



**Università
di Catania**

UNIVERSITÀ DEGLI STUDI DI CATANIA
DIPARTIMENTO DI MATEMATICA ED INFORMATICA
CORSO DI LAUREA MAGISTRALE IN INFORMATICA

**Sviluppo di Sistemi di Navigazione
Robotica in Ambienti Dinamici basati su
Deep Reinforcement Learning**

Danilo Leocata

RELATORE
Prof. Giovanni Maria Farinella
CORRELATORE
Dott. Marco Rosano

Anno Accademico 2022/23

Indice dei contenuti

Indice dei contenuti	1
1 Introduzione	2
1.1 Piattaforme disponibili in commercio	6
1.2 Introduzione della tesi	8
2 Navigazione robotica: metodi classici e metodi basati sul learning	10
2.1 Algoritmi di navigazione classici	11
2.1.1 Acquisizione dell'ambiente e creazione della mappa	14
2.1.2 Path planning	19
2.2 Algoritmi di navigazione basati sul learning	25
2.2.1 Breve storia del Machine Learning e del Deep Learning	29
2.2.2 Reinforcement Learning	32
2.2.2.1 Markov Decision Process	35
2.2.2.2 Deep Reinforcement Learning	37
2.2.3 Navigazione in presenza di persone	38
2.2.3.1 LSTM-RL (Long Short-Term Memory)	41
2.2.3.2 CADRL	42
2.2.3.3 SA-CADRL	43
3 CrowdSim-Real: un simulatore avanzato per Social Navigation in scenari realistici	46
3.1 Il simulatore originale e i modelli di DRL disponibili	47
3.1.1 L'ambiente di simulazione	52
3.1.2 Modifica delle policy degli umani	58
3.1.3 Simulazione del LiDAR	60
3.1.4 Modifiche del modello	63
3.1.5 Esperimenti	64
4 Conclusioni	73
5 Ringraziamenti	76
Bibliografia	79

Introduzione

Nell’ambito della robotica moderna, la navigazione in ambienti complessi e dinamici è diventata un argomento di forte interesse per la comunità scientifica. Con il crescente sviluppo della tecnologia robotica e dell’intelligenza artificiale è stata dedicata sempre più attenzione a questo campo al fine di ottenere robot autonomi, capaci di navigare in modo sicuro ed efficiente in contesti complessi e che siano in grado di interagire con l’ambiente circostante e con altri agenti con cui condividono lo spazio. Ad oggi, molte attività un tempo svolte solo dagli umani sono supportate, o addirittura sostituite, dai robot. Essi, grazie alla loro versatilità e flessibilità, possono essere impiegati in diversi settori per svolgere compiti di varia natura. Le piattaforme robotiche possono presentarsi in svariate forme e dimensioni, in base alla destinazione d’uso, spaziando da modelli standard per applicazioni industriali a modelli impiegati per compiti più complessi, come l’esplorazione spaziale. Una delle caratteristiche fondamentali di questi sistemi è che devono possedere un certo grado di autonomia operativa. Indipendentemente dalla specifica struttura meccanica, in tutti i tipi di applicazioni un robot raggiunge un obiettivo generico per mezzo di movimenti specifici [18]. In particolar modo, rendere autonomo il loro operato è un obiettivo fondamentale per essere in grado di operare senza la necessità di un controllo umano costante e supportare gli operatori durante lo svolgimento di operazioni critiche, come l’esplorazione in ambienti pericolosi o irraggiungibili, riducendo notevolmente il rischio di incidenti e infortuni. Le piattaforme robotiche si rivelano molto utili anche nello svolgimento di operazioni ripetitive. Infatti, non richiedendo pause o riposo, possono aumentare la produttività e ridurre i tempi di esecuzione delle attività. Esse possono essere in grado di adattarsi a diverse situazioni e condizioni dell’ambiente in cui operano, raccogliendo informazioni, utilizzando sensori,

elaborando e adattando le proprie azioni di conseguenza in modo da renderli idonei ad affrontare sfide e compiti in ambienti dinamici e variabili (adattabilità). Lo studio della navigazione autonoma è stato oggetto di ricerca fin dagli anni '80. Nel corso degli anni, sono stati tentati diversi approcci per la risoluzione del problema e, nonostante i notevoli progressi raggiunti, esistono ancora diversi problemi aperti che richiedono lo sviluppo ulteriori soluzioni sempre più innovative. Una delle soluzioni proposte è stata quella del *path planning* (pianificazione del percorso) in ambienti particolarmente affollati, dove molte persone si muovono simultaneamente ed imprevedibilmente. Questo scenario porta alla presenza di ulteriori complessità come la necessità di evitare collisioni con gli agenti circostanti, la presenza di ostacoli dinamici non previsti e il calcolo di percorsi efficienti per raggiungere l'obiettivo desiderato. In questo scenario, uno dei problemi più noti e ricorrenti è lo *stuck problem* (problema dello stallo), che si verifica quando un robot si trova in una situazione in cui non più è in grado di progredire o prendere decisioni appropriate per raggiungere il suo obiettivo. Questa situazione può verificarsi ad esempio quando l'agente robotico si avvicina eccessivamente ad un ostacolo ed ogni azione per allontanarsi dalla situazione critica viene ritenuta inefficace dal sistema di guida, oppure se il sistema percettivo degli ostacoli non si aggiorna alla frequenza desiderata, dando vita a degli artefatti sulla mappa degli ostacoli, che porta il robot a credere di essere bloccato in quell'area dello spazio. Problematiche simili presentano una sfida significativa in quanto possono compromettere l'efficienza e l'efficacia del robot nell'eseguire le attività previste e risolverle richiede lo sviluppo di strategie e algoritmi che consentano al robot di rilevare lo stato di stallo e di adottare azioni appropriate per superarlo. Per realizzare questi algoritmi di navigazione o modelli (spesso chiamati così in quanto rappresentazione matematiche per l'elaborazione dei dati), vengono solitamente utilizzati dei simulatori in grado di emulare le caratteristiche e le condizioni dell'ambiente reale in cui un robot opera, usati per addestrare e valutare i modelli di navigazione autonoma. Creare un ambiente controllato e riproducibile per la generazione dei dati di addestramento porta numerosi vantaggi per accelerare il processo di addestramento e ridurne i costi, nonostante i dati raccolti in un ambiente simulato non rappresentino accuratamente tutte le sfumature e le varia-

zioni dell’ambiente reale, come si evince nella **Figura 1.1**. Infatti, i simulatori non riescono a riprodurre tutte le *features* (caratteristiche) visive e fisiche dell’ambiente reale, che può presentare variazioni di illuminazione, superfici dalla texture particolare, superfici scivolose, tutti fattori che influenzano negativamente sulla capacità del modello di operare direttamente nel mondo reale.



