sciame. Un aggiornamento della tecnica ha portato alla suddivisione in segmenti dell'intera traiettoria e l'ottimizzazione è stata eseguita in punti di connessione chiamati punti nodali che comprendono angoli, velocità e accelerazioni angolari. Tramite l'aggiunta di un fattore di apprendimento dinamico i parametri delle particelle vengono ottimizzati per ottenere la soluzione migliore per ciascuna particella [22]. Si analizza il caso applicativo del paper [23] "Reduction in Robotic Arm Energy Consumption by Particle Swarm Optimization" [23] in cui si applica l'algoritmo evolutivo PSO per l'ottimizzazione delle traiettorie di un braccio robotico UR3 sotto il punto di vista energetico. Mediante l'utilizzo della curva di Bezier si ricavano sette punti di controllo utilizzati per modellare la traiettoria che il braccio robotico deve seguire. Questi punti vengono poi ottimizzati con l'algoritmo PSO che li modifica iterativamente per ridurre il consumo energetico dell'UR3. Dei sette punti, due sono il punto inziale e finale che non saranno soggetti ad ottimizzazione; i due punti di controllo vicini all'inizio e alla fine della traiettoria sono ottimizzati su un intervallo più piccolo per evitare bruschi avvii e arresti; gli ultimi tre punti di controllo sono utilizzati per ottimizzare l'efficienza energetica della traiettoria;

- Genetic Algorithm (GA), algoritmo di ricerca basato sulla genetica ed il concetto di sopravvivenza del più adatto. Il processo inizia con una popolazione di soluzioni rappresentate da una stringa di dati, i cromosomi, che codificano le variabili del problema. Le soluzioni candidate vengono generate in modo casuale e gli viene assegnato un valore valutato dalla funzione fitness, che misura quanto bene una soluzione risolve il problema. Elementi chiave di GA includono la rappresentazione cromosomica, la selezione, il crossover, la mutazione ed il calcolo della funzione fitness. Il processo inizia con una popolazione di soluzioni rappresentate da una stringa di dati, i cromosomi, che codificano le variabili del problema. Le soluzioni candidate vengono generate in modo casuale e gli viene assegnato un valore valutato dalla funzione fitness, che misura quanto bene una soluzione risolve il problema. Le soluzioni con valore maggiore vengono selezionate per la riproduzione, in cui si combinano tra loro scambiandosi parti dei loro dati tramite un processo chiamato "crossover". Possono essere introdotte piccole modifiche casuali, dette "mutazioni", per aiutare a esplorare nuove soluzioni potenziali. Questi nuovi individui sostituiscono parte della popolazione esistente, creando una nuova generazione di soluzioni candidate, con ciclo di selezione, crossover e mutazione che si ripete per molte generazioni con l'obiettivo di migliorare continuamente le soluzioni nel tempo. L'algoritmo cerca di convergere verso una soluzione ottimale per il caso studio tramite l'esplorazione e lo sfruttamento di informazioni ottenute dalle generazioni precedenti. La pianificazione della traiettoria nello spazio articolare utilizza i GA per ridurre al minimo le vibrazioni e/o il tempo di esecuzione per il movimento [22];

- Ant Colony Optimization (ACO), algoritmo di ottimizzazione ispirato alla natura comunemente utilizzato per trovare soluzioni approssimate a problemi di ottimizzazione combinatoria. Modella il comportamento di foraggiamento delle formiche per scoprire percorsi o soluzioni ottimali. Le formiche artificiali costruiscono soluzioni in modo incrementale, lasciando tracce di feromoni per comunicare le scoperte. I feromoni hanno un certo livello, indicatore della qualità del percorso, aggiornato ogni volta che tutte le formiche completano l'esplorazione. Nel corso del tempo queste tracce guidano altre formiche verso soluzioni migliori. Alla base del comportamento delle formiche si ha una regola decisionale che influenzerà la loro scelta del percorso da esplorare [22]. Sul paper "Ant Colony Robot Motion Planning "[24] viene riportato il caso applicativo per la pianificazione della traiettoria del robot con l'utilizzo dell'algoritmo ACO. Le formiche sono distribuite in due gruppi, quelle che partono dalla configurazione iniziale del robot, e quelle che partono dalla configurazione obiettivo del robot. Ognuna di queste formiche cercherà di raggiungere l'altra configurazione rilasciando lungo il percorso tracce, considerate come informazioni condivise che influenzeranno le scelte delle formiche successive. Le formiche decidono il percorso da seguire basandosi sulla distanza fisica e sulle tracce lasciate dalle formiche precedenti, preferendo percorsi

con tracce più intense e percorsi brevi. Tramite la memoria collettiva che guida le scelte delle formiche si facilita la scoperta e l'uso di percorsi più efficienti nel problema multi-obiettivo della pianificazione di percorsi robotici

- Differential Evolution (DE), evoluzione differenziale, si basa sull'imitazione del processo di selezione naturale per migliorare gradualmente un gruppo di potenziali soluzioni. In ogni iterazione, DE genera nuove soluzioni candidate combinando le informazioni di tre individui scelti casualmente dalla popolazione attuale. Queste nuove soluzioni sono in seguito confrontate con quelle attuali e, se si dimostrano migliori sulla base di una specifica misura di fitness, sostituiscono le vecchie soluzioni. È una strategia versatile poiché impiega operazioni di mutazione (variante differenziale) e crossover (variante esponenziale) per esplorare efficacemente lo spazio delle soluzioni [22]. Per una migliore comprensione dell'algoritmo viene analizzato il caso applicativo del paper "optimum robot manipulator path generation using Differential Evolution" [25]. È applicato l'algoritmo per ottimizzare il percorso di un manipolatore robotico PUMA a sei gradi di libertà. L'ottimizzazione è definita in termini di errori minimi di posizione ed orientamento dell'End effector del robot che saranno i parametri su cui si baserà la funzione fitness. L'algoritmo inizia con una configurazione iniziale approssimata dei giunti, poi ottimizzata attraverso diverse generazioni di membri della popolazione iniziale, rappresentante una possibile configurazione dei giunti del robot, distribuita attorno al percorso appreso. Selezionati un sottoinsieme di membri della popolazione vengono combinati tramite crossover per creare una nuova generazione di individui. Questi passaggi vengono ripetuti più volte, ad ogni ciclo l'algoritmo cerca di migliorare la soluzione avvicinandosi sempre più al percorso ottimale;

- Artificial Bee Colony (ABC), algoritmo di ottimizzazione ispirato al comportamento di foraggiamento delle api mellifere basato sulla popolazione. L'algoritmo inizializza la popolazione ed utilizza tre tipologie di api (impiegate, curiose, scout) per la ricerca delle migliori fonti di cibo, cioè dei percorsi ottimali. Le api scout sono responsabili della scoperta di nuove fonti di cibo che verranno esplorate dalle api impiegate su cui eseguono ricerche locali condividendo le informazioni con le api curiose che sceglieranno le fonti di cibo da sfruttare basandosi sui dati ricevuti. Se una fonte di cibo non è stata migliorata per un certo numero di iterazioni, viene abbandonata e sostituita con una soluzione generata casualmente. La migliore soluzione trovata dalle api impiegate e curiose viene aggiornata se è scoperta una soluzione migliore. L'algoritmo ripete le fasi precedenti per un numero specificato di iterazioni o finché non soddisfa una condizione di terminazione. ABC è un algoritmo progettato per la risoluzione dei problemi di evitamento degli ostacoli, con caratteristica importante della capacità di regolare il coefficiente di evitamento degli ostacoli, consentendo l'ottimizzazione della traiettoria e una maggiore adattabilità dei robot in varie

situazioni. Questo algoritmo offre anche l'esecuzione parallela dimostrando una forte robustezza nella risoluzione di questi problemi [22]. Viene di seguito illustrato l'applicazione dell'algoritmo ABC "Motion Planning of Dual-Chain Manipulator Based on Artificial Bee Colony Algorithm" [26]. con l'obiettivo di ottimizzazione della configurazione iniziale del manipolatore a doppia catena. Viene inizializzata la popolazione con la generazione casuale di diversi insiemi di angoli iniziali dei giunti e viene calcolata la funzione fitness che tiene conto della differenza tra la posa target dell'End effector della doppia catena e la posa effettiva. Tramite ABC vengono impiegate api lavoratrici che selezionano ed ottimizzano i set di angoli giunto iniziali, le api osservatrici valutano le soluzioni ottimizzate e selezionano le configurazioni vicine alla posa target. Dopo varie iterazioni si ottengono diversi set di configurazioni iniziali che soddisfano i vincoli e tra le quali viene selezionata quella ottimale.

2.3 Cobot UR3

Per il nostro studio si è utilizzato un robot collaborativo della famiglia dell'Universal Robot, l'UR3 CB-series (Figura 1). Gli UR3 sono robot leggeri, piccoli e compatti, per un peso complessivo di 11 kg, carico garantito di 3 kg e portata di 0.5 metri, perfetto per le attività da banco. Essendo l'ingombro minimo può essere installato facilmente nei pressi dei macchinari o in spazi di lavoro ristretti [27].

Figura 1: UR3

È un cobot costituito da 6 articolazioni rotanti, in grado di effettuare rotazioni a + o – 360 gradi, sesto asse, invece, presenta rotazione infinita.

I cobot UR3 sono progettati per lavorare a stretto contatto con l'operatore grazie alle 17 funzioni di sicurezza (safety native) e le funzioni di tempo e distanza d'arresto personalizzabili. Inoltre, i robot possono essere equipaggiati con sensori che accrescono la sicurezza per la registrazione dell'operatore, controllando movimenti e tempi di reazione del cobot [10]. Sono riportate le specifiche tecniche del cobot all'interno della Tabella 1.

SPECIFICHE TECNICHE DEL COBOT UR3	
Peso	11 kg
Carico utile	3 kg
Portata	500 mm
Rotation	+/- 360 °/s
	Rotazione infinita articolazione terminale
Velocità	Articolazioni Polso 360 °/s
	Altre Articolazioni 180°/s
	End Effector max = 5000 mm/s
Ripetibilità	+/- 0.1 mm
Gradi di Libertà	6 articolazioni rotanti
Consumo corrente	Tipicamente 100 watt
Materiale	Alluminio, plastica PP
Temperatura	Intervallo di 0-50 °C
Alimentazione corrente	100-240 VAC, 50-60 Hz

Tabella 1: Scheda tecnica cobot UR3 [28]

La programmazione di questi cobot è molto semplice e può avvenire tramite Teach Pendant o Free Drive. Il Teach Pendant è il tablet di programmazione con funzionalità touch screen, contiene graficamente il template che consente di impostare rapidamente i programmi [10].

La modalità "Free Drive" permette la programmazione del cobot tramite movimento manuale del braccio nello spazio registrando dei waypoints specifici che serviranno a ripetere il movimento programmato [10].

3. COLLEGAMENTO MATLAB-ROS-URSIM/UR3

Per la realizzazione del lavoro è stato creato per prima cosa un canale di comunicazione tra computer e robot che permettesse di avere un'interfaccia efficace, in modo da inviare bi-direzionalmente le informazioni tra i due dispositivi. Lo scopo del collegamento è quello di manovrare agevolmente il robot sfruttando l'ambiente Matlab, il quale con le sue funzioni dedicate consente una facile programmazione e validazione delle traiettorie. Inoltre, la possibilità di simulare il robot permette di verificare la traiettoria di inviare l'effettivo comando al robot.

Lo studio di pianificazione di traiettoria è stato eseguito con l'utilizzo di software appositi, quali Matlab e Simulink, che hanno permesso di ottenere studi approfonditi e precisi delle casistiche con l'utilizzo opportuno delle funzioni delle librerie fornite da Mathworks, i Toolbox.

Tramite la Robotics System toolbox si è reso possibile il collegamento di Matlab e Simulink con il software per la robotica Robot Operating System (ROS), che a sua volta ha permesso di creare il canale diretto con l'UR3. Oltre alla possibilità di utilizzare il robot reale è possibile simulare l'interfacciamento con esso tramite l'apposito software (URSim) fornito da Universal Robot in modo da verificare le traiettorie prima dell'esecuzione reale.

Una volta creato il collegamento è stato effettuato uno studio di pianificazione di traiettoria in spazio giunti e spazio cartesiano di movimenti semplici, con lo scopo di verificare la correttezza del collegamento avvenuto. Si è utilizzato un pacchetto di funzioni specifico, fornito da Universal Robot, e necessario per la comunicazione e manipolazione dei dispositivi UR.

Durante le prove di movimento sono stati raccolti i dati reali del robot e utilizzati in post process per un confronto con i dati ottenuti dai nostri studi. Questi confronti sono stati eseguiti per comprendere al meglio le capacità del collegamento e apprendere i possibili errori.

3.1 ROS

ROS, Robot Operating System, è un software open source che rende possibile la comunicazione ed il controllo dei robot e facilita lo sviluppo e la condivisione. Supporta diversi linguaggi di programmazione, tra cui Python e C++, ma può essere implementato con qualsiasi linguaggio moderno. ROS ricopre un ruolo fondamentale nel contesto dell'Industria 4.0 permettendo lo sviluppo di sistemi robotici avanzati (Figura 2) che siano in grado di percepire e riconoscere l'ambiente circostante, muoversi autonomamente e

collaborare sia con altri robot che con gli operatori, contribuendo a creare un ambiente di produzione più intelligente, efficiente e flessibile [31].

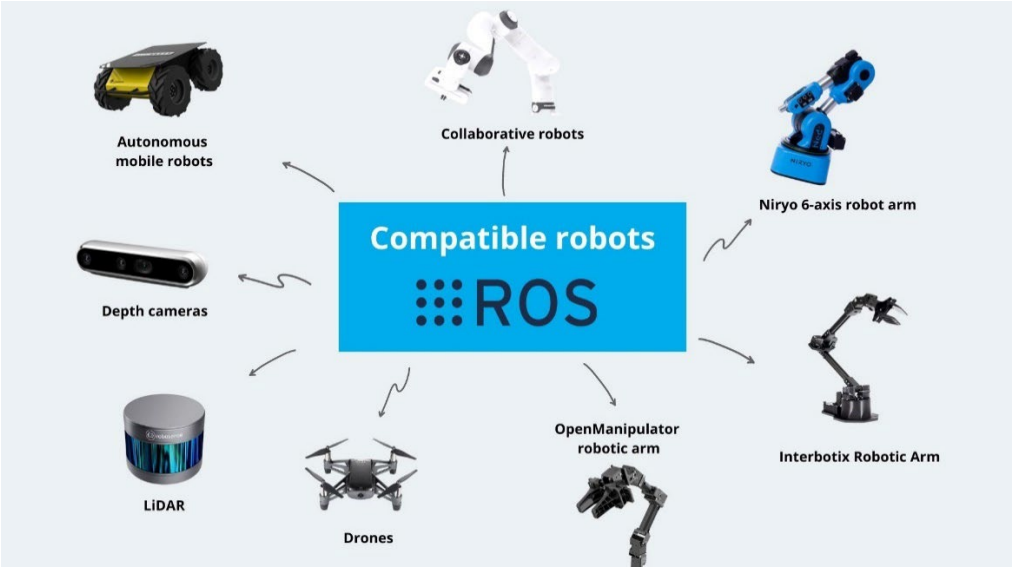

Figura 2: ROS [36]

Utilizza una filosofia peer-to-peer composta da numerose semplici applicazioni, chiamate nodi, che interagiscono tra loro scambiandosi messaggi senza l'uso di servizi di routing centrali. Ogni nodo rappresenta un programma che può gestire specifiche funzioni, come il controllo di dispositivi esterni, sensori o attuatori, e comunica tramite un programma centrale chiamato ROS Master coordinando le comunicazioni e permettendo ai nodi di scambiare tra loro informazioni. I nodi che inviano dati sono chiamati Publisher Nodes, mentre quelli che ricevono dati sono i Subscriber Nodes, e le informazioni vengono trasmesse sotto forma di messaggi ROS attraverso percorsi chiamati ROS Topics [29] (Figura 3).

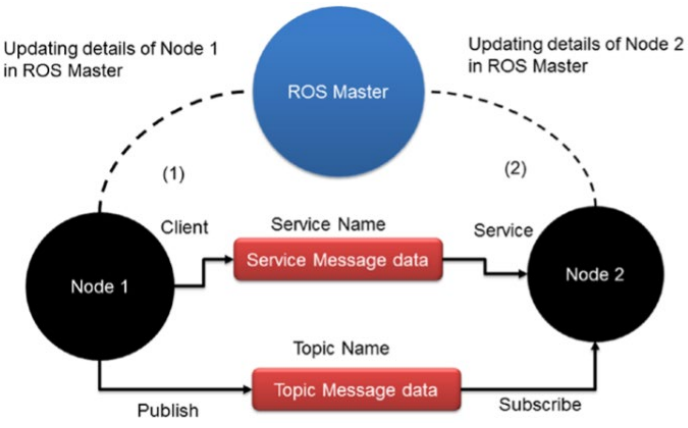

Figura 3: Struttura ROS con nodi e target [29]

ROS è costituito da circa 2000 pacchetti, ognuno dei quali offre funzionalità specifiche ed include strumenti per l'astrazione dell'hardware, i driver dei dispositivi, la comunicazione tra processi su più macchine, strumenti per test e visualizzazione che consentono agli

utenti di partizionare i codici in pacchetti modulari e riutilizzabili, facilitando la condivisione e la collaborazione per il loro miglioramento [30].

Il sistema operativo per consentire ai programmi di comunicare tra loro utilizza:

- **Azioni**: consentono agli utenti di dare indicazioni non sensibili al tempo per monitorare lo stato del server;

- **Topic**: canale di comunicazione utilizzato per trasmettere informazioni tra i nodi;

- **Servizi**: creano un canale sincrono tra i nodi per richiedere azioni specifiche ai robot e regolarne le impostazioni [30].

ROS è strutturato su tre livelli principali:

- **Filesystem**: gestisce le risorse incontrollabili del sistema operativo su disco e facilita l'organizzazione e la gestione delle risorse software, rendendo più semplice il riutilizzo e la condivisione del codice tra diversi progetti;

- **Computation Graph**: gestisce l'insieme di processi di calcolo distribuiti (nodi) che eseguono il codice e le sue interconnessioni. Questo livello gestisce come i nodi comunicano tra loro consentendo un'elevata modularità e scalabilità;

- **Community**: supporta la condivisione e la collaborazione all'interno della comunità ROS, consentendo lo sviluppo indipendente e la distribuzione di pacchetti. È supportata da una vasta gamma di risorse online, tra cui wiki, forum, mailing list, repository di codice e strumenti di gestione delle versioni. Questo ecosistema di supporto e collaborazione permette agli utenti di ROS di beneficiare delle conoscenze collettive, accelerando lo sviluppo e l'innovazione [32].

3.2 Matlab e Toolbox

Matlab è un ambiente di calcolo tecnico che mette a disposizione funzioni per la risoluzione di una vasta gamma di tipologie di problemi. Tutte le funzioni sono dotate di una documentazione esplicativa con esempi e descrizioni di input, output e sintassi [33].

Tramite Matlab è possibile scaricare i Toolbox, pacchetti software, utili per risolvere problemi specifici in moltissimi campi di studio ingegneristico, matematico, fisico, economico. Per la realizzazione del nostro collegamento si sono utilizzati due tipi di toolbox:

- Robotics System Toolbox;
- ROS Toolbox.

Robotics System Toolbox include algoritmi e strumenti per progettare, simulare, testare e distribuire applicazioni di manipolatori e robot mobili. Mette a disposizione la visualizzazione e la simulazione del movimento del robot tramite libreria che contiene strutture di robot comunemente impiegati nell'ambito industriale, oppure, dà la possibilità di importare file URDF o modelli Simscape Multibody per creare modelli di robot personalizzati. Fornisce comandi per lo studio della cinematica diretta, inversa e della dinamica del robot, con comandi relativi all'analisi delle auto-collisioni e delle collisioni con corpi esterni. Tramite il Robotics System Toolbox è possibile studiare la pianificazione del percorso tramite pianificatori personalizzabili e generare traiettorie per un movimento fluido evitando ostacoli (Figura 4) [34].

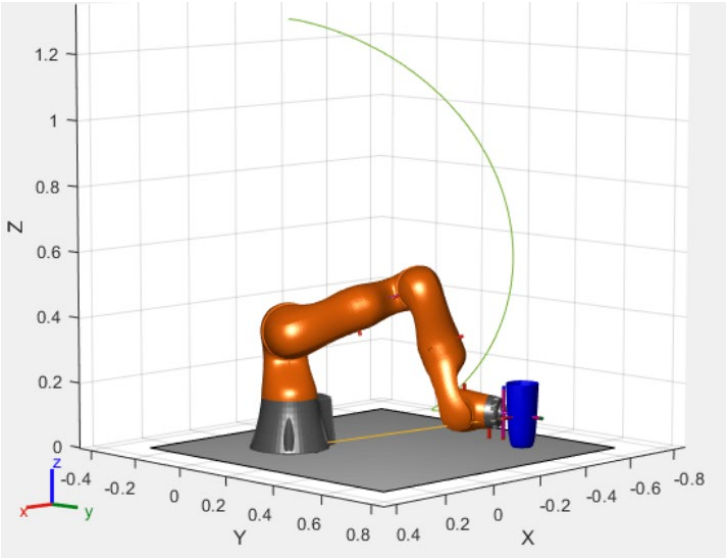

Figura 4: Modello di robot caricato e comandato tramite comandi del Robotics System Toolbox [35]

ROS Toolbox mette a disposizione un'interfaccia che collega Matlab e Simulink con il sistema operativo per la robotica ROS e ROS 2. Permette di progettare una rete di nodi ROS e combinare nodi ROS generati con Matlab con la propria rete ROS esistente. Include funzioni per la visualizzazione e analisi dati ROS tramite registrazione, importazione e riproduzione di file rosbag. Il toolbox consente di verificare i nodi ROS tramite simulazioni desktop e collegandosi a simulatori desktop collegandosi a simulatori robotici esterni [36].

3.3 Simulink

Simulink è un ambiente grafico di modellazione e di simulazione di sistemi dinamici. In questo ambiente i diagrammi vengono costruiti utilizzando blocchi, che possono rappresentare un componente fisico, un sistema o una funzioni specifiche. Un blocco risulta essere completo solo dopo aver definito i suoi input e output poiché tramite questi si definisce l'obiettivo del modello. È possibile il collegamento tra più blocchi per costruire sistemi e rappresentare funzionalità più complesse [37].

La funzione principale di Simulink è quella di simulare il comportamento dei componenti del sistema nel tempo, determinando l'ordine in cui i blocchi devono essere simulati e propagazione gli output calcolati nel diagramma a blocchi al blocco successivo.

Simulink gestisce i dati in tre categorie:

- Segnali: possono essere di input o di output del blocco, calcolati durante la simulazione;

- Stati: sono valori interni che rappresentano la dinamica del blocco e che vengono calcolati durante la simulazione;

- Parametri: sono valori che influenzano il comportamento di un blocco e vengono controllati dall'utente [38].

Simulink calcola nuovi valori per i segnali e gli stati ad ogni passo temporale, mentre i parametri sono specificati quando si costruisce il modello ed alcune volte sono modificabili durante l'esecuzione della simulazione.

3.4 URSim

Fornito da Universal Robot, URSim è un software di simulazione (Figura 5), realizzato per il sistema operativo Linux, utilizzato per la programmazione offline e la simulazione di programmi robot (Figura 6).

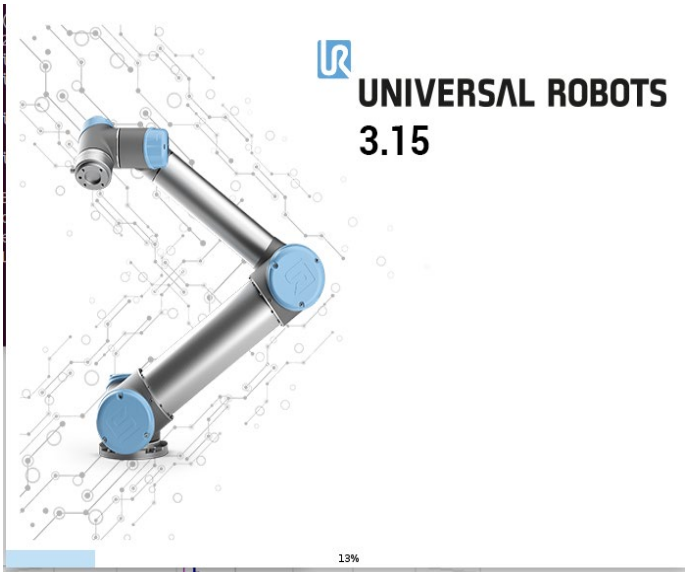

Figura 5: Schermata URSim versione 3.15

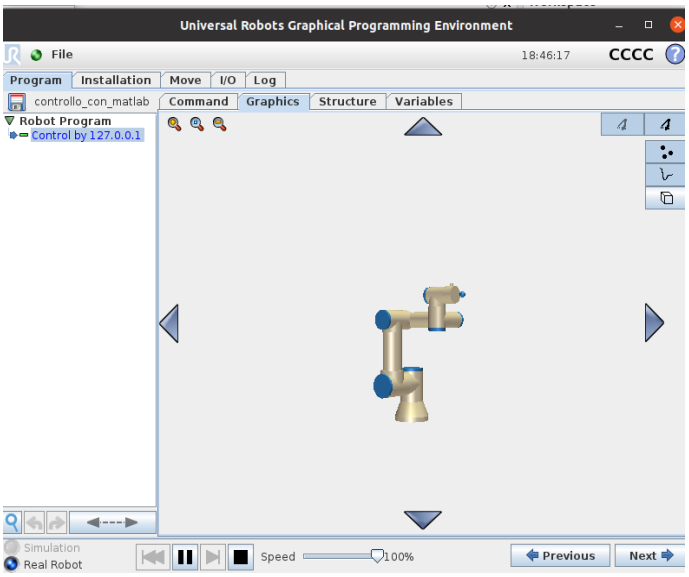

Figura 6: UR3 simulato su URSim

Per eseguire il simulatore in un altro sistema operativo è necessaria una macchina virtuale. Una macchina virtuale è un programma in cui possono essere installati più sistemi operativi, e quelli indicati dal sito dell'UR sono due:

- VMWare Player
- VirtualBox

Tramite questo software è possibile verificare e programmare il movimento del manipolatore anche quando il robot fisico non è presente. URSim mette a disposizione i comandi nominali che servono per muovere il robot manualmente o tramite display (Figura 7).[39]

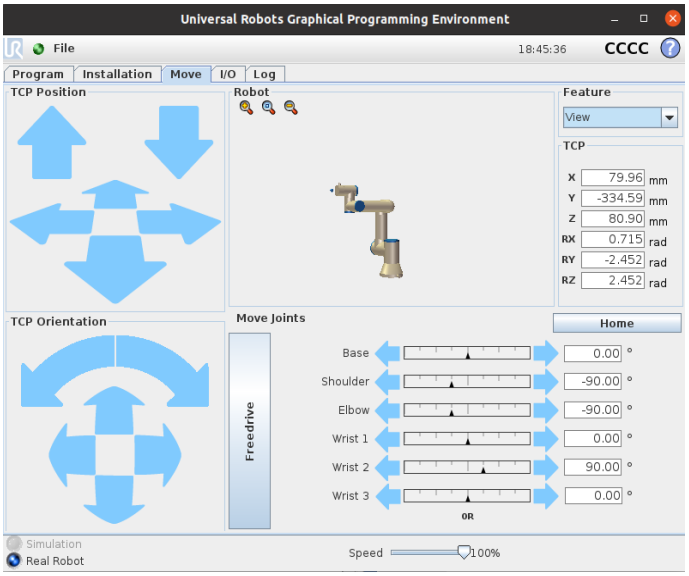

Figura 7: Opzioni di movimento tramite URSim

3.5 Collegamento software

Per la realizzazione del collegamento Matlab-ROS-URSim è stata seguita la guida fornita da Mathworks [40]. Come si era indicato in precedenza il software ROS necessita di essere installato sul sistema operativo Linux. L'approccio permette di utilizzare Matlab su Windows che comunicherà con la macchina virtuale e di conseguenza con ROS e con URSim senza la necessità di apportare partizioni di memoria del dispositivo ed evitando di ridurne le prestazioni.

Si è deciso di passare all'utilizzo di un dispositivo con sistema operativo principale Ubuntu Linux su cui avere tutti i software di studio: Matlab-ROS-URSim.

I software installati ed utilizzati sono Matlab versione R2024a, ROS Noetic, e URSim versione 3.15.8 Offline Linux, fornito da Universal Robots. Da guida Mathworks si è eseguita la configurazione dell'URSim per comunicare con i driver ROS. Per il controllo esterno dei cobot serie UR deve essere scaricato il pacchetto "matlab_externalcontrol-1.0.0.urcap" fornito da Mathworks. Dopo il download si è inserito il pacchetto all'interno della cartella del software URSim, in particolare nella cartella "Programs.UR3". È stato eseguito URSim tramite terminale, della directory di installazione dell'URSim inserendo il comando "start-ursim.sh" che di default genera un cobot UR5, invece per visualizzare un cobot UR3 bisogna specificare il modello "start-ursim.sh UR3" [41]. Si eseguirà l'installazione del Matlab

URCap per il controllo esterno utilizzando il menu delle impostazioni per poi configurarlo e connettersi al master ROS.

Host IP indica l'IP del computer con cui comunicherà il cobot.

Robot IP l'indirizzo identificativo del robot.

Sarà necessario cambiare il valore dell'Host IP con l'IP del device utilizzato per la connessione, poiché sia i driver ROS che URSim sono sullo stesso dispositivo per comunicare viene utilizzato l'indirizzo IP dell'host locale 127.0.0.1 (Figura 8)

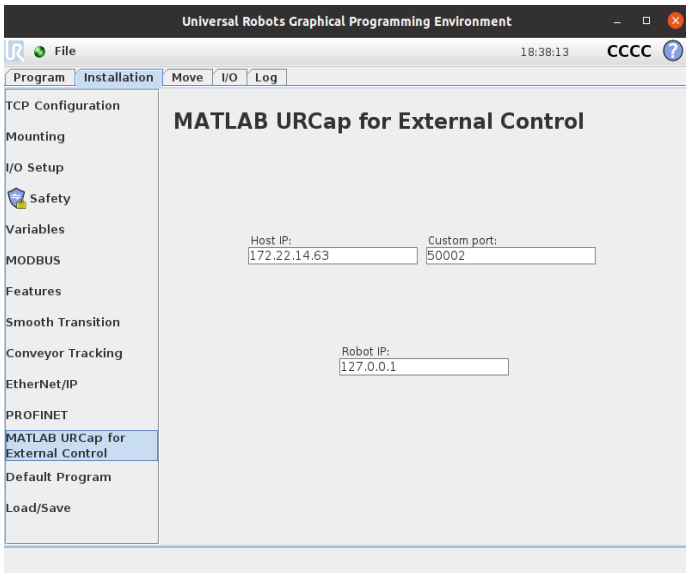

Figura 8: Host IP e Robot IP

In seguito alla configurazione, all'avvio di URSim apparirà il messaggio di errore "No Controller", si dovrà aprire il terminale nella directory di installazione di URSim e avviare "sudo starturcontrol.sh".

Avvenuta la configurazione di URSim si passa alla creazione del collegamento con Matlab. Questo viene realizzato seguendo l'esempio di Mathworks per il controllo di un cobot UR5[40].

Per la configurazione di Matlab è richiesta l'installazione del Robotics System Toolbox, del pacchetto relativo ai cobot serie UR, ed il ROS Toolbox. Come primo passo viene caricato il modello RigidBodyTree del manipolatore UR3 tramite comando "ur3=loadrobot ('universalUR3')'. Bisognerà indicare l'interfaccia che si sta utilizzando, "hardware" si riferisce al robot fisico e URSim invece al simulatore, il ROSDeviceAddress, l'indirizzo IP del PC in cui è installato ROS e i driver richiesti ed il robotAddress indirizzo IP del robot fisico o simulato.

Il comando "rqt_graph" di ROS è uno strumento utilizzato per la visualizzazione del grafo computazionale che fornisce un'interfaccia grafica per osservare i collegamenti relativi ai

Nodi di natura Publisher, Subscriber ed i Topic. È stato segmentato il grafo in Sezione 1 (Figura 9) in cui si analizza la comunicazione tra UR hardware e Matlab e Sezione 2 (Figura 10) in cui si osserva la struttura interna del passaggio di informazioni tra interfaccia e componenti dell'UR hardware

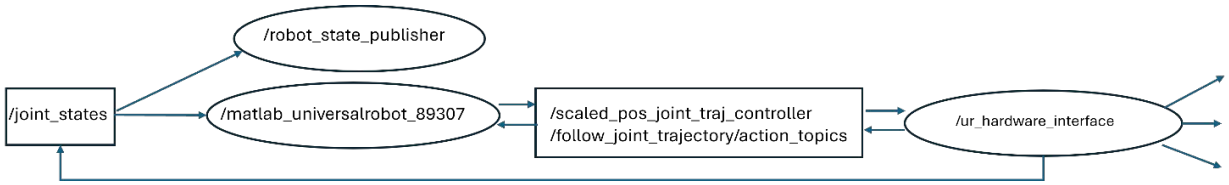

Figura 9:Sezione 1 grafo, in cui si osservano i nodi ed i topic del collegamento Matlab-URSim

Nella prima sezione del grafo è rappresentato il collegamento tra Matlab e URSim grazie all'utilizzo del sistema operativo ROS. Si osserva che, nel caso della programmazione su matlab per fornire comandi all'UR, il nodo matlab risulta essere di natura Publisher, pubblicando informazioni ad un gruppo di Topics, relativi ai campi che si vogliono modificare del robot di natura dinamica. Da questo gruppo di Topics si passerà al nodo UR, nodo Subscriber, che recepisce le informazioni programmate tramite matlab.
Nella condizione in cui matlab richiede i dati dal robot il nodo UR si comporterà da Publisher e pubblicherà i dati nella direzione opposta al caso precedente, passando per il gruppo di Topics arrivando al nodo di matlab che si comporterà da Subscriber.
Dalla "**Errore. L'origine riferimento non è stata trovata.**" si osserva come il nodo UR ricopra un ruolo di nodo Publisher anche per un ulteriore Topic, "joint_states". Ciò avviene dopo aver ricevuto informazioni da matlab ed aver eseguito le azioni indicate andando a ricoprire una nuova configurazione statica in spazio giunti. I dati vengono pubblicati sul topic e arriveranno al nodo matlab e ad un ulteriore nodo Subscriber "robot_state_publisher".

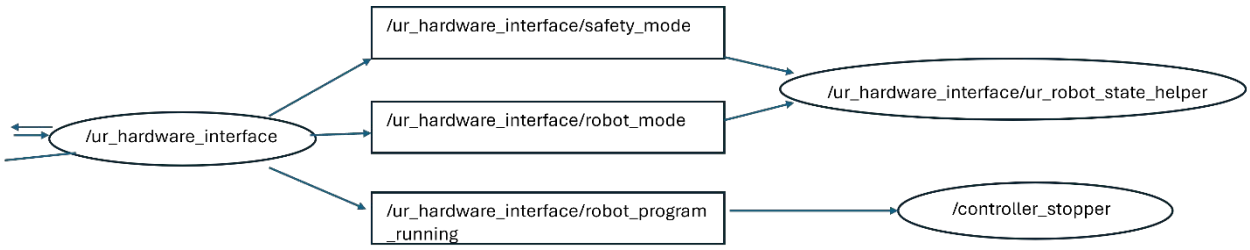

Figura 10: Sezione 2 rappresentante lo scambio di informazioni interne dell'UR

Nella sezione 2 (Figura 10) del grafo di ROS si osserva la presenza di altri collegamenti rappresentanti il passaggio d'informazioni tra interfaccia e componenti dell' UR. Si evince che il nodo UR ricopre ancora il ruolo di nodo Publisher per la pubblicazione di informazioni e dati per Topic "safety_mode", "robot_mode", collegati al nodo Subscriber "ur_robot_state_helper, e al Topic "robot_program_running", collegato al nodo Subscriber "controller_stopper", relativi alla gestione delle impostazioni di sicurezza del robot, delle

sue modalità di protezione e per il controllo dell'esecuzione del programma che invia messaggi di avvertimento fino allo stop del controller per questioni di sicurezza.

3.6 Verifica del Collegamento software

Creato il collegamento Matlab-ROS-URSim/UR3, si esegue una verifica tramite studio della pianificazione di traiettorie di casi semplici. Nel particolare, si sono applicati comandi in spazio giunti e in spazio cartesiano, per poi studiare la pianificazione di traiettoria in modo che il robot esegua movimenti semplici, quali l'esecuzione di un percorso rettilineo, triangolare e poi di un percorso circolare per l'end effector. La verifica si è basata sull'effettivo movimento del robot che ci ha fornito i feedback in tempo reale di parametri significativi per l'esecuzione della traiettoria, poi confrontati con i dati da noi studiati ed inviati al robot per l'esecuzione della traiettoria. Questo confronto ci permette di comprendere di quanto si discosta il movimento effettivo del robot rispetto al movimento studiato.

Nella "Figura 11" sono rappresentati i due sistemi di riferimento a cui si farà riferimento per la pianificazione delle traiettorie dei casi applicativi. Sono stati considerati il sistema di riferimento della base "base" ed il sistema di riferimento dell'End Effector "ee_link":

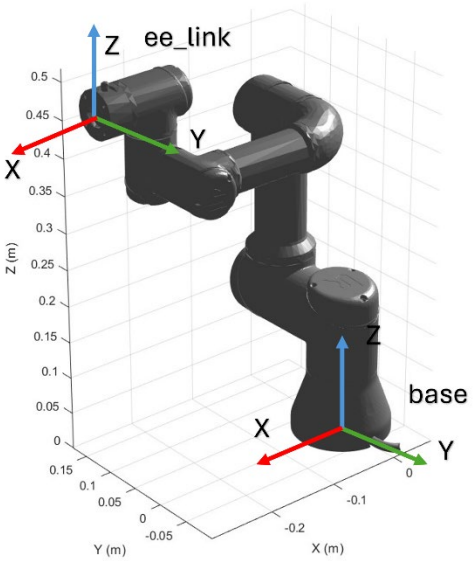

Figura 11: Sistemi di riferimento utilizzati per la pianificazione della traiettoria di questo capitolo

3.6.1 Funzioni Universal Robot

La realizzazione dei codici, per l'analisi del percorso e per il comando del robot, è stata possibile grazie all'utilizzo del Robotics System Toolbox, con funzioni relative alla pianificazione e studio di robot generali e funzioni specifiche per il movimento dei manipolatori serie UR dell'Universal Robot. Di seguito vengono riportate le funzioni da noi utilizzate con relative istruzioni per l'uso ed il relativo scopo di utilizzo.

Funzioni per manipolatori serie UR [42]:

- **getJointConfiguration(ur,timeout)** comando utilizzato per ottenere l'attuale configurazione in spazio giunti del cobot, restituirà un vettore di angoli 1x6 in radianti.

  $$q = [q1, q2, q3, q4, q5, q6]$$

  In input richiede il fattore "ur" e "timeout". "ur" indica la connessione al cobot fisico o simulato di Universal Robots specificato tramite urROSNodeObject. "timeout", indica il valore limite in secondi per ottenere informazioni sulla configurazione, altrimenti si visualizzerà un errore, e se non indicato corrisponderà al valore di default di 5 secondi;

- **getEndEffectorVelocity(ur,timeout)** comando per ottenere le velocità attuali dell'end effector del robot, restituite come vettore 1x6 costituito dai tre valori di velocità di rotazione delle tre velocità lineari, rispettivamente in rad/s e m/s

  [thetaz(dot), thetay(dot), thetax(dot), Vx, Vy, Vz]

  In input richiede l'oggetto "ur" e "timeout";

- **getJointVelocity(ur,timeout)** comando per ottenere le velocità attuali dei giunti del robot, restituisce come vettore 1x6 della velocità angolare dei singoli giunti in rad/s

  [qd1, qd2, qd3, qd4, qd5, qd6]

- **getCartesianPose(ur,timeout)** comando utilizzato per acquisire informazioni relative alla posa corrente dell'End Effector. Restituirà un vettore 1x6 costituito dai tre valori dell'orientazione e tre valori di posizione, rispettivamente in radianti e metri

  [theta(z), theta(y), theta(x), x, y, z]

  "thetaz" "thetay" "thetax" sono angoli di Eulero, rappresentano le rotazioni attorno agli assi principali z, y e x del sistema di riferimento "ee_link" rispetto al sistema di

riferimento "base". Sono utilizzati per descrivere l'orientamento del cobot nello spazio tridimensionale.

- **[result, state] = getMotionStatus(ur)** comando per ottenere lo stato attuale del movimento del robot tramite parametro logico "result", e lo stato dell'obiettivo finale del cobot "state". Nel caso di avvenuto movimento in modo corretto la variabile "state" sarà pari alla voce succeded ed il parametro "result" darà il valore logico true;

- **followTrajectory(ur, q, qd, qdd, trajtimes)** questa funzione permette di specificare una traiettoria tramite i valori di posizione, velocità e accelerazione angolare dei giunti, da fornire come vettori 6xN. Inoltre, per completare l'informazione di traiettoria devono essere forniti gli istanti di tempo a cui corrispondono i valori cinematici sopra indicati, questi vanno forniti con vettore 1xN;

- **followWaypoint (ur, taskwaypoints, waypointtimes)** Permette di specificare dei punti in spazio cartesiano che il robot dovrà seguire e la relativa orientazione. In input il comando richiede "ur", "taskwaypoints" e i "waypointtimes". I "taskwaypoints" sono i punti nello spazio cartesiano da seguire nella traiettoria e vengono forniti sottoforma di matrice di dimensione Nx6 con unità di misura rispettivamente radianti e secondi

  [thetaz, thetay, thetax, x, y, z]

  "thetaz" "thetay" "thetax" sono angoli di Eulero, rappresentano le rotazioni attorno agli assi principali z, y e x del sistema di riferimento "ee_link" rispetto al sistema di riferimento "base". Sono utilizzati per descrivere l'orientamento del cobot nello spazio tridimensionale.
  I "waypointtimes", rappresentano il tempo assoluto da ciascun segmento del "waypoint" per completare il movimento, rappresentato come vettore numerico 1xN, specificato in secondi. Se una traiettoria dovesse contenere 5 "waypoints" e passare per un punto ogni secondo il vettore fornito dovrebbe essere [0, 1, 2, 3, 4], il primo "waypoint" rappresenterebbe la posa iniziale per non avere errori da parte del robot.
  Fra gli input possono essere aggiunte altre voci. "InterpolationMethod", utilizzato per indicare il metodo di interpolazione da considerare per generare i comandi di movimento utilizzando le indicazioni sulle pose specificate, in questo caso solo il tempo è considerato il fattore di controllo per la forma della traiettoria desiderata. I metodi d'interpolazione corrispondono agli stessi comandi indicati tra le funzioni del Robotics System Toolbox (cubicpolytraj, bsplineploytraj, quinticpolytraj, trapveltraj). Altra voce aggiungibile tra gli input corrisponde ai NumberOfSamples, numero di campioni utilizzati come punti dati per interpolare la traiettoria.

Aumentando il numero di questo parametro si garantisce un tracciamento regolare nello spazio cartesiano ma con più tempo di calcolo per il robot per il calcolo della traiettoria;

- **sendJointConfiguration(ur,jointconfig)** funzione utilizzata per comandare al robot di spostarsi nella configurazione giunto desiderata. Richiede in input un vettore 1x6 con gli angoli in radianti dei giunti che descrivono la configurazione che il cobot dovrà avere al termine del movimento

  [q1, q2, q3, q4, q5, q6]

  Tra gli input può essere aggiunto il fattore "endtime" in cui indicare un valore numerico che indicherà entro quanto dovrà essere eseguita l'azione prima di ricevere un segnale di errore se non eseguita in tempo;

- **recordRobotState** comando utilizzato per la registrazione dei parametri chiave dello stato del robot durante il movimento di quest'ultimo;

Funzioni principali del "Robotics System Toolbox" [43]:

- **loadrobot (robotname)** carica un modello di robot identificato tramite "robotname" dalla libreria Mathworks contenente diversi impianti robotici come "rigidBodyTree". Il "**RigidBodyTree"** è una rappresentazione strutturale della connettività del robot, in cui le informazioni sono disposte a struttura ad albero, con ogni nodo che rappresenta un corpo rigido ed ogni collegamento tra i nodi rappresenta un giunto. Questa struttura permette di definire e calcolare la cinematica e la dinamica del robot, facilitando il controllo del robot;

- **getTransform(robot, configuration, sourcebody, targetbody)** calcola la trasformazione del **sistema** di riferimento del corpo sorgente rispetto il sistema di riferimento del corpo di destinazione, nella configurazione specificata dalla variabile "configuration". La variabile robot deve essere di tipo "RigidBodyTree". Si otterrà tramite questo comando la matrice di trasformazione in forma omogenea;

- **geometricJacobian (robot, configuration, "body_name")** calcola la matrice Jacobiana nella la configurazione indicata in "configuration" per il corpo specificato in "body_name".