Figura 1.1: Un confronto tra la stanza virtuale (a sinistra) e la stanza nel mondo reale (a destra). L’impiego di dati reali durante il processo di addestramento è una pratica comune e necessaria al fine di adattare il dominio du addestramento (simulato) a quello di test (reale) [29].

Pertanto, è necessario un adeguato e robusto processo di allenamento e validazione dei modelli ottenuti per verificare la loro effettiva applicabilità nel contesto reale. Con il termine “dominio” ci si riferisce all’*environment* (ambiente) in cui un algoritmo di *Machine Learning* (apprendimento automatico) viene addestrato e testato, incluse le caratteristiche, regole e relazioni che definiscono i dati di input e di output del sistema di apprendimento. Solitamente viene fatta una distinzione tra dominio di addestramento e di test:

- Il dominio di addestramento è il set di dati utilizzato per fare addestrare un modello a svolgere una specifica attività. È composto da *samples* (esempi) e le relative *labels* (etichette), che rappresentano i risultati desiderati per ciascun esempio. Durante il processo di addestramento, l’algoritmo impara a riconoscere i *pattern* (una regolarità o una struttura ricorrente che può essere identificata all’interno di un insieme di dati, eventi o fenomeni) e a fare previsioni o classificazioni basate sui dati di addestramento. È importante avere un

dominio di addestramento di alta qualità e rappresentativo, con una copertura adeguata dei casi possibili ed etichette corrette.

- Il dominio di test, d'altra parte, è formato da un set di dati separato, indipendente dal dominio di addestramento. Viene utilizzato per misurare l'accuracy e l'efficacia del modello nel fare previsioni o classificazioni su nuovi dati che non sono stati utilizzati durante l'addestramento ed è fondamentale per valutarne e stimare il comportamento su dati non presenti nel dominio di addestramento.

Nel corso del tempo, sono state proposte diverse tecniche modelli di navigazione in grado di generalizzare con successo su ambienti reali, che presentano caratteristiche diverse da quelle presenti nel dominio di addestramento. Una di queste è nota come la tecnica di *domain adaptation* (adattamento del dominio), che mira a mitigare la discrepanza tra il dominio simulato e quello reale, cercando di catturare l'essenza dell'immagine e della scena ignorando le proprietà variabili. Ad esempio, una stanza può apparire in modo diverso se illuminata da luce naturale o luce artificiale, oppure se deocarata o meno con dei quadri. Queste differenze potrebbero portare a identificare diverse stanze, pure essendo sempre la stessa. Infine, è auspicabile considerare, durante l'implementazione di queste modelli, anche l'incorporazione del rispetto delle cosiddette norme sociali nel contesto della navigazione sociale, che rappresentano le regole e le convenzioni che guidano il comportamento umano in contesti sociali, inclusi gli spostamenti in ambienti affollati, in modo da garantire un'interazione sicura e fluida con gli altri agenti presenti nell'ambiente. Permettere all'agente robotico il rispetto di queste ultime rappresenta una sfida complessa a causa della loro varietà e complessità. Per ottenere dei buoni risultati, è necessario che il robot abbia una sufficiente capacità percettiva in modo tale da essere in grado di riconoscere ad ogni istante gli agenti intorno a sé, il contesto in cui esso sta operando e che riesca a sfruttare queste informazioni per pianificare meglio il percorso verso la destinazione.

1.1 Piattaforme disponibili in commercio

Per potersi muovere in modo autonomo i robot devono essere in grado di percepire l’ambiente circostante ed utilizzare le informazioni ottenute per prendere decisioni. La percezione avviene per mezzo di una combinazione di sensori, adoperati per misurare le proprietà fisiche dell’ambiente e solitamente suddivisi in:

- **sensori di visione:** telecamere e sensori a infrarossi per rilevare gli ostacoli e riconoscere gli oggetti nell’ambiente circostante, utili per la navigazione in ambienti ben illuminati e non affollati;
- **sensori di prossimità:** sensori a ultrasuoni o laser per rilevare gli ostacoli vicini, utili per la navigazione in ambienti affollati e poco illuminati;
- **sensori geometrici:** come il LiDAR 2d o LiDAR 3d, camere depth che catturano la geometria dell’ambiente, hanno un range percettivo più ampio e una risoluzione maggiore dei sensori di prossimità aggiungendo informazioni;
- **sensori di posizione:** i robot utilizzano sensori GPS o di odometria per determinare la loro posizione nell’ambiente, utili per la navigazione in ambienti all’aperto o su grandi distanze;
- **sensori tattili:** sensori di pressione o di forza per rilevare le sue proprietà utili anche con l’utilizzo di bracci meccanici per il rilevamento del contatto con gli oggetti da afferrare

L’uso di sensori sempre più accurati, combinato all’utilizzo di sistemi basati sul machine learning, ha portato ad un notevole miglioramento delle capacità robotiche di elaborazione ed azione. La loro combinazione permette al robot di avere una percezione ambientale più precisa, e di adattarsi autonomamente alle nuove situazioni, riducendo gli errori e portando ad una migliore efficienza globale nelle attività robotiche. In base alle esigenze specifiche del contesto di applicazione vengono fatte delle scelte sul tipo di sensori e soprattutto sul tipo di modello e training da utilizzare. In ambienti strutturati e prevedibili, come gli ambienti industriali,

possono essere utilizzati algoritmi di controllo pre-programmati o tecniche di programmazione tradizionali. D’altro canto, in ambienti dinamici e non strutturati, è necessario impiegare metodi di addestramento più complessi, basati sull’uso di tecnologie di machine learning ed intelligenza artificiale, per consentire di apprendere dall’esperienza e di adattarsi a situazioni impreviste e mutevoli. Oggi sul mercato sono disponibili diverse piattaforme robotiche progettate per soddisfare esigenze specifiche con funzionalità e caratteristiche diverse in base al contesto di utilizzo, come si vede dalla **Figura 1.2**.



Figura 1.2: Esempi di robot moderni. In alto a sinistra: *Atlas*, robot umanoide sviluppato da *Boston Dynamics*. In alto al centro: *Roomba*, un robot dotato di ruote usato per la pulizia domestica. In alto a destra: *Bluefin-21*, robot sottomarino usato per l’esplorazione oceanica. In basso a sinistra: *Spot*, robot quadrupede sviluppato da *Boston Dynamics*. In basso a destra: *DJI Phantom*, drone volante per riprese fotografiche e video

Per la sorveglianza, la consegna dei pacchi a domicilio e le riprese video sono utilizzati i droni (robot volanti), mentre quelli impiegati per la pulizia e la consegna della merce nei magazzini sono dotati di ruote. In contesti più complessi come cantieri e magazzini, d’altro canto, vengono adoperati i robot quadrupedi (dotati di quattro zampe) o umanoidi (due gambe), questi ultimi ispirati al funzionamento del corpo umano. Solitamente queste piattaforme vengono assemblate utilizzando componenti modulari, in modo da personalizzarle e consentire agli sviluppatori di creare soluzioni specifiche in base alle esigenze del caso, come si evince dalla **Figura 1.3**.

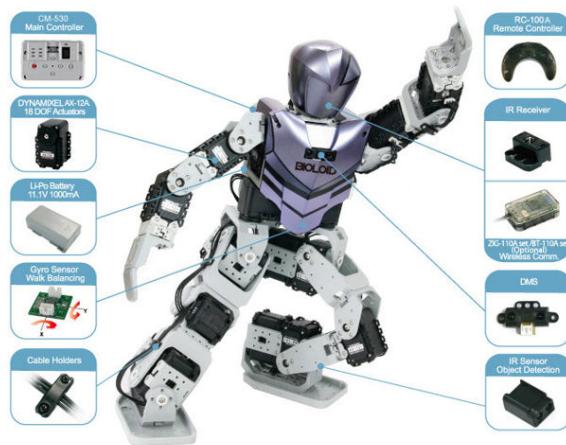


Figura 1.3: Un esempio di piattaforma modulare, il *Bioloid Premium Structure*. È un kit di robotica prodotto dalla **ROBOTIS**, progettato per l’educazione e lo sviluppo di robot autonomi. Il kit include una serie di componenti come servomotori, sensori, strutture meccaniche e una scheda di controllo che consentono di costruire una varietà di robot, come bracci robotici, bipedi, rover e altro ancora. Sono stati integrati diversi per permettere il movimento e garantire una percezione dell’ambiente soddisfacente per la risoluzione dei propri task

1.2 Introduzione della tesi

La presente tesi si focalizza sullo sviluppo di un modello di navigazione autonoma per robot in ambienti affollati, con l’obiettivo di consentire loro di muoversi in modo sicuro ed efficiente verso obiettivi prefissati. Questo studio si colloca all’interno del campo della robotica e dell’intelligenza artificiale, in particolare nel contesto dei sistemi di navigazione robotica. Il resto della tesi è strutturata come segue.

Nel **Capitolo 2**, vengono analizzati i metodi tradizionali di navigazione robotica che si basano su algoritmi per la creazione della mappa dell’ambiente e di pianificazione del percorso. Vengono evidenziate le limitazioni di tali approcci, soprattutto quando si tratta di ambienti complessi o dinamici. Successivamente, vengono introdotti i metodi basati sul Machine Learning (ML), con un focus particolare sul Reinforcement Learning (RL) e il suo utilizzo per la navigazione autonoma. Successivamente, si approfondisce l’utilizzo del reinforcement learning nel contesto della

navigazione in ambienti affollati. Si esaminano alcuni approcci proposti in letteratura, come il CADRL [11], il SA-CADRL [10] e il LSTM-RL [3], che si concentrano sulla navigazione in presenza di persone o oggetti mobili. Questi approcci sfruttano il Machine Learning per consentire al robot di adattarsi dinamicamente alle condizioni dell’ambiente e di raggiungere gli obiettivi in modo sicuro.

In questo lavoro, si mira a sviluppare un modello di navigazione autonoma che consenta al robot di apprendere autonomamente e migliorare le sue prestazioni nel tempo. L’obiettivo finale è dotare il robot di capacità di navigazione simili a quelle umane, in modo che possa interagire con sicurezza con le persone ed in particolare muoversi in ambienti affollati in maniera efficiente, un fattore che attualmente rappresenta ancora un problema aperto, parzialmente risolto. Ciò può essere ottenuto allenando un modello di navigazione in un contesto quanto più vicino a quello reale. In particolar modo, in questo lavoro sono state estese le funzionalità di un simulatore preesistente in modo tale da simulare ed effettuare il training in un ambiente che permetta di gestire la generazione randomica di ostacoli, l’importazione automatica di planimetrie di ambienti reali e di integrare un sensore LiDAR, per fornire al robot la percezione dell’ambiente circostante. Crediamo che uno strumento simile possa apportare un contributo significativo alla comunità scientifica, in quanto non è ancora presente un software/simulatore che comprende tutte queste funzionalità e che possa quindi permettere di effettuare il training di un agente in un contesto così complesso.