- **checkCollision(robot, config)** comando che controlla se il modello di robot in formato rigidBodyTree è in autocollisione alla configurazione specificata con output corrispondente ad un vettore logico. Aggiungendo agli input un oggetto specificato con parentesi graffe "{nome_corpo}", il comando verificherà se c'è la presenza di

autocollisione o della collisione con l'oggetto specificato indicando come output un vettore logico a due elementi relativi ai due tipi di collisione;

- **inverseKinematics** comando utilizzato per creare un risolutore di cinematica inversa (IK) per calcolare le configurazioni dei giunti per una posa desiderata del link selezionato in base ad un modello "RigidBodyTree". Questo modello definisce tutti i vincoli dei giunti che il risolutore impone. L'utilizzo avviene tramite linea di codice

  [q_sol, q_info] = ik('ee_link',T_des, weights, Initialguess)

  Come input bisogna indicare il nome del sistema di riferimento per il quale si vuole calcolare la cinematica inversa e fornire la posa desiderata a cui si dovrà portare il robot. La posa è da specificare come matrice omogenea, nell'esempio è indicata con il nome "T_des". Il parametro "weights" rappresenta le tolleranze di errore rispetto la posa desiderata. Esso è specificato come vettore a sei elementi, di cui i primi tre corrispondono ai pesi dell'errore di orientazione mentre gli altri tre corrispondono ai pesi nell' errore di posizione rispetto x y z. La variabile "Initialguess" indica la configurazione iniziale del robot specificato come struttura o vettore, utilizzata come guida per il risolutore verso la configurazione desiderata del robot. Come output si otterrà "configSol", configurazione del robot restituita come struttura, e "solInfo", conterrà le informazioni sulla soluzione.

In entrambi i Toolbox sono disponibili diversi comandi per l'interpolazione delle traiettorie, ognuno dei quali consente di ottenere una traiettoria sotto forma di configurazione dei giunti "q", velocità "qd" e accelerazione "qdd". Questi comandi offrono varie metodologie di interpolazione per adattarsi a specifiche esigenze di movimento del robot, garantendo fluidità, precisione e controllo ottimale. Vengono di seguito elencati i comandi relativi ai metodi più utilizzati [42] [43]:

- **[q,qd,qdd] = cubicpolytraj(wayPoints, timePoints, tSamples)** genera un polinomio di terzo ordine che transita per il set di pose specificate in "waypoints" negli istanti di tempo forniti nella variabile "timePoints". La funzione restituisce posizione, velocità e accelerazioni angolare dei giunti ai campioni di tempo indicati in "tSamples";

- **[q,qd,qdd] = bsplineploytraj(controlPoints, tInterval, tSamples)** genera una traiettoria B-spline cubica a tratti che rientra nel poligono di controllo definito da "controlPoints". La traiettoria è campionata uniformemente tra i tempi di inizio e fine indicati in "tInterval". La funzione restituisce posizione, velocità e accelerazioni angolare dei giunti ai campioni di tempo dati, "tSamples";

- **[q,qd,qdd] = quinticpolytraj(wayPoints, timePoints, tSamples)** genera un polinomio di quinto ordine che raggiunge un dato set di "waypoints" di input con corrispondenti punti temporali. La funzione restituisce posizione, velocità e accelerazioni angolare dei giunti ai campioni di tempo dati, "tSamples";

- **[q,qd,qdd] = trapveltraj(wayPoints, numSamples)** genera una traiettoria attraverso un dato set di "waypoints" di input che seguono un profilo di velocità trapezoidale. La funzione restituisce posizione, velocità e accelerazioni angolare dei giunti ai campioni di tempo dati, "tSamples".

3.6.2 Cobot comandato in spazio giunti

Nel primo caso applicativo è stata definita la configurazione che i sei giunti devono raggiungere al termine del movimento. Tale configurazione è indicata definendo gli angoli dei giunti in radianti, sottoforma di vettore 1x6, [q1, q2, q3, q4, q5, q6]. Nella "Figura 12" è riportata la configurazione di partenza e la configurazione finale del movimento:

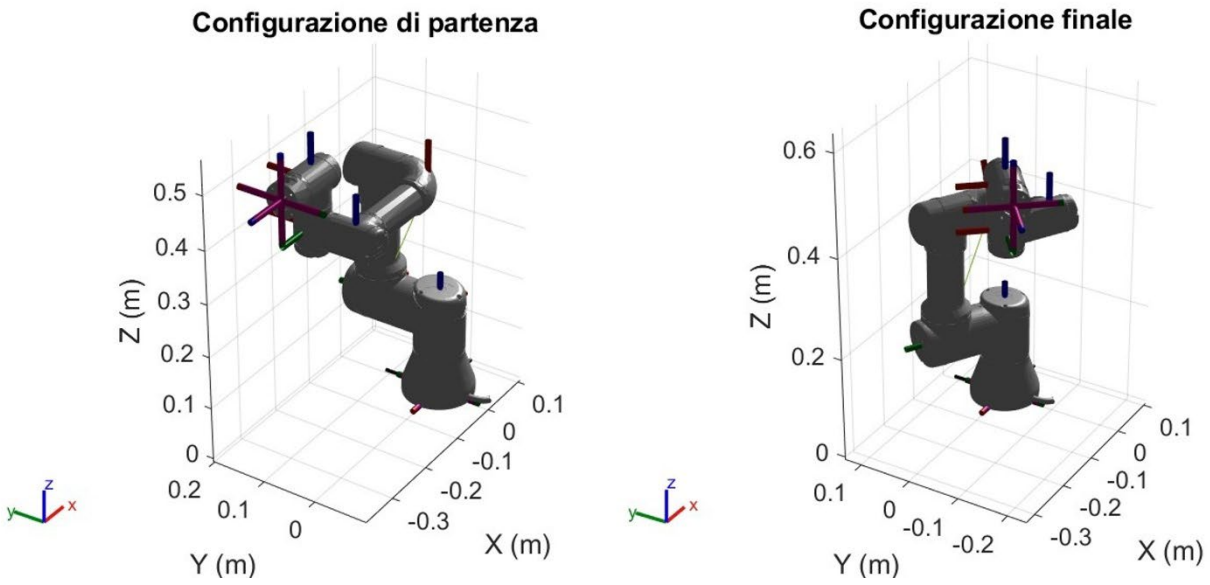

Figura 12: Configurazione del cobot di iniziale e finale

Il cobot è comandato tramite la funzione "sendJointConfiguration" indicando come input il modello UR3 precedentemente caricato tramite comando "loadrobot", e la configurazione finale dei giunti.

Grazie al collegamento Matlab-ROS-UR3 si ha un canale di comunicazione tra il robot e l'ambiente di calcolo Matlba. Utilizzando le funzioni del "Universal Robots Toolbox" è stato

possibile effettuare le acquisizioni dei dati durante l'esecuzione del movimento. In particolare, si utilizza la funzione "recordRobotState" per registrare i parametri del robot in movimento, per poi salvarli tramite funzioni specifiche per i parametri d'interesse.

Nella "Figura 13" si osserva l'andamento degli angoli dei giunti durante l'esecuzione del movimento, rappresentativi delle configurazioni del cobot in ogni istante temporale di campionamento. Queste informazioni sono ottenute tramite il comando "getJointConfiguration".

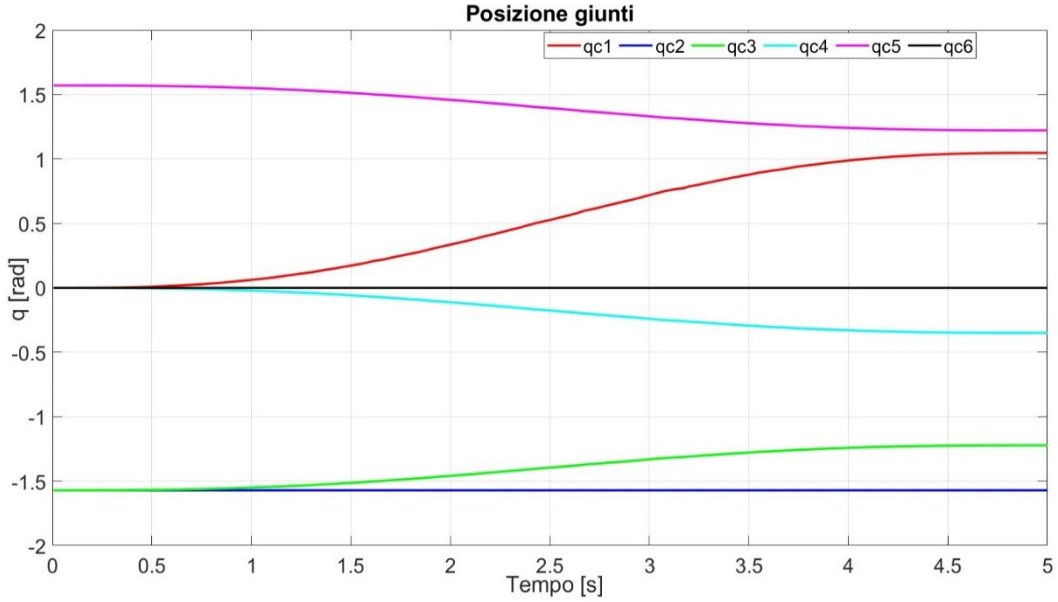

Figura 13: Andamento della posizione dei giunti durante il movimento eseguito con comando "sendJointConfiguration"

Mediante il comando "getJointVelocity" di acquisiscono le informazioni relative alle velocità angolari dei giunti (Figura 14)

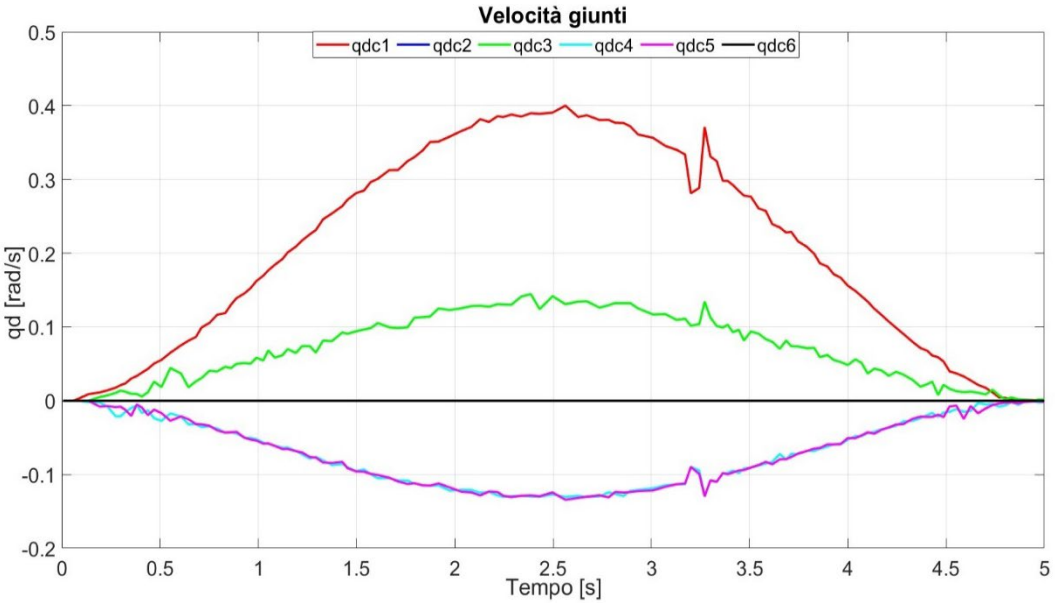

Figura 14: Andamento delle velocità angolari nei giunti durante il movimento eseguito con comando "sendJointConfiguration"

Si effettua l'acquisizione dei dati delle coordinate in spazio cartesiano (x, y, z) dell'End Effector utilizzando il comando "getCartesianPose", tramite cui si ricavano informazioni in ogni istante di tempo della posa relativa al movimento del terminale. I parametri acquisiti sono diagrammati nella "Figura 15"

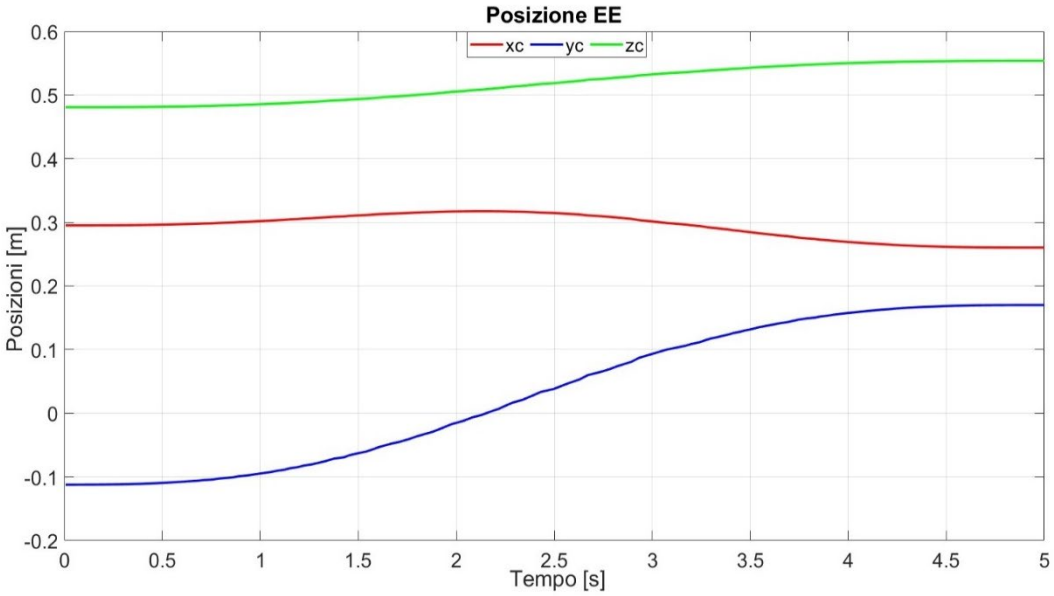

Figura 15: Andamento della posizione dell'End effector durante il movimento eseguito con comando "sendJointConfiguration"

3.6.2 Traiettoria rettilinea

Si effettua una pianificazione della traiettoria per eseguire un percorso rettilineo, fornendo le informazioni necessarie in spazio giunti. Per ottenere un percorso lineare si è reso necessario partire dalla definizione dei punti nello spazio cartesiano convertendoli poi in informazioni in spazio giunti tramite la cinematica inversa. L'orientazione dell'End Effector è stata mantenuta fissa. Dopo aver definito il tempo totale di percorrenza, questo è stato suddiviso in intervalli temporali considerando la frequenza di campionamento del cobot, pari a 125 Hz, e quindi un tempo di campionamento di 1/125 = 0.008 s. Le configurazioni dei giunti determinate dalla cinematica inversa sono state interpolate utilizzando un comando fornito dalla Robotics System Toolbox per ottenere informazioni in spazio giunti sulla configurazione, la velocità angolare e l'accelerazione angolare a ogni intervallo temporale.

Il comando d'interpolazione utilizzato nel caso applicativo è "bsplinepolytraj" necessario a creare traiettorie basate su spline B cubiche, definite da polinomi di terzo grado, garantendo traiettorie lisce e continue, prive di oscillazioni. Come input del comando sono inserite informazioni sulle configurazioni dei giunti ottenute dalla cinematica inversa, l'intervallo di tempo totale ed il vettore di tempi specifici in cui calcolare le configurazioni, velocità e accelerazioni angolari dei giunti.

Tramite il canale di comunicazione Matlab-ROS-UR3, il cobot UR3 è comandato utilizzando la funzione *FollowTrajectory* inserendo come informazioni di input la posizione, velocità e

accelerazione angolare dei giunti ottenute tramite interpolazione, e l'informazione dei tempi di percorrenza delle configurazioni indicate. L'acquisizione dei parametri si effettua tramite funzioni "Universal Robots Toolbox".

Dal cobot si ottengono informazioni sulle configurazioni dei giunti, "qc", confrontate graficamente con le configurazioni ottenute dall'interpolazione, "qj" (Figura 16):

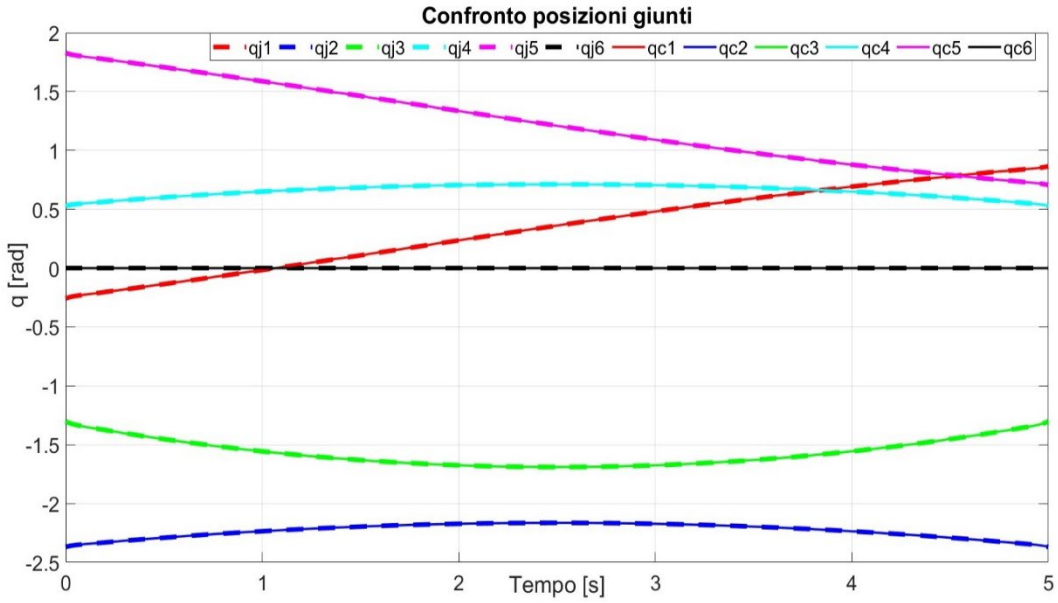

Figura 16: Confronto delle configurazioni dei giunti nell'esecuzione della traiettoria rettilinea

Nella "Figura 17" è rappresentato il confronto tra le informazioni delle velocità angolari dei giunti "qdc" acquisite dal robot, e i dati relativi alle velocità angolari dei giunti ottenute da interpolazione "qdj" (Figura 17):

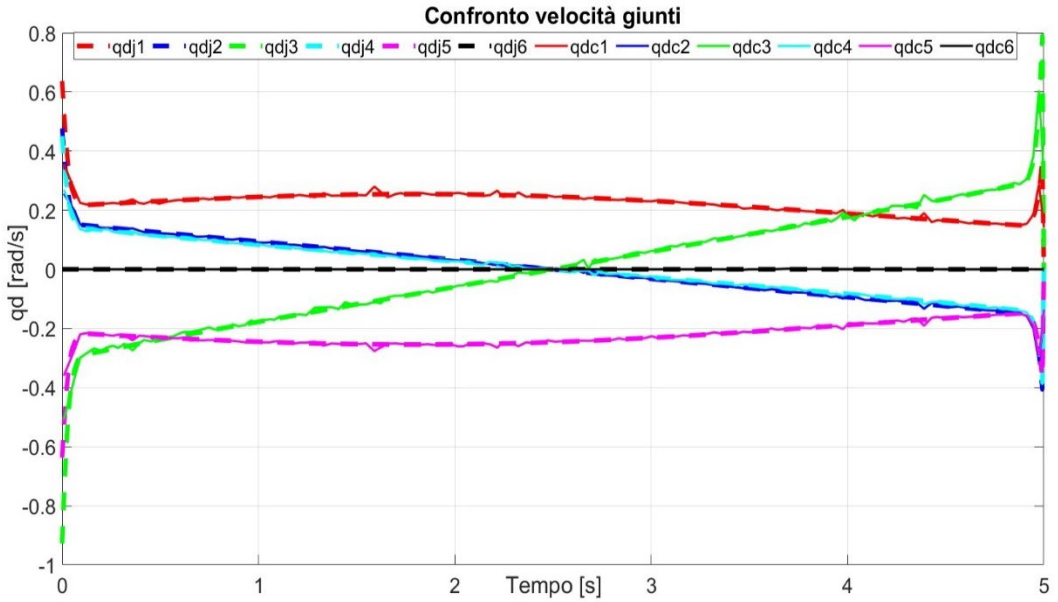

Figura 17: Confronto delle velocità angolari dei giunti nell'esecuzione della traiettoria rettilinea

Acquisite le informazioni sulle posizioni dell'End Effector nello spazio cartesiano (xc, yc, zc) sono confrontate graficamente con i punti (xj, yj, zj) che descrivono il percorso (Figura 18).

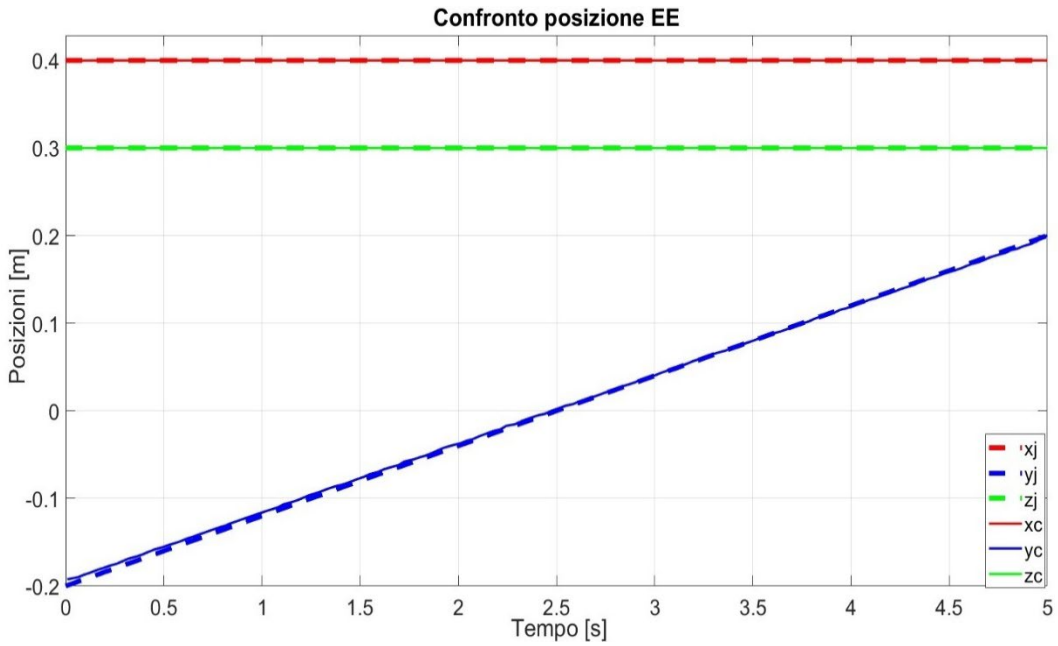

Figura 18: Confronto delle coordinate dell'End Effector nell'esecuzione della traiettoria rettilinea

Di seguito è riportato il percorso effettivo con grafico 3D (Figura 19) utilizzando le informazioni acquisite per le posizioni dell'End Effector:

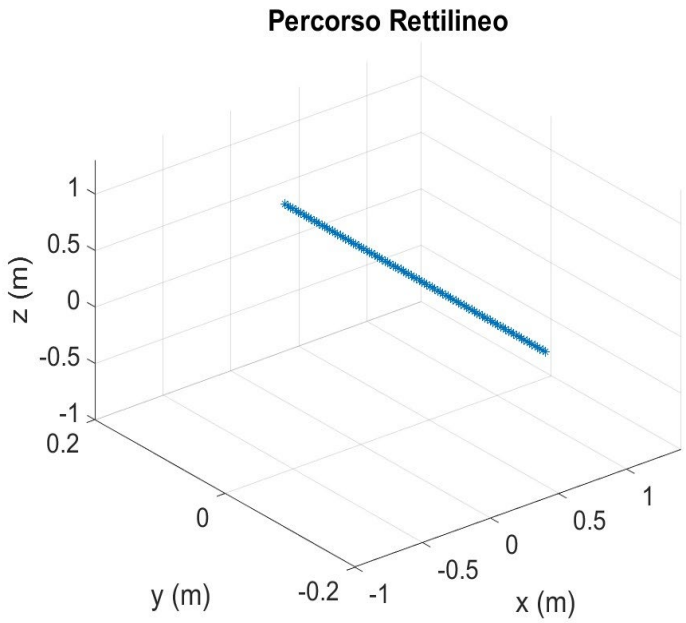

Figura 19: Traiettoria rettilinea nello spazio 3D

3.6.4 Traiettoria triangolare

La traiettoria triangolare è stata realizzata fornendo informazioni in spazio cartesiano ed eseguita mediante l'utilizzo del comando *FollowWayPoints*. Si definisce la posa nello spazio cartesiano indicando l'orientazione dell'End Effector (rotz, roty, rotx), mantenuta fissa, e le coordinate relative alla posizione (x,y,z), fatte variare disegnando nello spazio un percorso triangolare.

Per utilizzare il comando FollowWayPoints si indicano come input le informazioni relative al percorso triangolare in coordinate cartesiane, sottoforma di matrice (rotz, roty, rotx, x, y, z), i tempi in cui l'End Effector deve trovarsi in ogni punto e il metodo d'interpolazione utilizzato per calcolare il movimento tra le pose indicate.

Il comando per l'interpolazione applicato nel caso studio è "quinticpolytraj", il quale genera una traiettoria polinomiale che passerà per le posizioni specificate garantendo continuità di posizione, velocità e accelerazione.

Una volta definiti tutti i parametri necessari al comando *FollowWayPoints*, lo si usa per inviare le informazioni di movimento all'UR3 e ricevere feedback durante l'esecuzione della traiettoria. L'acquisizione è resa possibile dall'utilizzo del codice costruito tramite funzioni "Universal Robots Toolbox". Questi sono di seguito riportati graficamente con lo scopo di comprendere le potenzialità del collegamento e le possibilità dei codici con cui si può comandare il cobot.

Eseguita l'acquisizione si riportano le informazioni relative alle configurazioni dei giunti per ogni istante di tempo (Figura 20):

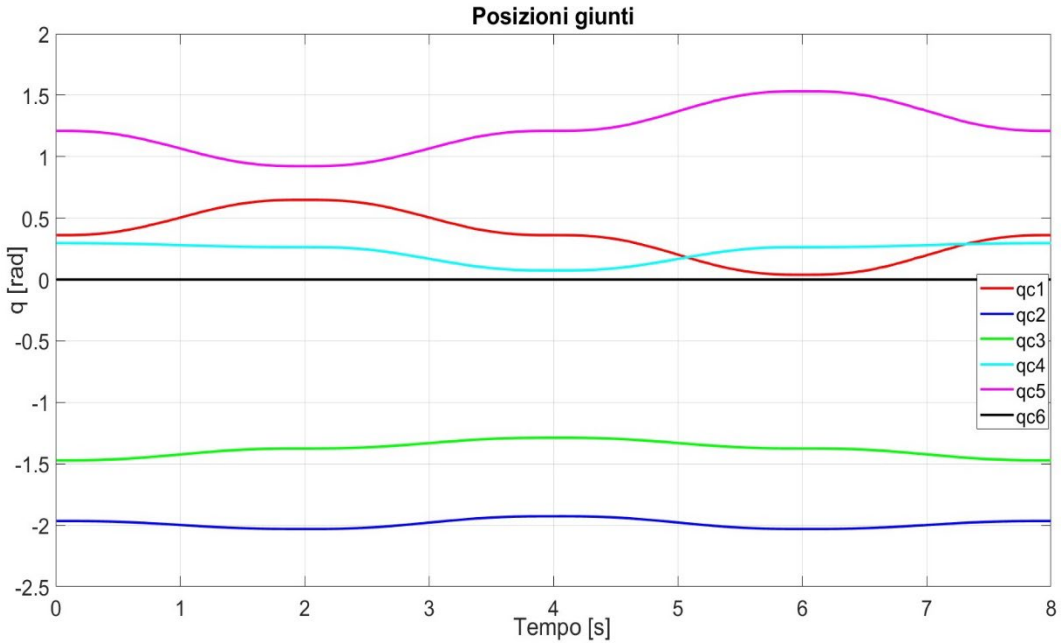

Figura 20: Rappresentazione delle configurazioni dei giunti acquisite durante l'esecuzione della traiettoria triangolare

Nella “Figura 21” sono diagrammate le informazioni relative alle velocità dei giunti nel tempo acquisite tramite il collegamento. Da questo andamento si evidenzia il comportamento del robot in prossimità dei punti forniti per disegnare il percorso triangolare, dove si ha una diminuzione della velocità fino al valore nullo, in cui il robot si ferma per poi ripartire:

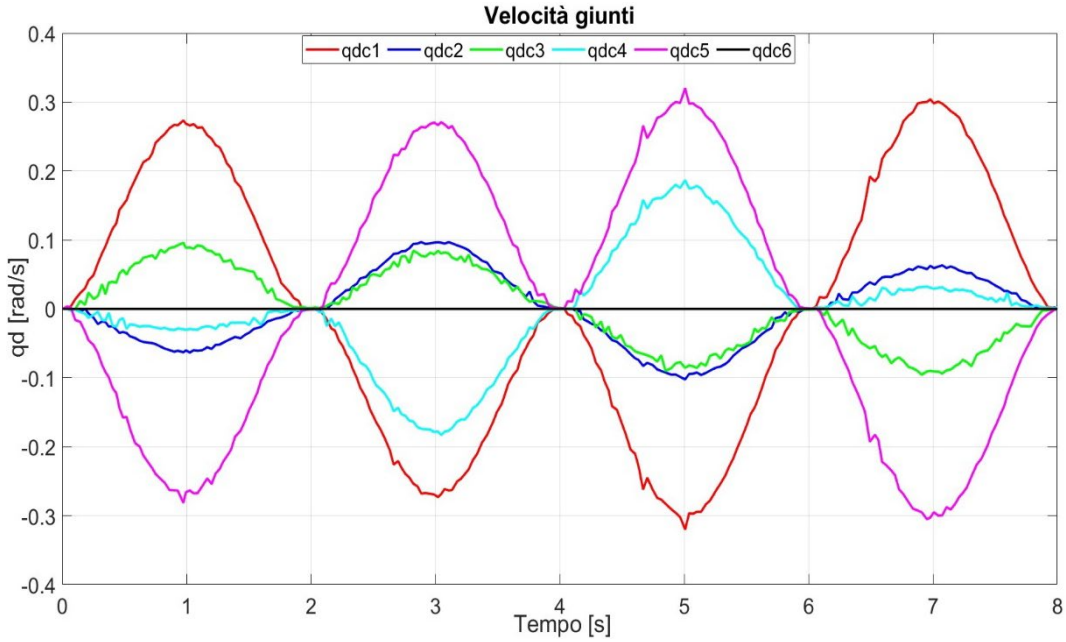

Figura 21: Rappresentazione delle velocità angolari nei giunti acquisite durante l’esecuzione della traiettoria triangolare

Le informazioni acquisite relative alle posizioni dell’End Effector come coordinate cartesiane (xc,yc,zc) e confrontate con le coordinate fornite dai WayPoints (xj, yj, zj). Si osserva (Figura 22) un andamento a tratti per quanto riguarda le informazioni dei WayPoints poiché vengono forniti solo 5 punti per definire il percorso triangolare, poi aumentati in indicazioni per le configurazioni dei giunti tramite interpolazione:

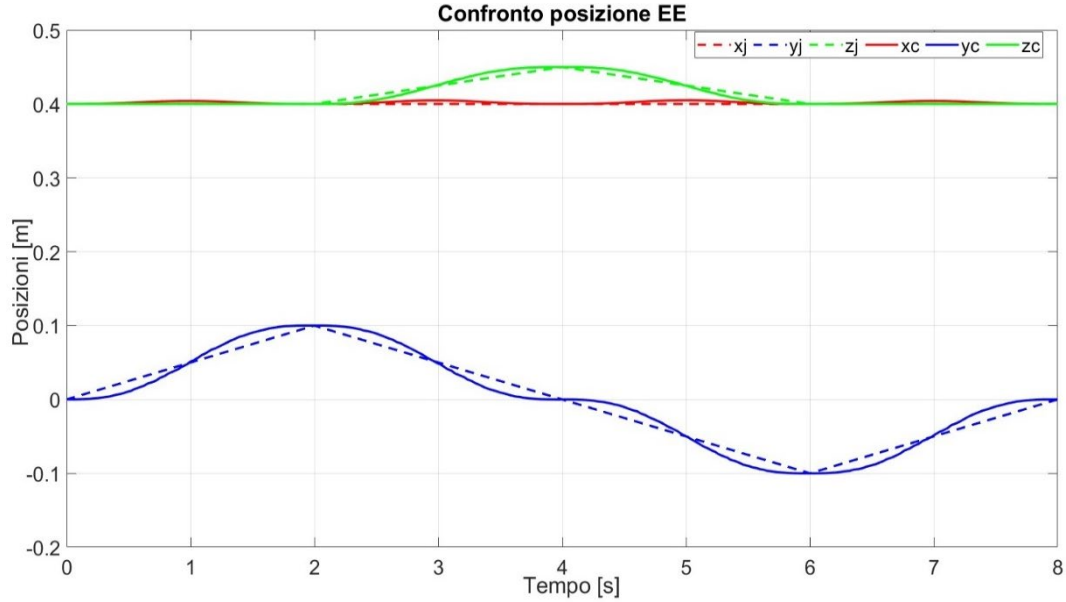

Figura 22: Confronto della posizione dell'End Effector per la traiettoria triangolare

È riportato il percorso triangolare in 3D ottenuto dall'acquisizione delle posizioni (Figura 23)

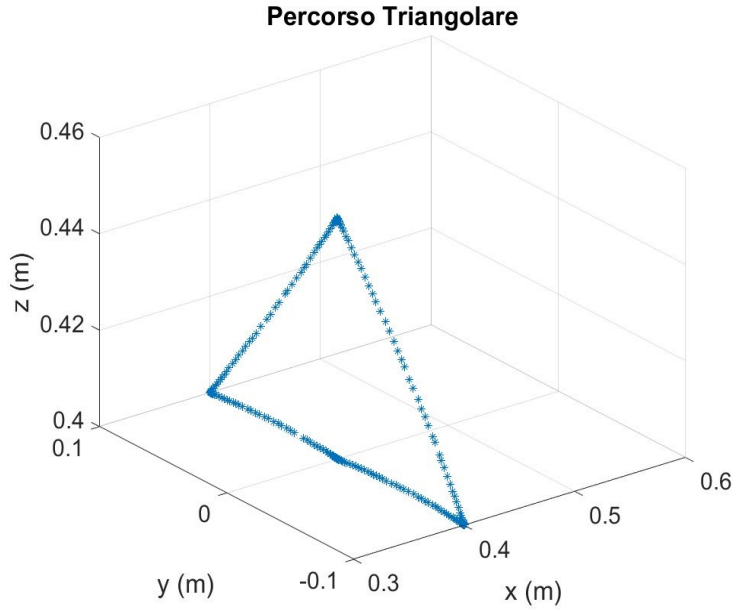

Figura 23: Rappresentazione 3D della traiettoria triangolare

3.6.5 Traiettoria circolare

Nel seguente caso studio si realizza una traiettoria circolare utilizzando il comando *FollowTrajectory*, partendo dal profilo di velocità desiderato, a differenza di quanto fatto per la traiettoria rettilinea, che era basata sulle posizioni.

È stato definito il profilo di velocità desiderato in spazio cartesiano, poi convertito in velocità in spazio giunti "qd" utilizzando la Jacobiana inversa, calcolata a partire dalla configurazione iniziale dei giunti. Successivamente, integrando la velocità "qd" si è ottenuta la configurazione "q", derivandola si è ottenuta l'accelerazione "qdd". Dalla configurazione giunti relativa all'istante di tempo si ricalcola la Jacobiana inversa e si ripete la procedura fino al termine della conversione del profilo di velocità. Al termine dell'operazione si ottengono le matrici relative a "q", "qd" e "qdd" necessarie al comando *FollowTrajectory* utilizzato per comandare il movimento. Sono acquisiti dati, grazie al collegamento creato tramite ROS, per comprendere l'effettivo movimento del robot e per eseguire un confronto con i dati teorici ricavati dallo studio descritto.

Di seguito è riportato il confronto grafico tra le configurazioni dei giunti acquisite (qc) con le configurazioni dei giunti ottenute dall'interpolazione (qj) ed utilizzate come input del comando (Figura 24):

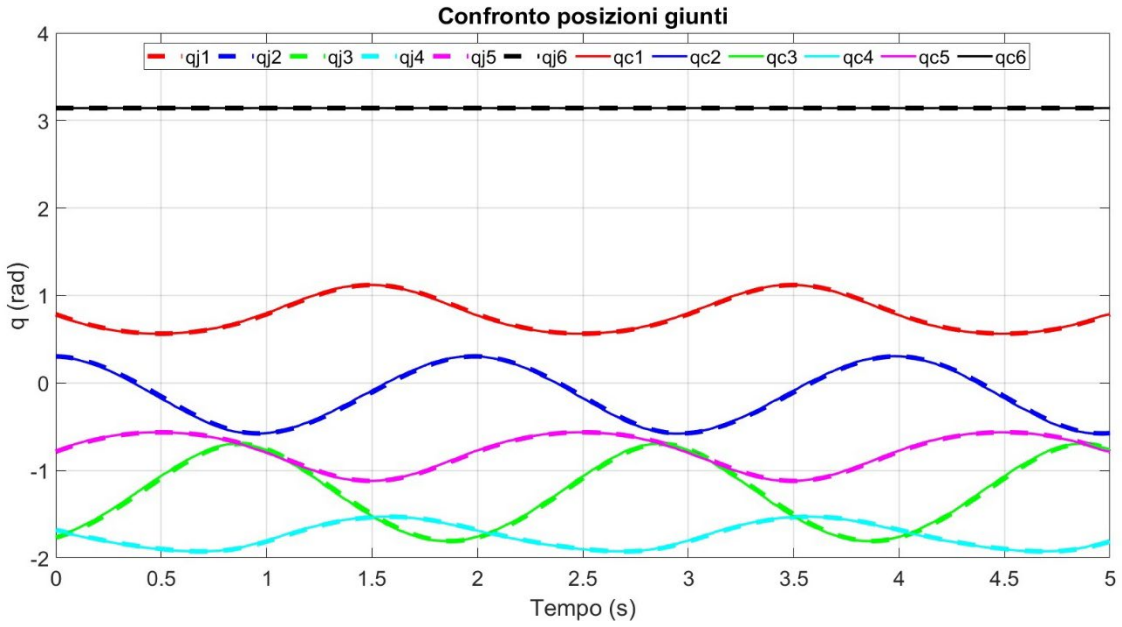

Figura 24: Confronto delle configurazioni dei giunti della traiettoria circolare

Di seguito sono confrontate graficamente le velocità giunti acquisite (qdc) con le velocità giunti ottenute dallo studio descritto (qdj) (Figura 25):

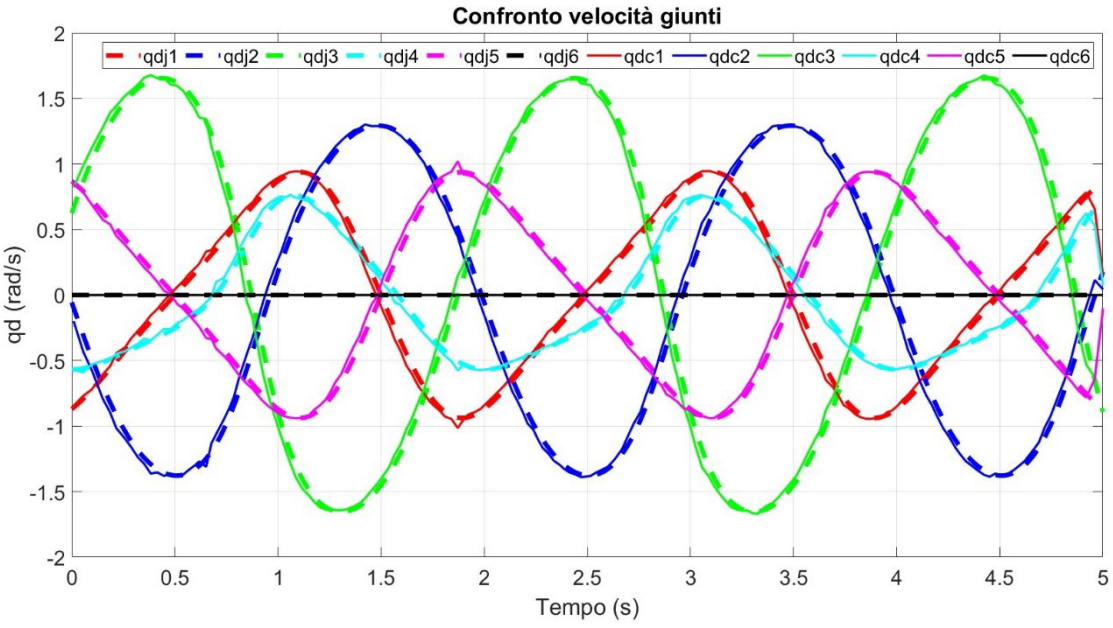

Figura 25: Confronto delle velocità dei giunti della traiettoria circolare

Acquisite le posizioni in spazio cartesiano dell'End Effector (xc, yx, zc) sono confrontate graficamente con le posizioni ricavate da studio preliminare (xj, yj, zj) (Figura 26):

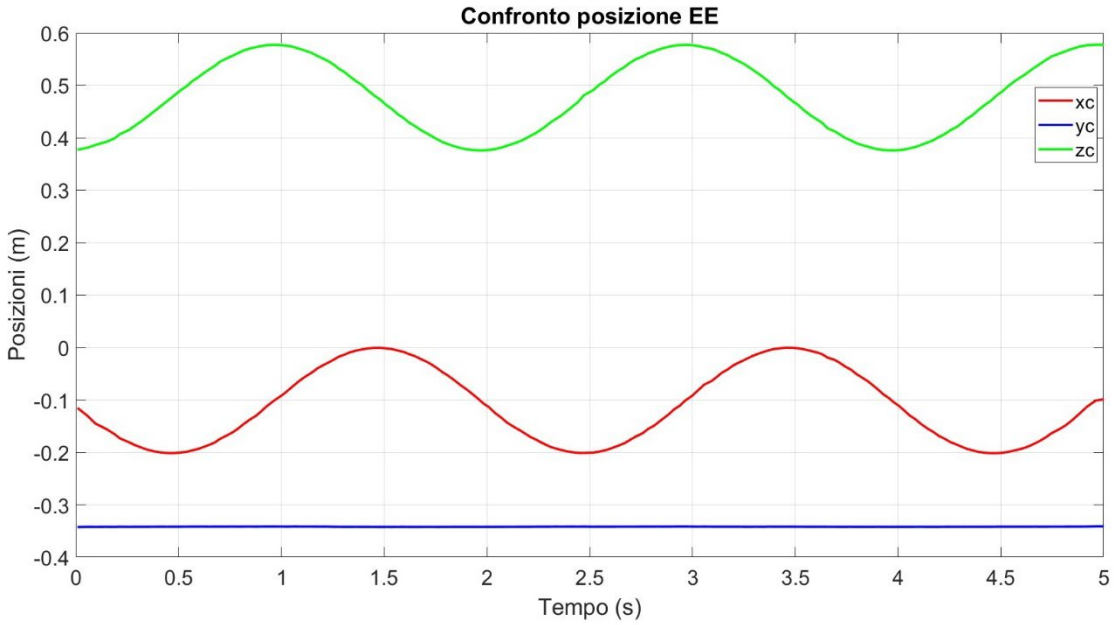

Figura 26: Confronto delle posizioni dell'End Effector per la traiettoria circolare

Mediante l'acquisizione dei dati delle posizioni in spazio cartesiano, sono riportate le coordinate su un grafico 3D ottenendo una visualizzazione del movimento circolare effettivo eseguito dall'End Effector (Figura 27):

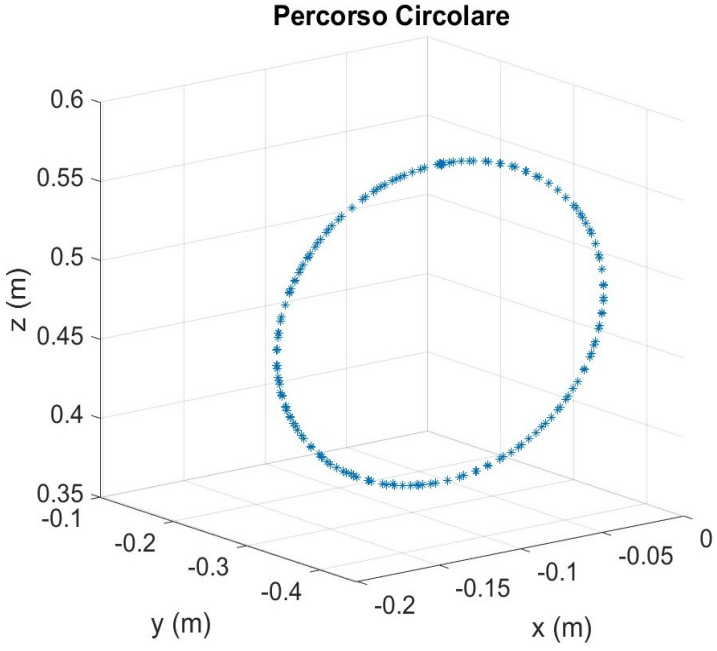

Figura 27: Rappresentazione 3D della traiettoria circolare

3.7 Conclusioni

Dallo studio dei casi applicativi si è osservata la potenzialità del collegamento creato Matlab-ROS- URSim/UR3 e le diverse possibilità di comando del cobot rappresentate dalle funzioni del “Universal Robots Toolbox”. Dai grafici ottenuti confrontando i dati inviati al robot e i dati effettivi ottenuti da acquisizione, si osserva la sovrapposizione delle variabili, con variazioni impercettibili, prova che il cobot esegue correttamente ciò che gli viene comandato. Bisogna prestare attenzione ai sistemi di riferimento prima di comandare il cobot tramite collegamento, è necessario verificare che i sistemi di riferimento caricati tramite modello Mathworks coincidano con quelli rilevati da ROS. Durante la verifica dei movimenti si sono comprese alcune limitazioni dei comandi appartenenti al pacchetto Universal Robots. Nel particolare, si è appreso come per un uso corretto dei comandi “FollowWayPoints” e “FollowTrajectory” sia necessario fornire come primo dato la configurazione in cui si trova il robot. Nel caso in cui si indichi una posa iniziale diversa dalla posa pre-movimento del robot, questo riscontrerà un errore, attiverà i freni entrando in “Safety Mode” e si disconnetterà dal collegamento. Al contrario, tramite il comando “sendJointConfiguration” il robot riconosce la configurazione in cui si trova ed esegue il movimento per assumere la configurazione indicata.