Navigazione robotica: metodi classici e metodi basati sul learning

Prima di tutto è necessario introdurre il concetto di navigazione, che è la capacità di determinare un percorso da un punto A ad un punto B. Per raggiungere un punto di destinazione è possibile effettuare diverse scelte e tra queste ve ne sono alcune migliori di altre. L'obiettivo degli algoritmi di navigazione è trovare il percorso più sicuro ed efficiente (che impiega meno tempo) per raggiungere un obiettivo fissato. Gli algoritmi tradizionali utilizzati in questo contesto, richiedono una rappresentazione accurata del mondo circostante e una serie di regole predefinite per eseguire compiti specifici come la pianificazione del percorso. Successivamente, essi sono stati combinati a sistemi per la costruzione di una mappa dell'ambiente, con l'idea che un robot potesse seguire un percorso prefissato sulla base della mappa. Questa tipologia di algoritmi, è stata trovata particolarmente adatta per ambienti strutturati e dotati di poca mutabilità, tuttavia, quando sono stati applicati in ambienti dinamici o con ostacoli mobili, ne sono stati scoperti i limiti, in quanto non sono in grado di adattarsi dinamicamente ai cambiamenti nelle condizioni dell'ambiente o di apprendere da esperienze precedenti. Spesso infatti è difficile adattarli a nuovi scenari o ambienti complessi, in quanto è richiesta una conoscenza approfondita del dominio e soprattutto non sono in grado di imparare in autonomia dai propri errori o esperienze passate. In aggiunta, la gestione di mappe dettagliate richiede un notevole sforzo manuale per aggiornare e mantenere le informazioni geografiche. Nonostante queste limitazioni, ad oggi, i metodi di navigazione classici continuano ad essere impiegati in contesti semplici. Più avanti, l'avvento del Machine Learning

(apprendimento automatico) ed in particolare delle Artificial Neural Networks (reti neurali artificiali) ha rivoluzionato i sistemi di navigazione, in grado di apprendere dai dati forniti e migliore le loro prestazioni in base all'esperienza accumulata, oltre a poter essere combinati con gli algoritmi di navigazione tradizionali per ottenere risultati migliori.

2.1 Algoritmi di navigazione classici

Questa tipologia di algoritmi è basata sulla conoscenza totale dell'ambiente circostante. È comune utilizzare rappresentazioni basate su grafi per gestire l'ambiente circostante, in modo da rappresentare le connessioni tra le diverse posizioni o celle nell'ambiente: su questi concetti sono stati ideati i primi algoritmi di navigazione. Il più conosciuto (considerato il primo sviluppato) è chiamato “Algoritmo di Dijkstra”, dal nome del suo inventore, l'olandese Edsger W. Dijkstra. Proposto per la prima volta nel 1956, rappresenta un algoritmo di ricerca del percorso minimo in un grafo pesato ed orientato [14] [26], assegnando a ciascun nodo del grafo una distanza provvisoria che rappresenta la stima della distanza minima dal nodo di partenza. Inizialmente, la distanza provvisoria del nodo di partenza viene impostata a 0 e la distanza provvisoria di tutti gli altri nodi viene impostata a un valore infinito. Procede iterativamente selezionando il nodo con la distanza provvisoria minima e aggiornando le distanze provvisorie dei suoi nodi adiacenti. Inizialmente, il nodo di partenza viene selezionato come nodo corrente e calcola la distanza (provvisoria) per ogni nodo adiacente al nodo corrente sommando il peso dell'arco che li collega al nodo corrente con la distanza provvisoria del nodo corrente. Se la nuova distanza provvisoria è inferiore alla distanza provvisoria attuale del nodo adiacente, la distanza provvisoria viene aggiornata. Questo processo viene ripetuto fino a quando non sono stati visitati tutti i nodi o fino a quando non viene trovato il nodo di destinazione. Durante ogni iterazione, il nodo corrente viene contrassegnato come visitato, in modo che non venga più considerato nelle iterazioni successive. L'algoritmo continua finché non viene trovata la distanza minima tra il nodo di partenza e il nodo di destinazione o finché tutti i nodi raggiungibili sono stati visitati. Una

volta terminato, la distanza provvisoria di ogni nodo rappresenta la distanza minima dal nodo di partenza. In aggiunta, viene costruito un percorso ottimo dal nodo di partenza a ciascun nodo raggiungibile, registrando il predecessore di ciascun nodo lungo il percorso che può essere ricostruito risalendo dai nodi di destinazione ai nodi di partenza utilizzando i predecessori registrati. Questo algoritmo viene ampiamente utilizzato in vari contesti, come il calcolo dei percorsi più brevi nelle reti di trasporto, il routing dei pacchetti nelle reti di computer e la pianificazione di itinerari per robot autonomi. La sua implementazione richiede l'utilizzo di una struttura dati adeguata per memorizzare i nodi, le distanze provvisorie ed i predecessori (solitamente con una coda di priorità o un heap binario) per garantire la ricerca efficiente del nodo con la distanza minima durante ogni iterazione. Ad oggi, rappresenta ancora un importante punto di partenza per comprendere i fondamenti della pianificazione del percorso e ha svolto un ruolo cruciale nello sviluppo di soluzioni di navigazione autonoma e robotica mobile. Un altro algoritmo classico fondamentale è A* (A-star), proposto da Peter Hart, Nils Nilsson e Bertram Raphael nel 1968 [21] che combina la ricerca in ampiezza (breadth-first search) con una valutazione euristica (utilizzata per stimare il costo rimanente per raggiungere la destinazione e guidare la ricerca verso le soluzioni più promettenti) per determinare il percorso più breve tra due punti in un grafo, ampiamente utilizzato nella pianificazione di percorsi ed i giochi. Combina, come Dijkstra, la ricerca basata su costo con una stima euristica del costo rimanente per raggiungere la destinazione. L'algoritmo, valuta una funzione di costo $F(n)$ per ogni nodo n del grafo, è definita come la somma di due componenti: il costo $g(n)$ per raggiungere il nodo n dal nodo di partenza e l'euristica $h(n)$ che stima il costo rimanente per raggiungere la destinazione dal nodo n . Durante l'esecuzione, vengono esplorati i nodi in base al valore della funzione $F(n)$ selezionando sempre quello con il valore ($F(n)$) più basso. L'algoritmo tiene traccia di due insiemi di nodi: il set aperto, che contiene i nodi da esplorare, e il set chiuso, che contiene i nodi già esplorati. Inizia inserendo il nodo di partenza nel set aperto con il valore $F(n)$ iniziale calcolato e successivamente viene selezionato il nodo con il valore $F(n)$ più basso dal set aperto che viene esaminato. Se il nodo selezionato è il nodo di destinazione, l'algoritmo termina e viene costruito il percorso ottimo risalendo dai nodi di

destinazione ai nodi di partenza utilizzando i predecessori registrati durante la ricerca, altrimenti, per ogni nodo adiacente al nodo corrente, vengono calcolati i valori di $g(n)$ e $h(n)$ e viene calcolato il valore $F(n)$. Se il nodo adiacente è già presente nel set aperto o nel set chiuso e il nuovo valore $F(n)$ è maggiore del valore precedente, viene ignorato. Altrimenti, il nodo adiacente viene aggiunto al set aperto con il nuovo valore $F(n)$ e il nodo corrente viene impostato come suo predecessore. Dopo aver esaminato tutti i nodi adiacenti al nodo corrente, il corrente viene spostato dal set aperto al set chiuso indicando che è stato completamente esplorato. L'algoritmo continua fino a quando viene raggiunto il nodo di destinazione o fino a quando il set aperto è vuoto, indicando che non esiste un percorso dal nodo di partenza al nodo di destinazione. A* è efficiente perché utilizza l'euristica $h(n)$ per guidare la ricerca nella direzione corretta, concentrandosi sui nodi che sembrano essere più promettenti per raggiungere la destinazione, nonostante la scelta di quest'ultima può influenzare notevolmente le prestazioni dell'algoritmo. La Figura 2.1 mostra un esempio della ricerca del percorso ottimale calcolato dagli algoritmi Dijkstra ed A*. Per rendere chiara la differenza tra i due viene mostrato (in basso) il costo ed il numero di nodi visitati durante la ricerca del percorso ottimale.

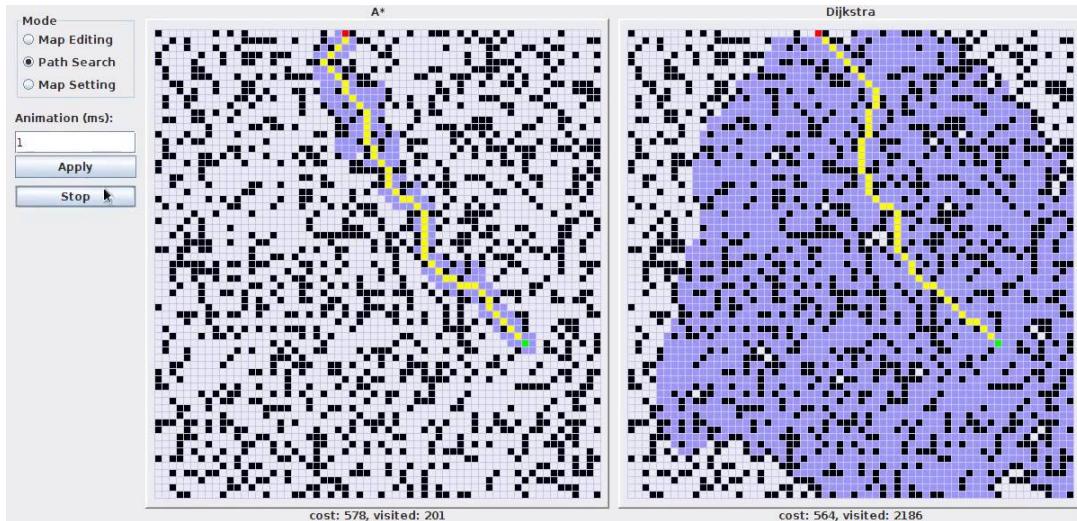


Figura 2.1: Percorsi ottimali calcolati utilizzando A* e Dijkstra, dove la cella verde indica la partenza, la cella rossa la destinazione, le celle nere gli ostacoli ed in viola le celle prese in considerazione dall'algoritmo per la costruzione del percorso ottimale in giallo. Immagine tratta dall'esecuzione di [54].

L'algoritmo A * utilizza l'euristica, quindi è quasi sempre più veloce dell'algoritmo Dijkstra, tuttavia, il percorso che trova potrebbe non essere il più breve/il più ottimale. Anche se l'algoritmo Dijkstra è arrivato prima di A*, questo può essere visto come una forma più generale di Dijkstra, un “Dijkstra con euristica”. Come accennato precedentemente, questo tipo di algoritmi richiedono una rappresentazione accurata del mondo circostante ed è necessario avere un modo per acquisire e rappresentare le informazioni sull'ambiente in cui il robot si trova e ciò viene fatto con sensori come telecamere, LiDAR o sonar per rilevare ostacoli, muri, strade e altre caratteristiche dell'ambiente. È fondamentale, quindi, trovare una rappresentazione del mondo che può essere espressa attraverso una mappa, una griglia o un modello geometrico.

2.1.1 Acquisizione dell'ambiente e creazione della mappa

L'ambiente circostante viene acquisito attraverso diverse tecniche e sensori, a seconda delle caratteristiche dell'ambiente stesso e degli obiettivi specifici del sistema. Alcune delle tecniche comuni includono l'utilizzo di telecamere, LiDAR (Light Detection and Ranging), sensori di profondità, sonar e sensori di prossimità. Grazie ai dati acquisiti, si possono ricreare i modelli 3D dell'ambiente e le planimetrie per rappresentare e memorizzare le informazioni sull'ambiente circostante. I modelli 3D, ad esempio, possono essere memorizzati in diversi formati, come file CAD (Computer-Aided Design) o file di mesh tridimensionali (ad esempio, nel formato OBJ o STL), consentendo di rappresentare in modo dettagliato la geometria dell'ambiente, inclusi gli oggetti, le superfici e le strutture presenti. Le planimetrie, d'altra parte, rappresentano una vista bidimensionale dell'ambiente e possono essere create tramite disegno manuale, rilevamento topografico o utilizzando software di modellazione 2D, spesso utilizzate per fornire una rappresentazione schematica dell'ambiente, indicando la disposizione degli ostacoli, dei corridoi, delle stanze o delle strade. La scelta delle tecniche e dei sensori di acquisizione dipende dalle specifiche esigenze del sistema di navigazione, dalle caratteristiche dell'ambiente e dalla precisione richiesta per le applicazioni desiderate. È importante scegliere i sensori appropriati e