4. REINFORCEMENT LEARNING

Il tema principale di questo capitolo è l'applicazione del Reinforcement Learning ad un caso reale. L'obiettivo principale del lavoro è lo sviluppo di un pianificatore di movimento in ambiente dinamico con l'utilizzo del Reinforcement Learning. Lo scopo è quello di comprendere le potenzialità ed i limiti di questo modello. All'inizio del capitolo viene descritta la struttura del modello e dei suoi componenti costitutivi, con la presentazione dell'applicazione fornita da Mathworks, che facilita l'impostazione e l'utilizzo del metodo. Si passa poi all'analisi del problema applicativo, su cui si è focalizzato lo studio ed il relativo approccio risolutivo sviluppato con la definizione di tutte le componenti necessarie con relativa costruzione e ragionamento. In seguito, è stato suddiviso il problema in sottoproblemi più semplici, analizzati e costruiti con la definizione di tutte le componenti necessarie per l'attuazione del metodo del Reinforcement Learning [44] [45].

4.1 Cos'è il Reinforcement Learning

Il Reinforcement Learning è un metodo di apprendimento automatico basato sulla psicologia dell'apprendimento animale, in cui un Agent, viene addestrato a svolgere attività di complessità differente tramite ripetute interazioni di tipo "trial-and-error" in ambienti dinamici cercando di massimizzare la Reward. È quindi in grado di affrontare problemi complicati relativi al processo decisionale sequenziale senza conoscere informazioni sulla dinamica oggettiva e sotto l'influenza di disturbi esterni [46]. Gli algoritmi alla base del Reinforcement Learning migliorano le proprie prestazioni in modo adattivo con l'aumento del numero dei campioni disponibili per l'apprendimento e, tramite l'utilizzo di reti neurali, approssimano le funzioni di valore e le politiche che l'Agent deve apprendere per essere in grado di affrontare attività complesse. Il significato di Reward e Agent verrano spiegati nel dettaglio nei prossimi paragrafi [47].

I tempi di addestramento, anche delle azioni più semplici, possono andare da qualche minuto a delle ore, fino a giornate intere. Formulare i problemi in modo corretto potrebbe essere difficile in quanto bisogna prendere tutta una serie di decisioni di progettazione e, prima di raggiungere il risultato desiderato, potrebbero servire diverse iterazioni. Tra queste si ha la selezione dell'architettura appropriata per le reti neurali, la regolazione degli iper-parametri e la progettazione della funzione di ricompensa, basata sul compito che si vuole far compiere all'Agent.

I componenti fondamentali del Reinforcement Learning sono (Figura 28):

- Environment
- Reward
- Agent
- Training

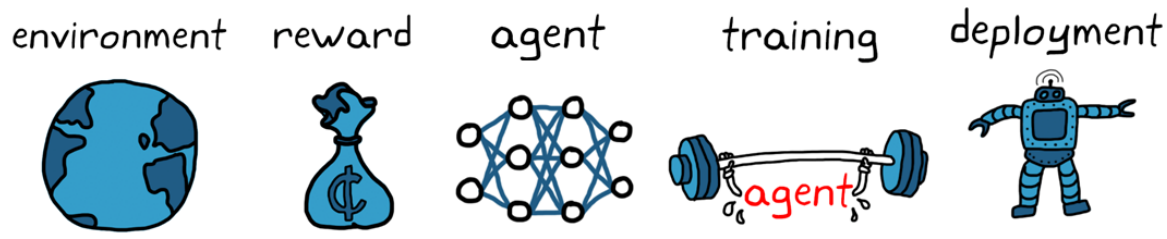

Figura 28: Struttura del Reinforcement Learning [47]

4.1.1 Environment

L'environment, o ambiente, modella il sistema esterno, il mondo, ed i segnali con cui l'Agent interagisce. Agent ed Environment interagiscono tra loro in ogni sequenza di passaggi temporali
I componenti necessari per la definizione dell'Environment sono [48]:

- **Observation**, rappresentano lo stato attuale dell'ambiente, indicando i parametri necessari che si vogliono fornire all'Agent per aiutarlo nelle sue valutazioni. Tra questi valori sarà presente anche la Reward e la Last Action, ultima azione presa dall'Agent. Le osservazioni possono essere suddivise in uno o più canali ciascuno dei quali trasporta gruppi di singoli elementi numerici discreti o continui. Ciascun gruppo può essere organizzato secondo un numero qualsiasi di dimensioni;

- **Action**, rappresentano le possibili azioni che può intraprendere l'Agent come strategia per massimizzare la sua ricompensa futura. Ogni azione eseguita dall'Agent modifica lo stato dell'Environment che a sua volta formulerà una nuova observation. Per le Action è consentito un unico canale e quindi unico gruppo di qualsiasi dimensione;

- **Reset function**, ha il compito di riportare l'ambiente ad uno stato iniziale, inizializzando i parametri e le variabili del problema. Funzione chiamata all'inizio di ogni episodio di allenamento o quando l'ambiente raggiunge uno stato terminale.

Si può comprendere meglio il funzionamento dell'environment considerando un'interazione Agent-Environment con passaggi temporali discreti (Figura 29). Ad un passo temporale t, l'ambiente sarà in uno stato S(t), che risulta nell'osservazione O(t). In base a O(t) e alla politica interne, l'Agent calcolerà l'Action A(t). Basandosi sullo stato S(t), sull'Action A(t) e in base alla sua dinamica interna, l'environment aggiornerà il suo stato a S(t+1), presente nella successiva osservazione O(t+1). Sulla base di S(t), A(t) e S(t+1),

l'environment calcolerà una ricompensa scalare R(t+1) indicatrice della bontà dell'Action A(t). Al passo temporale successivo t+1 l'agent riceve l'osservazione O(t+1) e la ricompensa R(t+1). Sulla base delle osservazioni ricevute l'algoritmo di apprendimento aggiornerà i parametri della policy dell'Agent tentando di migliorarla. In base a O(t+1) l'Agent calcolerà l'azione successiva A(t+1) [48].

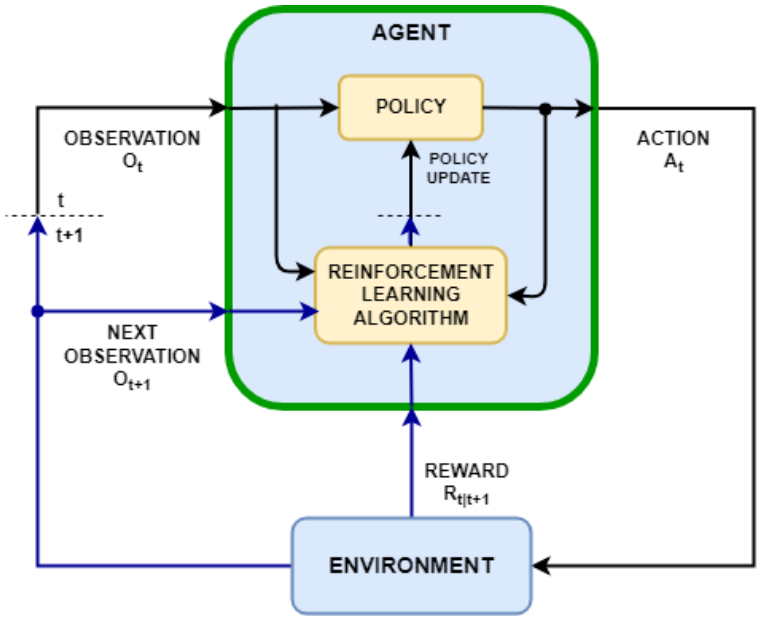

Figura 29: Rappresentazione del processo di scelta e valutazione nel Reinforcement Learning [48]

4.1.2 Agent

Come precedentemente detto, il Reinforcement Learning ha come obiettivo quello di addestrare un Agent a completare un'attività in un ambiente incerto. L'Agent, ad ogni intervallo di tempo, riceve osservazioni dall'ambiente, tra cui la Reward, le elabora e sviluppa un Action da mandare all'environment.
I componenti costituenti dell'Agent sono:

- **Policy**: è il cuore del processo decisionale, supportata da modelli matematici avanzati che trasformano le osservazioni dell'ambiente in azioni concrete attraverso degli approssimatori di funzione, come le reti neurali profonde;

- **Algoritmo di apprendimento**: migliora continuamente le decisioni dell'Agent in base all'esperienza acquisita attraverso l'interazione con l'ambiente, con l'obiettivo finale di ottenere una politica ottimale che massimizzi la ricompensa complessiva nel tempo.

A seconda dell'Agent l'algoritmo di approssimazione opera su uno o più approssimatori di funzioni parametrizzate che apprendono la politica. Gli approssimatori possono essere utilizzati come Critic o Actor.

Gli Agent che utilizzano approssimatori esclusivamente Critic per la determinazione delle Action si basano su una politica detta indiretta. Questo approccio porta a rappresentare una funzione valore basata sulle osservazioni o una funzione Q che esprime il valore in base all'osservazione e all'azione. Questi Agent tendono a funzionare meglio con spazi di azione discreti ma possono essere computazionalmente costosi quando devono gestire spazi di azione continui.

Gli Agent che si affidano solo ad approssimatori Actor utilizzano direttamente la Reward, hanno cioè una politica diretta che può essere deterministica o stocastica. Possono gestire spazi di azione continui ma l'algoritmo di addestramento può essere sensibile al rumore e rischiare di convergere ai minimi locali.

Gli Agent che utilizzano sia Actor che Critic sono definiti Agent Actor-Critic. In particolare, l'Actor apprende l'Action migliore da intraprendere utilizzando i feedback del Critic. Allo stesso tempo, il Critic apprende la funzione valore dalle Reward in modo da fornire un'adeguata valutazione all'Actor. Questi Agent sono adatti alla gestione di spazi d'azione discreti e continui [49].

4.1.3 Reward

Nel Reinforcement Learning, la Reward guida l'Agent verso l'apprendimento di una politica ottimale. È progettata per fornire feedback relativi alla bontà delle decisioni dell'Agent in base agli obiettivi dell'attività.

Per la costruzione della Reward, in generale, viene fornito un valore positivo se l'azione eseguita ha portato ad un risultato buono con lo scopo di incoraggiare l'Agent, mentre riceve valore negativo quando si vogliono scoraggiare Action che si discostano dall'attività che deve compiere. Se ben progettata, la Reward guiderà l'Agent a massimizzare la Reward cumulativa a lungo termine guidandolo nell'apprendimento e a svolgere l'attività desiderata.

Nel caso in cui l'esecuzione dell'attività preveda di rispettare più obiettivi ed aspetti applicativi, si costruirà una Reward a più parametri. Ognuno di questi sarà caratterizzato da un valore scalare che indicherà il peso del fattore. Questo valore sarà determinato in base ad una scala di priorità definita dall'operatore. Il bilanciamento dei pesi per ogni obiettivo non è un compito semplice e potrebbe portare l'Agent a trascurare alcuni obiettivi favorendone altri. Il segnale della Reward può essere continuo, discreto o misto [50].

- **Reward continua:** varia continuamente con i cambiamenti relativi alle observations e alle actions dell'Environment. Questo tipo di segnale migliora la convergenza durante la formazione e possono portare a strutture di rete più semplici;

- **Reward discreta:** varia in modo discontinuo con i cambiamenti nelle observations e delle actions. Questi tipi di segnale possono rallentare la convergenza e richiedere strutture di rete più complesse. Pur rallentando la convergenza, questo tipo di Reward possono guidare l'Agent verso regioni con migliori Reward nello spazio;

- **Reward mista:** è costituita da una combinazione di componenti di ricompensa continue e discrete. Quella discreta serve ad allontanare il sistema da stati negativi mentre il segnale di ricompensa continua può migliorare la convergenza con un andamento graduale vicino agli stati target [50].

4.1.4 Training

Il Training è la fase in cui tutti gli elementi precedentemente descritti interagiscono tra loro con lo scopo di addestrare l'Agent ad eseguire l'attività d'interesse. Il Reinforcement Learning, come precedentemente detto, è basato sulla filosofia "trail and error", metodo di problem solving che consiste nel tentare varie soluzioni per risolvere un problema, scartando quelle che non funzionano e affinando gradualmente l'approccio fino a trovare una soluzione ottimale. Infatti, tramite più sessioni di allenamento e l'interazione Agent-Environment, l'Agent esplora l'ambiente in cui si trova alla ricerca di soluzioni ottimali per poi eseguire fasi di affinamento per consolidarle. La singola sessione di allenamento è costituita da Episode (episodi) a loro volta costituiti da una sequenza di azioni compiute dall'Agent, Step, per passare da uno stato iniziale ad uno finale. Ogni decisione dell'Agent, e quindi ad ogni Step, viene valutata e classificata con un valore scalare, la Reward. La somma di tutte le Reward degli Step costituisce la Reward cumulata rappresentativa del risultato del singolo Episode e tramite cui verrà valutato. Da un Episode al successivo, agisce la Reset function che riporta i parametri nelle condizioni iniziali, per poter analizzare una nuova serie di decisioni dell'Agent.

Per la fase di Training Mathworks mette a disposizione un'app presente nel Reinforcement Learning Toolbox, il "Reinforcement Learning Designer". Quest'app permette di impostare in modo semplice le funzioni di allenamento dell'Agent e fornisce un'interfaccia di visualizzazione per l'andamento progressivo della fase. Al termine della sessione di addestramento propone come risultati il modello di Agent allenato, ed il grafico relativo all'andamento della reward cumulata [51].

Nell'app del RL Toolbox è possibile creare o importare l'Environment e l'Agent. La creazione dei due elementi è supportata dalle librerie messe a disposizione da Mathworks per caricare modelli già definiti. L'importazione viene effettuata nel caso in cui si vogliano utilizzare

Environment e/o Agent costruiti dall'utente. Particolare attenzione per quanto riguarda il blocco per la creazione dell'Agent (Figura 30):

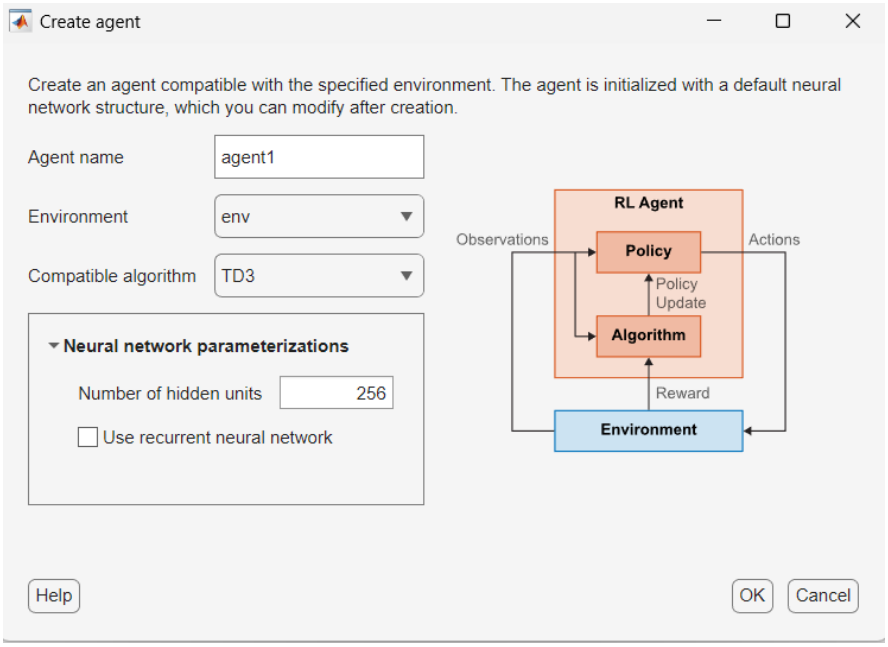

Figura 30: Menù di creazione dell'Agent

Bisogna specificare il nome che vogliamo dare all'Agent, l'ambiente in cui deve essere creato e l'algoritmo di training selezionato direttamente da un elenco di algoritmi compatibili con l'ambiente caricato, fornito dalle librerie.

Bisogna poi specificare il Number of hidden units e lo Use recurrent neural network; Il primo permette di indicare il numero di unità di ciascun livello completamente connesso delle reti di Actor e Critic mentre l'altra opzione permette di selezionare la possibilità di creazione di Actor e Critic con reti neurali ricorrenti [52].

Una volta generato l'Agent nell'environment si passa a definire le impostazioni della sessione di addestramento (Figura 31Figura 31: Impostazioni per la sessione di Train). Premendo in alto il pulsante "Train" ci apparirà la seguente schermata con i relativi parametri

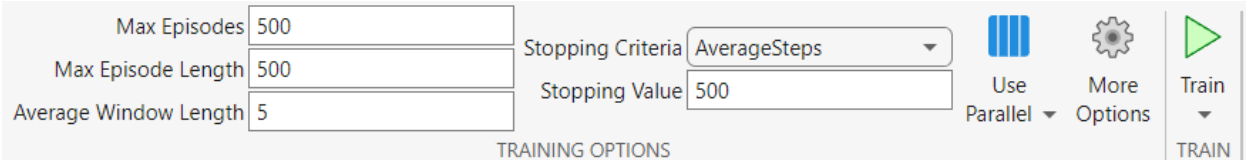

Figura 31: Impostazioni per la sessione di Train

- **Max episodes**: indica il numero massimo di episodi che dovranno essere eseguiti per la seguente sessione di addestramento. Questo numero dipenderà dalla complessità dell'ambiente e dal momento dell'addestramento in cui ci troviamo. All'inizio dell'addestramento, è opportuno eseguire un gran numero di episodi per dare la possibilità all'Agent di esplorare opportunamente l'ambiente in cui si trova. Facendo ciò l'Agent prenderà decisioni in maniera casuale per associare ad ogni

azione o comportamento una ricompensa e poter individuare le possibili scelte ottimali. Dopo aver superato questa fase, che non è detto richieda una sola sessione di simulazione, se il numero degli episodi indicati non è sufficiente, si passerà alla fase di affinamento delle scelte ottimali in cui non sarà necessario avere un elevato numero di episodi;

- **Max Episode Length:** rappresenta la lunghezza massima di ogni singolo episodio, il numero di step massimo che all'Agent può fare per arrivare a conclusione dell'episodio;

- **Average Window Length:** indica su quanti episodi verrà eseguita la media che darà il valore alla linea dell'andamento medio dell'allenamento;

- **Stopping Criteria:** indica il fattore per cui si ha la conclusione anticipata della sessione di addestramento, questo valore è importante nella prevenzione di effetti di overfitting.

Nell'app del Reinforcement Learning Designer è possibile apportare modifiche alle opzioni dell'Agent in utilizzo (Figura 32) [52]

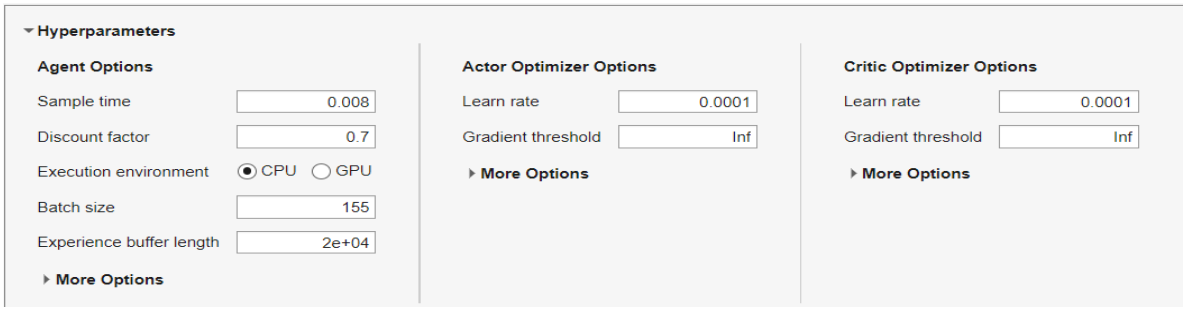

Figura 32: Impostazioni dell'Agent

Si osservano i parametri che influenzano le opzioni dell'algoritmo dell'Agent

- **Sample Time**: rappresenta il tempo di campionamento dell'Agent, periodo in cui l'Agent esegue l'Action

- **Discount Factor**: fattore di sconto è un parametro che determina quanto l'Agent valorizzi le ricompense future rispetto a quelle immediate, influisce su come le ricompense future sono considerate nel processo decisionale dell'Agent. Per valori vicino allo 0 l'Agent darebbe molta più importanza alle ricompense immediate rispetto a quelle future; questa scelta può portare a politiche in cui ci si focalizza su azioni che producono ricompense immediate piuttosto che su quelle che potrebbero portare ricompense maggiori in futuro e quindi evidenziano un comportamento sub-ottimale nel lungo periodo trascurando l'opportunità che porterebbe benefici maggiori. Nel caso invece di un Discount Factor alto con valore

vicino all'1, l'Agent considera molto le ricompense future, quasi allo stesso livello di quelle immediate, incoraggiando l'Agent a pianificare e prendere decisioni che massimizzino il valore cumulativo a lungo termine. Si avrà una politica di bilanciamento delle ricompense immediate con quelle future per ottenere il miglior risultato complessivo ma con il possibile problema della convergenza durante allenamento portando ad un aumento della complessità del problema di apprendimento [53];

- **Batch size:** indica il numero di campioni di dati che il modello elabora prima di aggiornare i suoi parametri. Questo parametro può influenzare la stabilità di apprendimento riducendo la varianza degli aggiornamenti se si inserisce un valore grande e garantire anche un'esplorazione dello spazio delle azioni più bilanciato. Per valori bassi garantisce un rinnovo dei dati più frequente e quindi una dinamica di apprendimento più rapido ma porta ad alta varianza. Per valori alti si risolve il problema della varianza stabilizzando l'apprendimento ma rallentandolo;

- **Experience Buffer Length**: indica lo spazio disponibile per memorizzare le esperienze passate dell'Agent in seguito campionate casualmente durante l'addestramento per aggiornare la Policy.

Scorrendo la schermata si arriverà alla definizione dell'andamento della Gaussian Noise e del Target Policy Smoothing, tecniche utilizzate nel reinforcement learning per il miglioramento dell'esplorazione e la stabilizzazione dell'apprendimento.

- **Gaussian Noise:** è una forma di rumore aggiunto alle azioni eseguite dal modello per migliorare l'esplorazione dell'ambiente (Figura 33). La definizione di rumore gaussiano è un rumore generato secondo una distribuzione gaussiana con media 0 e una deviazione standard. Varianza elevata di questo parametro porta ad una maggiore esplorazione alla ricerca delle soluzioni ottimali ma se eccessivo potrebbe causare instabilità impedendo l'apprendimento di una policy stabile. Nel caso di varianza bassa il metodo porta ad un affinamento e miglioramento della strategia concentrandoci sull'ottimizzazione ma rischia di far cadere l'agent in strategie subottimali se non si è esplorato abbastanza;

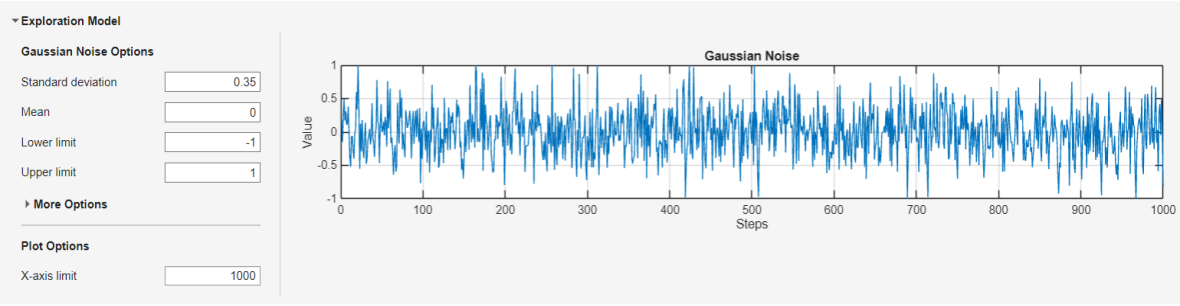

Figura 33: Gaussian Noise Options

- **Target Policy Smoothing:** tecnica utilizzata per il miglioramento della stabilità del processo di aggiornamento delle politiche (Figura 34). Per valori di varianza alta si ha riduzione del rischio di sovrastime della Q-function portando ad un modello più stabile e robusto ma potrebbe rallentare l'apprendimento poiché limita l'ampiezza delle correzioni possibili in ogni aggiornamento. Con valori bassi di varianza gli aggiornamenti saranno più rapidi e più netti del Q-function ma aumenterà il rischio di sovrastime ed instabilità del modello [54].

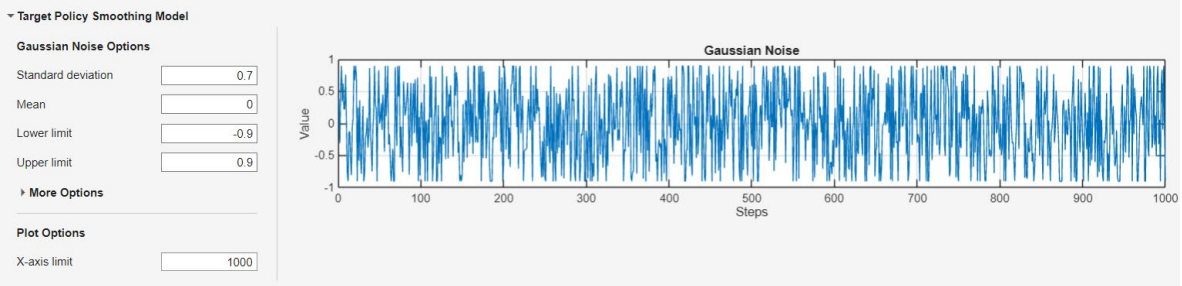

Figura 34: Target Policy Smoothing Model

Impostati i parametri dell'Agent ed i parametri relativi all'addestramento e all'esplorazione si esegue il Train. Durante la fase di addestramento il Reinforcement Learning Design mostrerà un'interfaccia grafica (Figura 35) per monitorare l'andamento dell'addestramento. Di seguito lo screen della visualizzazione output.

Figura 35: Rappresentazione grafica del Training

Come “Training Results” si visualizza un grafico a due assi: in ascissa gli “Episode Number”, in ordinata “Episode Reward”. Si osserveranno gli andamenti di tre parametri, “Episode reward”, “Average reward” ed “Episode Q0” indicati con colori diversi [55].

- **Episode reward**: indicato graficamente con la linea celeste, rappresenta il valore della reward del singolo episodio ottenuto dalla somma di tutte le reward dei singoli step dell’Episode. Tramite questo valore possiamo osservare l’esplorazione dell’Agent e se si avvicina nei singoli Episode al valore obiettivo;

- **Average reward**: indicato graficamente dal colore blu, valore che rappresenta la ricompensa media corrente. Questo fattore viene calcolato facendo la media su un numero di episodi da noi indicato in precedenza tramite l’impostazione “Average Window Length”. Lo stabilizzarsi di questo andamento potrebbe indicare l’apprendimento da parte dell’Agent di una politica efficace che massimizza le ricompense nell’ambiente; dall’altra, se dovesse stabilizzarsi per valori bassi o negativi, potrebbe indicare che l’Agent ha incontrato una politica subottimale o che non è riuscito ad imparare una strategia efficace per massimizzare le ricompense;

- **Episode Q0**: indicato graficamente dal colore giallo, nel caso di Agent con filosofia Actor-Critic indica la stima della ricompensa a lungo termine scontata all’inizio di ogni episodio, data l’osservazione iniziale dell’ambiente. Con l’avanzamento della formazione se il Critic è ben strutturato, apprenderà con successo e sarà evidenziabile dall’avvicinarsi dell’Episode Q0 al vero valore del premio scontato a lungo termine. Caso non scontato (Discount Factor = 1), l’Episode Q0 si avvicinerà alla ricompensa effettiva dell’episodio [55].

A destra dell’interfaccia, si trova il “Training Progress” (Figura 36) in cui sono riportati il numero di episodi che sono stati svolti su quelli totali della sessione di addestramento. È riportata la data e l’ora d’inizio, la durata e lo stato dell’addestramento. Questi parametri servono per comprendere le prestazioni del dispositivo con cui si sta eseguendo l’addestramento, in modo da tarare i parametri per ottenere simulazioni efficienti cercando di minimizzare i tempi di addestramento. In questo riquadro è presente anche il pulsante di “Stop Training”, necessario per mettere in pausa la simulazione oppure per concluderla prima dell’arrivo al numero massimo di episodi.

Opzione aggiuntiva è quella del “More Details” (Figura 36) utilizzato per ottenere dettagli specifici della simulazione.

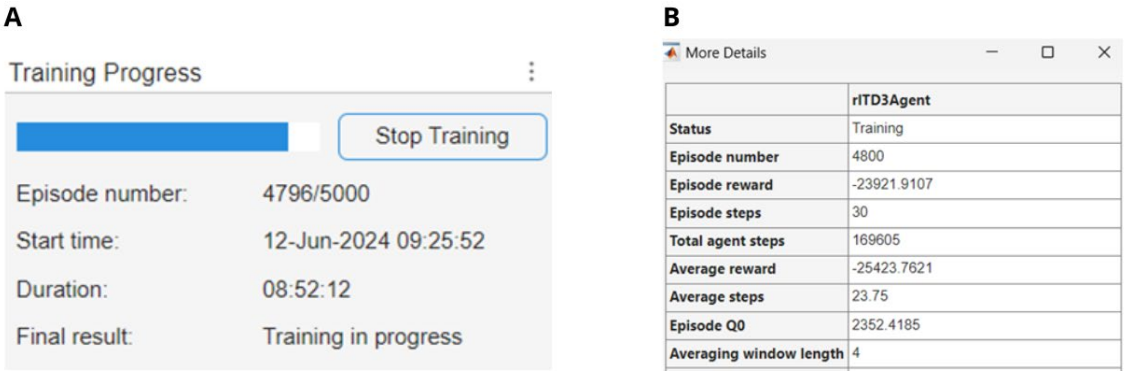

Figura 36: A) Training Progress; B) More Details

Alla conclusione della sessione di allenamento verrà proposto il grafico ottenuto e la possibilità di accettare o rifiutare la sessione eseguita (Figura 37). In caso di conferma verrà salvato nella “Agent” (Figura 37), l’Agent da noi utilizzato nella sua versione aggiornata “Trained”.

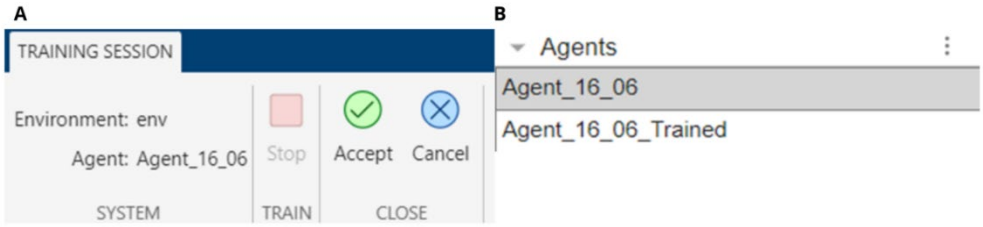

*Figura 37 **A**) Training Session **B**) Elenco Agent creati*

Nella sezione risultati sarà caricato il grafico ottenuto dall’addestramento, utile in fase di impostazioni di nuove sessioni di allenamento per la visualizzazione degli andamenti della formazione dell’Agent, dando anche la possibilità di zoomare ed accedere alle informazioni tecniche dei singoli episodi d’interesse identificati graficamente da punti (Figura 38).

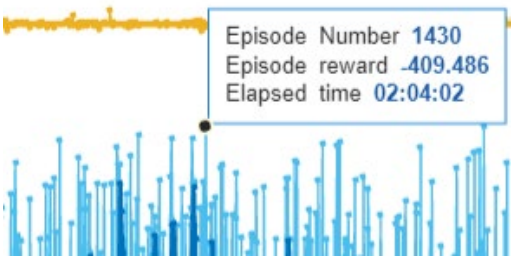

Figura 38: Informazioni sugli episodi

Per la successiva fase di allenamento, dovrà essere utilizzata la nuova versione dell’Agent, “Agent_Trained”, in modo da continuare la sua formazione. Si avrà la possibilità di variare i suoi valori caratteristici ed impostare nuovi parametri di allenamento.

Durante il Training si è riscontrato il problema dell'eccessiva durata dell'esecuzione del Train. Per le prime sessioni di addestramento l'Agent deve esplorare sufficientemente l'ambiente dinamico per poter apprendere ed individuare le possibili soluzioni ottimali, per poi approfondirle in una fase successiva del Training.

Mathworks mette a disposizione uno strumento progettato per facilitare il calcolo parallelo e distribuire le attività computazionali su processori multicore, GPU o cluster di calcolo: il Parallel Computing Toolbox.

Questo toolbox permette di sfruttare tutta la potenza di elaborazione dei desktop multicore e abilitati per GPU eseguendo applicazioni sui thread e sui processi worker, motori di calcolo di Matlab, in esecuzione locale, così da poter affrontare simulazioni ad elevato numero di episodi e quindi un'elevata mole di dati (Figura 39)[56].

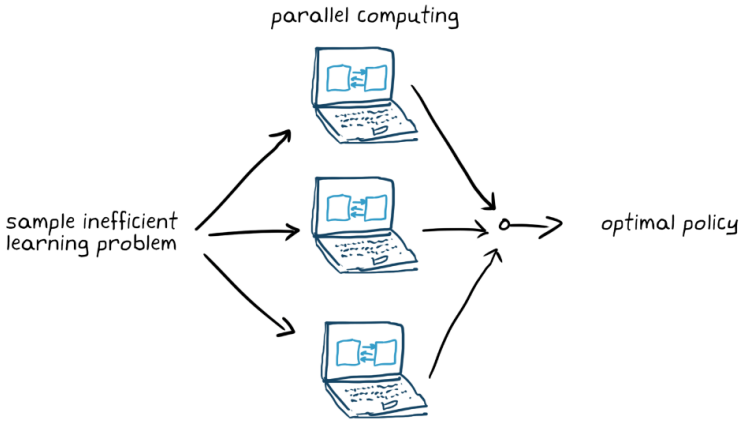

Figura 39: Parallel Computing [56]

Nell'applicazione del metodo per i casi studio si è optato per l'utilizzo del calcolo parallelo applicato tramite CPU. Il dispositivo utilizzato mette a disposizione 4 workers, passando da simulazioni di 100 episodi in 50 minuti a soli 5 minuti.

4.2 Caso Applicativo

In questo elaborato è stato applicato ed analizzato il metodo del Reinforcement Learning per la realizzazione di un pianificatore di traiettoria che segua un percorso indicato con la capacità di scegliere autonomamente la traiettoria ottimale in condizioni di presenza dell'operatore nello spazio di lavoro durante l'esecuzione del movimento.

Dall'analisi dell'obiettivo si è cercato di tradurre il comportamento desiderato del cobot in specifici criteri operativi e requisiti tecnici necessari per garantire il corretto funzionamento e l'efficace esecuzione del compito in linea con gli obiettivi prefissati.

La descrizione del metodo del Reinforcement Learning, precedentemente discussa, evidenzia la necessità di tradurre questi aspetti significativi nella funzione Reward. Questa deve infatti riflettere gli obiettivi desiderati, incentivando comportamenti che migliorino la sicurezza, l'efficienza e la precisione rispecchiando la capacità del cobot di adattarsi alla dinamicità dell'ambiente.

4.2.1 Reward

Come visto in precedenza la Reward è l'elemento su cui si basa l'Agent per comprendere la correttezza delle decisioni che prende. La Reward che viene realizzata per questo caso applicativo tiene in considerazione diversi aspetti tradotti in più componenti.

R1 – Percorso

Si vuole realizzare un pianificatore di traiettoria a cui venga indicato un percorso da seguire e che abbia autonomia sulle scelte della traiettoria per garantire la sicurezza dell'operatore in questo caso studio denominato "corpo". Se la presenza del corpo non influisce sul movimento del cobot si vuole che quest'ultimo sia motivato a seguire il percorso indicatogli: si è fornito un percorso di riferimento e non una traiettoria poiché non si vuole imporre al cobot dove essere in un determinato istante di tempo bensì lasciare a lui la scelta con obiettivo di prediligere l'ottimo nel tempo.

Per addestrare l'Agent (robot) a seguire il percorso indicatogli, bisogna impostare una funzione che dia un valore negativo quando la posizione che assume dopo l'Action è distante dal percorso ed un valore positivo quando la sua posizione risulta sul percorso. La componente della Reward per il percorso è stata realizzata utilizzando come fattore di riferimento la distanza minima del corpo dal percorso. In particolare, questa viene ricavata in ogni istante temporale analizzando la distanza tra la posizione dell'End Effector e tutti i punti appartenenti al percorso selezionando il valore minimo.

Per maggior chiarezza è riportato nella "Figura 40" una rappresentazione grafica di quello che accade in ogni istante temporale per il calcolo della distanza. È rappresentato l'End Effector "EE" in una posizione non appartenente al percorso e tenendo conto solo di alcuni punti del percorso, indicati nell'immagine con dei punti rossi. Si calcolano tutte le distanze dalla posizione dell'End Effector "EE" ai punti del percorso per poi prendere tra queste la

distanza più piccola “m” che sarà utilizzata per il calcolo della Reward in quell’istante temporale. Il processo verrà ripetuto per tutti gli istanti temporali del processo.

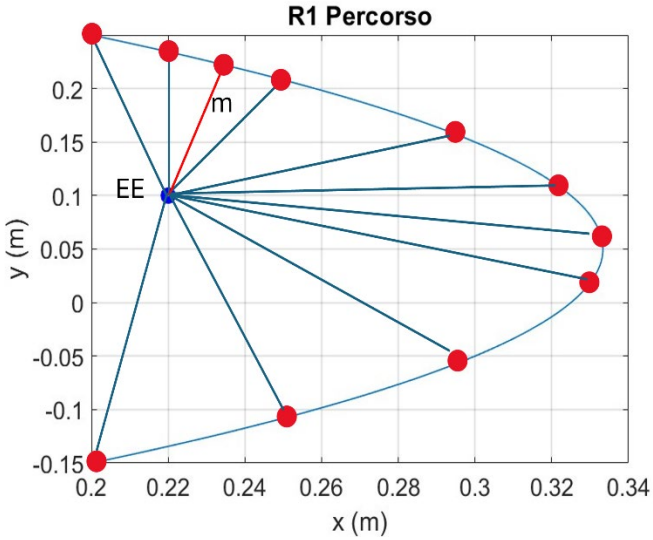

Figura 40: Calcolo della distanza minima dal percorso

È riportato l’andamento della funzione nello spazio 2D (Figura 41) generando una griglia di punti e inserendovi il percorso. Per ognuno di questi punti è riportato il valore della componente della Reward R1 osservabile tramite grafico a colori:

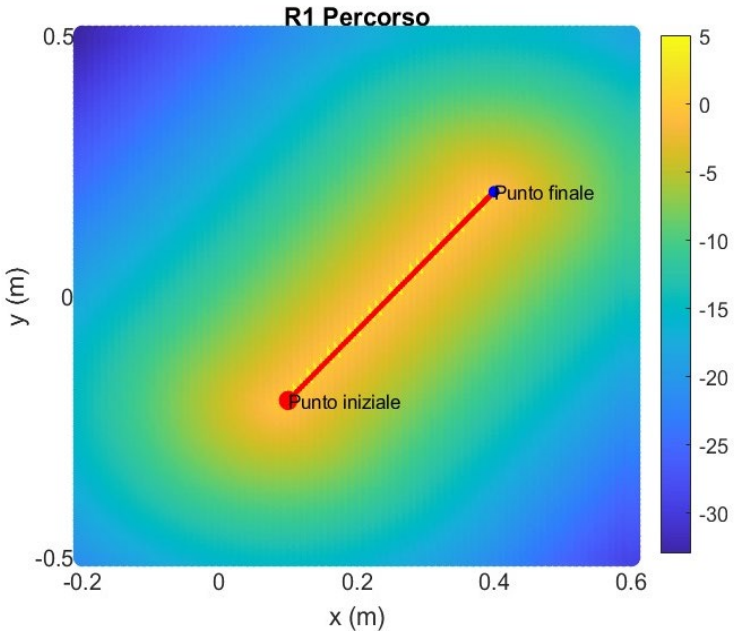

Figura 41: R1 Percorso

È riportata la funzione della reward relativa al percorso (R1)

$$(1) \qquad R1 = -k * m$$

- **m** rappresenta la minore distanza dal percorso trovata in ogni istante di tempo;

- **k** è il peso della R1, parametro da regolare per definire una scala d'importanza tra le componenti della Reward. In questo caso è stato preso k=50;

R2 – Corpo

Si deve ora gestire il comportamento del cobot in condizioni in cui la presenza del corpo influisca sulla sua traiettoria. L'obiettivo è evitare che il movimento del robot possa essere dannoso per l'operatore. Per garantire questo aspetto, si è deciso di regolare la velocità del cobot. In particolare, si è scelto di generare intorno al corpo una zona di sicurezza in cui il cobot debba rispettare una velocità di riferimento, di valore basso, in modo da non costituire una minaccia per l'operatore. Per lo sviluppo della funzione della R2 sono presi in considerazione il parametro "dist_corpo", indicatore della distanza dell'End Effector dal corpo nell'istante temporale, ed il fattore "g", rappresentante la differenza tra velocità del cobot in quell'istante e la velocità di sicurezza.

La funzione è progettata in modo che, quando il cobot si trova al di fuori dalla zona di sicurezza la funzione non risulta significativa non influendo sulla velocità del dispositivo che potrà viaggiare a velocità elevate con fattore "g" ininfluente. Invece, se si avvicina al corpo la funzione diventa importante, il valore negativo diventa significativo, rendendo la differenza di velocità tra quella del cobot e quella di sicurezza significativa. In funzione di un valore della distanza, definito molto prossimo al corpo ed interna alla zona di sicurezza, si avrà una zona di non percorrenza, impostata per evitare la collisione tra dispositivo ed operatore.