implementare algoritmi di elaborazione dei dati adeguati in modo da ottenere una rappresentazione affidabile e utile dell’ambiente circostante. Le suddette mappe del mondo vengono spesso rappresentate utilizzando griglie o i grafi, che consentendo di discretizzare lo spazio e di rappresentare le informazioni geografiche in modo strutturato. Ad esempio, un’occupancy grid, mostrata in Figura 2.2 può essere utilizzata per mappare una planimetria o un ambiente. L’occupancy grid è una rappresentazione spaziale che suddivide un ambiente in una griglia di celle, ciascuna delle quali rappresenta una piccola area dello spazio dove, in ciascuna cella, viene registrato lo stato dell’occupazione. Utilizzando successivamente i dati sensoriali, come letture da sensori laser o telecamere, è possibile aggiornare dinamicamente lo stato dell’occupazione delle celle in base alle informazioni rilevate nell’ambiente. Questo processo di aggiornamento consente di creare una mappa dettagliata dell’occupazione dello spazio, che può essere utilizzata per la navigazione, la localizzazione e la pianificazione del percorso di robot o veicoli autonomi.

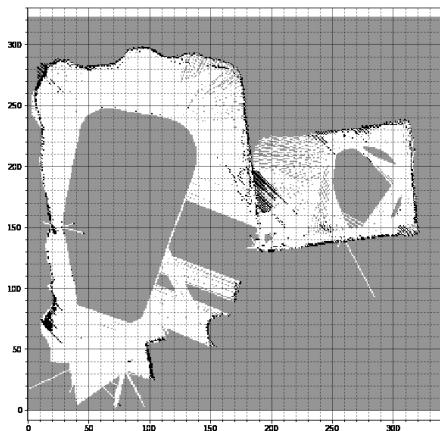


Figura 2.2: Esempio di occupancy grid di una mappa di più spazi, uniti in una mappa comune [47].

Gli ostacoli sono rappresentati come celle bloccate in una griglia, o come nodi inaccessibili in un grafo, ed una volta definiti i punti e gli ostacoli è necessario creare le connessioni tra i punti sulla mappa che rappresentano i possibili spostamenti o percorsi tra i vari punti (un esempio in Figura 2.3).

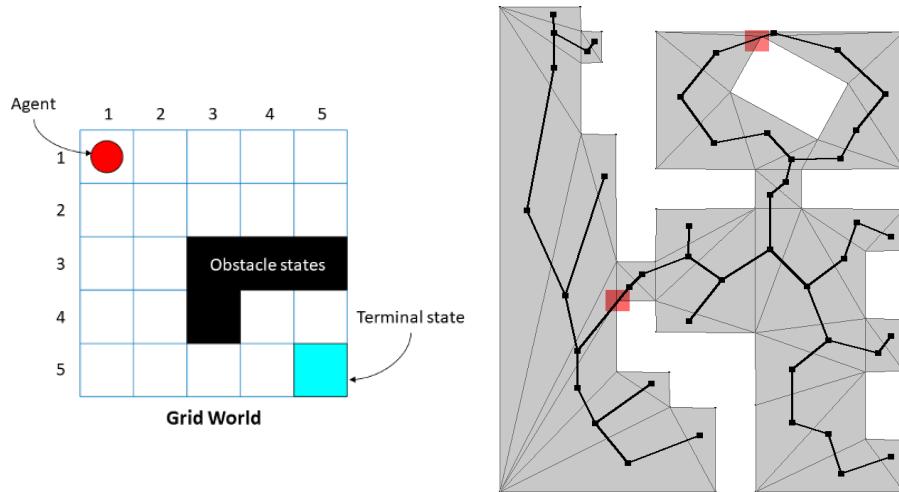


Figura 2.3: A sinistra un esempio di rappresentazione in gridworld [35], a destra un esempio di grafi direzionali in una mappa [5].

Nelle mappe a griglia, sono rappresentate da celle adiacenti orizzontalmente, verticalmente o diagonalmente. Nei grafi sono rappresentate come archi tra i nodi. È fondamentale inoltre assegnare costi ai percorsi sulla mappa che rappresentano la distanza o il tempo necessario per spostarsi da un punto all'altro in modo tale da garantire una "differenziazione" tra i tipi di percorso. Una volta creata la mappa, è possibile applicare gli algoritmi di ricerca del percorso. Tra le strutture dati comuni ci sono le seguenti:

- Griglia: una struttura dati rettangolare composta da celle dove ogni cella può rappresentare una posizione sulla mappa e può avere attributi come lo stato (libero o occupato), il costo del percorso, le coordinate;
- Grafo: una struttura dati composta da nodi e archi. I nodi rappresentano i punti sulla mappa, mentre gli archi rappresentano le connessioni tra i punti, ognuno può avere attributi come le coordinate, il costo del percorso **Figura 2.4 (a)**;
- Matrice di adiacenza: una tabella bidimensionale che rappresenta le connessioni tra i punti sulla mappa. Le righe e le colonne della matrice rappresentano i nodi, mentre i valori all'interno della matrice indicano se due nodi sono collegati o meno **Figura 2.4 (e)**;

- Lista di adiacenza: una struttura dati che elenca le connessioni di ciascun nodo. Ogni nodo ha una lista di nodi adiacenti a cui è collegato Figura 2.4 (f);.

Queste, consentono di rappresentare l'ambiente e di implementare gli algoritmi di ricerca del percorso come Dijkstra o A*. Utilizzandole con gli algoritmi appropriati, è possibile trovare il percorso ottimo o approssimato tra il punto di partenza e la destinazione sulla mappa creata.