Nella "Figura 42" sono riportate la zona di sicurezza e la zona di non percorrenza sopra descritte. È rappresentato un esempio pratico del comportamento che si cerca di ottenere nella condizione di presenza dell'operatore nello spazio di lavoro. Dalla figura si osserva il dispositivo che viaggia ad una velocità sostenuta di v_ur=1 m/s. Una volta entrato nella zona di sicurezza, in cui è stato impostato il limite di velocità pari a v_rif=0.3 m/s, dovrà rispettare l'indicazione sulla velocità e rallentare. In prossimità del corpo il cobot dovrà eseguire un cambio di direzione per evitare la zona di non percorrenza e quindi la collisione con l'operatore. Una volta uscito dalla zona di sicurezza si prevede che il cobot aumenterà la sua velocità, andando ad un valore maggiore rispetto a quella pre-regolazione per recuperare il tempo perso

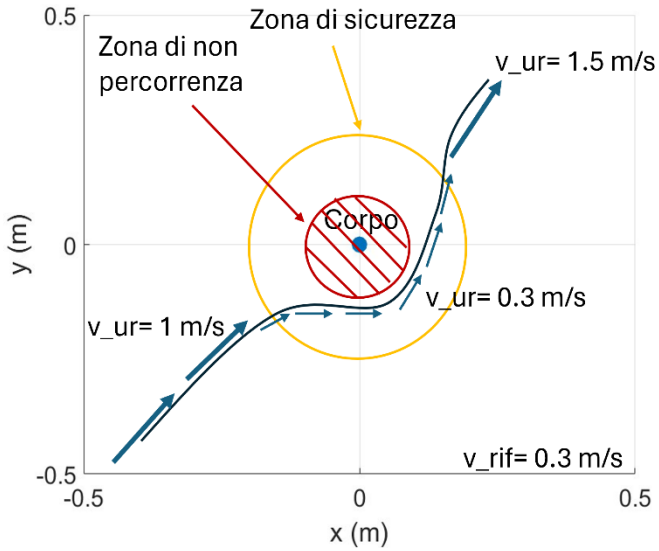

Figura 42: Comportamento atteso per la gestione dell'operatore nello spazio di lavoro

Dalle precedenti affermazioni, si progetta la funzione R2 in modo da essere principalmente regolata dal fattore "dist_corpo". Considerando la penalità dovuta al fattore "g" delle velocità, il parametro "dist_corpo" la renderà trascurabile se si è lontani dal corpo e importante all'avvicinarsi al corpo, fino ad arrivare alla zona di "non percorrenza" con penalità elevata per far desistere il robot a percorrere quella zona (Figura 43).

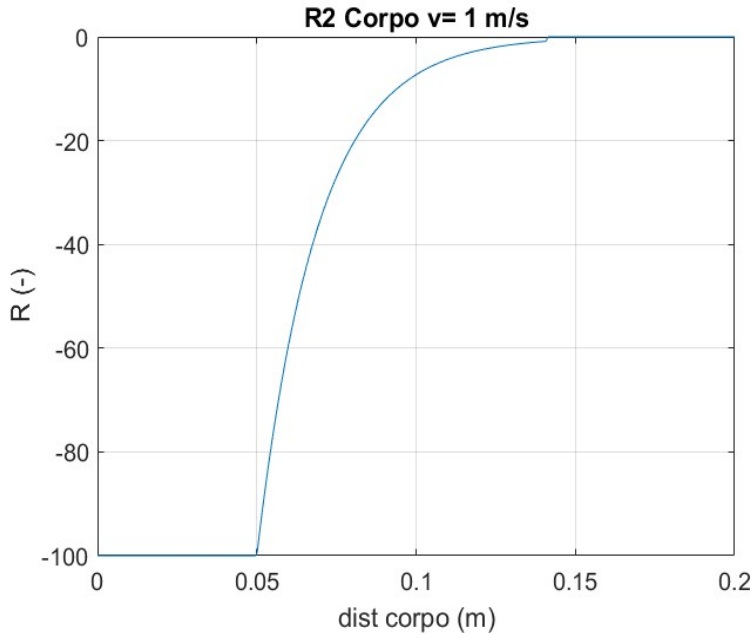

Figura 43: Rappresentazione grafica della R2 penalità relativa al corpo

La distanza nella quale si entra nella zona di sicurezza e quindi il valore di "dist_corpo" per il quale diventa importante la penalità, non è fisso ma varierà in funzione del fattore "g". Questo parametro influenzerà la zona di sicurezza rendendola più grande se il dispositivo

viaggia a velocità elevate, in modo da iniziare la regolazione della velocità ad una distanza maggiore per evitare variazioni improvvise, o più piccola se la velocità è prossima a quella di sicurezza, necessitando di meno spazio e meno tempo per raggiungere la velocità indicata. Verrà calcolato il parametro x_var indice del raggio della zona di sicurezza (Figura 44).

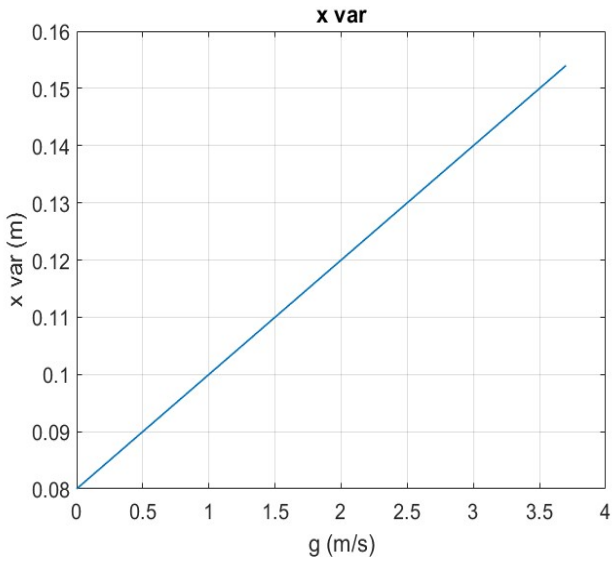

Figura 44: Variazione della coordinata dell'area di sicurezza

Ipotizzando il dispositivo all'interno della zona di sicurezza e con velocità di valore interno ai valori di tolleranza della velocità di sicurezza, la penalità verrà annullata e per comunicare al cobot che il comportamento è quello corretto, verrà fornito un valore positivo di ricompensa. Tutto ciò è vero fin quando si è al di fuori della zona di non percorrenza dove la penalità rimarrà fissa. Di seguito l'andamento della R2 al variare del fattore "dist_corpo" (Figura 45):

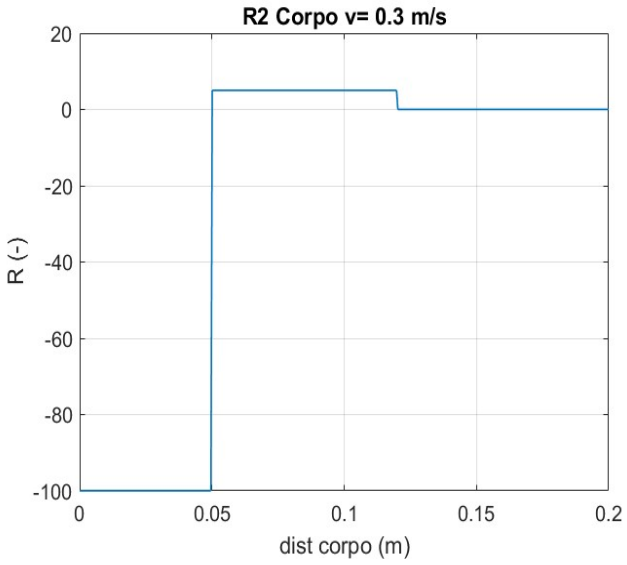

Figura 45: Andamento reward per v=0.3 m/s

Nel caso di percorrenza nella zona di sicurezza e velocità maggiore di quella di riferimento, si avrà una penalità crescente al diminuire del fattore indicatore della distanza dal corpo. Sono riportati i grafici dell'andamento della Reward R2 al variare della velocità dell'UR.

Velocità di sicurezza v_rif= 0.3 m/s e v_ur=0.36 m/s (Figura 46):

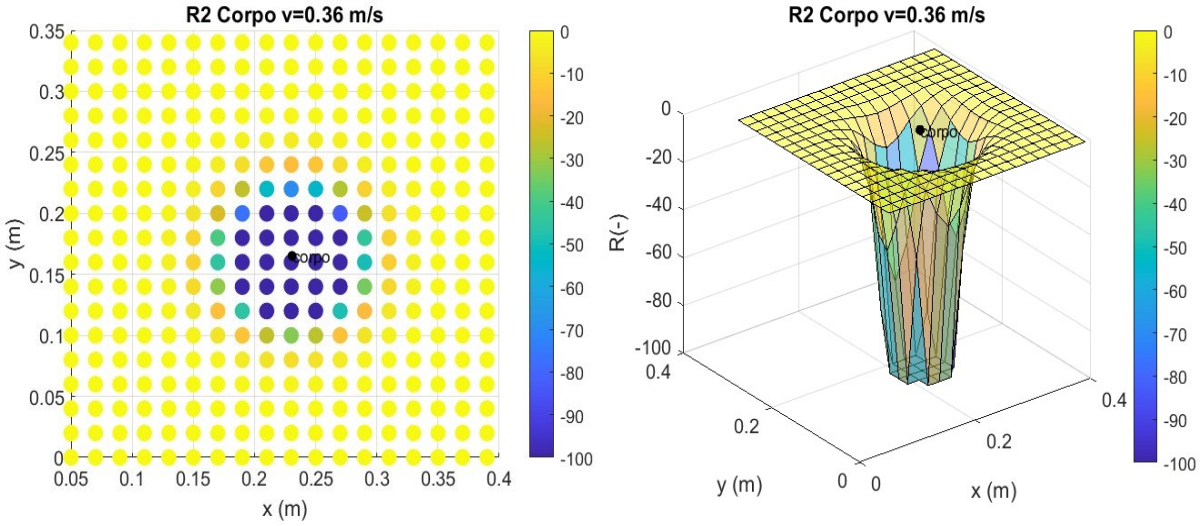

Figura 46: Rappresentazione su grafico 2D (sinistra) e 3D (destra) dell'andamento della Reward con v=0.36 m/s

Velocità di sicurezza v_rif= 0.3 m/s e v_ur=1 m/s (Figura 47):

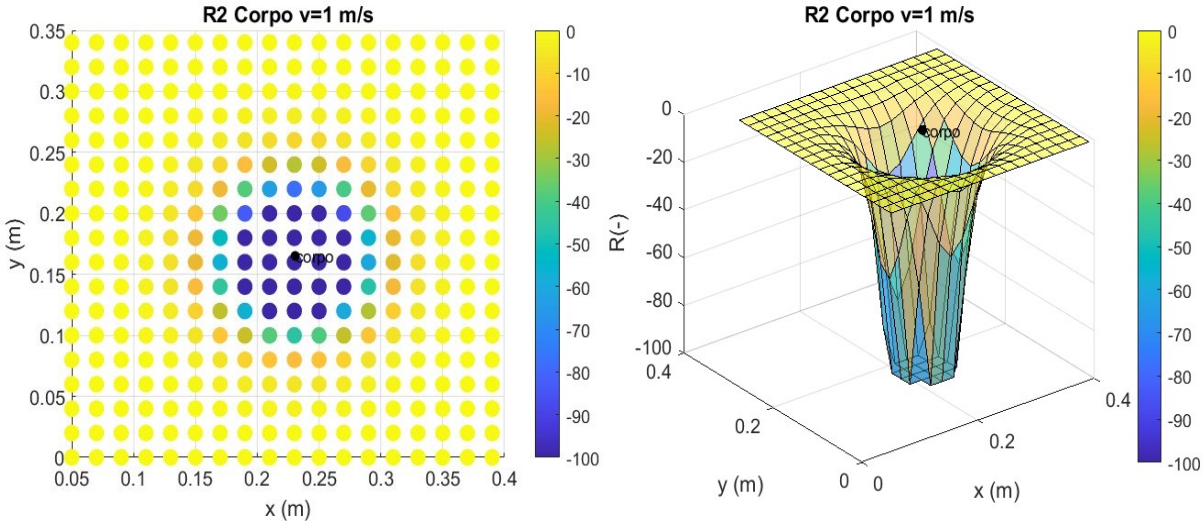

Figura 47: Rappresentazione su grafico 2D (sinistra) e 3D (destra) dell'andamento della Reward con v=1 m/s

Velocità di sicurezza v_rif= 0.3 m/s e v_ur=2 m/s (Figura 48):

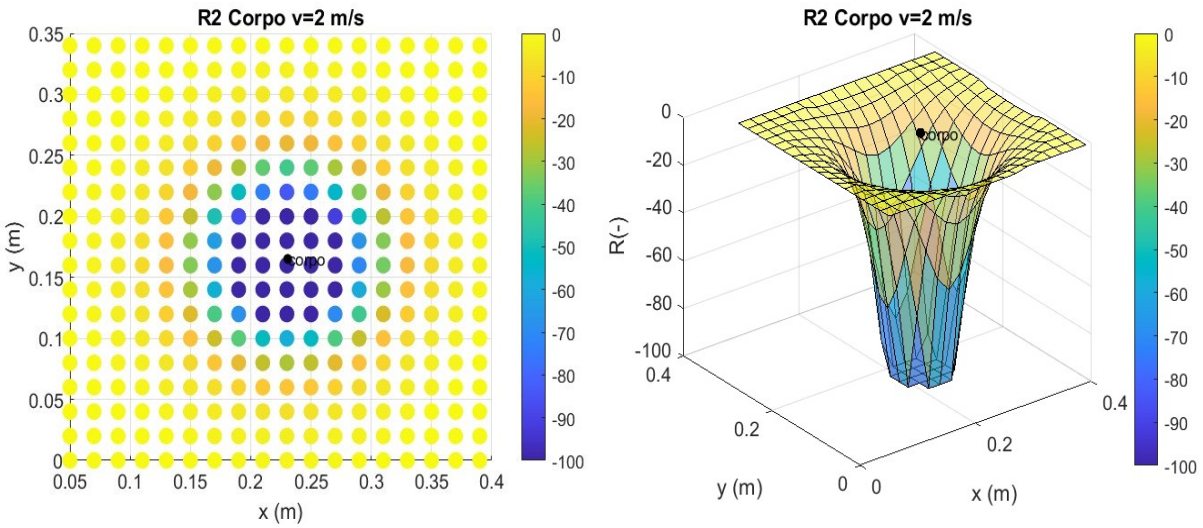

Figura 48: Rappresentazione su grafico 2D (sinistra) e 3D (destra) dell'andamento della Reward con v=2m/s

Velocità di sicurezza v_rif= 0.3 m/s e v_ur=3 m/s (Figura 49):

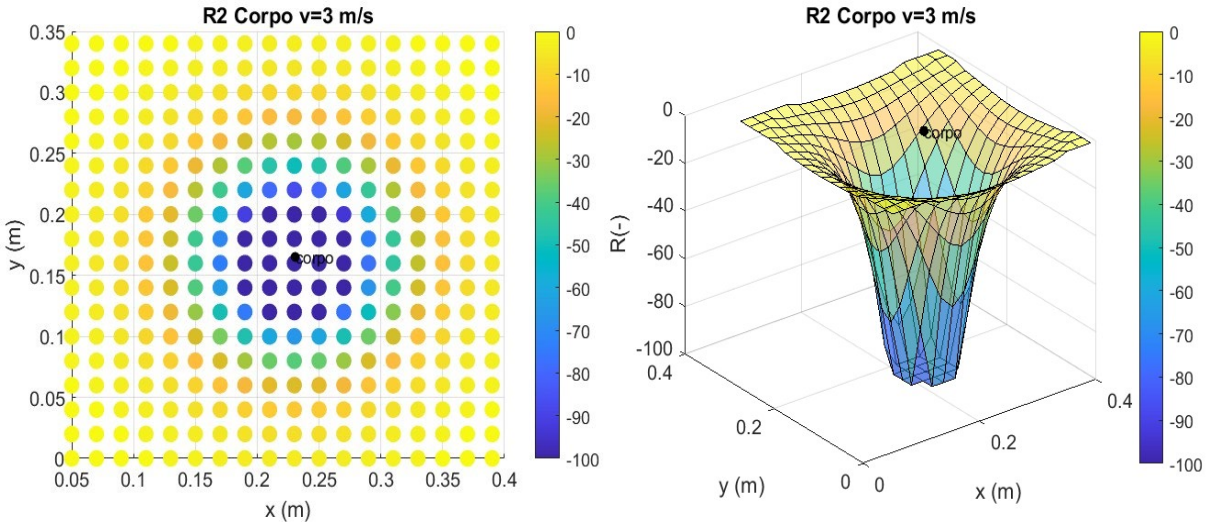

Figura 49: Rappresentazione su grafico 2D (sinistra) e 3D (destra) dell'andamento della Reward con v=3 m/s

È stato considerato anche il caso in cui il cobot riduca troppo la sua velocità, rimanendo al di sotto della velocità di sicurezza, assumendo un comportamento inefficiente per l'esecuzione della traiettoria. La funzione è stata costruita in modo da fornire una penalità

anche in questa circostanza. Per dimostrazione, di seguito è riportato un caso applicativo con velocità di sicurezza di 0.3 m/s e velocità dell'UR di v=0.1 m/s (Figura 50).

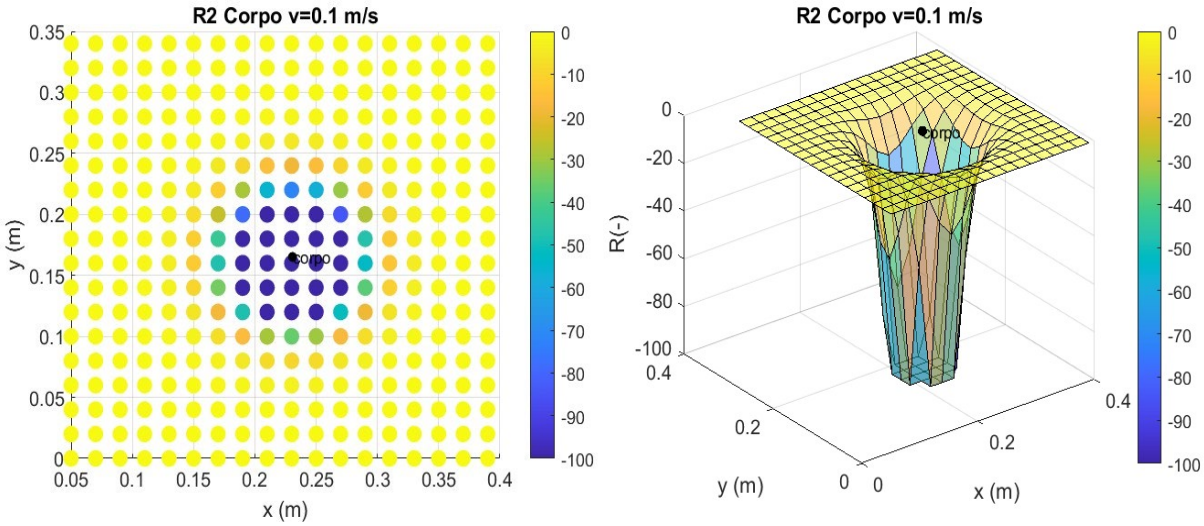

Figura 50: Rappresentazione su grafico 2D (sinistra) e 3D (destra) dell'andamento della Reward con v=0.1 m/s

Di seguito è riportata la funzione della R2 con relativi parametri

$$(2) \qquad g = abs(0.3 - vel.sic)$$

vel_sic= è la velocità che deve essere rispettata nella zona di sicurezza

$$(3) \qquad x_{var} = 0.1 + (g - 1) \cdot (0.02)$$

$$(4) \qquad b = \frac{\log(10)}{(0.05 - x_{var})} \quad (g > 0.05) \quad b = 0 \quad (g \leq 0.05)$$

$$(5) \qquad a = \frac{1}{\left(e^{(b \cdot x_{var})}\right)} \quad (g > 0.05) \quad a = \ -w \quad (g \leq 0.05)$$

I fattori a e b sono i componenti necessari per ottenere sempre una funzione ad andamento parabolico tra x=0.05 dove si avrà valore y=10, ed il valore variabile x_var in cui si otterrà sempre valore y=1 per gestire le differenti velocità del cobot, estendendo o diminuendo la zona di sicurezza. Inoltre, si osserva in questo fattore che, se è rispettato il limite di velocità della zona di sicurezza il cobot riceverà una ricompensa w, nel nostro caso w=0.5 che andrà ad essere moltiplicato per il fattore peso.

$$(6) \qquad R2 = -k \cdot \left(a \cdot e^{(b \cdot dist\ corpo)}\right) (dist\ corpo > 0.05)$$

$$(7) \qquad R2 = -100\ (dist\ corpo \leq 0.05)$$

Il fattore k è il valore relativo al peso della penalità che verrà data al cobot. I grafici riportati sono stati eseguiti considerando il parametro k = 10.

R3 – Velocità

Da scheda tecnica dell'Universal Robotics si comprende che il dispositivo UR3 possa viaggiare tra velocità comprese tra gli 0 ed i 5 m/s, con precisazione sulla velocità di utilizzo tipica pari a 1 m/s. Si è voluto considerare questo aspetto costruendo una funzione R3, in modo da dare una penalità in caso di velocità eccessiva da parte del dispositivo, con valori negativi crescenti all'aumentare del parametro della velocità dell'End Effector. La penalità andrà a saturare in corrispondenza della velocità di 5 m/s per addestrare il robot a non operare in prossimità di questo valore ed evitare il superamento di un limite fisico del dispositivo. Di seguito si osserva la funzione costruita per R3:

$$R3 = -q \cdot (vel_{ur} - 1) \cdot \left(\frac{sign(vel_{ur}-1)}{10} + \frac{1}{10}\right) \quad (8)$$

q rappresenta il peso che viene deciso in base alla scala d'importanza degli aspetti considerati nella Reward.

È riportato il grafico della R3 con i valori scelti per verificare l'andamento della componente (Figura 51):

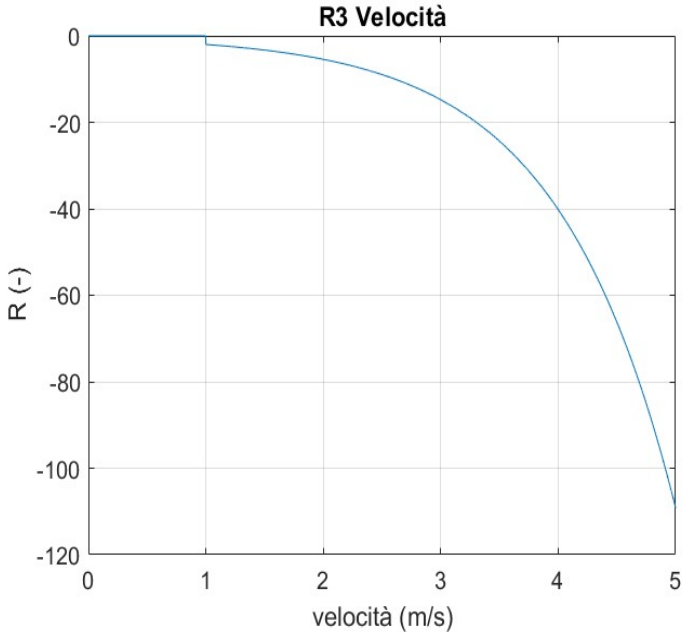

Figura 51: Andamento della reward della velocità R3

In questo caso si è impostata la funzione in modo da avere Reward nulla fino al valore della velocità 1 m/s, velocità tipica indicata da catalogo.

R4 – Collisioni

Per garantire l'integrità strutturale del robot e la continuità operativa è importante che il robot eviti di eseguire movimenti o che assuma configurazioni che causano auto-collisioni, cioè collisioni tra i propri link. Inoltre, è essenziale prevenire collisioni con oggetti nell'area di lavoro per mantenere la precisione e la sicurezza del movimento. La definizione di un'adeguata componente di Reward (R4) che penalizzi le collisioni è cruciale per l'addestramento del robot, assicurando comportamenti ottimali e sicuri. Questa viene sviluppata tramite funzione che, in caso di collisione, concluderà l'episodio ed assegnerà un valore negativo alle azioni dell'Agent istruendolo ad evitare tali configurazioni. Invece verrà premiato se al raggiungimento del punto target non sarà entrato in collisione nemmeno una volta. Nel caso applicativo viene preso in considerazione come oggetto presente nell'ambiente il pavimento, modellato tramite comando "CollisionBox" (Figura 52):

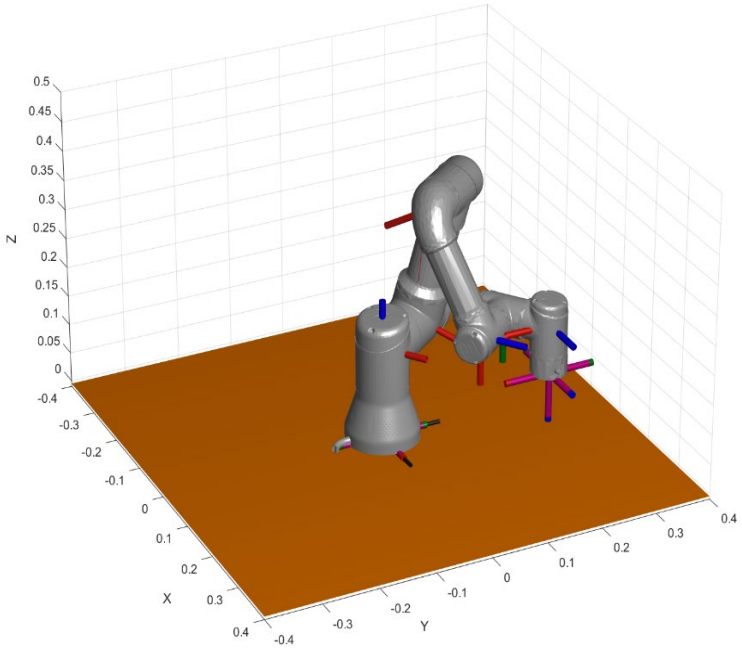

Figura 52: Visualizzazione del cobot + pavimento

La verifica di avvenuta collisione è stata eseguita tramite comando "checkCollision", appartenente al Robotics System Toolbox, che sarà utilizzato per analizzare in ogni step di campionamento temporale se la configurazione assunta porta il robot ad essere in auto-collisione o in collisione con il pavimento.

R5 - Tempo

Fattore importante per il caso applicativo è il tempo. Si vuole ottenere l'esecuzione del movimento dal punto iniziale al punto finale in modo efficiente dal punto di vista temporale, rispettando la sicurezza dell'operatore. Per evitare che il cobot rimanga fermo in un punto a Reward positiva, è stata introdotta una componente che tenga conto dello scorrere del

tempo, incentivando il dispositivo a cercare una soluzione che ottimizzi la traiettoria temporalmente in qualsiasi condizione di applicazione. A tal scopo, è stata quindi progettata la componente della Reward R5, corrispondente ad un valore costante negativo che l'Agent riceverà ad ogni step temporale, incentivandolo quindi a raggiungere il punto finale nel minor tempo possibile per così concludere l'episodio.

R6 - Avanzamento

L'obiettivo del caso applicativo è quello di ricercare soluzioni efficienti ed ottimali in situazioni generali di movimento da parte del cobot. Si vuole quindi premiare le soluzioni che avvicinano il cobot al punto finale anche se non dovesse seguire il percorso indicato. Viene definita una componente della Reward che tenga conto del movimento del dispositivo ed abbia come parametro di riferimento la sua distanza dal punto finale. Questa componente fornirà una ricompensa, rappresentata da valore positivo, se la decisione presa dal cobot lo porterà ad una distanza dal punto finale minore rispetto all'intervallo temporale precedente. Sarà invece nulla nel caso in cui il movimento non corrisponde ad un avvicinamento al punto finale, questo perché, non è detto che il percorso datogli come indicazione non preveda degli spostamenti con allontanamento dal punto target.

Nella "Figura 53" rappresentata un esempio pratico di come funziona questa componente di Reward. È riportata la posizione dell'End Effector negli istanti temporali successivi "i", "i+1" e "i+2". Nell'istante "i+1" l'Agent si porta ad una distanza dal punto finale minore rispetto a quella dell'istante "i" ottenendo così una Reward positiva. Osservando l'istante temporale "i+2" l'Agent avrà preso una decisione che porterà un allontanamento dal punto finale e non sarà fornita una Reward nulla.

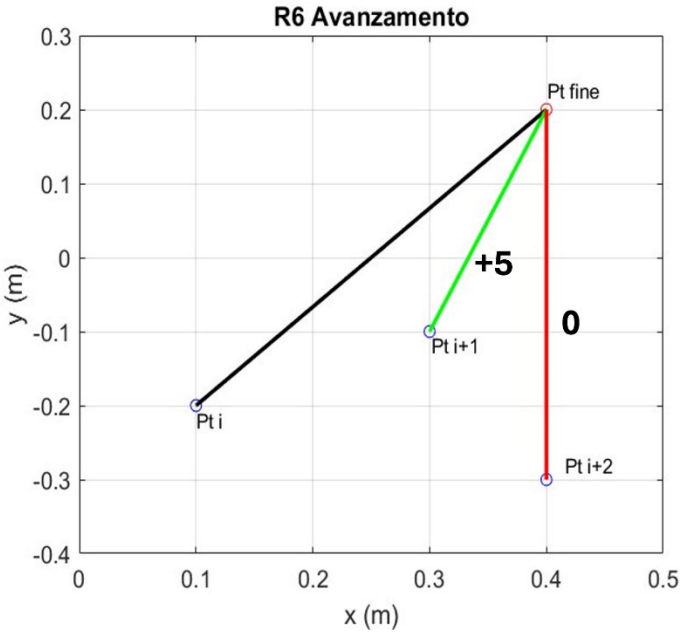

Figura 53: Reward R6 Avanzamento

R7 – Punto finale

È aggiunta una componente relativa al raggiungimento del punto finale della traiettoria per premiare il cobot per essere arrivato al punto target. È stato definito come valore costante positivo pari a +100 che viene assegnato se, tramite le sue decisioni, l'Agent si ritrova dentro una sfera di tolleranza, definito pari al raggio di 0.005 metri, costruita attorno al punto finale definendo un valore di distanza accettabile.

4.2.2 Descrizione Environment

Per la creazione dell'Environment bisogna definire le Observations e l'Action, per poi passare alla costruzione della Reset function. Per la struttura dell'Environment si è optato per l'utilizzo di un modello Simulink che, grazie alla sua struttura a blocchi, permette di realizzare un percorso logico delle operazioni e dei passaggi che devono essere eseguiti ad ogni step temporale.

Per il caso applicativo, le Observations sono state raggruppate in un vettore 19x1. Sono riportati tutti i canali necessari in prospettiva dell'ampliamento del problema fino al caso complesso, obiettivo dell'elaborato. Queste informazioni sono necessarie all'Agent per la scelta dell'Action ed il raggiungimento del target

- **Configurazione giunti (6)**, per fornire informazioni dirette sulle posizioni attuali di ciascun giunto.

- **Velocità del TCP (6),** per avere un'informazione diretta di come il TCP si sta muovendo nello spazio;

- **Coordinate del punto finale (3)**, per conoscere il punto target da raggiungere;

- **Coordinate dell'ostacolo (3),** per conoscere la posizione corrente dell'altro oggetto presente nell'ambiente di cui deve tenere in considerazione l'Agent nel suo percorso;

- **Step (1),** utilizzato per quantificare il tempo che scorre, tenendo in considerazione che ogni unità corrisponde al valore del Sample Time, valore del tempo di campionamento.

Si è deciso di progettare l'Agent in modo che prenda decisioni, l'Action, relative alle velocità dei giunti del robot. È stata definita con vettore 6x1, dove ciascuna dimensione rappresenta la velocità angolare dei singoli giunti del robot. Sono stati definiti i limiti inferiori e superiori di queste velocità secondo specifiche tecniche dell'UR3, in cui per i primi tre giunti si indica una velocità compresa tra -π e + π rad/s, gli altri tre una velocità compresa tra –2π e +2 π rad/s.

Nella “Figura 54” sono rappresentati i due sistemi di riferimento rispetto cui sono stati eseguiti i calcoli dei casi applicativi. Sono stati considerati il sistema di riferimento dell’ambiente “world” ed il sistema di riferimento del “wrist_3_link”:

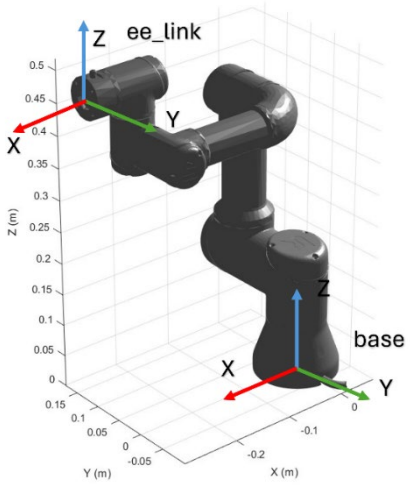

Figura 54:Sistemi di riferimento utilizzati per la pianificazione della traiettoria di questo capitolo

4.2.3 Modello Simulink Caso Applicativo

Nella “Figura 55” è mostrato il modello Simulink rappresentante l’Environment costruito per l’addestramento dell’Agent. Si utilizza il blocco “RL Agent” (Figura 56: RL Agent) fornito dal “Reinforcement Learning Toolbox” [57].

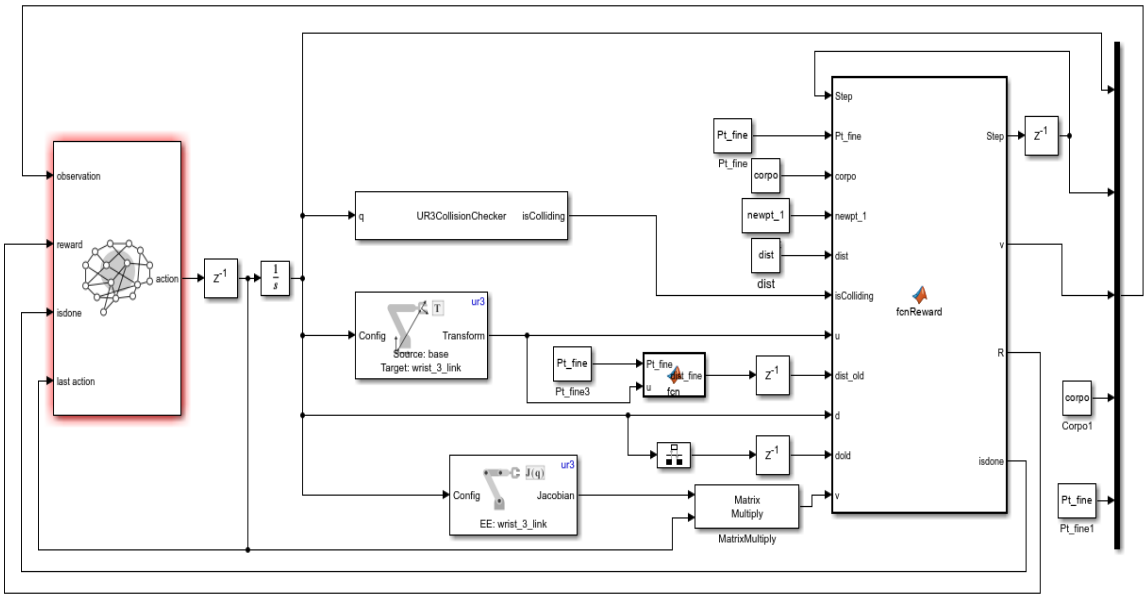

Figura 55: Modello Simulink caso applicativo complesso

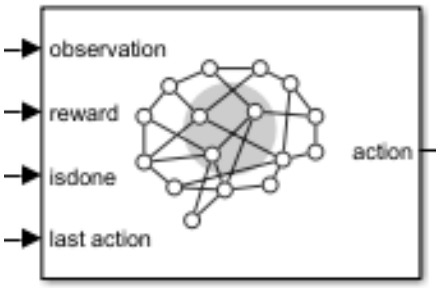

Figura 56: RL Agent [57]

È utilizzato per simulare e addestrare un Agent. Ha le porte di input per ricevere le observations, il valore della Reward, il comando "isdone" per concludere la simulazione e "last action" per ottenere informazioni sulla precedente decisione presa dall'Agent. Queste sono informazioni necessarie all'Agent per elaborare una nuova decisioneper il successivo step temporale. Come detto in precedenza, l'Action nel caso applicativo è un vettore di sei elementi relativi alle velocità angolari dei giunti del cobot "qd". Proseguendo con la descrizione del modello Simulink, si osserva l'informazione Action, qd, che viene integrata per ottenere informazioni sulla configurazione giunti attuale "q". All'interno del blocco d'integrazione, indicato con simbolo "1/s" si ha l'informazione sulla configurazione iniziale dei giunti all'inizio di ogni episodio "Initial_Condition", definita nella Reset Function, descritta successivamente.

La configurazione attuale dei giunti, oltre ad essere un parametro di input per le Observations e per la Reward function, sarà l'input di altri blocchi per l'ottenimento di parametri necessari allo studio. Tramite il blocco "GetTrasform" si passerà dallo spazio giunti a quello cartesiano ottenendo una matrice 4x4 della posa del end effector, indicata con il parametro "u", input del blocco della Reward function. La "u" viene utilizzata in un ulteriore blocco per il calcolo della distanza con il punto finale, necessaria allo sviluppo di una delle componenti della Reward.

Il blocco della Jacobian genera una matrice Jacobiana 6x6 basandosi sulla configurazione attuale dei giunti "q". Questa matrice viene pre-moltiplicata per il vettore delle velocità dei giunti "qd" per ottenere un vettore di sei elementi relativi alle velocità nello spazio cartesiano dell'End Effector "v". Il parametro "v" sarà un input della "Reward function" e delle Observations.

Il parametro "q" passa per il blocco "Matlab System" [58]. Il modello simula il blocco utilizzando il motore di esecuzione di Matlab. Il codice costruito su Matlab permette, tramite comando "checkCollision", di verificare in ogni step, se la configurazione assunta "q" dal cobot lo porta in auto-collisione o in collisione con il pavimento. L'output di questo parametro sarà un valore logico che verrà elaborato nella Reward Function.

I blocchi "Pt_fine", "newpt_1", "corpo", "dist", sono valori calcolati tramite Reset function, tra cui anche "Initial Condition" non visibile. "Pt_fine" raccoglie le coordinate relative al

punto finale del percorso, “newpt_1” i punti del percorso, “corpo” le coordinate relative alla posizione dell’operatore e “dist” indica la distanza tra punto di partenza e quello finale.

4.2.4 Reset Function

Ora bisogna definire la Reset Function necessaria per riportare l’ambiente a uno stato iniziale predefinito, garantendo che ogni episodio di addestramento inizi in modo uniforme e coerente. Si individuano perciò le variabili necessarie da introdurre all’inizio di ogni episodio per ottenere la corretta formazione del robot.

All’interno della Reset Function viene sviluppato il calcolo per la definizione del percorso tra il punto di partenza ed il punto target. La generazione del percorso avviene seguendo una funzione matematica che descrive la forma da ottenere nello spazio. Il percorso da seguire è stato creato in modo da mantenere le stesse proporzioni di forma per qualsiasi punto finale ed iniziale. Questo permette di insegnare all’ Agent non un percorso specifico ma una forma da seguire per muoversi da un punto iniziale ad uno finale.

Come primo passo per la costruzione del percorso si genera una serie di punti lungo l’asse x normalizzato, creando un intervallo equispaziato tra 0 e 1. Questi valori sono poi utilizzati in una funzione geometrica per ottenere le coordinate z che definiranno la forma del percorso lungo l’intervallo normalizzato:

$$z_{norm} = \mathrm{a} * \mathrm{x}_{\mathrm{norm}} - (\mathrm{x}_{\mathrm{norm}} - 0.8)^2 + 0.8^2 - \mathrm{b} * \mathrm{x}_{\mathrm{norm}}^3 \tag{9}$$

Nella “Figura 57” sono diagrammati i punti ottenuti a mostrare il percorso normalizzato:

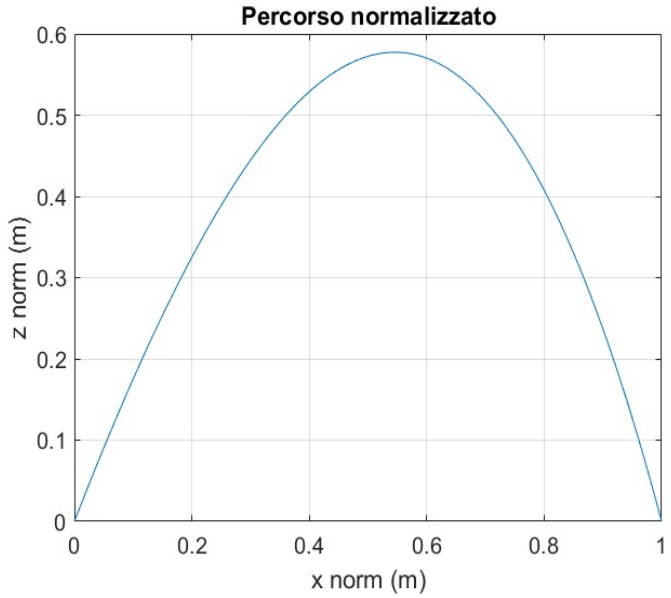

Figura 57: Percorso Normalizzato

Si calcola la distanza euclidea totale tra punto iniziale e finale e la si usa per moltiplicare le coordinate x e z, ottenendo la funzione scalata, rappresentata nella “Figura 58” in modo da coprire la distanza totale tra i due punti estremi del percorso:

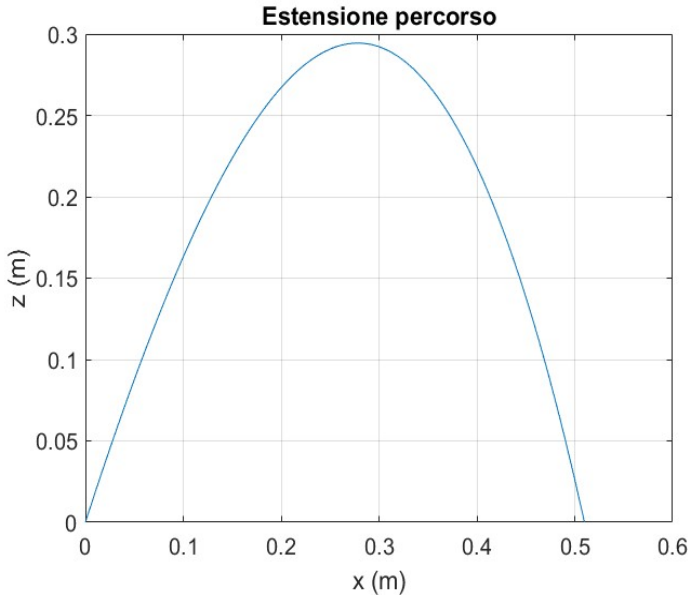

Figura 58: Estensione del percorso

Devono essere eseguite delle opportune rotazioni per posizionare il percorso nello spazio 3D. Considerando il punto iniziale e finale si effettua la rotazione lungo l’asse y per l’inclinazione ottenuta utilizzando la tangente inversa del rapporto tra la differenza di z e la distanza nel piano xy, inclinando il percorso in salita o in discesa come mostrato in “Figura 59”

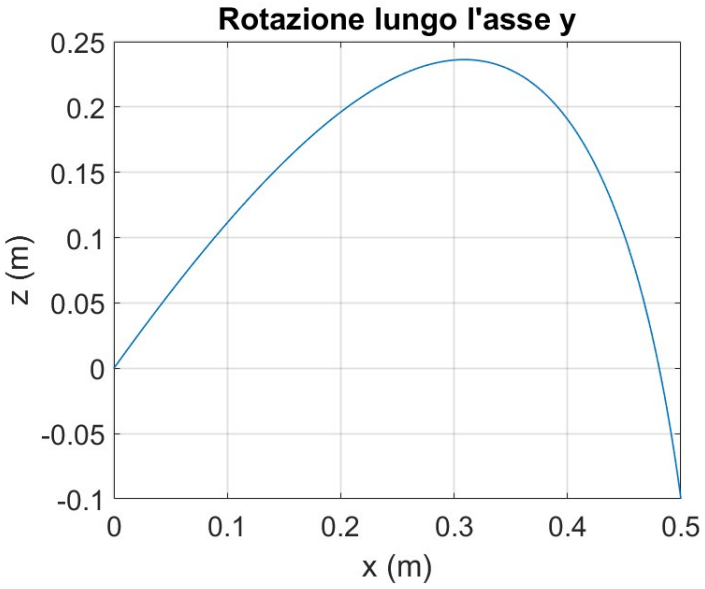

Figura 59: Rotazione del percorso lungo l'asse y

Dopodiché, si effettua un'ulteriore rotazione lungo l'asse z, utilizzando l'angolo ottenuto dalla tangente inversa delle differenze tra le coordinate y e x, ottenendo la rotazione del percorso verso sinistra o destra come mostrato nella "Figura 60":

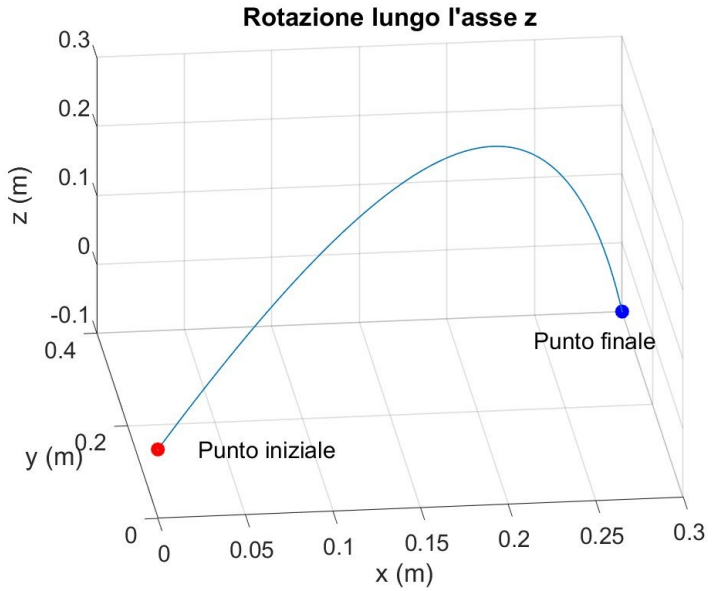

Figura 60: Rotazione del percorso lungo l'asse z

Da queste operazioni di rotazione si otterrà il percorso correttamente orientato nello spazio tridimensionale. Come ultimo passaggio si sommano le coordinate iniziali di x, y e z a tutti i punti del percorso, spostandolo alla posizione corretta come rappresentato nella "Figura 61":

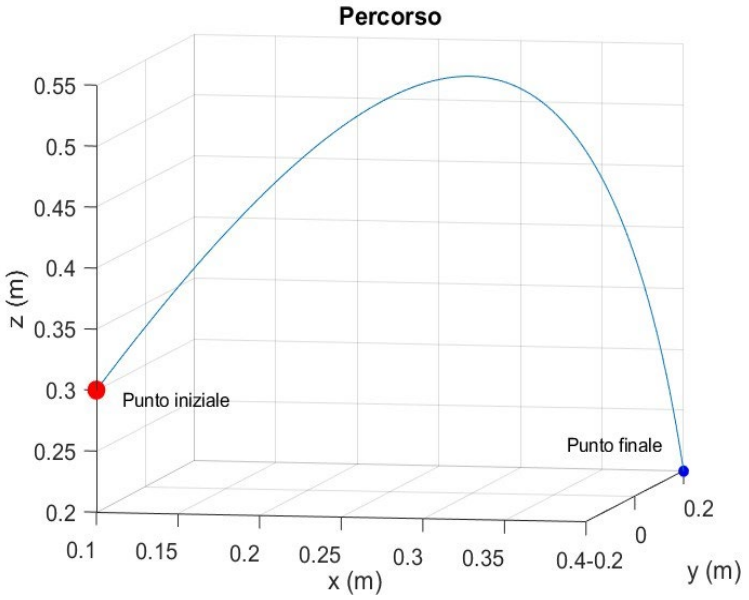

Figura 61: Percorso 3D

Il percorso generato si adatterà alle diverse coordinate del punto iniziale e finale mantenendo la forma della funzione matematica. Lo scopo dello studio e

dell'addestramento è perciò quello di ottenere un cobot versatile, in grado di affrontare qualsiasi tipo di situazione nello spazio.

Per considerare situazioni ancora più generali, nella Reset Function il punto inziale e finale vengono generati in modo casuale in una semisfera di lavoro, ottenendo per ogni episodio un percorso differente. All'interno dello stesso spazio verrà generato il corpo statico indicatore della posizione dell'operatore. Tramite il punto iniziale verrà poi eseguita la cinematica inversa per la determinazione della configurazione iniziale che dovrà assumere il cobot. Questa configurazione è stata progettata in modo da ottenere l'UR3 con End Effector che ha la flangia in direzione del punto finale (Figura 62).

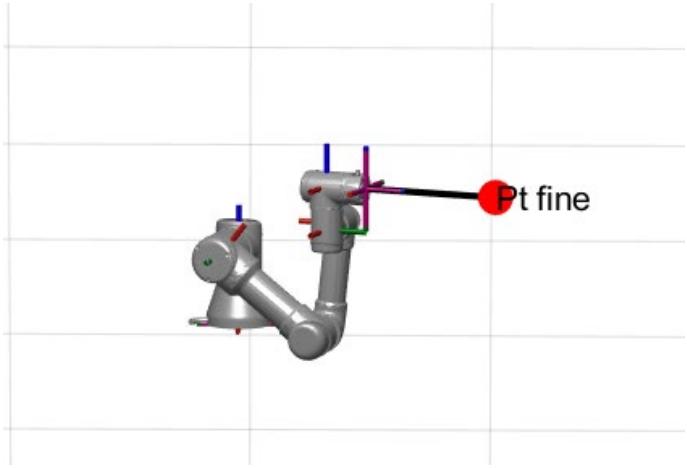

Figura 62: Visualizzazione della configurazione dell'UR3

All'interno di questa funzione sono stati inseriti dei controlli per verificare la correttezza dei punti del percorso generati casualmente. Il primo controllo viene effettuato sul punto iniziale, in particolare sulla configurazione di partenza, ottenuta tramite cinematica inversa. Questa viene verifica tramite comando "checkCollision" per controllare che non sia in auto-collisione o in collisione con il pavimento. In caso sia presente collisione si effettua il ricalcolo del punto inziale e la verifica della collisione fin quando non risulterà una configurazione senza compenetrazioni. Si passa poi al successivo controllo relativo ai punti del percorso in cui si effettua un'analisi per verificare che tutti i punti del percorso appartengano allo spazio di lavoro del robot, quindi, che non ci siano punti nell'area al di sopra della base, la quale non appartiene allo spazio di lavoro del robot. L'area di non percorrenza è stata definita considerando la distanza dall'origine inferiore a 0.08 metri dal punto di origine (0,0,0). Se un punto del percorso risulta non rispettare questo vincolo, si effettua un ricalcolo del punto iniziale e finale, processo ripetuto fino all'ottenimento di punti al di fuori della zona di non percorrenza.

4.2.5 Scelta dell'Agent

Altra fase importante per impostare l'ambiente in cui deve essere creato è l'algoritmo di training selezionato direttamente da un elenco di algoritmi compatibili con l'ambiente caricato [59].

L'Agent, in questo caso studio, è stato creato tramite modello predefinito fornito da app "Reinforcement Learning Toolbox"[60]. Tra le possibili opzioni si è optato per l'utilizzo dell'algoritmo TD3, preferito agli altri per la sua capacità di impedire la sovrastima della funzione valore e garantire la stabilità dell'apprendimento grazie alla struttura del suo algoritmo. Il TD3 (Twin Delayed Deep Deterministic Policy Gradient) è un algoritmo avanzato, progettato per problemi con spazi di azione continui ed appartenente alla famiglia dei metodi Actor-Critic, senza modello, online e off-policy [61]. Le componenti principali sono:

- **Actor**: rete neurale che mappa gli stati dell'ambiente alle azioni. Determina quale azione compiere dato uno stato attuale, approssimando una politica deterministica;

- **Critic**: due reti neurali indipendenti che stimano la ricompensa cumulativa attesa per una data coppia stato-azione.

La particolarità del Critic, costituito da due reti neurali, contraddistingue l'algoritmo TD3, infatti, con l'utilizzo di due reti neurali per il Critic si riduce la sovrastima dei valori Q prendendo il minimo dei due valori stimati durante gli aggiornamenti della policy. Inoltre, la politica dell'Actor viene aggiornata meno frequentemente rispetto ai Critic, ciò stabilizza l'apprendimento, riducendone le variazioni rapide [62].

4.2.6 Complessità del sistema ed Approccio Progressivo

Essendo il caso applicativo caratterizzato da una Reward a più componenti, devono essere tarati opportunamente i pesi di ogni componente, basandoci su una scala di importanza. Assegnando valori sbilanciati si rischierebbe di ottenere un Agent che dia più importanza ad un singolo aspetto ignorando le restanti componenti di comportamento. La determinazione dei pesi deve passare attraverso un processo iterativo di sperimentazione. Inoltre, Il caso applicativo è caratterizzato da un'elevata generalizzazione, basti pensare la sola scelta dell'Action, composta da sei valori della velocità dei giunti. Eseguendo un calcolo combinatorio delle velocità possibili, considerando una discretizzazione di 0.1 sui valori possibili, tenendo conto che i primi tre giunti possono avere un valore di velocità compreso tra i -π e i +π, otterremo 63 possibili scelte per ogni giunto e altri tre con valori compresi tra -2π e +2π, avremo 127 possibili scelte per ogni giunto.

$$63 \times 63 \times 63 \times 127 \times 127 \times 127 = 512192024001 \quad (10)$$

512192024001 possibili combinazioni per un singolo step.

Considerando ora che il percorso indicato è costituito da 129 punti, le combinazioni possibili saranno

$$(11) \quad (512192024001)^{129}$$

Tutto ciò deve essere anche considerato in un caso applicativo in cui il percorso, con relativi punti inziali e finali, cambia ad ogni episodio come anche la posizione del corpo. Si ottiene un problema di complessità elevata.

Dopo un'attenta riflessione relativa alla complessità del modello, si è deciso di adoperare una tecnica di addestramento progressivo, suddividendo il problema in sottoproblemi più semplici, ed ottenendo un beneficio dal punto di vista computazionale, della stabilità del processo di addestramento riducendo il rischio di comportamenti sub-ottimali. Si prevede di ottenere una migliore convergenza, più rapida ed affidabile di ogni singolo comportamento [63].

4.3 Path following ed evitamento dell'ostacolo

Seguendo la filosofia dell'Approccio Progressivo, il caso applicativo viene suddiviso in sottoproblemi per analizzare nel modo opportuno le potenzialità del modello. Si è deciso di affrontare il problema eliminando l'elevata generalità dello studio partendo da un problema semplice per poi aumentarne la complessità.

In questa sezione viene analizzato il problema del path following in cui si è addestrato l'UR3 a seguire un singolo percorso, privo di controlli relativi alle collisioni, in assenza del corpo e senza la generalizzazione del percorso.

Lo step successivo dello studio si è basato ragionando sull'obiettivo del caso applicativo complesso in cui l'aspetto fondamentale è quello di adattarsi alla presenza del corpo all'interno della zona di lavoro. È stato aggiunto al problema del path following di un percorso singolo il corpo direttamente inserito in un punto fisso della traiettoria. Si vuole osservare se l'Agent è in grado con l'addestramento di generare un'ottima traiettoria supplementare, in alternativa a quella indicata, per garantire la sicurezza dell'operatore.

Per la realizzazione dell'Environment di questi casi applicativi, vengono utilizzate le stesse Observations e Action del caso complesso. Ciò viene fatto in prospettiva di integrare pezzi e funzioni per aumentare la complessità del modello, fino ad ottenere il sistema complesso obiettivo. È quindi necessario prevedere la presenza di tutte le variabili e dei parametri che verranno utilizzati nei casi studio successivi. La differenza risiede nel modello Simulink utilizzato, nella Reward e nella Reset Function. Diversamente dal caso iniziale, è effettuato un cambio per quanto riguarda la Reward, in cui l'Agent dovrà seguire una traiettoria definita tramite profilo di velocità.

4.3.1 Path following: singolo percorso

Si parte dall'analisi del problema del path following più semplice in cui si vuole addestrare il cobot ad avere un determinato profilo di velocità tra punto iniziale e finale sempre nelle stesse coordinate. Da qui, si ritroverà a seguire in ogni istante di tempo dei punti fissi e determinati direttamente dal profilo di velocità fornitogli.

Environment

Per la casistica analizzata di percorrenza di uno specifico percorso sono stati individuati i parametri da inizializzare ad ogni episodio con cui si è costruita la Reset function. In questa funzione vengono indicate le coordinate cartesiane del punto iniziale, del punto finale e dei punti del percorso, trovati tramite studio pre-addestramento. Inoltre, è fornita la configurazione iniziale in spazio giunti del robot, in modo che l'UR3 parta con la flangia rivolta perpendicolarmente al terreno (Figura 63).

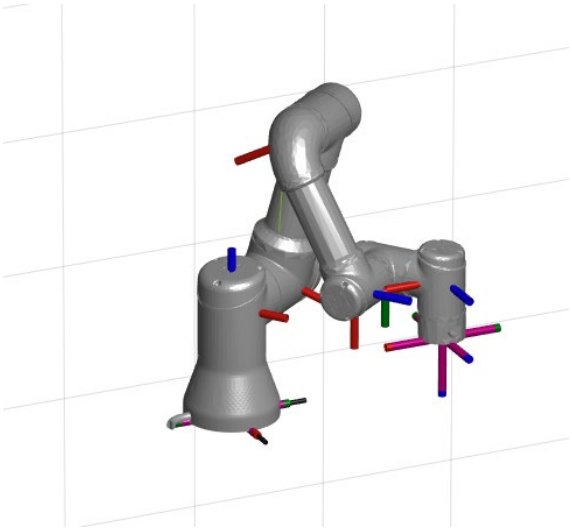

Figura 63: Visualizzazione della configurazione iniziale dell'UR3

L'analisi preliminare eseguita prima dell'addestramento riguarda la determinazione del punto iniziale e finale, della configurazione iniziale dei giunti e la definizione dei punti del percorso. Quest'analisi corrisponde a quella effettuata per ogni episodio tramite Reset Function del caso complesso. In questa applicazione viene eseguita solo la prima volta per scegliere una casistica che non dia problemi di collisione o della generazione del percorso costituito da punti appartenenti a zone dove il robot non può arrivare.

Per l'addestramento del path following è stato scelto un percorso simile a quello del caso complesso con la differenza che sarà eseguito sul piano fisso z=0.2 m come rappresentato nella "Figura 64":

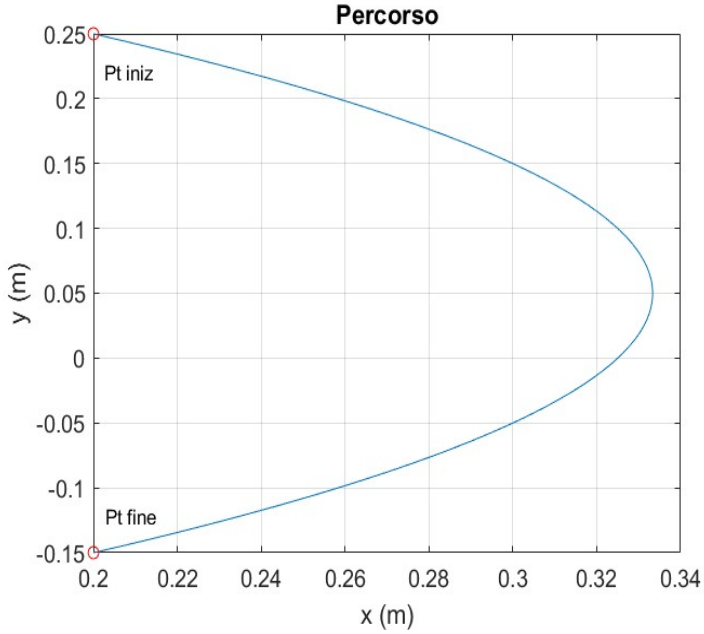

Figura 64: Percorso per l'addestramento del caso applicativo Path Following

Sono stati indicati i profili di velocità di percorrenza desiderati che il robot dovrà imparare a seguire. Velocità nella coordinata x (Figura 65) e velocità nella coordinata y (Figura 66).

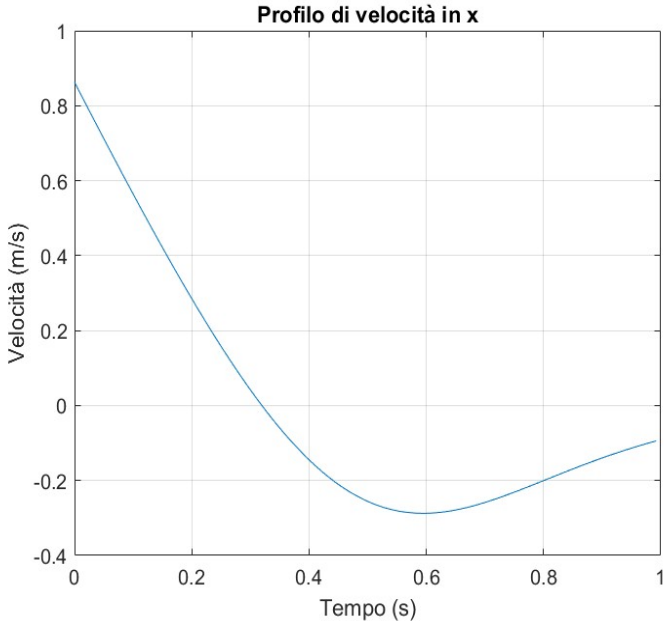

Figura 65: Profilo di velocità nella coordinata x

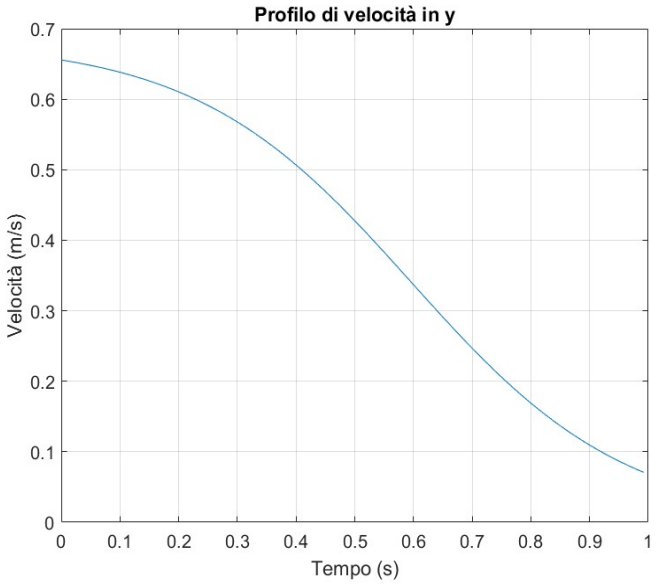

Figura 66: Profilo di velocità nella coordinata y

Questi profili di velocità sono indicativi dell'andamento necessaria a percorrere una distanza euclidea, corrispondente alla retta di congiunzione dal punto iniziale al finale, pari a 0.4 m in 1 secondo. Essendo ottenuti da esponenziali, i profili di velocità non terminano allo zero, come si nota dalle rappresentazioni grafiche. Per correggere questo problema sono stati aggiunti due punti al percorso coincidenti con le coordinate del punto finale, in modo da addestrare l'Agent a concludere la traiettoria regolando la velocità, rallentando e terminando con valore nullo.

L'Environment è stato rappresentato tramite modello Simulink, illustrato nella "Figura 67", con cui si comprenderanno i passaggi logici e matematici effettuati.

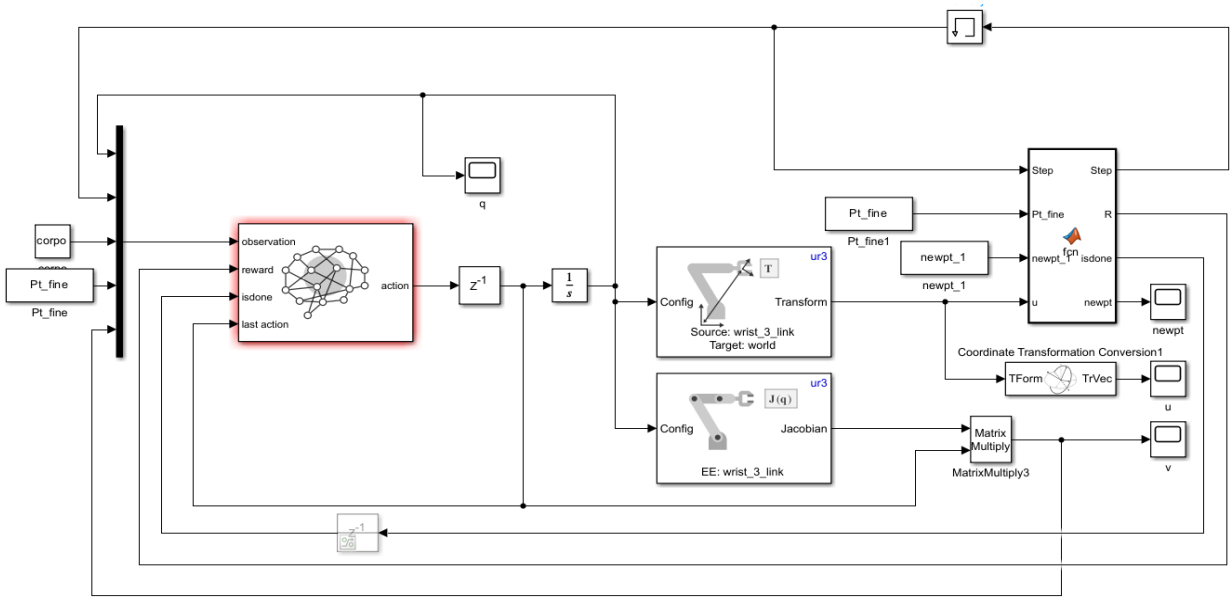

Figura 67: Modello Simulink Path Following

I blocchi “Pt_fine”, “corpo”, “newpt_1” sono relativi ai valori resettati ad ogni episodio tramite I blocchi “Pt_fine”, “corpo”, “newpt_1” sono relativi ai valori assegnati all’inizio di ogni episodio tramite Reset function. Dal blocco “RL_Agent” si osserva l’uscita del canale “Action”, corrispondente al vettore 6x1 delle velocità angolari dei singoli giunti definiti dall’Agent. Questo vettore dovrà essere integrato tramite il blocco indicato come “1/s” per ottenere la configurazione dei giunti q. Dentro al blocco dell’integrazione viene indicata la configurazione iniziale in spazio giunti “Initial_condition” specificata nella Reset function. La configurazione dei giunti q servirà ad ottenere la posa attuale del robot “u” e di conseguenza la posizione attuale del cobot nello spazio cartesiano, utilizzando il blocco “Get Trasform”. Inoltre, tramite l’utilizzo del blocco “Jacobian”, si calcolerà la Jacobiana che sarà poi moltiplicata per il vettore delle velocità dei giunti per ottenere “v” la velocità del robot in spazio cartesiano.

I blocchi Scope, relativi ai parametri “newpt_1”, “u”, “v” saranno poi necessari nella fase di analisi dei risultati per verificare l’addestramento.Per evitare l’incorrenza di loop algebrici sono stati inseriti i blocchi memory e 1 step delay.

Reward

La funzione Reward è stata costruita considerando l’obiettivo di addestrare l’Agent in modo che segua un profilo di velocità assegnato. Come precedentemente illustrato nella fase di studio pre-addestramento, dal profilo di velocità si è ottenuto un percorso costituito da punti. Questi indicano la posizione ed il momento esatto in cui deve trovarsi l’Agent per far in modo che sia rispettato il profilo di velocità desiderato. Per la realizzazione dell’addestramento è stata utilizzata la componente del percorso R1 della Reward progettata per il caso applicativo iniziale.

Nella “Figura 68” è mostrata una sessione di allenamento con la Reward impostata:

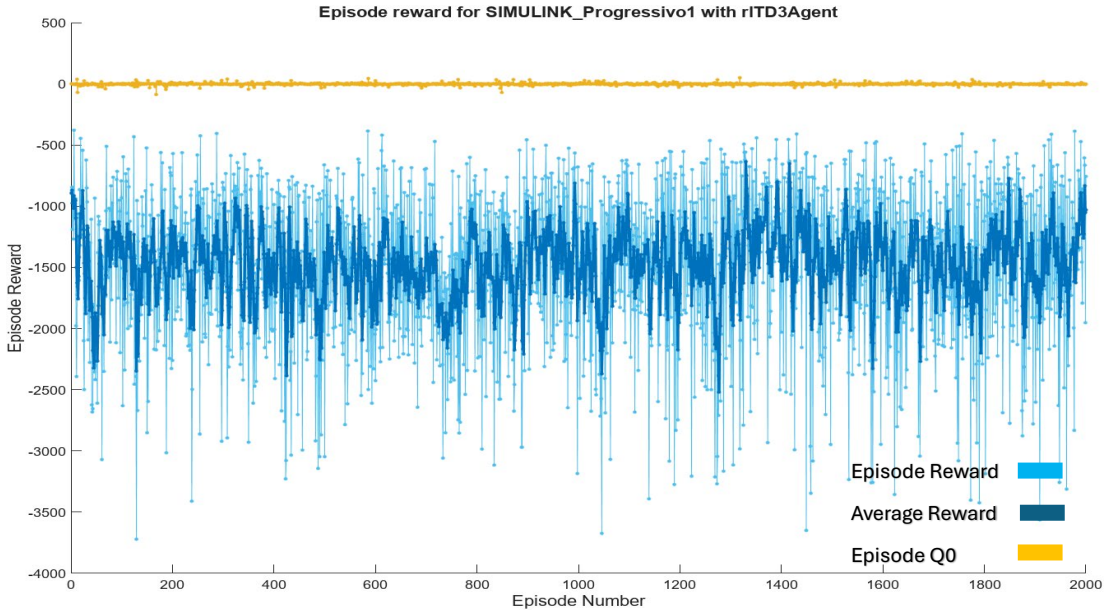

Figura 68: Training con tolleranza di posizionamento pari a 0 mm

Si è poi optato per l'utilizzo di un valore positivo pari a +1 in caso di distanza pari a zero, per poi passare ad accettare un errore di posizionamento di 1 mm rispetto al punto prefissato. Questa strategia non ha sortito gli effetti desiderati, come si evince dalla "Figura 69" si osservano evidenti fluttuazioni significative nelle decisioni dell'Agent.

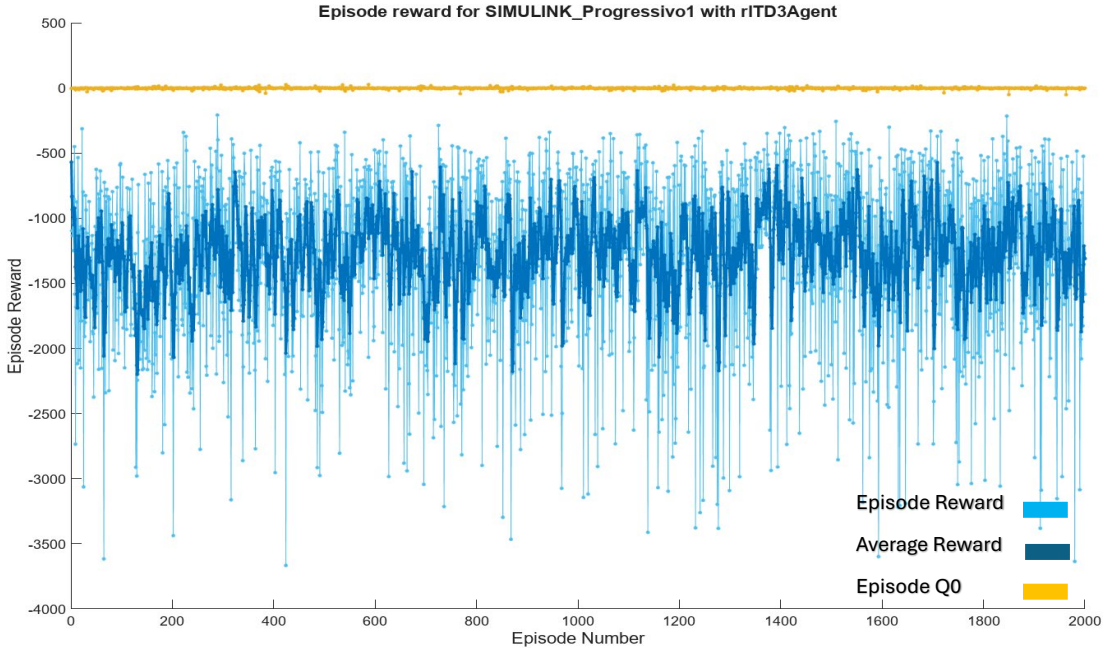

Figura 69: Training con tolleranza di posizionamento di 1 mm

Infine, considerando accettabile un errore effettivo di posizionamento di 5 mm rispetto alla posizione desiderata, e assegnando una ricompensa di +5 se l'Agent si ritrova all'interno di questa zona, si ottiene un grafico di training accettabili, come illustrato nella "Figura 70", tramite cui è eseguito l'allenamento.

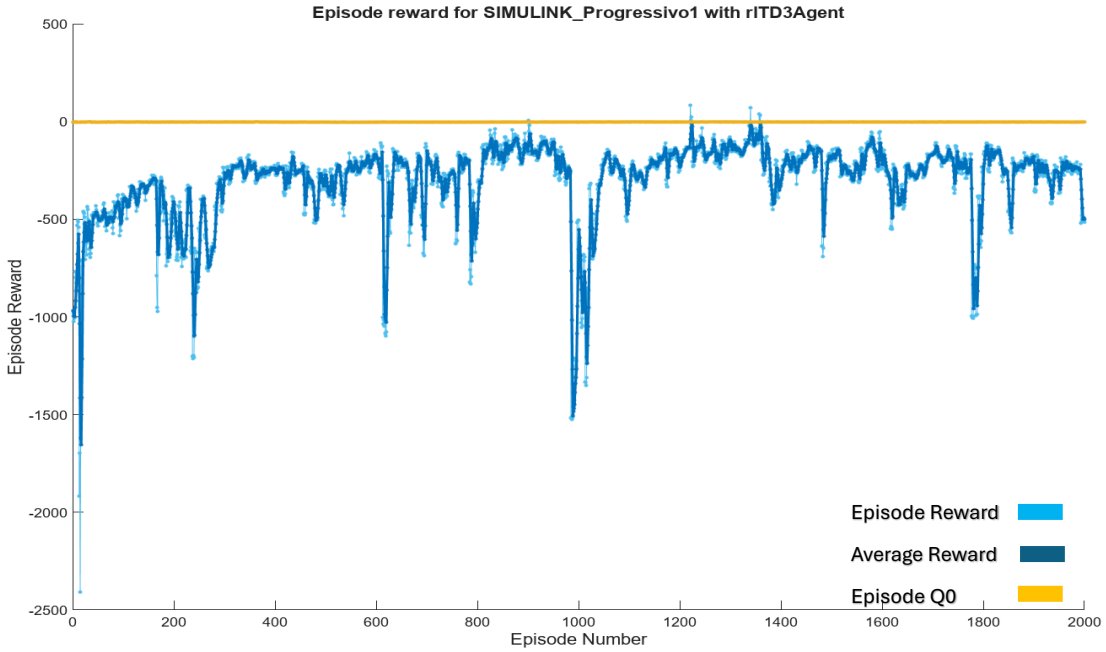

Figura 70: Training con tolleranza di posizionamento di 5 mm

È di seguito riportata l'equazione della Reward al variare della distanza dal punto del percorso "m":

$$(12) \quad R = -50 \cdot m \ (m > 0.005) \qquad R = +5 \ (m \leq 0.005)$$

La rappresentazione in 2D della "Figura 71" raffigura l'andamento della Reward nello spazio. Nel particolare, il punto rosso rappresenta un punto della traiettoria e a seconda della posizione dell'end effector la funzione R assume un valore diverso, maggiore se è vicino al punto voluto e negativo se lontano

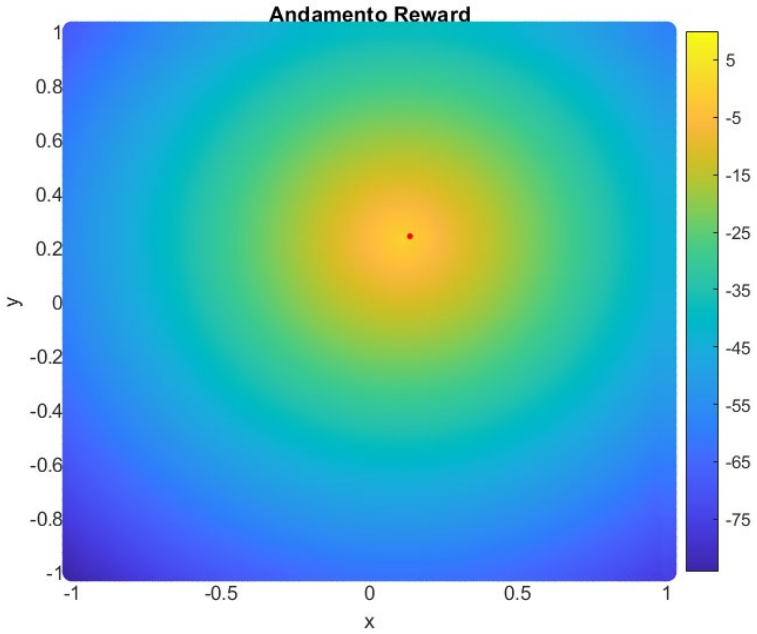

Figura 71: Andamento della Reward Path Follow

A questa funzione Reward si è aggiunto un fattore di ricompensa di +100 in caso di arrivo in prossimità del punto finale. L'assegnazione di questa ricompensa dipenderà dal fattore distanza calcolato con il punto finale.

Dopo l'addestramento con questa Reward si è optato per un cambio della funzione di penalità poiché incapace di gestire opportunamente i valori della distanza di piccole dimensioni, quindi incapace di ottenere un cobot con un'adeguata precisione. Si è optato per la costruzione della seguente nuova Reward

$$(13) \quad R = a * \log(b * m + 1) \ \ (m > 0.005) \qquad R = +5 \ (m \leq 0.005)$$

È stata scelta questa funzione perché permette di dare un maggiore gradiente alla reward nella direzione voluta.

La funzione così ottenuta prevede un aumento veloce della Reward a basse distanze e con valori significativi per far capire al cobot che non si è ancora arrivati a dei risultati ottimali (Figura 72). Passando da 0.1 a 1 ci sarà un attenuarsi della crescita del fattore di penalità per evitare che valori eccessivi impediscano un corretto addestramento, anche in previsione

dell'aggiunta delle componenti della Reward relative alle casistiche più complesse (Figura 73).

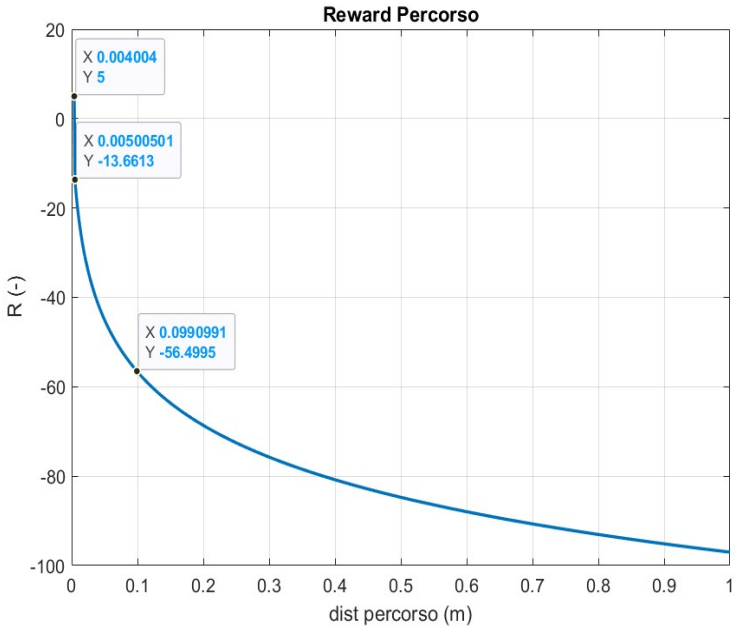

Figura 72: Nuovo andamento della funzione Reward del Percorso

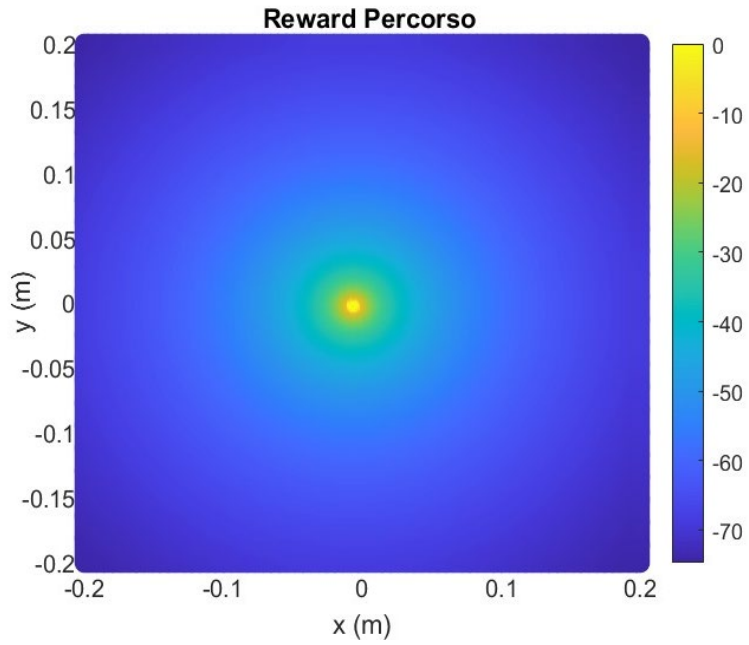

Figura 73: Grafico Colorbar per visualizzare l'andamento della Reward

Training

Costruito il modello si passa al caricamento dell'Environment nell'app "Reinforcement Learning Designer" e alla creazione dell'Agent TD3, mantenendo i parametri predefiniti. Si impostano i parametri dell'Agent e del "Train" per le sessioni di addestramento ragionando sulle definizioni teoriche dei parametri e sul caso applicativo in esame.

Considerando l'approccio del path following, bisogna seguire la traiettoria costituita da 128 punti generata nella Reset Function. Si è perciò preso in considerazione questo valore come parametro per il "Max Episode Length". Per "Average Window Length" si è indicato a 5 il numero di episodi su cui basare l'andamento della media della Reward. Come criterio di stop della simulazione è stato inserito "Episode Reward", indicando il valore della Reward che il singolo episodio deve ottenere per terminare la sessione di allenamento. Per questo caso applicativo si è calcolato il valore della Reward massima ottenibile nel caso ottimale, considerando una ricompensa di +5 per ogni punto del percorso centrato dal cobot e il +100 relativo alla ricompensa di arrivo al punto finale. Informazioni riportate nella Tabella 2

Max Episode Length	128
Average Window Length	5
Episode Reward stop	740

Tabella 2: Impostazioni per definire le dimensioni della sessione di Training

Si è inserito un Sample Time considerando la frequenza di campionamento dell'UR3, che corrisponde a 125 Hz, ottenendo il valore di 0.008 secondi dalla formula per il periodo T=1/f.

Per il Discount Factor è stato fissato a 0.7 basando la scelta sulle nozioni teoriche riportate nel Capitolo 4.1. Impostando il fattore a 0.7 l'Agent valuta le ricompense future con peso ridotto del 30% rispetto alle ricompense immediate. Si sono effettuate diverse simulazioni variando questo parametro ma non si sono riscontrati differenze sostanziali nell'allenamento dell'Agent.

Il Batch size è stato raddoppiato, passando dal valore standard di 64 al valore di 128, per poi passare a valori anche superiori riscontrando un aumento dei tempi di Training dovuti all'aumento del numero di campioni con cui il modello elaborerà il suo aggiornamento.

Dei parametri relativi alle reti neurali Actor, Critic è stato gestito il "Learn rate", tasso di apprendimento delle reti neurali. Si è partiti da un valore pari a 0.001 per poi scendere a sessioni di allenamento di 0.0001, fino al valore di 0.00008. Con questo passaggio graduale si è riscontrato il rallentamento delle sessioni di Training ma con una maggiore stabilità di apprendimento. I dati utilizzati sono riportati nella "Tabella 3" riportata di seguito:

Sample Time	0.008 s
Discount Factor	0.7
Batch Size	155
Learn Rate	0.00008

Tabella 3: Impostazioni del Train

La strategia di addestramento adottata consiste nella suddivisione dell'allenamento in due fasi, fase di esplorazione e fase di affinamento. Ognuna di queste fasi è caratterizzata da più sessioni di allenamento impostando differenti valori di "Max Episode Number" e variando i parametri relativi al "Gaussian Noise", gestendo il rumore per una maggiore esplorazione, e del "Target Policy Smoothing", per garantire la stabilità di apprendimento.

Per il Target Policy Smoothing, dopo un elevato numero di sessioni di allenamento si è trovato un andamento che rende stabile l'apprendimento senza eccessive variazioni. Ottenuto questo risultato non è stato più variato in questo caso applicativo.

Nella "Tabella 4" sono riportati i valori di tutte le sessioni di allenamento eseguite per l'addestramento dell'Agent per il problema del Path Following:

Max Episode	Gaussian Std. deviation	Gaussian Upper/Lower Limit	Target Std. deviation	Target Upper/Lower Limit
10000/10000	0.8	(+/-) 4	0.7	(+/-) 0.9
4000/4000	0.8	(+/-) 3	0.7	(+/-) 0.9
4000/4000	0.7	(+/-) 3	0.7	(+/-) 0.9
2000/2000	0.6	(+/-) 2	0.7	(+/-) 0.9
1000/1000	0.5	(+/-) 2	0.7	(+/-) 0.9
896/1000	0.4	(+/-) 1	0.7	(+/-) 0.9
156/500	0.3	(+/-) 0.5	0.7	(+/-) 0.9
41/500	0.2	(+/-) 0.5	0.7	(+/-) 0.9

Tabella 4: Sessioni di addestramento con relative impostazioni per l'Agent

Per il parametro "Max Episode" sono indicati il numero degli episodi eseguiti sul numero di episodi massimi da eseguire nella sessione di allenamento. È stato fornito in questa forma poiché alcune simulazioni sono terminate prima del numero massimo di episodi avendo raggiunto il valore indicato per il "criterio termine simulazione" scelto, corrispondente al valore massimo della Reward dell'episodio, pari in questa casistica a +740.