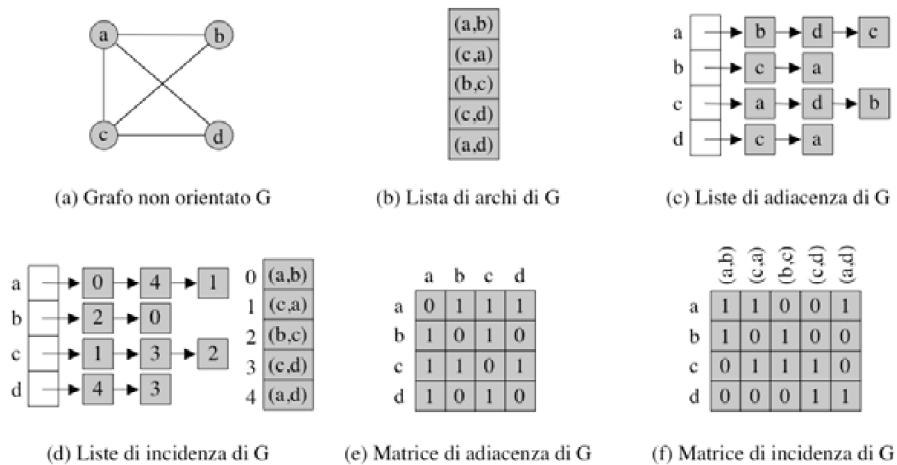


Figura 2.4: Esempi di strutture dati utilizzate per memorizzare un grafo [13]

Fino agli anni '80 e '90, quando il campo della robotica mobile ha cominciato a prendere forma, la maggior parte dei robot operava in ambienti strutturati e predefiniti, ma c'era una crescente necessità di sviluppare robot che potessero operare in ambienti complessi e in evoluzione, come le strade di una città o gli interni di un edificio. L'avanzamento delle tecnologie dei sensori, come le telecamere e i sensori di profondità, ha reso possibile la raccolta di dati dettagliati sull'ambiente circostante. Questi dati sono stati utilizzati per sviluppare algoritmi di SLAM (Simultaneous Localization and Mapping) in grado di combinare informazioni sulla posizione del robot con la creazione di una mappa dell'ambiente, raggiungendo un livello superiore di complessità. Infatti, i moderni sistemi SLAM sono in grado di mappare un ambiente sconosciuto e determinare la posizione di un agente mobile all'interno di tale ambiente, combinando dati provenienti da sensori come telecamere, LiDAR, sonar o sensori inerziali per creare una mappa e stimare la posizione

dell'agente simultaneamente. I dati raccolti vengono poi elaborati per estrarre le caratteristiche significative dell'ambiente. Ad esempio, utilizzando algoritmi Computer Vision (CV), le telecamere possono rilevare punti di riferimento visivi o bordi degli oggetti. I dati del LiDAR possono essere utilizzati per identificare punti di scansione tridimensionali o ostacoli nell'ambiente. Le features estratte dai diversi frame o istantanee dei sensori vengono confrontate per trovare corrispondenze tra di esse. Ad esempio, si cerca di identificare gli stessi punti di riferimento visivi o gli stessi punti di scansione LiDAR in diverse istantanee, questo passo è importante per determinare come le diverse istantanee sono collegate tra loro nello spazio tridimensionale. Utilizzando le corrispondenze delle feature e i dati dei sensori, SLAM stima la posizione dell'agente mobile all'interno dell'ambiente e aggiorna la mappa dell'ambiente in tempo reale. Questo processo coinvolge l'utilizzo di algoritmi di localizzazione, come l'estensione del filtro di Kalman o i metodi di ottimizzazione basati su grafi, in modo da stimare la posizione dell'agente rispetto alla mappa esistente. Man mano che l'agente si sposta nell'ambiente, i dati dei sensori vengono continuamente acquisiti, le feature estratte, le corrispondenze vengono trovate e la posizione e la mappa vengono aggiornate. Questo processo di aggiornamento continuo consente di migliorare gradualmente la precisione della mappa e della stima della posizione, l'obiettivo finale è ottenere una mappa accurata dell'ambiente e una stima affidabile della posizione dell'agente all'interno di essa. Questi moderni sistemi hanno reso possibile il mapping e la localizzazione in tempo reale, aprendo nuove possibilità per una vasta gamma di applicazioni.

2.1.2 Path planning

Con il concetto di *planning* (pianificazione) si intende un processo che consiste nella determinazione di una sequenza di azioni da intraprendere per spostarsi da una posizione di partenza a una destinazione desiderata, evitando ostacoli e seguendo determinati criteri di ottimalità. La definizione formale è "*trovare un movimento privo di collisioni tra una configurazione iniziale (start) e una configurazione finale (goal) all'interno di un ambiente specifico*" [18]. Il concetto di planning può essere diviso in due sotto-concetti: path planning (tecniche per la determinazione delle traiettorie che il robot deve percorrere per raggiungere la configurazione finale evitando gli ostacoli) e path following (tecniche per l'esecuzione delle traiettorie generate dal path planning evitando gli ostacoli imprevisti). Il path planning, può essere suddiviso a sua volta in due categorie: globale e locale ed è dimostrato che utilizzare una combinazione di queste due tecniche è utile in quanto permette all'agente di focalizzarsi sugli obiettivi prossimi mantenendo comunque l'obiettivo globale in considerazione. Nella Figura 2.5 si osserva un agente robotico che ha un obiettivo specifico da raggiungere, la cui destinazione è evidenziata dalla freccia in rosso. La curva che va dal robot alla freccia rappresenta la pianificazione globale, ovvero il percorso generale che il robot deve seguire per raggiungere la destinazione desiderata. In aggiunta, l'immagine evidenzia anche un'area circoscritta in blu, all'interno del quale vengono considerate informazioni dettagliate sugli ostacoli, le condizioni del terreno e altre variabili importanti. Il path planning locale sfrutta questa area dettagliata e permette al robot di prendere decisioni in tempo reale per evitare ostacoli, seguire il percorso globale in modo sicuro e adattarsi a situazioni impreviste [45].