Sono riportati gli andamenti dei Training di tre Agent:

- Agent 1: ottenuto dopo aver effettuato le prime tre sessioni di allenamento riportate in tabella corrispondenti alle sessioni di esplorazioni iniziali (Figura 74):

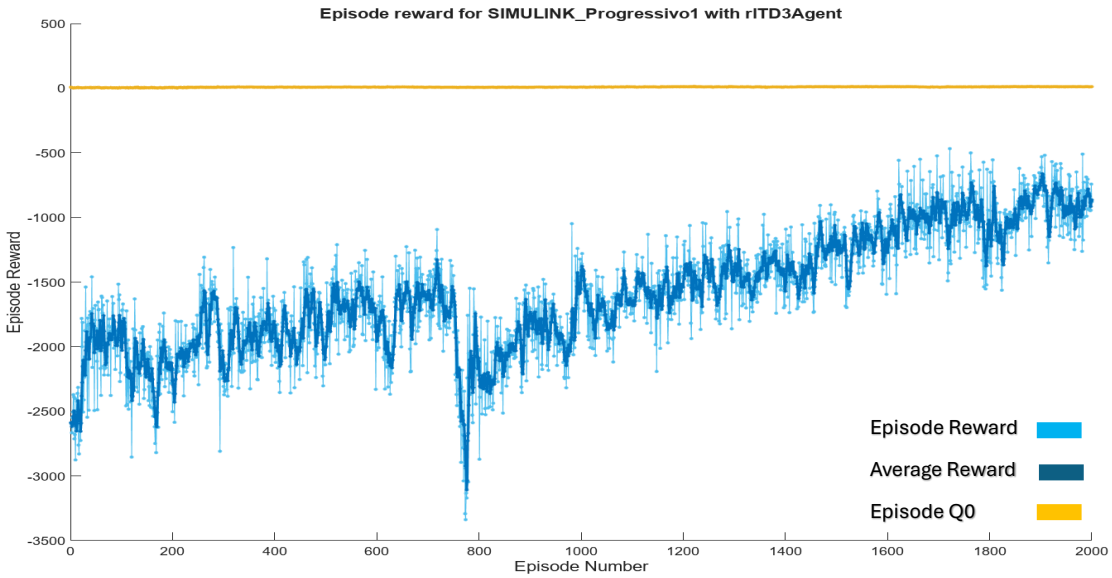

Figura 74: Training Agent 1

- Agent 2: ottenuto dopo aver effettuato le prime cinque sessioni di allenamento riportate in tabella. Da questo Agent si passa ad una strategia di affinamento (Figura 75):

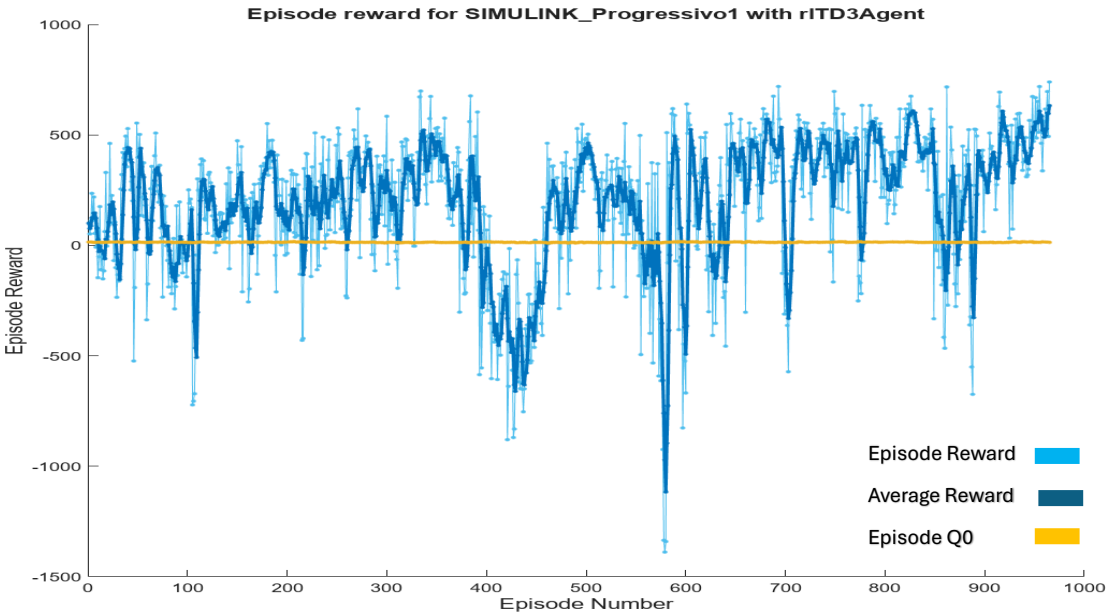

Figura 75: Training Agent 2

- Agent 3: ottenuto dopo aver effettuato tutte le sessioni di allenamento riportate in tabella (Figura 76):

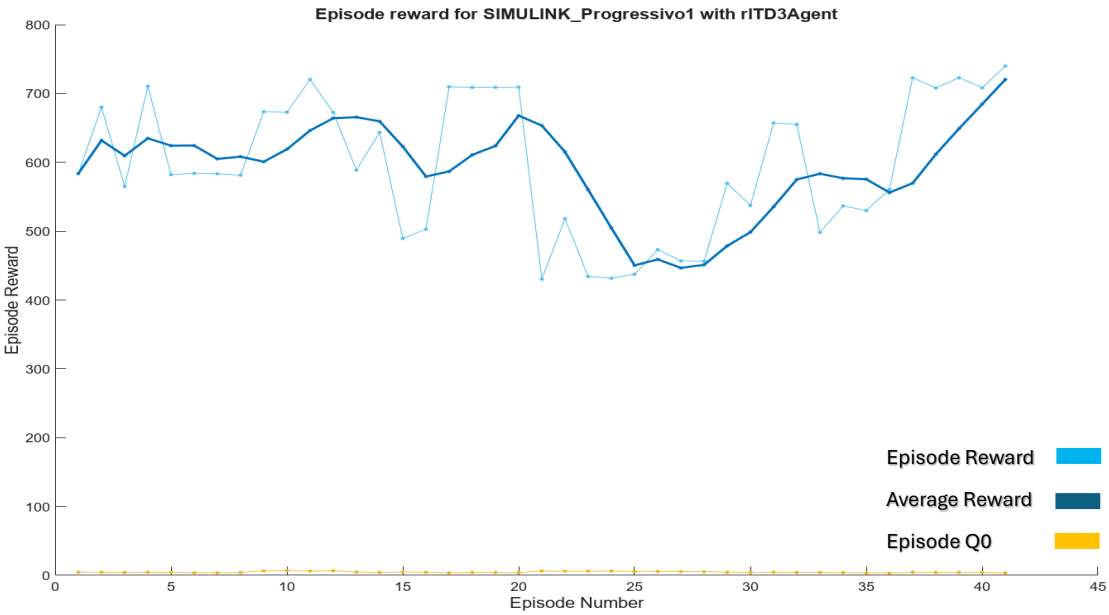

Figura 76: Training Agent 3

Risultati

Si riportano i risultati grafici del "Agent 3" ottenuto al termine delle sessioni di allenamento per il Path Following dove si confrontano i dati che rispecchiano la traiettoria desiderata con i dati ricavati dalla simulazione dell'Agent addestrato.

Nella "Figura 77" è riportato l'andamento delle coordinate che descrivono il percorso fornito per l'addestramento, indicate con denominazione "th" indicate con linea continua, confrontate con l'andamento delle coordinate percorse dall'Agent durante la simulazione indicate con la denominazione "real" e rappresentate con linea tratteggiata.

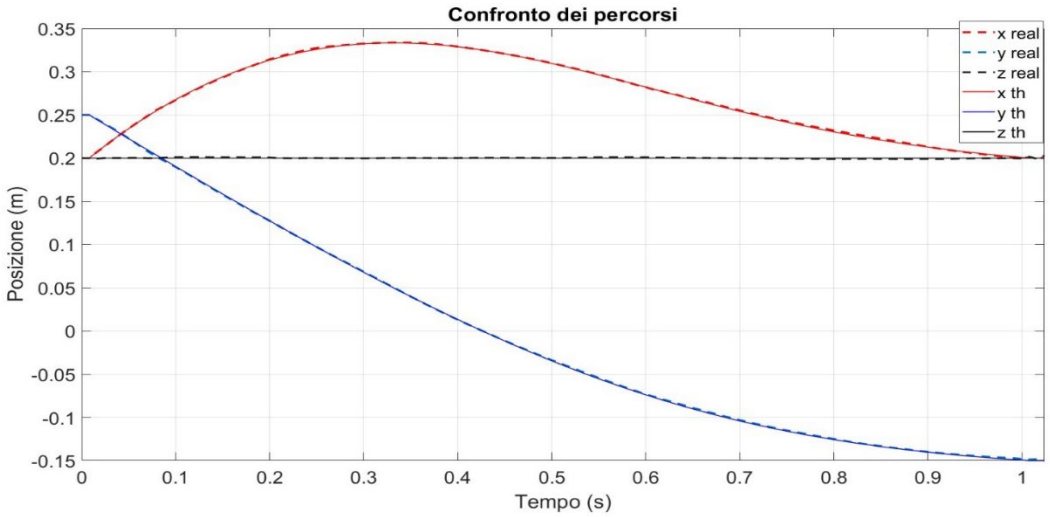

Figura 77: Grafico di confronto tra le coordinate del percorso fornite per l'addestramento e le coordinate percorse dall'Agent 3 addestrato

Dall'ingrandimento del tratto terminale, rappresentato nella "Figura 78", è possibile osservare il discostamento tra i due percorsi, spiegabile dal fatto che per giungere in corrispondenza del punto finale in modo preciso debbano essere percorsi in modo corretto i punti precedenti.

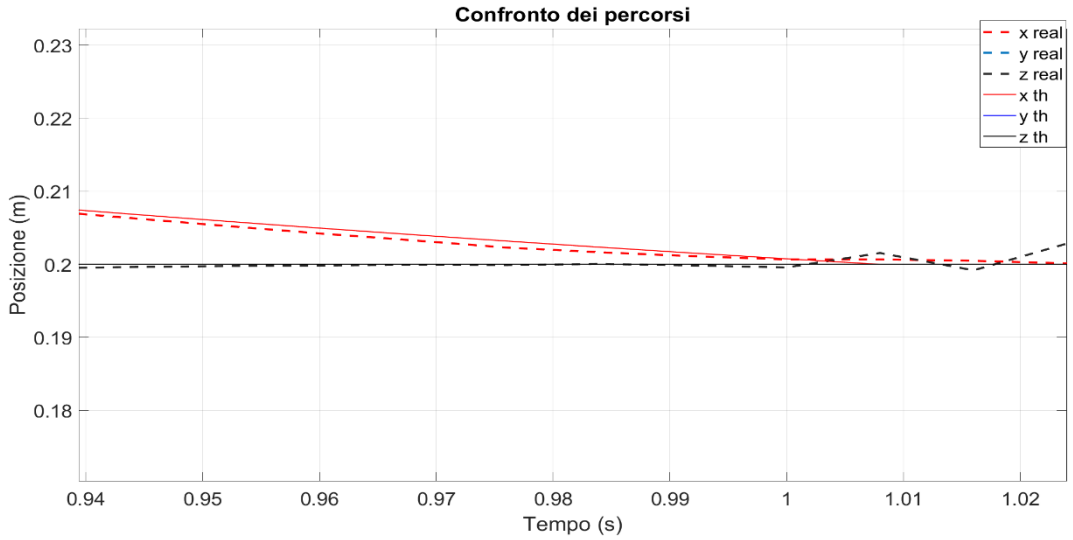

Figura 78:Ingrandimento del confronto per evidenziare un leggero discostamento in prossimità del punto finale

È possibile risolvere il problema con l'esecuzione di altre sessioni di allenamento dove la tolleranza sull'errore di posizionamento viene diminuito arrivando ad un valore prossimo allo zero.

Nel grafico 2D in "Figura 79" è riportato l'andamento della componente di velocità x, in cui si confronta l'andamento della velocità fornita per l'addestramento "vx th" con la velocità che ha l'Agent lungo la coordinata x durante la simulazione "vx real":

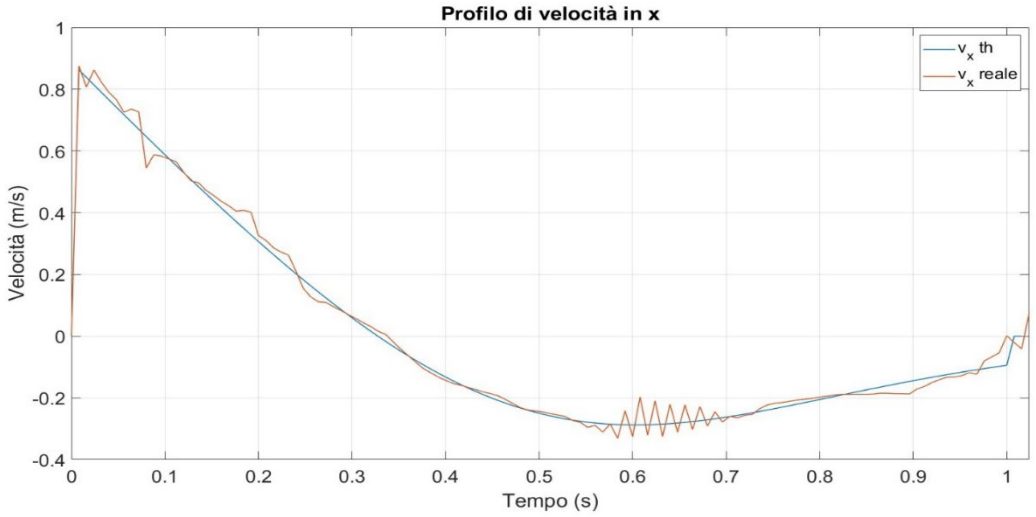

Figura 79: Confronto tra profilo di velocità della vx fornita e profilo di velocità risultante ottenuto dall'Agent 3 addestrato

Nel grafico 2D "Figura 80" si osserva l'andamento della componente di velocità y, in cui è confrontato l'andamento della velocità fornita per l'addestramento "vy th" con la velocità che ha l'Agent lungo la coordinata x durante la simulazione "vy real":

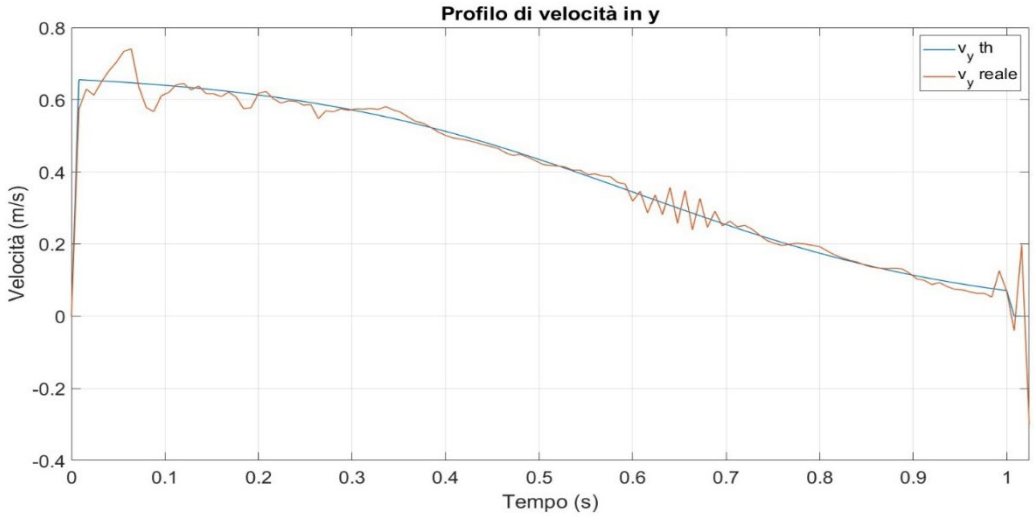

Figura 80: Confronto tra profilo di velocità della vy fornita e profilo di velocità risultante ottenuto dall'Agent 3 addestrato

Nella “Tabella 5” sono riportati gli errori medi delle informazioni relative al discostamento tra i dati ottenuti dall’Agent allenato e i dati che descrivono la traiettoria desiderata.

Agent	Errore sul percorso (m)	Errore sul profilo di velocità in x (m/s)	Errore sul profilo di velocità in y (m/s)	Errore sulla velocità (m/s)
Agent 1	0.051	0.1679	0.2385	0.3276
Agent 2	0.0021	0.0332	0.0629	0.0938
Agent 3	0.0012	0.0183	0.0208	0.0399

Tabella 5: Errori relativi al confronto tra i dati ottenuti dall'Agent addestrato e i dati forniti

Si esegue un confronto tra tre Agent, corrispondenti all’Agent addestrato considerato in tre diversi momenti dell’allenamento, infatti, si osservano valori di errore alto nel caso dell’ “Agent1”, corrispondente ad un Agent che non ha esplorato opportunamente l’Environment. L’”Agent 2”, corrispondente all’Agent che ha terminato la fase di esplorazione, raggiunge valori positivi della Reward, prossimi al valore target. Dal confronto tra “Agent 2” ed “Agent 3” si osservano errori più piccoli per l’”Agent 3” che è stato sottoposto alla fase di affinamento, importante per garantire un’opportuna precisione nell’esecuzione della traiettoria.

4.3.2 Path following: singolo percorso e gestione dell'orientazione

Dall'addestramento del cobot per il path following si otterrà il dispositivo che segue il percorso indicato, ma non avendo fornito indicazioni sull'orientazione durante il percorso l'UR3 eseguirà rotazioni dell'End Effector. La "Figura 81" rappresenta l'andamento dell'orientazione sottoforma di angoli di Eulero, convenzione ZYX, del "wrist_3_link" rispetto al "world":

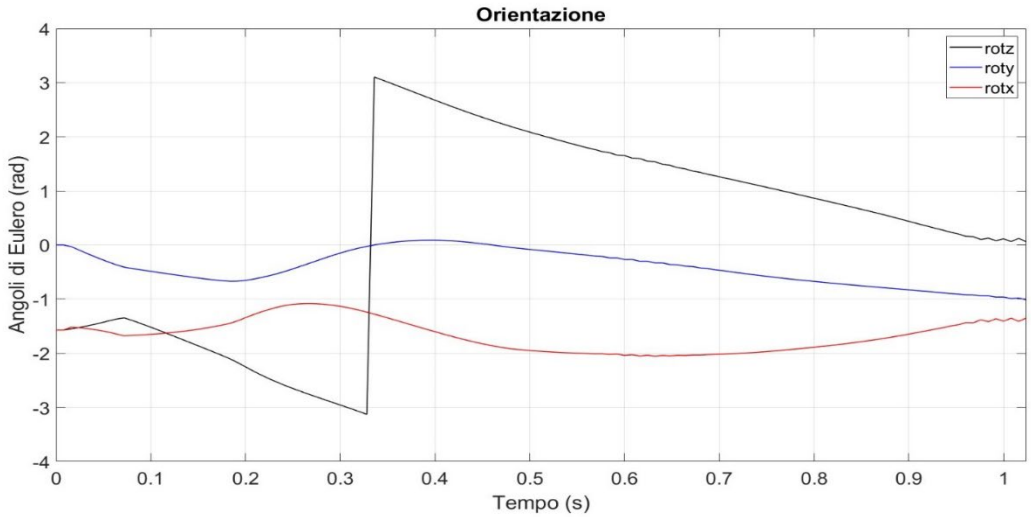

Figura 81: Andamento dell'orientazione sottoforma degli angoli di Eulero (ZYX) durante l'esecuzione del percorso con l'Agent addestrato per il Path Following

È stato quindi impostato un ulteriore addestramento per regolare l'orientazione dell'Agent ottenuto dal path following. Considerando che il movimento del cobot deve essere eseguito su un piano ad una quota fissa ed avendo impostato la configurazione iniziale dell'End Effector, con asse uscente dalla flangia perpendicolare al pavimento, si vuole che l'UR3 mantenga l'orientazione di partenza. È stata perciò progettata una nuova componente della Reward in grado di confrontare la matrice 3x3 dell'orientazione nell'istante di tempo con quella della posa iniziale, assegnando una penalità di valore costante in caso di discostamento, tramite comando "if". Questa penalità si aggiungerà a quella relativa del path following. È di seguito riportata la "Tabella 6" indicativa della sessione di allenamento per l'orientazione:

Max Episode	Gaussian Std. deviation	Gaussian Upper/Lower Limit	Target Std. deviation	Target Upper/Lower Limit
5000/5000	0.8	(+/-) 3	0.7	(+/-) 0.9

Tabella 6: Sessione di addestramento per l'orientazione

In seguito allo svolgimento di alcune sessioni di allenamento è stata inserita una zona di tolleranza corrispondente ai 0.01 radianti, entro cui l'Agent riceve una ricompensa di +1. Questa aggiunta è stata fatta dopo aver osservato un andamento instabile della Reward complessiva, con una media in prossimità di valori di Reward negativi e lontani dalle zone

di positività. Dopo l'esecuzione di altre sessioni di addestramento riportate nella "Tabella 7" è stata ampliata la zona di tolleranza.

Max Episode	Gaussian Std. deviation	Gaussian Upper/Lower Limit	Target Std. deviation	Target Upper/Lower Limit
6000/6000	0.8	(+/-) 3	0.7	(+/-) 0.9
4000/4000	0.7	(+/-) 2	0.7	(+/-) 0.9

Tabella 7: Sessione di allenamento per l'orientazione con Reward modificata

Tramite esplorazione per 10000 episodi, si ha la stabilizzazione della Reward complessiva su valori negativi pari a -1200, per l'esecuzione di un'operazione relativamente più semplice del seguire un percorso indicato (Figura 82).

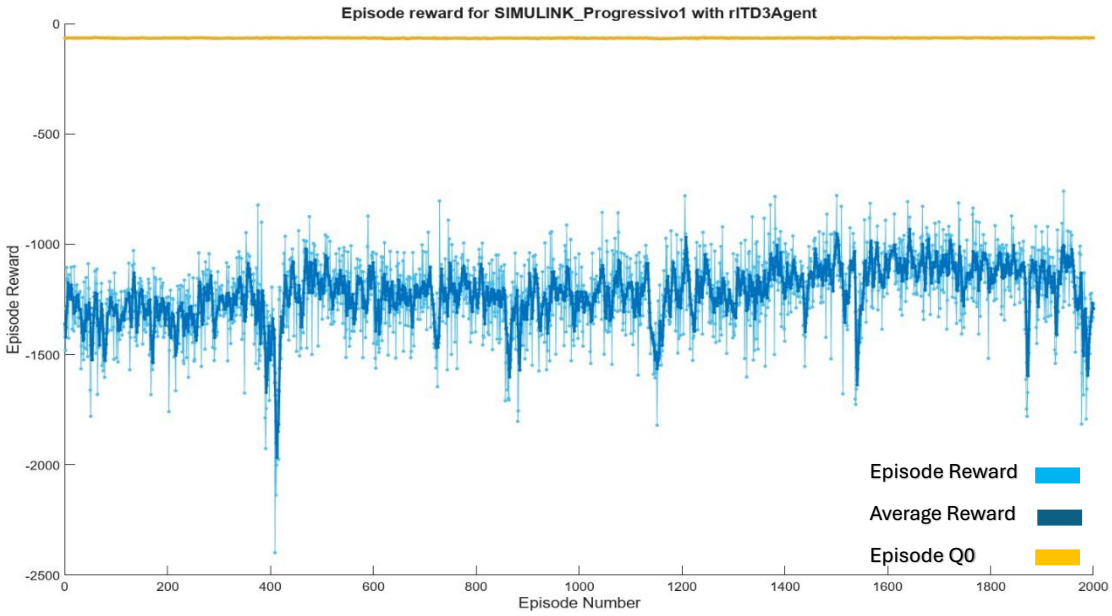

Figura 82: Grafico di Training per l'orientazione con andamento stabilizzato per Reward negative

Per migliorare l'andamento della reward e favorire l'apprendimento del robot si è sviluppata una funzione di penalità graduale e continua in grado di gestire anche valori di discostamento minimi dall'orientazione voluta, in modo da ottenere l'esecuzione di compiti che prevedono precisione nei movimenti. Essendo il valore massimo di discostamento tra due matrici di rotazione pari al valore di 3.14 radianti, dovendo gestire valori anche dell'ordine di 0.01, si è optato per la costruzione di una funzione di reward logaritmica:

$$(14) \quad R = -(a * \log(b * disc) + c) \quad (disc \geq 0.1) \qquad R = +1 \quad (disc < 0.1)$$

disc = corrisponde al valore del discostamento calcolato come differenza tra la matrice di orientazione assunto dall'Agent e la matrice di orientazione di riferimento.

È rappresentato nella “Figura 83” l’andamento rappresentato:

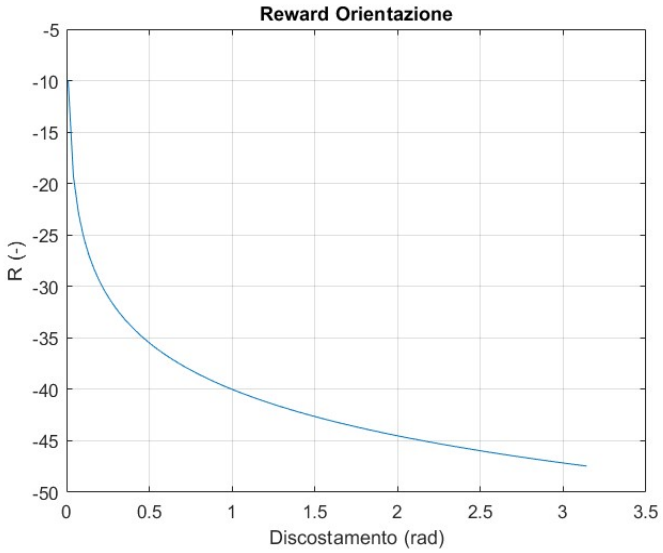

Figura 83: Funzione logaritmica della Reward per la gestione dell'orientazione

Sono elencate nella “Tabella 8” le sessioni di allenamento svolte con la Reward costituita dalla funzione del path following, del caso applicato precedente, e quella logaritmica progettata per l’orientazione. L’addestramento è svolto utilizzando come Agent di partenza quello ottimo, ottenuto dagli addestramenti del Path Following.

Max Episode	Gaussian Std. deviation	Gaussian Upper/Lower Limit	Target Std. deviation	Target Upper/Lower Limit
5000/5000	0.7	(+/-) 3	0.7	(+/-) 0.9
2000/2000	0.6	(+/-) 2	0.7	(+/-) 0.9
1000/1000	0.6	(+/-) 2	0.7	(+/-) 0.9
1000/1000	0.5	(+/-) 1.5	0.7	(+/-) 0.9
500/500	0.4	(+/-) 1	0.7	(+/-) 0.9
452/500	0.3	(+/-) 1	0.7	(+/-) 0.9
1/500	0.2	(+/-) 1	0.7	(+/-) 0.9

Tabella 8: Sessioni di allenamento eseguite per regolare l'orientazione

Sono presi in esame tre momenti diversi dell’allenamento dell’Agent, numerati da 1 a 3, di cui si riportano gli andamenti dei Training:

- È indicato come “Agent 1” l’Agent ottenuto dopo la prima sessione di allenamento riportata nella “Tabella 8”. Nella “Figura 84” è mostrato l’andamento del Training:

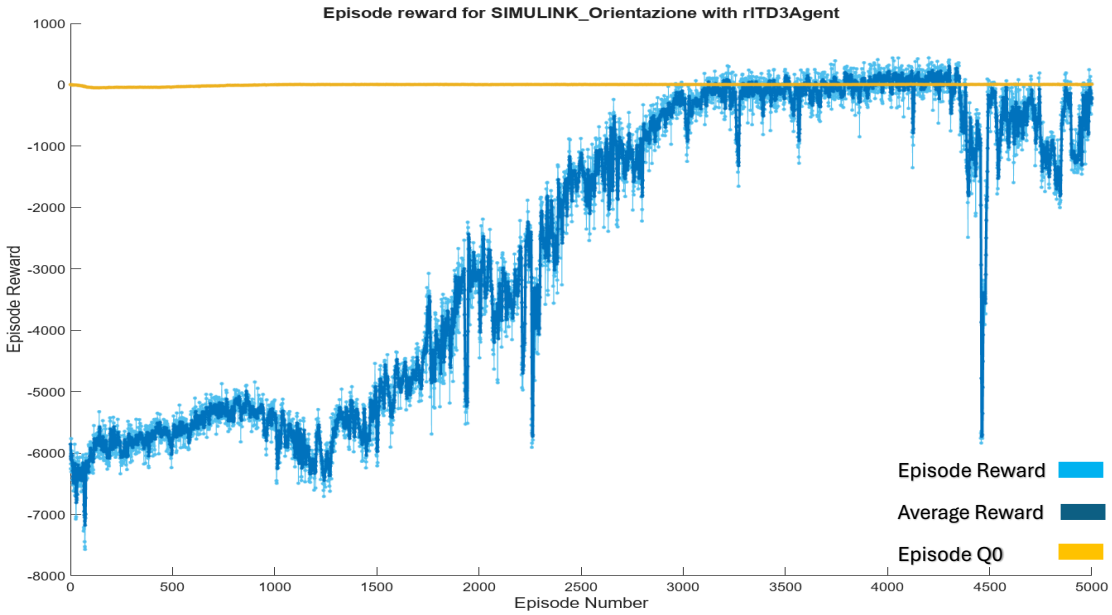

Figura 84: Training per l'ottenimento dell'Agent 1

- È indicato come “Agent 2” l’Agent ottenuto dopo aver eseguito le sessioni di allenamento riportate nella “Tabella 8” fino alla riga 4 compresa. Nella “Figura 85” è mostrato l’andamento del Training:

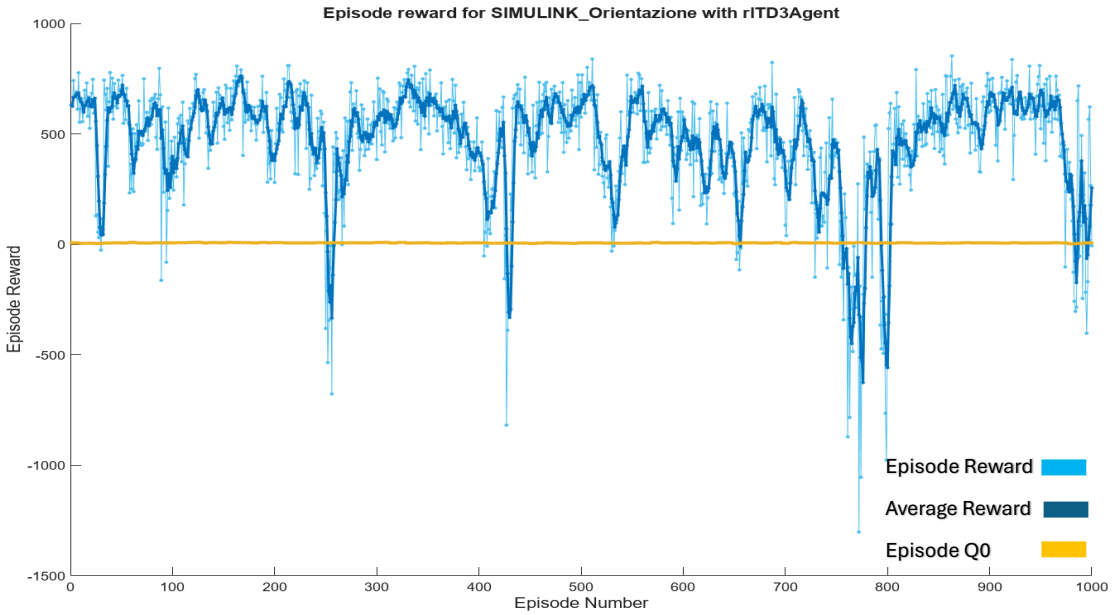

Figura 85:Training per l'ottenimento dell'Agent 2

- È indicato come “Agent 3” l’Agent ottenuto dopo aver eseguito tutte le sessioni di allenamento riportate nella “Tabella 8”. Non è riportato l’andamento del Training per cui si è ottenuto questo Agent poiché converge immediatamente al valore di massimo della Reward, graficamente corrispondente ad un solo punto, indicativo dell’avvenuto affinamento con cui si otterrà l’Agent ottimale.

Risultati

Sono riportati i risultati grafici del “Agent 3” ottenuti al termine delle sessioni di allenamento per il Path Following di un singolo percorso con gestione dell’orientazione, confrontati con i dati forniti indicativi della traiettoria desiderata.

Nella “Figura 86” è rappresentato l’andamento dell’orientazione per l’Agent addestrato ed affinato con una tolleranza sul discostamento di 0.1 radianti:

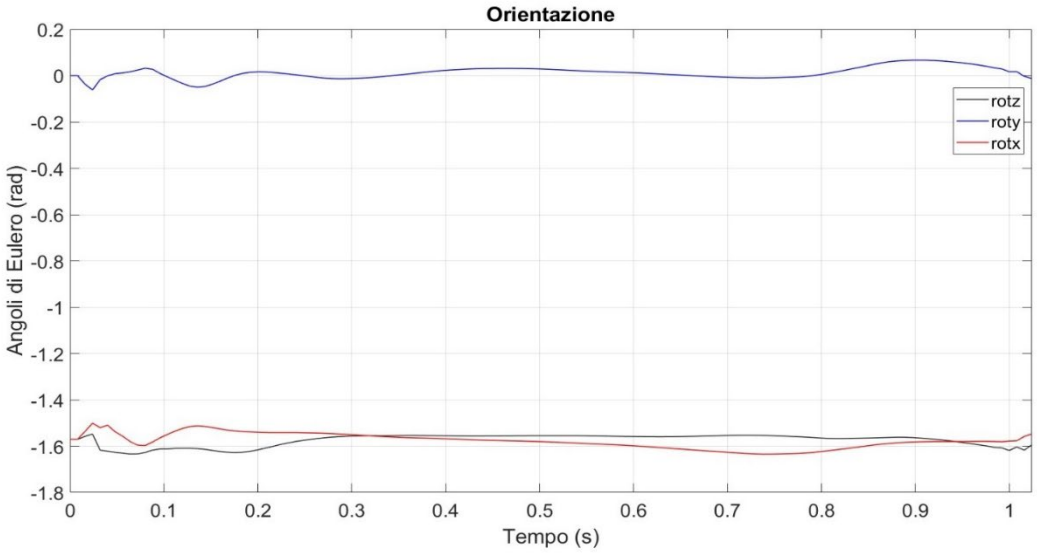

Figura 86: Andamento dell'orientazione del "wrist_3_link" dopo le sessioni di addestramento con tolleranza di discostamento di 0.1 radianti

Dai risultati grafici si può affermare che l’addestramento è andato a buon fine e che l’Agent ha imparato in seguito a queste sessioni a gestire l’orientazione, confermando la correttezza della funzione della Reward progettata.

È riportato nella “Figura 87” il percorso sottoforma di coordinate x y z eseguito dall’Agent 3 e confrontato con le coordinate del percorso fornito per mostrare l’efficacia del metodo ponendo più componenti della Reward:

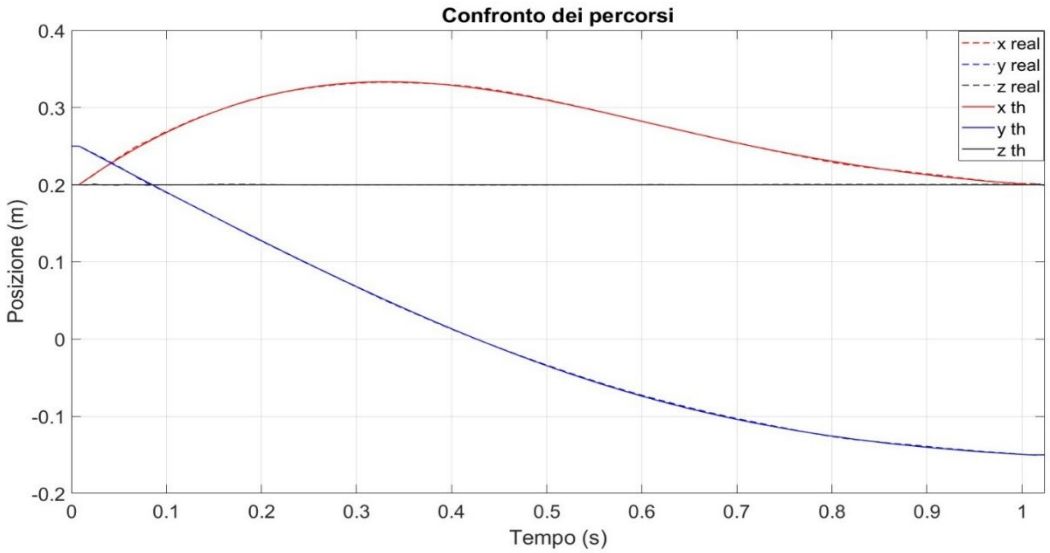

Figura 87: Grafico di confronto tra le coordinate del percorso fornite per l'addestramento e le coordinate percorse dall'Agent 3 addestrato per path following di un singolo percorso e gestione dell'orientazione

Il grafico 2D in "Figura 88" è raffigura il confronto tra l'andamento della velocità fornita per l'addestramento "vx th" con la velocità che ha l'Agent lungo la coordinata x durante la simulazione "vx real":

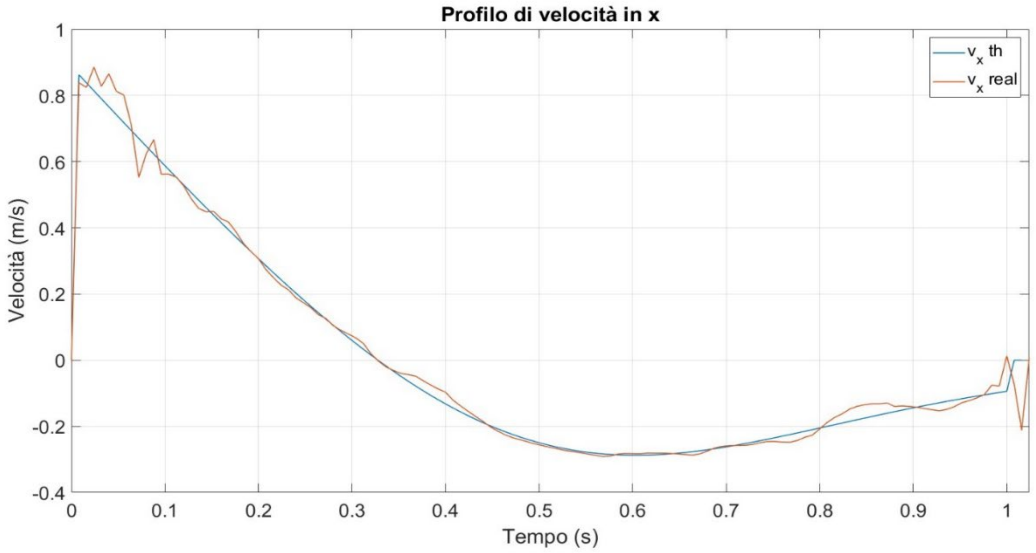

Figura 88: Confronto tra profilo di velocità della vx fornita e profilo di velocità risultante ottenuto dall'Agent 3 addestrato

Nel grafico 2D “Figura 89” si osserva il confronto tra l’andamento della velocità fornita per l’addestramento “vx th” con la velocità che ha l’Agent lungo la coordinata x durante la simulazione “vx real”:

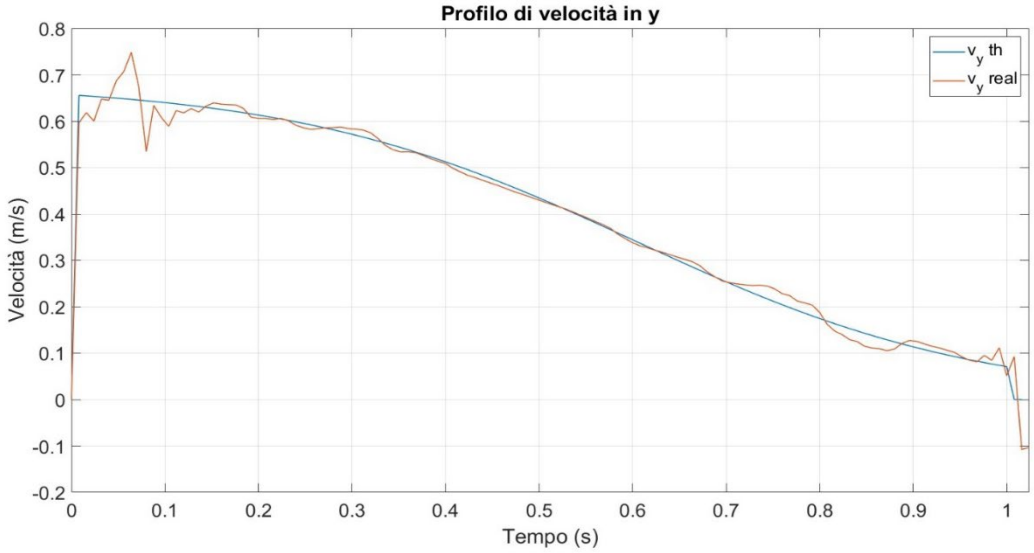

Figura 89:Confronto tra profilo di velocità della vy fornita e profilo di velocità risultante ottenuto dall'Agent 3 addestrato

Dalle rappresentazioni grafiche dei dati forniti dall’”Agent 3” si può confermare la bontà dell’approccio progressivo, osservando un andamento delle posizioni e dei profili di velocità più in linea con le indicazioni fornite rispetto ai dati precedentemente analizzati dell’Agent testato al termine del caso applicativo del solo Path Following.

Nella “Tabella 9” sono riportati gli errori di scostamento dei tre Agent campioni rispetto ai dati forniti per l’addestramento. Sono riportati gli errori di posizionamento, di scostamento dai profili di velocità, e l’errore relativo all’orientazione:

Agent	Errore sul percorso (m)	Errore sul profilo di velocità in x (m/s)	Errore sul profilo di velocità in y (m/s)	Errore sulla velocità (m/s)	Errore sull’orientazione (rad)
Agent 1	0.0011	0.0545	0.0715	0.1213	0.0524
Agent 2	0.0013	0.0239	0.0204	0.0394	0.0745
Agent 3	0.0009	0.0148	0.0186	0.0298	0.0467

Tabella 9: Errori relativi al confronto tra i dati ottenuti dall'Agent addestrato e i dati forniti

Si osserva come con le sessioni di affinamento l’orientazione vada migliorando. Un ulteriore miglioramento si potrebbe riscontrare eseguendo sessioni di allenamento con una diminuzione graduale sul valore della tolleranza di discostamento. Inoltre, osservando la “Tabella 10” si conferma l’efficacia dell’approccio progressivo dal miglioramento delle

prestazioni con aumento della precisione di posizionamento per il path following e dei profili di velocità.

Agent	Errore sul percorso (m)	Errore sul profilo di velocità in x (m/s)	Errore sul profilo di velocità in y (m/s)	Errore sulla velocità (m/s)	Errore sull'orientazione (rad)
Agent su singolo percorso	0.0012	0.0183	0.0208	0.0399	/
Agent con gestione orientazione	0.0009	0.0148	0.0186	0.0298	0.0467

Tabella 10: Confronto tra l'Agent ottimale per la sola gestione della posizione e l'Agent ottimale con l'integrazione della gestione dell'orientazione

4.3.3 Path following di un singolo percorso con gestione dell'orientazione e presenza del corpo

Ottenuto e consolidato un buon risultato sullo studio del path following di un singolo percorso con gestione dell'orientazione, si aggiunge la presenza dell'operatore, qui denominato "corpo", con coordinate coincidenti con un punto del percorso. Lo scopo di questo caso applicativo è quello di formare un Agent che sia in grado di valutare una traiettoria supplementare, ottimale, in modo da evitare il corpo e giungere al punto target. Questa casistica è stata esaminata per valutare se con l'addestramento e la struttura assegnata della Reward e del Reinforcement si raggiunge l'obiettivo prefissato di evitare il corpo per garantire la sicurezza dell'operatore.

Nel caso applicativo complessivo, descritto nel Capitolo 4.2, la componente R2 considerava nello stesso momento la zona di sicurezza, in cui si voleva una regolazione della velocità da parte del cobot, con al centro una zona di non percorrenza impostata per evitare la collisione con l'operatore. In linea con quanto detto per l'approccio progressivo si vuole comprendere la capacità e le possibilità dell'Agent di trovare traiettorie alternative. Si è quindi svolto l'addestramento considerando la sola zona di non percorrenza, per poi aggiungervi la regolazione della velocità nella zona di sicurezza.

Nella "Figura 90" è mostrato il caso studio del percorso da seguire con il corpo posto in un punto appartenente ad esso.

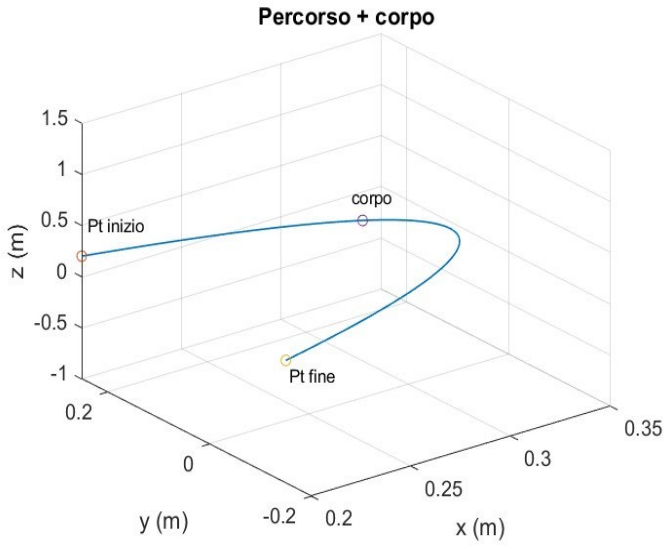

Figura 90: Rappresentazione grafica 3D del percorso da seguire e del corpo

La definizione dell'Environment rimane la stessa dei casi precedenti, con le Observations e l'Action fisse, e con l'uso dello stesso modello Simulink. È mantenuta la Reset Function invariata con l'unica aggiunta della definizione delle coordinate del corpo, come precedentemente detto, coincidenti con un punto del percorso.

La funzione Reward del caso applicativo precedente è modificata con l'aggiunta della condizione della zona di non percorrenza. Mediante il comando "if" si imposta la condizione di penalità fissando un controllo sulla distanza. La zona è stata definita di raggio pari a 5 cm e penalità di -200, valore assegnato all'Agent in caso di Action che lo portano in posizioni interne a quest'area.

Nella "Figura 91" è riportato l'andamento della Reward costruita:

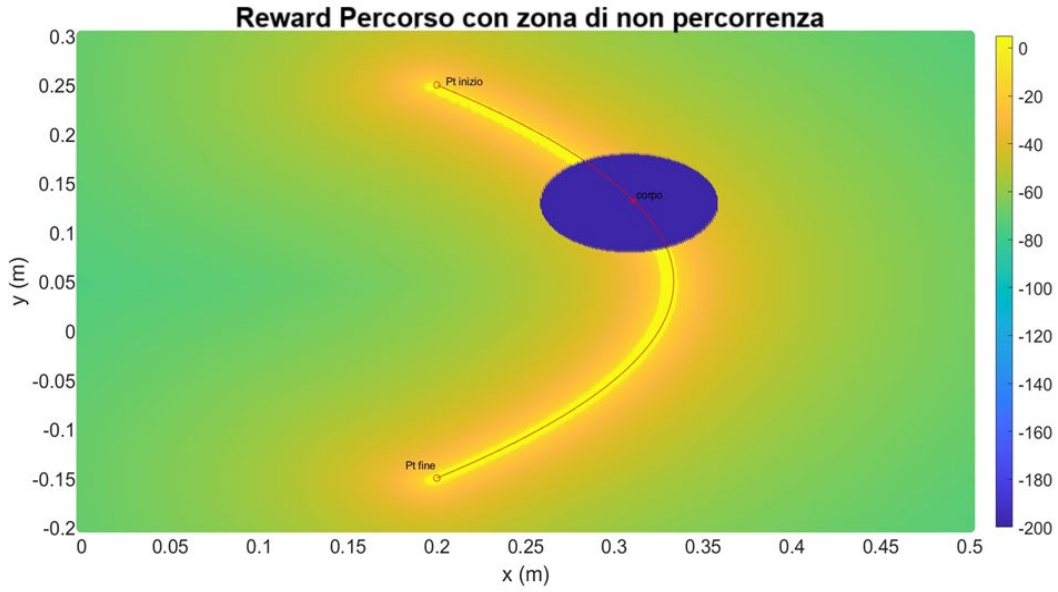

Figura 91: Reward per il path following e la zona di non percorrenza collocata nell'intorno del corpo

Nella "Tabella 11" sono riportate le sessioni di allenamento svolte per il caso applicativo in analisi:

Max Episode	Gaussian Std. deviation	Gaussian Upper/Lower Limit	Target Std. deviation	Target Upper/Lower Limit
10000/10000	0.8	(+/-) 4	0.7	(+/-) 0.9
10000/10000	0.7	(+/-) 3	0.7	(+/-) 0.9
8000/8000	0.8	(+/-) 3	0.7	(+/-) 0.9
6000/6000	0.7	(+/-) 3	0.7	(+/-) 0.9

Tabella 11: Sessioni di allenamento e i parametri utilizzati per l'addestramento

Rispetto ai casi applicativi precedenti sono state impostate più sessioni di allenamento con la finalità di esplorare opportunamente l'environment. Questo approccio è giustificato dalla complessità del caso applicativo, in cui l'Agent è costretto a discostarsi dalla traiettoria di riferimento per evitare la collisione con il corpo. Esso dovrà esplorare alla ricerca di tutte le possibili soluzioni e valutare quella ottimale basandosi sui risultati della Reward.

Nella “Figura 92” si mostra l’andamento del Training per l’addestramento dell’Agent nel caso applicato di evitamento del corpo:

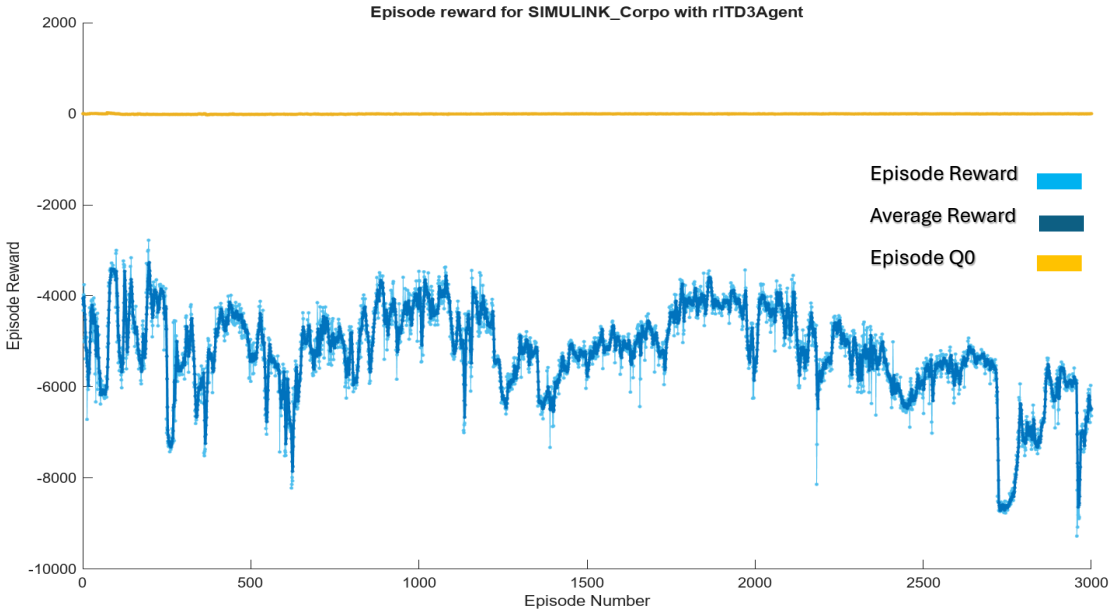

Figura 92: Sessione di allenamento per ottenere un Agent capace di evitare il corpo mantenendo una traiettoria simile a quella indicata

Dal Training raffigurato si può notare come l’allenamento non converga con andamento instabile su valori di Reward negativi. Si è effettuata una simulazione con l’Agent addestrato con lo scopo di osservare se riesce o meno ad evitare la zona di non percorrenza. Inoltre, si cerca di verificare che l’Agent percorra una traiettoria simile a quella indicata e che riesca a giungere al punto finale.

Nella “Figura 93” è raffigurato il confronto tra le coordinate del percorso seguito dall’Agent confrontate con le coordinate del percorso indicato per l’addestramento:

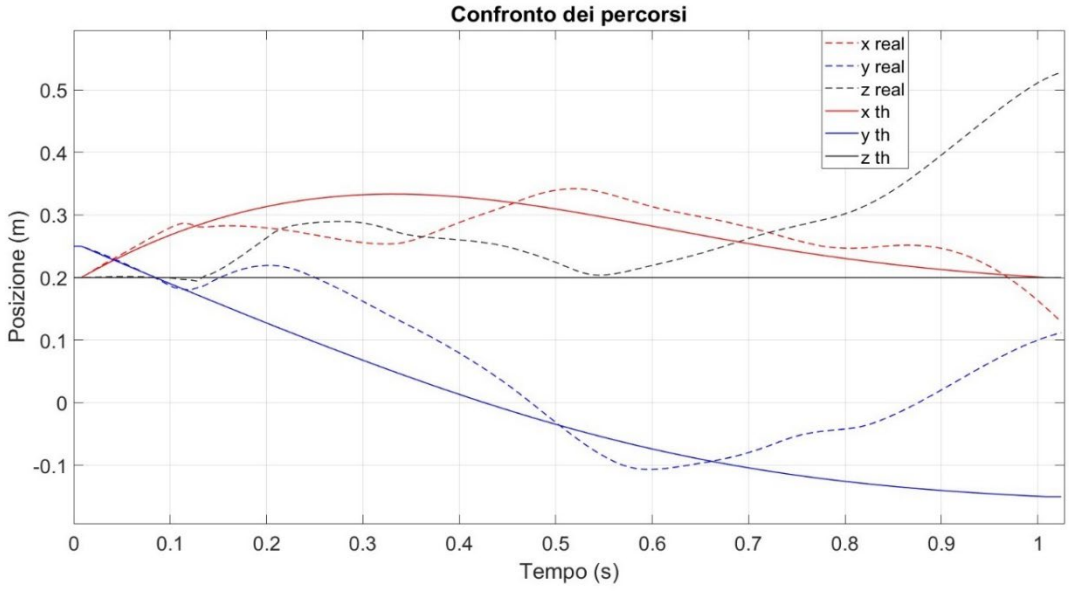

Figura 93: Confronto delle coordinate del percorso ottenuto con l'Agent rispetto alle coordinate fornite per il Training

Si osserva dal grafico come l'Agent addestrato non riesca a raggiungere il punto target della traiettoria. Essendo un caso complesso potrebbero essere necessarie più sessioni di allenamento per esplorare in modo opportuno l'Environment. Un'altra soluzione consisterebbe nel riformulare l'equazione della Reward, in modo che gestisca differentemente il corpo e che non porti l'Agent a trascurare il comportamento relativo al Path Following e alla gestione dell'orientazione.

4.3.4 Path following di un singolo percorso con gestione dell'orientazione, presenza del corpo e regolazione della velocità

Il passo successivo dello studio è l'inserimento di una zona di sicurezza attorno al corpo. L'obiettivo dell'integrazione è fornire un'ulteriore funzione di sicurezza per garantire l'incolumità dell'operatore. Si otterrà una zona detta di "sicurezza" in cui si vuole una regolazione di percorrenza del cobot che dovrà avere un valore di velocità nell'intorno del valore di riferimento indicata "v_rif". Questa funzione di "gestione della velocità" è aggiunta alla zona di non percorrenza, interna alla zona di sicurezza, poiché si vuole sempre evitare che il cobot, pur a velocità basse, collida con l'operatore.

Per la definizione dell'Environment vengono mantenute le Observations e l'Action inalterate e viene utilizzata la Reset function del caso applicativo precedente. Il modello di Simulink dell'Environment viene modificato (Figura 94), aggiungendo la velocità nello spazio cartesiano del cobot come input per la "Reward Function":

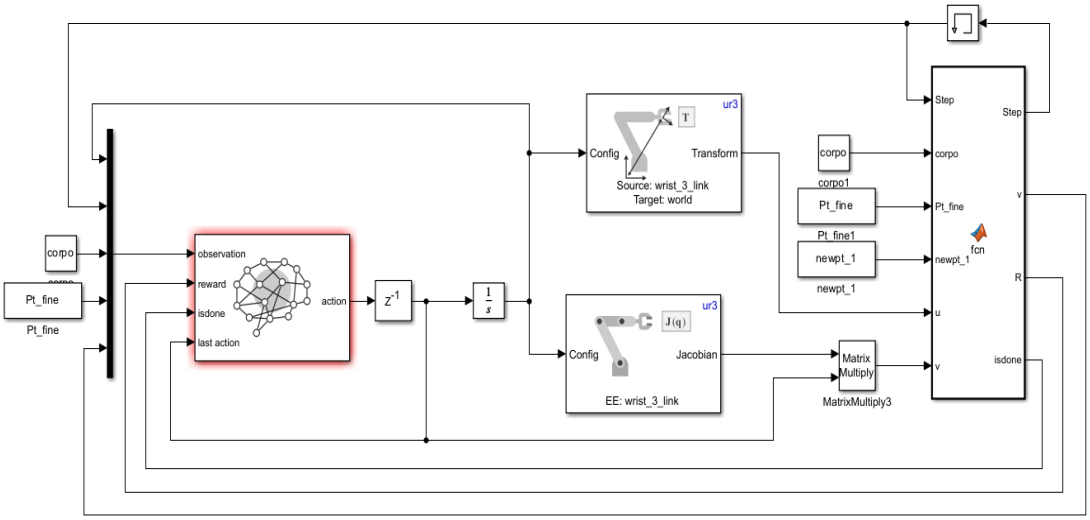

Figura 94: Modello Simulink per il caso applicativo di path following e presenza del corpo con gestione della velocità

Per la Reward viene considerata la componente R2 del caso complessivo, descritto nel capitolo 4.2, in cui si tiene conto della zona di sicurezza più la presenza della zona di non percorrenza.

$$(15) \quad R2 = -k \cdot \left(a \cdot e^{(b \cdot dist\ corpo)}\right) (dist\ corpo > 0.05)$$

$$(16) \quad R2 = -100\ (dist\ corpo \leq 0.05)$$

È riportato nella “Figura 95” il grafico 2D della Reward per il percorso imposto con la presenza del corpo e relativa zona di sicurezza più zona di non percorrenza. Per questo caso la velocità di riferimento è stata impostata pari a v_rif = 0.3 m/s e velocità dell’UR3 pari a v_ur=1m/s.

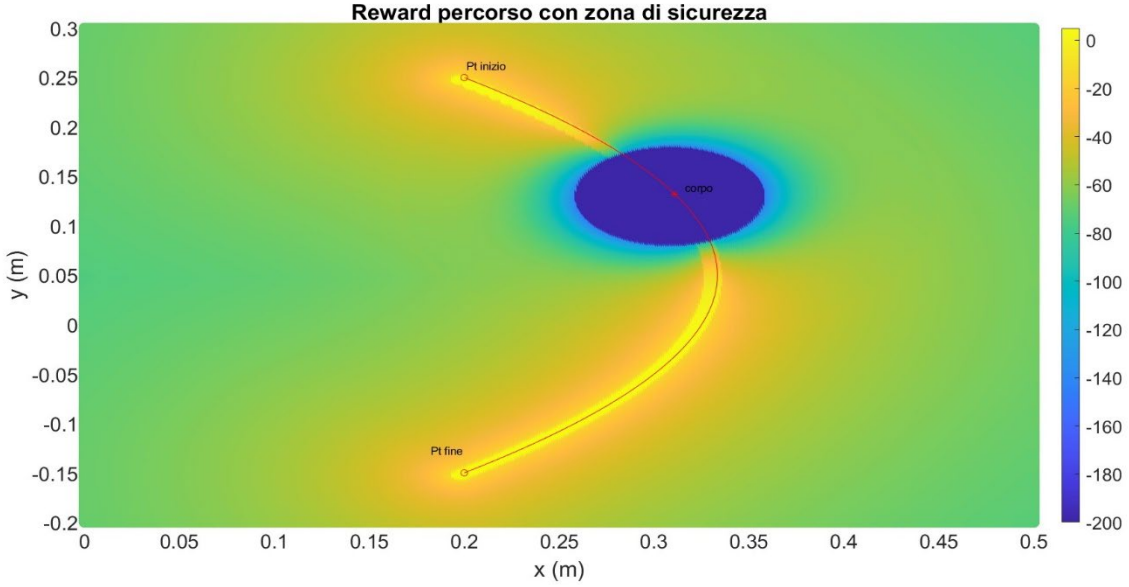

Figura 95: Andamento della Reward per il path following, presenza del corpo e gestione della velocità

4.3.5 Path following di un singolo percorso con gestione dell'orientazione, presenza del corpo, regolazione della velocità e verifica delle collisioni

Con i casi precedenti si ottiene un Agent in grado di gestire la presenza dell'operatore sul percorso trovando nuove traiettorie. Per queste nuove possibilità deve essere eseguito un controllo delle configurazioni assunte dal cobot per verificare l'assenza delle collisioni, con lo scopo di escludere quelle inadeguate che creano conflitto e la non corretta esecuzione del compito da parte del cobot.

La definizione delle Observations e dell'Action rimane la stessa, come la Reset Function che non viene modificata rispetto all'applicazione precedente. Nel modello Simulink (Figura 96) è inserito il blocco "Matlab System" in cui è stato progettato un codice per effettuare un controllo delle configurazioni assunte negli step temporali per l'esecuzione delle traiettorie alternative ricavate per evitare il corpo. Questo blocco verifica sia le collisioni interne, tra i link del cobot, e sia le collisioni con il pavimento.

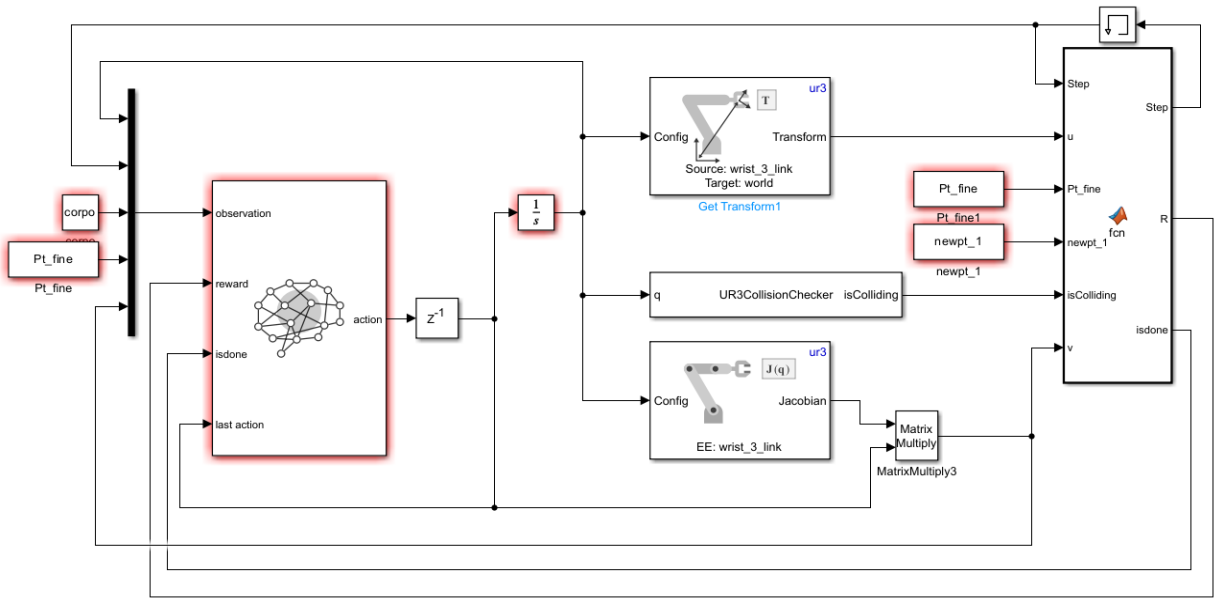

Figura 96: Modello Simulink per il caso applicativo di path following e presenza del corpo con gestione della velocità e delle collisioni

L'informazione ricavata da questo blocco diventa un input della "Reward Function" con cui addestrare il cobot.

La Reward è stata progettata in modo tale che, in caso di avvenuta collisione, viene portato al termine l'episodio prima del tempo tramite comando "isdone=true", La funzione di Reward è stata costruita considerando un peso negativo moltiplicato per la differenza tra il valore degli step eseguiti ed il valore degli step totali del percorso. Inizialmente, era stata assegnata una penalità costante per le collisioni. Tuttavia, si è osservato che l'Agent optava

di collidere all'inizio del percorso, evitando di accumulare le penalità relative ad altre componenti della reward sommate alla costante della penalità della collisione verso la fine del percorso, essendo la collisione una criticità. In caso di assenza di collisioni per tutto l'episodio, la Reward function assegna una ricompensa all'Agent in modo da addestrarlo ad escludere le traiettorie con collisioni.

4.4 Generalizzazione del modello

Per allenare l'Agent in modo da renderlo in grado di affrontare più situazioni di funzionamento, si agisce sulla definizione del modello cercando di generalizzare le casistiche di allenamento. A tale scopo si può lavorare sui punti estremi della traiettoria e sulle coordinate del corpo.

Questo studio è stato strutturato basando le scelte sull'Approccio Progressivo, affrontando la generalizzazione dei parametri singolarmente, in modo da garantire una buona stabilità del modello ed una maggiore comprensione delle limitazioni o delle potenzialità del Reinforcement Learning.

Le casistiche sviluppate e non ancora analizzate sono:

- Generalizzazione del percorso tramite definizione casuale del punto iniziale e finale per ogni episodio di addestramento. Nel particolare le coordinate di questi punti vengono selezionati dall'area di lavoro del cobot, ristretta ad una semicirconferenza su un piano di coordinata z fissa. Questa casistica potrà essere in seguito resa ancora più complessa rendendo variabile anche la coordinata z;

- Generalizzazione del corpo con la definizione delle sue coordinate generate casualmente per ogni episodio di addestramento. Per questa casistica sono state concepite più strategie in linea con la filosofia di approccio progressivo. L'approccio più generale consiste nella generazione casuale del corpo all'interno dell'area di lavoro del cobot, esaminando casistiche di assenza dell'influenza del corpo sulla traiettoria, parziale influenza per prossimità del corpo con regolazione di velocità, e presenza del corpo sul percorso del dispositivo.

Per la definizione del caso generalizzato viene impostata una Reset Function in cui si effettua la generazione casuale dei punti iniziale e finale con coordinata fissa z=0.2. Nella Reset Function è stato impostato un controllo per verificare la corretta generazione del punto iniziale e della configurazione che avrà il robot in questo punto. Assegnando l'orientazione desiderata, si calcola tramite cinematica inversa la configurazione iniziale in spazio giunti, poi verificata per collisioni interne tramite funzione "checkCollision". In caso di avvenuta collisione si procede al ricalcolo del punto iniziale, e quindi al ricalcolo casuale

delle sue coordinate x e y nell'area di lavoro, per poi eseguire la verifica di collisioni sulla nuova configurazione iniziale. Questo processo viene ripetuto fino all'ottenimento di una configurazione iniziale priva di collisioni.

Nel caso applicativo della generalizzazione non sono forniti i punti del percorso in modo diretto, ma si calcoleranno nella Reset Function partendo dal profilo di velocità desiderato per la percorrenza sulla coordinata x. Tramite integrazione del profilo di velocità in x si otterranno le coordinate in x da normalizzare ed inserire nella funzione matematica scelta, descrittiva del profilo del percorso (Figura 97). In questo caso la funzione scelta è stata ottenuta dal percorso utilizzato dal path following.

$$y\_norm=((-(2*x\_norm-1)^2)+1)/3 \quad (17)$$

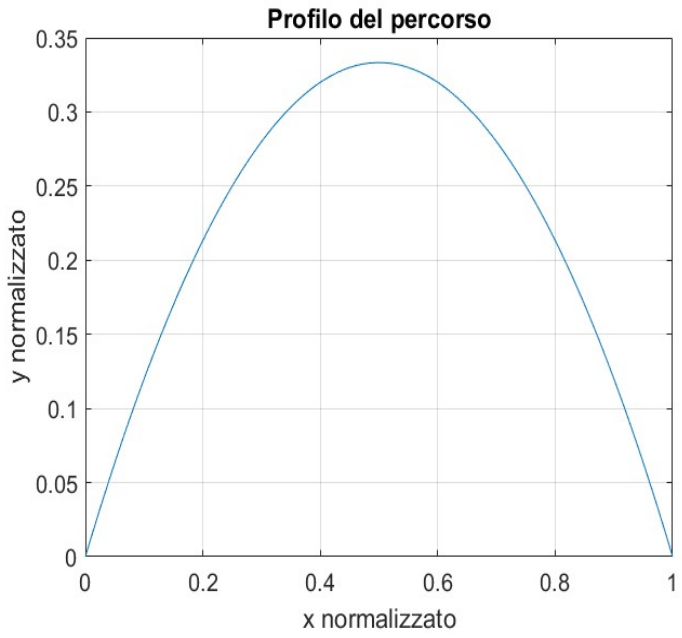

Figura 97: Percorso normalizzato ottenuto con l'utilizzo della funzione matematica y_norm

Da qui si estenderà il percorso considerando la distanza tra punto inziale e quello finale e si ricaverà il profilo di velocità in y. Infine, considerando le coordinate del punto iniziale e finale sarà calcolata l'inclinazione sul piano x-y e si eseguirà la rotazione del percorso e dei profili di velocità. Con questa procedura si otterrà sempre il percorso descritto dalla funzione matematica fissata e sarà eseguito con il profilo di velocità in x, fornito, e y, ricavato.

Ottenuta la traiettoria si effettua un'analisi per verificare che i punti del percorso appartengano all'area di lavoro del cobot, in particolare evitando la zona relativa a dove si trova la base del cobot. Si verifica che nessuno dei punti del percorso sia ad una distanza inferiore a 0.08 metri dal punto di origine (0,0,0). In caso contrario si effettua un ricalcolo

del punto iniziale e finale, processo ripetuto fino all'ottenimento di punti al di fuori dell'area interna alla circonferenza di raggio pari a 0.08 metri.

Al termine dell'analisi preliminare si otterranno i punti in cui dovrà essere il cobot ad ogni sample time, tempo di campionamento.

Una volta addestrato l'Agent per una delle due casistiche si integrerà la generalizzazione ancora non applicata. Questo viene fatto con lo scopo di avvicinare il modello al caso applicativo complesso, descritto nel capitolo 4.2, per ottenere dallo studio il modello di pianificatore di traiettoria vicino a quello target.

5. CONCLUSIONI

In questo elaborato si è posto l'obiettivo di sviluppare un pianificatore di traiettoria per robot antropomorfi basato sul Reinforcement Learning, in grado di adattarsi dinamicamente alle condizioni di lavoro e di gestire in sicurezza la presenza dell'operatore.

Il primo passo è stato creare un canale di comunicazione per comandare il cobot. Questo è stato realizzato collegando il cobot UR3 all'ambiente di programmazione Matlab/Simulink tramite il software ROS. Dopo aver stabilito il canale di comunicazione tra i due dispositivi, sono state sviluppate pianificazioni di traiettoria per eseguire movimenti semplici, analizzando le diverse funzioni fornite da Universal Robots per comandare l'UR3 tramite Matlab. L'osservazione dei feedback ottenuti dal robot, confrontati con i dati ricavati dalla pianificazione della traiettoria, ha confermato il corretto collegamento, con precisione dei movimenti.

Creato il collegamento è stata approfondita l'architettura del Reinforcement Learning, descrivendone dettagliatamente la struttura e le caratteristiche principali. Dopodiché, si è passato alla progettazione del modello per l'addestramento del pianificatore di traiettoria, con la definizione delle Observations, dell'Action e della Reset Function, tutti elementi necessari per la costruzione dell'Environment. È stata sviluppata una Reward function multicomponente, cruciale per guidare l'apprendimento dell'Agent verso il comportamento desiderato. Le componenti della Reward presentate nell'elaborato sono state progettate in modo da considerare le diverse esigenze del sistema, gestendo situazioni operative complesse, inclusa la presenza dell'operatore nell'ambiente, la variabilità delle traiettorie e la gestione di eventuali collisioni interne ed esterne.

Durante le sessioni di Training del caso complesso si è riscontrata la difficoltà nel verificare la corretta esecuzione del comportamento desiderato e l'elevata instabilità dell'andamento dell'Agent. Inoltre, essendoci più componenti della Reward, queste dovranno essere caratterizzate da un valore numerico indicativo del peso di priorità. Un bilanciamento non corretto porterebbe a privilegiare alcuni comportamenti e trascurarne degli altri.

Per affrontare queste problematiche, si è adottato un approccio progressivo, analizzando modelli semplificati del caso complesso, per verificare la correttezza delle funzioni assegnate alle componenti della Reward e i relativi pesi.

Dall'analisi del caso studio di path following su singolo percorso, dopo aver cambiato la funzione di Reward da una funzione lineare crescente ad una logaritmica, in grado di gestire anche le piccole distanze, si è riusciti ad ottenere un Agent capace di eseguire il percorso indicato in poche sessioni di allenamento. L'analisi è stata condotta testando l'Agent con un numero di sessioni di addestramento crescente e confrontando i feedback prodotti dall'Agent nelle diverse fasi di allenamento. Durante i test si è riscontrato un maggiore scostamento nel caso dell'Agent addestrato con meno sessioni. Questo errore si è visto poi decrescere aumentando il numero delle sessioni di allenamento, evidenziando

l'importanza della fase di affinamento necessaria per ottenere un comportamento in linea con le indicazioni fornite con un fattore di errore molto basso.

Ottenuto un buon risultato dal Path Following, è stata integrata la gestione dell'orientazione con l'obiettivo di mantenere la configurazione di partenza con il cobot con l'asse uscente dalla flangia perpendicolare al pavimento. Durante questa casistica è emersa l'inadeguatezza dell'utilizzo di funzioni di Reward discrete, che hanno portato alla stabilizzazione dell'addestramento in corrispondenza di valori negativi di Reward ad evidenziare un comportamento lontano da quello atteso. Questo ha portato alla definizione di una funzione logaritmica continua con l'obiettivo di guidare l'Agent verso condizioni di funzionamento ottimali. Tramite l'inserimento della componente della Reward relativa alla funzione logaritmica per la gestione dell'orientazione si sono ottenuti dei buoni risultati, migliorabili con la riduzione della tolleranza di errore. Inoltre, si è verificata l'efficacia dell'approccio progressivo osservando una riduzione nell'errore della posizione dell'Agent rispetto al percorso fornito per l'addestramento, indicando una continuazione dell'addestramento per il Path Following, non penalizzato dalla presenza della componente per la gestione dell'orientazione.

Nel caso del Path Following con controllo dell'orientazione le reward dei due comportamenti non implicano un peggioramento l'uno dell'altro, diversamente da quello che avviene nel caso applicativo dell'evitamento di un ostacolo posizionato sul percorso, dove la gestione della reward richiede maggiore attenzione. È essenziale ottenere un valore positivo quando il cobot esegue una traiettoria che, sebbene diversa da quella indicata, rimane vicina ad essa, rispettando in parte il path following e allo stesso tempo evitando l'ostacolo. In questo contesto, la Reward per gestire la presenza del corpo è stata affrontata tramite funzione discreta, fornendo una penalità alta quando il cobot entra nella zona di non transito. È necessario effettuare molte più sessioni di addestramento per validare l'efficacia della Reward discreta o per progettare una nuova funzione di Reward continua.

Nell'elaborato sono state progettate e strutturate le componenti necessarie alla costruzione dei modelli risolutivi delle casistiche successive dell'approccio progressivo. Queste possono essere integrate una alla volta ai casi precedenti per verificare la correttezza della loro definizione e gestire comportamenti che avvicinano l'Agent al comportamento descritto dal caso complesso. Sono state indicate anche le impostazioni per la generalizzazione del problema passando dall'esecuzione di un singolo percorso all'esecuzione di un qualsiasi percorso nello spazio di lavoro del cobot. Essendo un approccio progressivo e non avendo ottenuto una convergenza nell'analisi della presenza del corpo non è stato possibile procedere all'integrazione e verifica dei componenti successivi della Reward.

L'obiettivo della sperimentazione è quello di ottenere un pianificatore di traiettoria che esegua una traiettoria efficace gestendo la presenza di un corpo, ovvero, ottenere una traiettoria ottimale assegnati il punto inziale, finale e posizione dell'ostacolo. Il caso

applicativo risulta essere complesso e la risoluzione prevede l'utilizzo di un elevato numero di sessioni di allenamento. Per i casi semplificati si è riuscito ad ottenere l'addestramento di un Agent in linea con il comportamento indicato, a fronte di un elevato numero di sessioni di training. Si vuole sottolineare che all'aumentare della complessità del percorso desiderato le sessioni di addestramento dell'Agent sarò in numero ancora più elevato. Utilizzando dispositivi con maggior numero di processori e tramite l'utilizzo di server si potrebbero ottenere dei buoni risultati in minor tempo.

In conclusione, non si può affermare con certezza l'efficacia del Reinforcement Learning per il raggiungimento del nostro scopo. Si è giunti alla risoluzione di casi applicativi risolvibili anche con metodologie dirette ma è stato eseguito il primo passo per poter arrivare a questo obiettivo e per la comprensione delle potenzialità e dei limiti del metodo applicato a questo contesto. L'approccio del Reinforcement Learning permette di gestire un'elevata quantità di parametri regolabili nelle sessioni di training dell'Agent consentendo di affrontare casistiche di diversa complessità. Tuttavia, la corretta regolazione dei parametri richiede una conoscenza approfondita del metodo e una comprensione dettagliata delle variazioni indotte dalla modifica di questi. Una delle difficoltà riscontrate nell'utilizzo del metodo è stata quella di gestire gli allenamenti, i quali richiedono il corretto bilanciamento dei parametri oltre che tempi considerevoli per l'allenamento dell'Agent. Inoltre, l'incertezza nella definizione di un'opportuna funzione di Reward porta alla necessità di eseguire sessioni di Training di casi semplificati, come è stato eseguito nell'elaborato, per convalidare la funzione definita portando ad un ulteriore aumento dei tempi per l'addestramento dell'Agent desiderato. Il caso applicativo studiato è complesso e richiede un elevato numero di sessioni di addestramento, ma potrebbe essere semplificato definendo alcuni parametri della struttura del Reinforcement Learning in modo diverso, come l'Action, tramite cui in questo caso l'Agent definisce le velocità angolari dei sei giunti. Metodi alternativi potrebbero consistere nel fornire al cobot una traiettoria parametrizzata in cui l'Agent agisca fornendo il parametro, oppure limitare la gestione a un numero inferiore di giunti. Nonostante questo, si è riuscito ad ottenere con il Reinforcement Learning la risoluzione del caso complesso del Path following del singolo percorso tramite la gestione delle velocità dei sei giunti, confermando la capacità del metodo di ottenere buoni risultati per casistiche complesse. Non è possibile garantire che le soluzioni ottenute con l'utilizzo del Reinforcement Learning siano le migliori tra quelle ottenibili con l'utilizzo di altre metodologie. Tuttavia, il Reinforcement Learning rappresenta un approccio potente e versatile per la risoluzione del problema affrontato.

Appendice

Appendice A: Pianificazione della traiettoria

Codice Inizializzazione

```
%linee di codice necessarie per creare il collegamento tra software e
robot
clear;
ur3 = loadrobot('universalUR3');

%URSim
interface = "URSim"
ROSDeviceAddress = '192.168.76.210';
robotAddress = '127.0.0.1';

%robot real
% interface="Hardware"
% ROSDeviceAddress = '192.168.56.104';
% robotAddress = '192.168.56.103';

username = 'ur';
password = '***********';
ROSFolder = '/opt/ros/noetic';
WorkSpaceFolder = '~/ur_ws';
device = rosdevice(ROSDeviceAddress,username,password);
device.ROSFolder = ROSFolder;
generateAndTransferLaunchScriptGettingStarted(device,WorkSpaceFolder,i
nterface,robotAddress);


if ~isCoreRunning(device)
    w = strsplit(system(device,'who'));
    displayNum = cell2mat(w(2));

    system(device,['export SVGA_VGPU10=0; ' ...
        'export DISPLAY=' displayNum '.0; ' ...
        './launchUR.sh &']);
    pause(10);
end

ur = universalrobot(ROSDeviceAddress,'RigidBodyTree',ur3);
jointAngles = getJointConfiguration(ur,10);
show(ur.RigidBodyTree,jointAngles);
pose = getCartesianPose(ur);
jointVelocity = getJointVelocity(ur);
eeVelocity = getEndEffectorVelocity(ur);
```

Cobot comandato in spazio giunti

```
%CASO SENDJOINTCONFIGURATION
jointWaypoints = [60*((pi)/180) -90*((pi)/180) -70*((pi)/180) -
20*((pi)/180) 70*((pi)/180) 0*((pi)/180)];
sendJointConfiguration(ur,jointWaypoints);


[result,~] = getMotionStatus(ur);
confgiunti = [];
velgiunti = [];
time =[];
posizione=[];
posaistant= [];
tic;
while ~result
    [result,~] = getMotionStatus(ur);
    tempotrascorso=toc;
    time = [time, tempotrascorso];
    jointconf = getJointConfiguration(ur);
    confgiunti = [confgiunti,jointconf'];
    jointvel = getJointVelocity(ur);
    velgiunti = [velgiunti,jointvel'];
%verifica scostamento dalla traiettoria rettilinea
    posaistant= getCartesianPose(ur);
    posizione=[posizione,posaistant(:,4:6)'];
end
```

Traiettoria Rettilinea

```
%traiettoria rettilinea

%cinematica inversa
ik = inverseKinematics('RigidBodyTree', ur3);
%tempo impiegato per raggiungere ogni punto
weights = [1, 1, 1, 1, 1, 1];
x=0.4.*ones(1,100)';
y=linspace(-0.2,0.2,100)';
z=0.3.*ones(1,100)';
points =[x,y,z];


jointWaypoints = jointAnglesA;
numJoints = size(jointWaypoints',1);
numWaypoints = size(points,1);
%matrice di trasformazione dell'end effector rispetto alla base
T_home = getTransform(ur3, jointWaypoints, 'ee_link') ;
%matrice di zeri con righe pari al numero di punti della traiettoria
%e colonne pari al numero di giunti
```

```
qs = zeros(numWaypoints,numJoints);
i=1;
for i = 1:numWaypoints
    T_des = T_home;
    %si prendono le coordinate relative alla posizione
    T_des(1:3,4) = points(i,:)';
    [q_sol, q_info] = ik('ee_link', T_des, weights, jointWaypoints);
    qs(i,:) = q_sol(1:numJoints);
    jointWaypoints=q_sol(1:numJoints);
end
sendJointConfigurationAndWait(ur,qs(1,:),'EndTime',10);

timePoints=(0:5/(length(points(:,1))-1):5)';
trajTimes = (0:1/125:5);
[qt,qdt,qddt] = bsplinepolytraj(qs',[0 timePoints(end)],trajTimes');
followTrajectory(ur,qt,qdt,qddt,trajTimes);

[result,~] = getMotionStatus(ur);
confgiunti = [];
velgiunti = [];
time =[];
posizione=[];
posaistant= [];
tic;
while ~result
    [result,~] = getMotionStatus(ur);
    tempotrascorso=toc;
    time = [time, tempotrascorso];
    jointconf = getJointConfiguration(ur);
    confgiunti = [confgiunti,jointconf'];
    jointvel = getJointVelocity(ur);
    velgiunti = [velgiunti,jointvel'];
    posaistant= getCartesianPose(ur);
    posizione=[posizione,posaistant(:,4:6)'];
end
```

Traiettoria triangolare

```
%traiettoria triangolare
A=[0.4,0,0.4];
%cinematica inversa
ik = inverseKinematics('RigidBodyTree', ur3);

weights = [1, 1, 1, 1, 1, 1];
%acquisizione posizione angolare giunti del punto di partenza e di
arrivo
jointAnglesA = getJointConfiguration(ur,10);
pause(2);
jointWaypoints = jointAnglesA;
numJoints = size(jointWaypoints',1);
```

```
numWaypoints = size(A,1);
%matrice di trasformazione dell'end effector rispetto alla base
T_home = getTransform(ur3, jointWaypoints, 'ee_link') ;
%matrice di zeri con righe pari al numero di punti della traiettoria
%e colonne pari al numero di giunti
qs = zeros(numWaypoints,numJoints);

    T_des = T_home;
    %si prendono le coordinate relative alla posizione
    T_des(1:3,4) = A(1,:)';
    [q_sol, q_info] = ik('ee_link', T_des, weights, jointWaypoints);
    qs(1,:) = q_sol(1:numJoints);
    jointWaypoints=q_sol(1:numJoints);


sendJointConfigurationAndWait(ur,qs(1,:),'EndTime',10);



% definizione delle pose necessarie alla traiettoria triangolare su
piano
taskWayPoints = [-pi/2    0  -pi/2 0.4     0  0.4;
                 -pi/2    0  -pi/2 0.4     0.1  0.4;
                 -pi/2    0  -pi/2 0.4   0  0.45;
                 -pi/2    0  -pi/2 0.4     -0.1  0.4;
                  -pi/2    0  -pi/2 0.4  0 0.4;];


wayPointTimes = [0 2 4 6 8];
% comando per l'esecuzione del movimento
followWaypoints(ur,taskWayPoints,wayPointTimes,'InterpolationMethod','
quinticpolytraj','NumberOfSamples',100);
[result,~] = getMotionStatus(ur);


[result,~] = getMotionStatus(ur);
confgiunti = [];
velgiunti = [];
time =[];
posizione=[];
posaistant= [];
% ciclo while necessario per l'acquisizione dei dati durante
l'esecuzione del movimento da parte del robot
tic;
while ~result
    [result,~] = getMotionStatus(ur);
    tempotrascorso=toc;
    time = [time, tempotrascorso];
    jointconf = getJointConfiguration(ur);
    confgiunti = [confgiunti,jointconf'];
```

```
    jointvel = getJointVelocity(ur);
    velgiunti = [velgiunti,jointvel'];
    posaistant= getCartesianPose(ur);
    posizione=[posizione,posaistant(:,4:6)'];
end

%riposizionamento
sendJointConfigurationAndWait(ur,jointAnglesA,'EndTime',10);
```

Traiettoria circolare

```
%traiettoria circolare
ik = inverseKinematics('RigidBodyTree', ur3);
%per definire la traiettoria circolare
%definire centro e raggio
center = [0.3  0  0.4]; %[x y z]
radius = 0.05;
%tempo impiegato per raggiungere ogni punto
dt = 0.1;
tempo = (0:dt:10)';
%definizione dell'angolo e dei punti
weights = [1, 1, 1, 1, 1, 1];
theta = tempo*(2*pi/tempo(end))-(pi/2);
points = center + radius*[0*ones(size(theta)) cos(theta) sin(theta)];
numJoints = size(jointWaypoints',1);
numWaypoints = size(points,1);
%matrice di trasformazione dell'end effector rispetto alla base
T_home = getTransform(ur3, jointWaypoints, 'ee_link') ;
%matrice di zeri con righe pari al numero di punti della traiettoria
%e colonne pari al numero di giunti
qs = zeros(numWaypoints,numJoints);
i=1;
for i = 1:numWaypoints
    T_des = T_home;
    %si prendono le coordinate relative alla posizione
    T_des(1:3,4) = points(i,:)';
    [q_sol, q_info] = ik('ee_link', T_des, weights, jointWaypoints);
    qs(i,:) = q_sol(1:numJoints);
    jointWaypoints=q_sol(1:numJoints);
end
%posizionamento iniziale
sendJointConfigurationAndWait(ur,qs(1,:),'EndTime',10);

%comando per l’esecuzione del movimento
followTrajectory(ur,qt,qdt,qddt,trajTimes);

[result,~] = getMotionStatus(ur);
confgiunti = [];
velgiunti = [];
time =[];
```

```
posizione=[];
posaistant= [];
tic;
while ~result
    [result,~] = getMotionStatus(ur);
    tempotrascorso=toc;
    time = [time, tempotrascorso];
    jointconf = getJointConfiguration(ur);
    confgiunti = [confgiunti,jointconf'];
    jointvel = getJointVelocity(ur);
    velgiunti = [velgiunti,jointvel'];
%verifica scostamento dalla traiettoria rettilinea
    posaistant= getCartesianPose(ur);
    posizione=[posizione,posaistant(:,4:6)'];
end
```

Appendice B: Reinforcement Learning

Definizione Environment

Codice utilizzato in tutti i casi applicativi

```
ur3=loadrobot("universalUR3");
% Si apre il modello simulink relativo all'environment
open_system('SIMULINK_Progressivo1');

% Definizione delle Observations
ObservationInfo = rlNumericSpec([19 1]);
ObservationInfo.Name = "toss info";
ObservationInfo.Description = 'goal, feedback';

% Imposta i limiti per i primi tre giunti a +-pi e gli altri tre a +-
2*pi
ActionInfo= rlNumericSpec([6,1],"LowerLimit",[-pi;-pi;-pi;-2*pi;-
2*pi;-2*pi],"UpperLimit",[pi;pi;pi;2*pi;2*pi;2*pi]);
ActionInfo.Name = "velocità giunti";


% Generazione dell'Environment
env = rlSimulinkEnv("SIMULINK_Progressivo1","SIMULINK_Progressivo1/RL
Agent",ObservationInfo,ActionInfo);
% Collegamento della Reset Function
env.ResetFcn = @(in)ResetFcn(in);
ur3.DataFormat='column'
```

Caso Applicativo

Reset Function

```
function in=ResetFunc(in)
r_min=0.05;
r_max=0.45;
theta_min=-pi;
theta_max=+pi;


r=rand(1)*(r_max-r_min)+r_min;
theta=rand(1)*(theta_max-theta_min)+theta_min;
Pt_fine=[r*cos(theta),r*sin(theta),0.2]';


corpo =[rand(1)*0.4, rand(1)*0.4, rand(1)*0.4]';
ur3=loadrobot("universalUR3");


collisionFree=true;
while collisionFree==true
r=rand(1)*(r_max-r_min)+r_min;
theta=rand(1)*(theta_max-theta_min)+theta_min;
Pt_iniz=[r*cos(theta),r*sin(theta),0.2]';


ik=inverseKinematics('RigidBodyTree',ur3);
weights= [1,1,1,1,1,1];
jointWaypoints = homeConfiguration(ur3);


% Calcola il vettore direzione
vettore_direzione = Pt_fine - Pt_iniz;
% Calcola l'angolo tra l'asse x e il vettore direzione
angolo_z= atan2(vettore_direzione(2), vettore_direzione(1));
angolo_y = atan2(vettore_direzione(3), norm(vettore_direzione));
% Converti l'angolo di rotazione in forma di vettore di angoli di
Eulero
angoli_euleroz = [angolo_z, 0, 0];
% Converti gli angoli di Eulero in una matrice di trasformazione
utilizzando eul2tform
tformz = eul2tform(angoli_euleroz);
angoli_euleroy = [0, -angolo_y, 0];
% Converti gli angoli di Eulero in una matrice di trasformazione
utilizzando eul2tform
tformy = eul2tform(angoli_euleroy);
tform=tformy*tformz;
% %cambio la posizione
tform(1:3,4)=Pt_iniz';
 [q_sol,q_info]=ik('ee_link',tform,weights,jointWaypoints);
```

```
% %ottengo la configurazione iniziale dei giunti
for i=1:length(q_sol)
    Initial_condition(i)=q_sol(i).JointPosition;
end
% Imposta la proprietà DataFormat su 'column' o 'row'
ur3.DataFormat = 'column'; % o 'row'
collisionFree = checkCollision(ur3, Initial_condition');
ur3.DataFormat = 'struct';
end



%PERCORSO
x_norm=linspace(0,1,1000)';
a=0.26;
b=0.86;
y=zeros(size(x_norm));
z_norm=a*x_norm - (x_norm-0.8).^2+0.8^2-b.*x_norm.^3;
x_iniz=Pt_iniz(1);
x_fine=Pt_fine(1);
y_iniz=Pt_iniz(2);
y_fine=Pt_fine(2);
z_iniz=Pt_iniz(3);
z_fine=Pt_fine(3);
%inclinazione totale in rad
incl_y=(atan((z_fine-z_iniz)/sqrt((x_fine-x_iniz)^2+(y_fine-
y_iniz)^2)));
incl_z=(atan2((y_fine-y_iniz),(x_fine-x_iniz)));
%distanza
dist=sqrt((x_fine-x_iniz)^2+(y_fine-y_iniz)^2+(z_fine-z_iniz)^2);
x=[x_norm*dist];
z=abs([z_norm*dist]);
pt=[x,y,z]';
newpt=axang2rotm([0 1 0 -incl_y])*pt;
newpt_1=axang2rotm([0 0 1 incl_z])*newpt;
%si applica prima la rotazione e solo dopo si somma l'altezza iniziale
newpt_1(3,:)=newpt_1(3,:)+z_iniz;
newpt_1(2,:)=newpt_1(2,:)+y_iniz;
newpt_1(1,:)=newpt_1(1,:)+x_iniz;


% in = setBlockParameter(in, ...
%     "mdl1_robot/Corpo", ...
%     "Value",num2str(corpo));
in = setVariable (in, "corpo", corpo);
in = setVariable (in, "newpt_1", newpt_1);
in = setVariable (in, "Pt_fine", Pt_fine);
in = setVariable (in, "dist", dist);
```

```
in = setVariable (in,"Initial_condition",Initial_condition');

end
```

Reward Function

```
function [R,isdone, Step]  = fcnReward(u,v,d,q, corpo,newpt_1,Pt_fine,
dold,dist,dist_old, Step)
%u posizione EE
%v velocità EE
%d velocità dei singoli giunti
isdone=false;
T_step=0.008;

Step=Step+1;

%penalità tempo
r1=-10;

%non posso copiare e utilizzare le variabili posizione_TCP e qt poichè
sono
%state calcolate da un calcolo delle q qd e qdd
dist_percorso=zeros(1000,1);
for i=1:1000
dist_percorso(i)=sqrt((u(1,4)-newpt_1(1,i))^2+((u(2,4)-
newpt_1(2,i))^2)+(u(3,4)-newpt_1(3,i))^2);
end
% m distanza minima
m=min(dist_percorso);
c=100;
%c=costante
r2=-(c*(m)+1);


%penalità dist corpo
%le coordinate del corpo sono randomizzate
%La posizione del corpo è generata casualmente, la posizione da tenere
in
%considerazione per la verifica del fattore di rischio è la u che ci
danno
dist_corpo =sqrt((corpo(1)-u(1,4))^2+(corpo(2)-u(2,4))^2+(corpo(3)-
u(3,4))^2);
f=1./(dist_corpo);
v_rif=0.1;
v_ur=sqrt(v(1)^2+v(2)^2+v(3)^2);
k=1;
g=k*abs(v_rif-v_ur);
```

```
r3=-f*g;


%definizione dell'accelerazione
dd=(d-dold)./T_step;
limite = 1300; % punto di flessione
% Calcola la sigmoide
r4 = -sum(100 ./ (1 + exp(-(dd-limite))));

%dist definito nella ResetFnc definisce la distanza pt_iniz e pt_fine
dist_fine=sqrt((Pt_fine(1)-u(1,4))^2+(Pt_fine(2)-
u(2,4))^2+(Pt_fine(3)-u(3,4))^2);
if (dist_old-dist_fine)> 0
    r5=5;
else
    r5=0;
end
% end

%ricompensa premio quando arriva al punto finale
if u(1:3,4)==Pt_fine
    r6=200;
    isdone = true;
else
    r6=0;
end

%penalità dovuta all'eccessivo aumento di velocità
ru=sqrt(v(4)^2+v(5)^2+v(6)^2);
r7= -10*(exp(ru - 1).*(sign(ru - 1)/10 + 1/10));

%salvo
dold=d;

%reward tot
R=r1+r2+r3+r4+r5+r6+r7;
```

Path Following: singolo percorso

Reset Function

```
%RESETFcn del nuovo
function in = ResetFcn(in)
r_min=0.08;
r_max=0.4;
theta_min=-pi;
theta_max=+pi;
```

```
corpo =[rand(1)*0.4, rand(1)*0.4, rand(1)*0.4]';
ur3=loadrobot("universalUR3");

Pt_fine=[0.2,-0.15,0.2]';
Pt_iniz=[0.2,0.25,0.2]';


%configurazione iniziale
ik=inverseKinematics('RigidBodyTree',ur3);
weights= [0.1,0.1,0.1,0.1,0.1,0.1];
jointWaypoints = homeConfiguration(ur3);
jointiniz=[0,-pi/4,pi/2,-3*pi/4,-pi/2,0]';


for j=1:length(jointWaypoints)
jointWaypoints(1,j).JointPosition = jointiniz(j);
end

[q_sol,q_info]=ik('wrist_3_link',tform,weights,jointWaypoints);
% %ottengo la configurazione iniziale dei giunti
for i=1:length(q_sol)
    Initial_condition(i)=q_sol(i).JointPosition;
end
% Imposta la proprietà DataFormat su 'column' o 'row'
boxobj = collisionBox(2,2,0.1,Pose=trvec2tform([0,0,-0.055]));
ur3.DataFormat = 'column'; % o 'row'
collisionFree = sum(checkCollision(ur3, Initial_condition',{boxobj}));
%collisionFree = checkCollision(ur3, Initial_condition');
ur3.DataFormat = 'struct';
end


%DEFINIZIONE DEL PERCORSO
dist=sqrt((Pt_fine(1)-Pt_iniz(1))^2+(Pt_fine(2)-
Pt_iniz(2))^2+(Pt_fine(3)-Pt_iniz(3))^2);
k=linspace(0,1,126);
%flesso
c=0.6;
%distanza normalizzata
di=dist;
d=1/di;
%definizione del tempo in funzione della distanza
k=k/d;
for i=1:length(k)
%fattore inversamente proporzionale alla distanza e serve a variare lo
%spazio

%v_x è la velocità con profilo deciso da noi
    v_x(i) = (3./ (1 +exp(d*2*exp(1)*k(i)-2*exp(1)*c)))/1.7613;
```

```
    xt(i)=1.70329*k(i) - (0.313302*log(26.1001 +
exp(5.43656*d*k(i))))/d-(- (0.313302*log(26.1001 + exp(0)))/d);
end
x_norm=xt/max(xt);
x=x_norm.*dist;
%y_norm=a*x_norm - (x_norm-0.8).^2+0.8^2-b.*x_norm.^3;
y_norm=((-(2.*x_norm-1).^2)+1)/3;
x_iniz=Pt_iniz(1,1);
x_fine=Pt_fine(1,1);
y_iniz=Pt_iniz(2,1);
y_fine=Pt_fine(2,1);
z_iniz=Pt_iniz(3,1);
z_fine=Pt_fine(3,1);

%inclinazione totale in rad
incl_z=(atan2((y_fine-y_iniz),(x_fine-x_iniz)));
x=x_norm.*dist;
%definizione di y
y=abs(y_norm*dist);
%definizione di v_y tramite rapp_incrementale
v_y=diff(y)./diff(k);
v_y(end+1)=v_y(end);
z=zeros(size(x_norm));
pt=[x;y;z];

%definizione dei punti
newpt_1=axang2rotm([0 0 1 incl_z])*pt;
newpt_1(3,:)=newpt_1(3,:)+z_iniz;
newpt_1(2,:)=newpt_1(2,:)+y_iniz;
newpt_1(1,:)=newpt_1(1,:)+x_iniz;

newpt_1=[newpt_1(:,1),newpt_1(:,1:end)];
newpt_1(:,end+1)=newpt_1(:,end);
newpt_1(:,end+1)=newpt_1(:,end);

newpt_1(3,1)=z_iniz;
newpt_1(2,1)=y_iniz;
newpt_1(1,1)=x_iniz;

dist_origine=sqrt(newpt_1(1,:).^2+newpt_1(2,:).^2);
somma=sum(dist_origine<0.08);

in = setVariable (in, "corpo", corpo);
in = setVariable (in, "newpt_1", newpt_1);
in = setVariable (in, "Pt_fine", Pt_fine);
in = setVariable (in, "dist", dist);
in = setVariable (in,"Initial_condition",Initial_condition');
end
```

Reward Function

```
function [Step,R,isdone] = fcn(Step,Pt_fine,newpt_1,u,v,corpo)
isdone = false;
Step = Step+1;
i = Step(1,1);


%fattore che tiene conto del fatto di essere arrivato in prossimità
del
%punto finale. viene consegnato un premio al robot
dist_fine = sqrt((Pt_fine(1)-u(1,4))^2+(Pt_fine(2)-
u(2,4))^2+(Pt_fine(3)-u(3,4))^2);
R=0;
m = sqrt((u(1,4)-newpt_1(1,i))^2+((u(2,4)-newpt_1(2,i))^2)+(u(3,4)-
newpt_1(3,i))^2);

%DEFINIZIONE DELLA FUNZIONE REWARD DEL PATH FOLLOWING
% Definizione dei punti
x_data = [0, 0.005, 0.1, 1];
y_data = [0, 20, 50, 100];
% Funzione logaritmica modificata
log_func = @(params, x) params(1) * log(params(2) * x + 1);
% Funzione di errore da minimizzare
error_func = @(params) sum((log_func(params, x_data) - y_data).^2);
% Parametri iniziali
initial_guess = [1, 1];
% Ottimizzazione per trovare i parametri ottimali
options = optimset('Display','off');
params_opt = fminsearch(error_func, initial_guess, options);
% Estrazione dei parametri ottimali
c_opt = params_opt(1);
e_opt = params_opt(2);
% Definizione della funzione logaritmica con i parametri ottimali
f = @(m) c_opt * log(e_opt * m + 1);
% Calcola i valori y corrispondenti
y = f(m);
%Funzione Reward considerando una tolleranza di 0.003 m
R=-y.*(m>0.003)+5.*(m<=0.003)

if i==length(newpt_1(1,:))

%ricompensa premio quando arriva al punto finale

    if dist_fine <= 0.005
        R = R+100;
    end
```

```
    isdone=true;

end

end
```

Path Following di un singolo percorso con gestione dell'orientazione e presenza del corpo

Reset Function

```
%RESETFcn del nuovo
function in = ResetFcn(in)
r_min=0.08;
r_max=0.4;
theta_min=-pi;
theta_max=+pi;

corpo =[rand(1)*0.4, rand(1)*0.4, rand(1)*0.4]';

ur3=loadrobot("universalUR3");

Pt_fine=[0.2,-0.15,0.2]';
Pt_iniz=[0.2,0.25,0.2]';

%configurazione iniziale
ik=inverseKinematics('RigidBodyTree',ur3);
weights= [0.1,0.1,0.1,0.1,0.1,0.1];
jointWaypoints = homeConfiguration(ur3);
jointiniz=[0,-pi/4,pi/2,-3*pi/4,-pi/2,0]';

for j=1:length(jointWaypoints)
jointWaypoints(1,j).JointPosition = jointiniz(j);
end


tform=getTransform(ur3,jointWaypoints,"wrist_3_link");
orientaz=tform(1:3,1:3);
tform(1:3,4)=Pt_iniz';


[q_sol,q_info]=ik('wrist_3_link',tform,weights,jointWaypoints);
% %ottengo la configurazione iniziale dei giunti
for i=1:length(q_sol)
    Initial_condition(i)=q_sol(i).JointPosition;
End

% Imposta la proprietà DataFormat su 'column' o 'row'
boxobj = collisionBox(2,2,0.1,Pose=trvec2tform([0,0,-0.055]));
```

```
ur3.DataFormat = 'column'; % o 'row'
collisionFree = sum(checkCollision(ur3, Initial_condition',{boxobj}));
%collisionFree = checkCollision(ur3, Initial_condition');
ur3.DataFormat = 'struct';
end

%DEFINIZIONE DEL PERCORSO
dist=sqrt((Pt_fine(1)-Pt_iniz(1))^2+(Pt_fine(2)-
Pt_iniz(2))^2+(Pt_fine(3)-Pt_iniz(3))^2);
k=linspace(0,1,126);
%flesso
c=0.6;
%distanza normalizzata
di=dist;
d=1/di;
%definizione del tempo in funzione della distanza
k=k/d;

for i=1:length(k)
%v_x è la velocità con profilo deciso da noi
    v_x(i) = (3./ (1 +exp(d*2*exp(1)*k(i)-2*exp(1)*c)))/1.7613;

    xt(i)=1.70329*k(i) - (0.313302*log(26.1001 +
exp(5.43656*d*k(i))))/d-(- (0.313302*log(26.1001 + exp(0)))/d);
end
x_norm=xt/max(xt);
x=x_norm.*dist;
y_norm=((-(2.*x_norm-1).^2)+1)/3;


x_iniz=Pt_iniz(1,1);
x_fine=Pt_fine(1,1);
y_iniz=Pt_iniz(2,1);
y_fine=Pt_fine(2,1);
z_iniz=Pt_iniz(3,1);
z_fine=Pt_fine(3,1);


%inclinazione totale in rad
incl_z=(atan2((y_fine-y_iniz),(x_fine-x_iniz)));
x=x_norm.*dist;
%definizione di y
y=abs(y_norm*dist);
%definizione di v_y tramite rapp_incrementale
v_y=diff(y)./diff(k);
v_y(end+1)=v_y(end);
z=zeros(size(x_norm));
pt=[x;y;z];
```

```
%definizione dei punti
newpt_1=axang2rotm([0 0 1 incl_z])*pt;
newpt_1(3,:)=newpt_1(3,:)+z_iniz;
newpt_1(2,:)=newpt_1(2,:)+y_iniz;
newpt_1(1,:)=newpt_1(1,:)+x_iniz;


newpt_1=[newpt_1(:,1),newpt_1(:,1:end)];
newpt_1(:,end+1)=newpt_1(:,end);
newpt_1(:,end+1)=newpt_1(:,end);


newpt_1(3,1)=z_iniz;
newpt_1(2,1)=y_iniz;
newpt_1(1,1)=x_iniz;

dist_origine=sqrt(newpt_1(1,:).^2+newpt_1(2,:).^2);

in = setVariable (in, "orientaz", orientaz);
in = setVariable (in, "corpo", corpo);
in = setVariable (in, "newpt_1", newpt_1);
in = setVariable (in, "Pt_fine", Pt_fine);
in = setVariable (in, "dist", dist);
in = setVariable (in,"Initial_condition",Initial_condition');
end
```

Reward Function

```
function [Step,R,isdone,newpt] = fcn(Step,Pt_fine,newpt_1,orientaz,u)
isdone = false;
Step = Step+1;
i = Step(1,1);
newpt=newpt_1(:,i);

%fattore che tiene conto del fatto di essere arrivato in prossimità
del
%punto finale. viene consegnato un premio al robot
dist_fine =sqrt((newpt_1(1,end)-u(1,4))^2+(newpt_1(2,end)-
u(2,4))^2+(newpt_1(3,end)-u(3,4))^2);
R=0;

%distanza dal punto del percorso
m = sqrt((u(1,4)-newpt_1(1,i))^2+((u(2,4)-newpt_1(2,i))^2)+(u(3,4)-
newpt_1(3,i))^2);


% Definizione dei punti
x_data = [0, 0.005, 0.1, 1];
y_data = [0, 20, 50, 100];
```

```
% Funzione logaritmica modificata
log_func = @(params, x) params(1) * log(params(2) * x + 1);
% Funzione di errore da minimizzare
error_func = @(params) sum((log_func(params, x_data) - y_data).^2);
% Parametri iniziali
initial_guess = [1, 1];
% Ottimizzazione per trovare i parametri ottimali
options = optimset('Display','off');
params_opt = fminsearch(error_func, initial_guess, options);
% Estrazione dei parametri ottimali
c_opt = params_opt(1);
e_opt = params_opt(2);
% Definizione della funzione logaritmica con i parametri ottimali
f = @(m) c_opt * log(e_opt * m + 1);
% Calcola i valori y corrispondenti
y = f(m);
R=-y.*(m>0.003)+5.*(m<=0.003);

diff_orient = norm((orientaz-u(1:3,1:3)),2);
%Penalità logaritmica dell'orientazione
a = 30 / 4.605;
b = exp(10 / (a * log(pi)));
c = 40 - a * log(b);
% Funzione logaritmica risultante
r = -(a * log(b * diff_orient) +
c).*(diff_orient>0.1)+1.*(diff_orient<=0.1);
R=R+r;


%Reward per arrivo al punto finale
if i==length(newpt_1(1,1:end))

    if dist_fine <= 0.003  %ricompensa premio quando arriva al punto
finale
        R = R+100;
    end

    isdone = true;

end
end
```

Path Following di un singolo percorso con gestione dell'orientazione, presenza del corpo e regolazione della velocità

Reset Function

Viene utilizzata la stessa Reward del caso precedente

Reward Function

```
function [Step,R,isdone,newpt] = fcn(Step,Pt_fine,newpt_1,u,v,corpo)
isdone = false;
Step = Step+1;
i = Step(1,1);
newpt=newpt_1(:,i);

%fattore che tiene conto del fatto di essere arrivato in prossimità
del
%punto finale. viene consegnato un premio al robot
dist_fine = sqrt((Pt_fine(1)-u(1,4))^2+(Pt_fine(2)-
u(2,4))^2+(Pt_fine(3)-u(3,4))^2);
R=0;

m = sqrt((u(1,4)-newpt_1(1,i))^2+((u(2,4)-newpt_1(2,i))^2)+(u(3,4)-
newpt_1(3,i))^2);

% Definizione della Reward per il Path following singolo percorso
x_data = [0, 0.005, 0.1, 1];
y_data = [0, 20, 50, 100];
% Funzione logaritmica modificata
log_func = @(params, x) params(1) * log(params(2) * x + 1);
% Funzione di errore da minimizzare
error_func = @(params) sum((log_func(params, x_data) - y_data).^2);
% Parametri iniziali
initial_guess = [1, 1];
% Ottimizzazione per trovare i parametri ottimali
options = optimset('Display','off');
params_opt = fminsearch(error_func, initial_guess, options);
% Estrazione dei parametri ottimali
c_opt = params_opt(1);
e_opt = params_opt(2);
% Definizione della funzione logaritmica con i parametri ottimali
f = @(m) c_opt * log(e_opt * m + 1);
% Calcola i valori y corrispondenti
y = f(m);
R=-y.*(m>0.003)+5.*(m<=0.003)

diff_orient = norm((orientaz-u(1:3,1:3)),2);
%Penalità logaritmica dell'orientazione
a = 30 / 4.605;
b = exp(10 / (a * log(pi)));
```

```
c = 40 - a * log(b);
% Funzione logaritmica risultante
r = -(a * log(b * diff_orient) +
c).*(diff_orient>0.1)+1.*(diff_orient<=0.1);
R=R+r;


%Zona di non percorrenza per fargli evitare l'ostacolo
dist_corpo=sqrt((corpo(1)-u(1,4))^2+((corpo(2)-u(2,4))^2)+(corpo(3)-
u(3,4))^2);

%Regolazione velocità nella zona di sicurezza compresa tra gli 0.1 e
gli 0.05 metri dal corpo
r=0;
v=sqrt(v(4)^2+v(5)^2+v(6)^2);
g= (abs(0.3-v));
k=15;
x_var=0.1+(g-1.7)*(0.01);
   if dist_corpo<=1.3*x_var
      b = log(10)/(0.05-x_var).*(g>0.05)+0.*(g<=0.05);
      a=1/exp(b*x_var).*(g>0.05)-(3).*(g<=0.05);
      r=-k*((a*exp(b.*dist_corpo)));
   end
R=R+r;

%Fattore Reward per gestire la non percorrenza della zona prossima al
corpo compresa tra gli 0.05 e gli 0 metri

if dist_corpo<=0.05
    R=-200;
end

%Componente Reward per assegnare un premio per l'arrivo al punto
finale
if i==length(newpt_1(1,:))

    if dist_fine <= 0.005  %ricompensa premio quando arriva al punto
finale
        R = R+100;
    end

    isdone=true;
end
end
```

Path Following di un singolo percorso con gestione dell'orientazione, presenza del corpo, regolazione della velocità e verifica delle collisioni

Reset Function

Si utilizza lo stesso codice per la Reset Function utilizzata per il caso precedente

Reward Function

```
function [Step,R,isdone,newpt] =
fcn(Step,Pt_fine,newpt_1,orientaz,u,isColliding)
isdone = false;
Step = Step+1;
i = Step(1,1);
newpt=newpt_1(:,i);

%fattore che tiene conto del fatto di essere arrivato in prossimità
del
%punto finale. viene consegnato un premio al robot
dist_fine =sqrt((newpt_1(1,end)-u(1,4))^2+(newpt_1(2,end)-
u(2,4))^2+(newpt_1(3,end)-u(3,4))^2);
R=0;

if isColliding ~= 0
      R = -250*(length(newpt_1)-i);
      isdone=true;
end
%distanza dal punto del percorso
m = sqrt((u(1,4)-newpt_1(1,i))^2+((u(2,4)-newpt_1(2,i))^2)+(u(3,4)-
newpt_1(3,i))^2);


% Definizione dei punti
x_data = [0, 0.005, 0.1, 1];
y_data = [0, 20, 50, 100];
% Funzione logaritmica modificata
log_func = @(params, x) params(1) * log(params(2) * x + 1);
% Funzione di errore da minimizzare
error_func = @(params) sum((log_func(params, x_data) - y_data).^2);
% Parametri iniziali
initial_guess = [1, 1];
% Ottimizzazione per trovare i parametri ottimali
options = optimset('Display','off');
params_opt = fminsearch(error_func, initial_guess, options);
% Estrazione dei parametri ottimali
c_opt = params_opt(1);
e_opt = params_opt(2);
% Definizione della funzione logaritmica con i parametri ottimali
f = @(m) c_opt * log(e_opt * m + 1);
% Calcola i valori y corrispondenti
```

```
y = f(m);
R=-y.*(m>0.003)+5.*(m<=0.003);

diff_orient = norm((orientaz-u(1:3,1:3)),2);
%Penalità logaritmica dell'orientazione
a = 30 / 4.605;
b = exp(10 / (a * log(pi)));
c = 40 - a * log(b);
% Funzione logaritmica risultante
r = -(a * log(b * diff_orient) +
c).*(diff_orient>0.1)+1.*(diff_orient<=0.1);
R=R+r;


%Zona di non percorrenza per fargli evitare l'ostacolo
dist_corpo=sqrt((corpo(1)-u(1,4))^2+((corpo(2)-u(2,4))^2)+(corpo(3)-
u(3,4))^2);

    %Regolazione velocità corpo
     r=0;
     v=sqrt(v(4)^2+v(5)^2+v(6)^2);
     g= (abs(0.3-v));
     k=15;
     x_var=0.1+(g-1.7)*(0.01);
     if dist_corpo<=1.3*x_var
     b = log(10)/(0.05-x_var).*(g>0.05)+0.*(g<=0.05);
     a=1/exp(b*x_var).*(g>0.05)-(3).*(g<=0.05);
     r=-k*((a*exp(b.*dist_corpo)));
     end
     R=R+r;
     %evitamento
     if dist_corpo<=0.05
     R=-200;
     end

%Reward per arrivo al punto finale
if i==length(newpt_1(1,1:end))

    if dist_fine <= 0.003  %ricompensa premio quando arriva al punto
finale
        R = R+100;
    end


     if isColliding==0
          R = R+10;
     end
isdone = true;
```

```
end
end
```

Blocco verifica collisioni nel modello simulink

```
classdef UR3CollisionChecker < matlab.System
  % UR3CollisionChecker Esegue il check delle collisioni per UR3

    properties
        % Proprietà pubbliche, se necessario (ad esempio, nome del
link)
    end

    properties(Access = private)
        RigidBodyTree % Oggetto rigidBodyTree per UR3
    end

    methods(Access = protected)
        function setupImpl(ur)
            % Inizializza il rigidBodyTree
            ur.RigidBodyTree =
loadrobot('universalUR3','DataFormat','column');
        end

        function isColliding = stepImpl(ur, q)

            %solo nel caso in cui si considera anche il pavimento
            boxobj = collisionBox(2,2,0.1,Pose=trvec2tform([0,0,-
0.055]));
            isColliding = sum(checkCollision(ur.RigidBodyTree,
q,{boxobj}));

        end
    end
end
```

Generalizzazione del modello

Reset Function

```
%RESETFcn del nuovo
function in = ResetFcn(in)
r_min=0.08;
r_max=0.4;
theta_min=-pi;
theta_max=+pi;
```

```
%Generazione casuale del corpo nello spazio di lavoro
corpo =[rand(1)*0.4, rand(1)*0.4, rand(1)*0.4]';


%Check per le collisioni
ur3=loadrobot("universalUR3");
somma=1;
while somma ~=0
 r=rand(1)*(r_max-r_min)+r_min;
 theta=rand(1)*(theta_max-theta_min)+theta_min;
 Pt_fine=[r*cos(theta),r*sin(theta),0.2]';

collisionFree=1;
while collisionFree~=0
   r=rand(1)*(r_max-r_min)+r_min;
   theta=rand(1)*(theta_max-theta_min)+theta_min;
   Pt_iniz=[r*cos(theta),r*sin(theta),0.2]';

%configurazione iniziale
ik=inverseKinematics('RigidBodyTree',ur3);
weights= [0.1,0.1,0.1,0.1,0.1,0.1];
jointWaypoints = homeConfiguration(ur3);
jointiniz=[0,-pi/4,pi/2,-3*pi/4,-pi/2,0]';

for j=1:length(jointWaypoints)
jointWaypoints(1,j).JointPosition = jointiniz(j);
end


tform=getTransform(ur3,jointWaypoints,"wrist_3_link");
orientaz=tform(1:3,1:3);
tform(1:3,4)=Pt_iniz';


[q_sol,q_info]=ik('wrist_3_link',tform,weights,jointWaypoints);
% %ottengo la configurazione iniziale dei giunti
for i=1:length(q_sol)
    Initial_condition(i)=q_sol(i).JointPosition;
end


Initial_condition=Initial_condition;


% Imposta la proprietà DataFormat su 'column' o 'row'
boxobj = collisionBox(2,2,0.1,Pose=trvec2tform([0,0,-0.055]));
ur3.DataFormat = 'column'; % o 'row'
collisionFree = sum(checkCollision(ur3, Initial_condition',{boxobj}));
ur3.DataFormat = 'struct';
end
```

```
%DEFINIZIONE DEL PERCORSO
dist=sqrt((Pt_fine(1)-Pt_iniz(1))^2+(Pt_fine(2)-
Pt_iniz(2))^2+(Pt_fine(3)-Pt_iniz(3))^2);
k=linspace(0,1,126);
%flesso
c=0.6;
%distanza normalizzata
di=dist;
d=1/di;
%definizione del tempo in funzione della distanza
k=k/d;

for i=1:length(k)

%v_x è la velocità con profilo deciso da noi
    v_x(i) = (3./ (1 +exp(d*2*exp(1)*k(i)-2*exp(1)*c)))/1.7613;

%Si considera la funzione integrata della velocità ricavando le
coordinate x
 xt(i)=1.70329*k(i) - (0.313302*log(26.1001 + exp(5.43656*d*k(i))))/d-
(- (0.313302*log(26.1001 + exp(0)))/d);
end

x_norm=xt/max(xt);
x=x_norm.*dist;

y_norm=((-(2.*x_norm-1).^2)+1)/3;

x_iniz=Pt_iniz(1,1);
x_fine=Pt_fine(1,1);
y_iniz=Pt_iniz(2,1);
y_fine=Pt_fine(2,1);
z_iniz=Pt_iniz(3,1);
z_fine=Pt_fine(3,1);

%inclinazione totale in rad
incl_z=(atan2((y_fine-y_iniz),(x_fine-x_iniz)));
x=x_norm.*dist;

%definizione di y
y=abs(y_norm*dist);

%definizione di v_y tramite rapp_incrementale
v_y=diff(y)./diff(k);
v_y(end+1)=v_y(end);

z=zeros(size(x_norm));
```

```
pt=[x;y;z];

%definizione dei punti
newpt_1=axang2rotm([0 0 1 incl_z])*pt;
newpt_1(3,:)=newpt_1(3,:)+z_iniz;
newpt_1(2,:)=newpt_1(2,:)+y_iniz;
newpt_1(1,:)=newpt_1(1,:)+x_iniz;

newpt_1=[newpt_1(:,1),newpt_1(:,1:end)];
newpt_1(:,end+1)=newpt_1(:,end);
newpt_1(:,end+1)=newpt_1(:,end);

newpt_1(3,1)=z_iniz;
newpt_1(2,1)=y_iniz;
newpt_1(1,1)=x_iniz;

dist_origine=sqrt(newpt_1(1,:).^2+newpt_1(2,:).^2);
somma=sum(dist_origine<0.08);
end

in = setVariable (in, "orientaz", orientaz);
in = setVariable (in, "corpo", corpo);
in = setVariable (in, "newpt_1", newpt_1);
in = setVariable (in, "Pt_fine", Pt_fine);
in = setVariable (in, "dist", dist);
in = setVariable (in,"Initial_condition",Initial_condition');
end
```

Reward Function

Si utilizza la stessa Reward del caso applicativo precedente

Bibliografia

[1] Vicentini, F. (October 12, 2020). "Collaborative Robotics: A Survey." ASME. *J. Mech. Des*. April 2021; 143(4): 040802. https://doi-org.ezproxy.biblio.polito.it/10.1115/1.4046238

[2] Polonara, M.; Romagnoli, A.; Biancini, G.; Carbonari, L. Introduction of Collaborative Robotics in the Production of Automotive Parts: A Case Study. *Machines* **2024**, *12*, 196. https://doi.org/10.3390/machines12030196

[3] F. Sherwani, M. M. Asad and B. S. K. K. Ibrahim, "Collaborative Robots and Industrial Revolution 4.0 (IR 4.0)," *2020 International Conference on Emerging Trends in Smart Technologies (ICETST)*, Karachi, Pakistan, 2020, pp. 1-5, https://doi.org/10.1109/ICETST49965.2020.9080724

[4] Universal Robots. 2024. Automazione Industriale. Tratto da Sito Web Universal Robots: https://www.universal-robots.com/it/automazione-industriale/

[5] Universal Robots. 2024. Automazione Robotica. Tratto da Sito Web Universal Robots: https://www.universal-robots.com/it/automazione-robotica/

[6] Iina Aaltonen, T. S. (2018). Refining levels of collaboration to support the design and evaluation of human-robot interaction in the manufacturing industry. *Procedia CIRP*, 93-98. https://doi.org/10.1016/j.procir.2018.03.214

[7] Frisoli Antonio, *Robotica collaborativa, perché è la chiave per l'industria 5.0,* (April 21, 2021), tratto da sito Web: https://www.agendadigitale.eu/industry-4-0/robotica-collaborativa-perche-e-la-chiave-per-lindustria-5-0/

[8] Productive robotics, *What is a Cobot?,* 2023, tratto da sito web: https://www.productiverobotics.com/cobots-101

[9] WiredWorkers, *Cobot,*(s.d.), sito Web: *https://www.wiredworkers.io/cobot/*

[10] Universal Robots, *Robot collaborativi o cobot: cosa sono? La guida definitiva*, 2024, sito web: https://www.universal-robots.com/it/robot-collaborativi-o-cobot-cosa-sono-la-guida-definitiva/

[11] Collaborative Robotics Trends, *Collaborative Robots Comparison Tool,* 2024, sito web: https://www.cobottrends.com/cobot-comparison-tool/

[12] M. J. Rosenstrauch and J. Krüger, "Safe human-robot-collaboration-introduction and experiment using ISO/TS 15066," *2017 3rd International Conference on Control, Automation and Robotics (ICCAR)*, Nagoya, Japan, 2017, pp. 740-744, https://doi.org/10.1109/ICCAR.2017.7942795.

[13] Universal Robots, *Sicurezza del cobot: I quattro tipi di operazioni collaborative*, 2024, sito web: https://www.universal-robots.com/it/2020/sicurezza-del-cobot-i-quattro-tipi-di-operazioni-collaborative/?utm_referrer=https%3A%2F%2Fwww.universal-robots.com%2F

[14] WiredWorkers, *7 advantages of using cobots!,* (s.d.), sito Web: https://www.wiredworkers.io/blog/advantages-of-cobots/

[15] ticktocktech, *Collaborative Robots Vs Traditional Robots: CoBots Vs RoBots,* 2023, sito web: https://ticktocktech.com/blog/2022/09/01/collaborative-robots-vs-traditional-robots-cobots-vs-robots/

[16] H. Zhang, Y. Wang, J. Zheng and J. Yu, "Path Planning of Industrial Robot Based on Improved RRT Algorithm in Complex Environments," in *IEEE Access*, vol. 6, pp. 53296-53306, 2018, doi: 10.1109/ACCESS.2018.2871222.

[17] Pedro La Hera, Daniel Ortiz Morales, Omar Mendoza-Trejo. “A study case of Dynamic Motion Primitives as a motion planning method to automate the work of forestry cranes”. Computers and Electronics in Agriculture. vol 183. pp 1060372021. https://doi.org/10.1016/j.compag.2021.106037

[18] S. Wang, Y. Yuan, H. Shi and Y. Zhong, "Trajectory planning for robot arm based on the Improved Mixture of Motors Primitives," *2023 IEEE International Conference on Mechatronics and Automation (ICMA)*, Harbin, Heilongjiang, China, 2023, pp. 129-134, doi: 10.1109/ICMA57826.2023.10216231.

[19] Z. Wang, J. Chang, B. Li, C. Wang and C. Liu, "Application of Improved Rapidly-exploring Random Trees (RRT) algorithm for Obstacle Avoidance of Snake-like Manipulator," *2020 IEEE International Conference on Mechatronics and Automation (ICMA)*, Beijing, China, 2020, pp. 490-495, doi: 10.1109/ICMA49215.2020.9233573.

[20] Zou, K.; Guan, X.; Li, Z.; Li, H.; Gao, X.; Zhu, M.; Tong, W.; Wang, X. A Path Planning Strategy of Wearable Manipulators with Target Pointing End Effectors. *Electronics* **2022**, *11*, 1615. https://doi.org/10.3390/electronics11101615

[21] Mathworks, *Che cos’è il Reinforcement learning?,* 2024, sito web: https://it.mathworks.com/discovery/reinforcement-learning.html?s_tid=srchtitle_support_results_2_reinforcement%20learning

[22] Juříček, M.; Parák, R.; Kůdela, J. Evolutionary Computation Techniques for Path Planning Problems in Industrial Robotics: A State-of-the-Art Review. *Computation* **2023**, *11*, 245. https://doi.org/10.3390/computation11120245

[23] Vysocký, A.; Papřok, R.; Šafařík, J.; Kot, T.; Bobovský, Z.; Novák, P.; Snášel, V. Reduction in Robotic Arm Energy Consumption by Particle Swarm Optimization. *Appl. Sci.* **2020**, *10*, 8241. https://doi.org/10.3390/app10228241ù

[24] M. M. Mohamad, M. W. Dunnigan and N. K. Taylor, "Ant Colony Robot Motion Planning," *EUROCON 2005 - The International Conference on "Computer as a Tool"*, Belgrade, Serbia, 2005, pp. 213-216, doi: 10.1109/EURCON.2005.1629898.

[25] C. Gonzalez, D. Blanco and L. Moreno, "Optimum robot manipulator path generation using Differential Evolution," *2009 IEEE Congress on Evolutionary Computation*, Trondheim, Norway, 2009, pp. 3322-3329, doi: 10.1109/CEC.2009.4983366.

[26] Z. Zhou, J. Zhao, Z. Zhang and X. Li, "Motion Planning of Dual-Chain Manipulator Based on Artificial Bee Colony Algorithm," *2023 9th International Conference on Control, Automation and Robotics (ICCAR)*, Beijing, China, 2023, pp. 55-60, doi: 10.1109/ICCAR57134.2023.10151753.

[27] Universal Robots, Robot Collaborativi Universal Robots, 2024, sito web: https://www.universal-robots.com/it/prodotti/

[28] Universal Robots, UR3 Specifiche tecniche, 2016, sito web: https://www.universal-robots.com/media/240748/ur3_it.pdf

[29] Redazione Fare Elettronica, *Cos'è il Robot Operating System (ROS) e come può essere significativo nell'industria 4.0*, 01/2022, sito web: *https://farelettronica.it/cose-il-robot-operating-system-ros-e-come-puo-essere-significativo-nellindustria-4-0/*

[30] Eureka System, *Robot Operating System: il framework per eccellenza per applicazioni robotiche,* (s.d.), sito web: https://www.eurekasystem.it/blog/robot-operating-system-il-framework-per-eccellenza-per-applicazioni-robotiche/

[31] ROS.org, *ROS Introduction,* 2018, sito web: https://wiki.ros.org/ROS/Introduction

[32] ROS.org, *ROS Concepts,* 2022, sito web: https://wiki.ros.org/ROS/Concepts

[33] MathWorks, *Matlab,* 2024, sito web: https://it.mathworks.com/help/matlab/index.html?searchHighlight=matlab&s_tid=srchtitle_support_results_1_matlab

[34] MathWorks, *Robotics System Toolbox,* 2024, sito web: https://it.mathworks.com/products/robotics.html?s_tid=srchtitle_site_search_2_robotics%20system%20toolbox

[35] MathWorks, *Plan a Reaching Trajectory with Multiple Kinematic Constraints,* 2024, sito web: https://it.mathworks.com/help/robotics/ug/plan-a-reaching-trajectory-with-kinematic-constraints.html

[36] MathWorks, *ROS Toolbox,* 2024, sito web: https://it.mathworks.com/products/ros.html?s_tid=srchtitle_site_search_2_ROS%20toolbox

[37] MathWorks, *Simulink,* 2024, sito web: https://it.mathworks.com/help/simulink/index.html?s_tid=srchtitle_site_search_1_simulink

[38] MathWorks, *Diagrammi a blocchi di Simulink,* 2024, sito web: https://it.mathworks.com/help/simulink/gs/simulink-block-diagrams.html

[39] Universal Robots, *Offline Simulator- E- Series- UR sim for non Linux 5.12.6 LTS,* 2024, sito web: https://www.universal-robots.com/download/software-e-series/simulator-non-linux/offline-simulator-e-series-ur-sim-for-non-linux-5126-lts/

[40] MathWorks, *Getting Started with Connecting and Controlling a UR5e Cobot from Universal Robots,* 2024, sito web: https://it.mathworks.com/help/robotics/urseries/ug/getting-started-controlling-ur5e.html?s_tid=srchtitle_site_search_1_getting%20ur5

[41] MathWorks, *Set Up URSim Offline Simulator,* 2024, sito web: https://it.mathworks.com/help/robotics/urseries/ug/setup-ursim-offline-simulator.html

[42] MathWorks, *Get Started with ROS and ROS2 Connectivity Interface- Functions,* 2024, sito web: https://it.mathworks.com/help/robotics/referencelist.html?type=function&category=get-started-urseries&s_tid=CRUX_topnav

[43] MathWorks, *Robotics System Toolbox- Functions*, 2024, https://it.mathworks.com/help/robotics/referencelist.html?type=function&s_tid=CRUX_topnav

[44] Cheng, X.; Zhang, S.; Cheng, S.; Xia, Q.; Zhang, J. Path-Following and Obstacle Avoidance Control of Nonholonomic Wheeled Mobile Robot Based on Deep Reinforcement Learning. *Appl. Sci.* **2022**, *12*, 6874. https://doi.org/10.3390/app12146874

[45] B. Wang, Z. Liu, Q. Li and A. Prorok, "Mobile Robot Path Planning in Dynamic Environments Through Globally Guided Reinforcement Learning," in *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6932-6939, Oct. 2020, doi: 10.1109/LRA.2020.3026638.

[46] Wang, S., Yin, X., Li, P. *et al.* Trajectory Tracking Control for Mobile Robots Using Reinforcement Learning and PID. *Iran J Sci Technol Trans Electr Eng* **44**, 1059–1068 (2020). https://doi.org/10.1007/s40998-019-00286-4

[47] MathWorks, *Che cos'è il Reinforcement Learning?,* 2024, sito web: https://it.mathworks.com/discovery/reinforcement-learning.html?s_tid=srchtitle_support_results_2_reinforcement%20learning

[48] MathWorks, *Reinforcement Learning Environments,* 2024, sito web: https://it.mathworks.com/help/reinforcement-learning/ug/reinforcement-learning-

environments.html?s_tid=srchtitle_site_search_1_Environment%20Reinforcement%20learning

[49] MathWorks, *Reinforcement Learning Agents,* 2024, sito web: https://it.mathworks.com/help/reinforcement-learning/ug/create-agents-for-reinforcement-learning.html

[50] MathWorks, *Define Reward and Observation Signals in Custom Environments* 2024, sito web: https://it.mathworks.com/help/reinforcement-learning/ug/define-reward-and-observation-signals.html?searchHighlight=Reinforcement%20learning%20Reward&s_tid=srchtitle_support_results_1_Reinforcement%20learning%20Reward

[51] MathWorks, *Reinforcement Learning Toolbox,* 2024, sito web: https://it.mathworks.com/products/reinforcement-learning.html?s_tid=srchtitle_site_search_2_reinforcement%20learning%20toolbox

[52] MathWorks, *Create Agents Using Reinforcement Learning Designer,* 2024, sito web: https://it.mathworks.com/help/reinforcement-learning/ug/create-agents-using-reinforcement-learning-designer.html

[53] Amit Ron, Meir Ron, Ciosek Kamil, Discount Factor as a Regularizer in Reinforcement Learning, 07-2020, Ithaca:Cornell University Library, arXiv.org, EISSN: 2331-8422 DOI: 10.48550/arxiv.2007.02040

[54] Fujimoto, Scott, Herke van Hoof, and David Meger. "Addressing Function Approximation Error in Actor-Critic Methods." arXiv.org (2018): n. pag. Web. EISSN: 2331-8422, DOI: 10.48550/arxiv.1802.09477

[55] MathWorks, *Train Reinforcement Learning Agents,* 2024, sito web: https://it.mathworks.com/help/reinforcement-learning/ug/train-reinforcement-learning-agents.html?s_tid=srchtitle_site_search_1_Episode%20Q0%20Reinforcement%20Learning

[56] MathWorks, *Parallel Computing Toolbox,* 2024, sito web: https://it.mathworks.com/products/parallel-computing.html

[57] MathWorks, *RL Agent,* 2024, sito web: https://it.mathworks.com/help/reinforcement-learning/ref/rlagent.html?s_tid=srchtitle_site_search_1_RL%20Agent

[58] MathWorks, *MATLAB System,* 2024, sito web: https://it.mathworks.com/help/simulink/slref/matlabsystem.html?s_tid=doc_ta

[59] MathWorks, *Create Agents Using Reinforcement Learning Designer,* 2024, sito web: https://it.mathworks.com/help/reinforcement-learning/ug/create-agents-using-reinforcement-learning-designer.html

[60] MathWorks, *rlTD3Agent,* 2024, sito web: https://it.mathworks.com/help/reinforcement-learning/ref/rl.agent.rltd3agent.html?searchHighlight=Agent%20TD3&s_tid=srchtitle_support_results_1_Agent%20TD3

[61] Daniel Udekwe, Ore-ofe Ajayi, Osichinaka Ubadike, Kumater Ter, Emmanuel Okafor, *Comparing actor-critic deep reinforcement learning controllers for enhanced performance on a ball-and-plate system*, Expert Systems with Applications, 2024, vol. 245, pp. 123055, https://doi.org/10.1016/j.eswa.2023.123055

[62] Feiyu, Z., Dayan, L., Zhengxu, W. *et al*. Autonomous localized path planning algorithm for UAVs based on TD3 strategy. *Sci Rep* **14**, 763 (2024). https://doi.org/10.1038/s41598-024-51349-4

[63] MathWorks, *Train Reinforcement Learning Agents, 2024, sito web: https://it.mathworks.com/help/reinforcement-learning/ug/train-reinforcement-learning-agents.html*

Ringraziamenti

Desidero esprimere la mia più sincera gratitudine al relatore Prof. Stefano Paolo Pastorelli, per avermi dato la possibilità di lavorare a questo progetto. Il suo contributo è stato fondamentale al perfezionamento ed il continuo miglioramento di questo lavoro.

Inoltre, vorrei ringraziare l'Ing. Michele Polito e l'Ing. Valerio Cornagliotto che mi hanno supportato pazientemente durante questo percorso mettendo a disposizione le loro conoscenza e le loro abilità, creando nel Laboratorio di Meccanica (DIMEAS) un ambiente di lavoro stimolante e collaborativo.