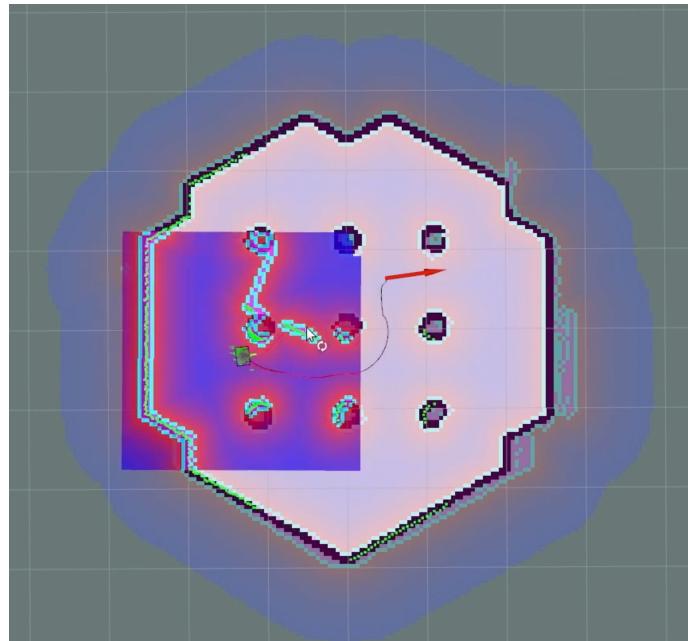


Figura 2.5: Screenshot del software ROS [15] dove si evidenzia che l’obiettivo (freccia rossa) ed il percorso da effettuare per raggiungerlo (curva che va dal robot alla freccia). Il path planning locale tiene in considerazione le informazioni riguaranti ciò che è prossimo a lui (riquadro in blu) e fornisce comandi i comandi di movimento al robot.

Il path planning globale si riferisce alla pianificazione di un percorso completo dall’inizio alla fine, considerando l’intero ambiente e tutti gli ostacoli presenti. Questo tipo di pianificazione è utile quando è richiesta una navigazione accurata e precisa in un ambiente noto, uno degli algoritmi utilizzati per questo caso A*. D’altra parte, il path planning locale è orientato alla pianificazione di azioni di navigazione immediate per evitare ostacoli imprevisti e reagire a situazioni in tempo reale. Quest’ultima è particolarmente utile quando il robot si muove in un ambiente dinamico e deve adattarsi alle condizioni in continua evoluzione. È di fondamentale importanza introdurre alcuni concetti utili per la descrizione e la pianificazione dei movimenti di un robot: spazio di configurazione (C-space), spazio delle configurazioni libere (C-free) e rappresentazione degli ostacoli nello spazio di configurazione (C-obs):

- Lo spazio di configurazione (C-space) rappresenta l’insieme di tutte le possibili configurazioni che un robot può assumere. Una configurazione è una specifica disposizione delle variabili che definiscono lo stato del robot, come posizione,

orientamento, velocità, giunti o altre caratteristiche rilevanti per il movimento del robot, è quindi un insieme multidimensionale che descrive tutte le possibili combinazioni di queste variabili;

- Lo spazio delle configurazioni libere (C-free) rappresenta l'insieme delle configurazioni nello spazio di configurazione che il robot può raggiungere senza collisioni con gli ostacoli presenti nell'ambiente. È l'insieme delle configurazioni che consentono al robot di muoversi liberamente senza incontrare ostacoli fisici (è quindi una sottoinsieme dello spazio di configurazione che soddisfa i vincoli di collisione);
- La rappresentazione degli ostacoli nello spazio di configurazione (C-obs) consiste nella mappatura degli ostacoli presenti nell'ambiente nello spazio di configurazione del robot, implica la definizione di una funzione o di un insieme di regole che determinano quali configurazioni del robot sono considerate collisioni con gli ostacoli. La rappresentazione degli ostacoli nello spazio di configurazione può essere utilizzata per valutare la fattibilità di un percorso o per generare un insieme di configurazioni valide per il movimento del robot.

Esistono anche degli algoritmi di planning globale basate su mappe geometriche [18], in particolare roadmap, decomposizione in celle e campi di potenziale.

Le tecniche di roadmap si basano sulla riduzione dello spazio di configurazione ad un insieme di percorsi unidimensionali all'interno dello spazio delle configurazioni libere (C-free) o nella sua chiusura, permettendo di mappare la connettività dello spazio libero utilizzando curve unidimensionali. La roadmap R ottenuta contiene un insieme di cammini che collegano la configurazione iniziale con quella finale, consentendo così di trovare un percorso percorribile tra le due configurazioni. Spesso, a quest'ultima, è associato un grafo nel quale i nodi rappresentano le configurazioni e gli archi rappresentano i collegamenti tra di essi. Prendendo come riferimento la **Figura 2.6**, nodi del grafo indicano le posizioni dei punti, mentre i bordi rappresentano le connessioni visibili tra i nodi. Le aree grigie indicano gli ostacoli da evitare.

È possibile utilizzare indici di ottimalità, come la distanza euclidea, per la ricerca del percorso ottimo all'interno del grafo.

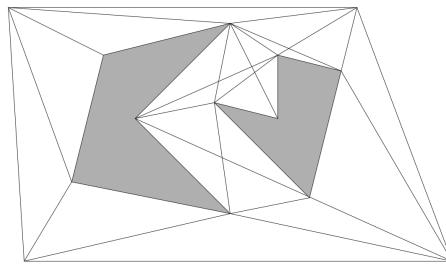


Figura 2.6: Grafo di visibilità (visibility graph) [18]

La decomposizione delle celle permette di utilizzare le tecniche di ricerca su grafi per trovare il percorso desiderato. In questo approccio, come si nota nelle immagini della **Figura 2.7**, lo spazio libero del robot viene suddiviso in diverse regioni (chiamate celle), scelte in modo tale che sia semplice generare un percorso tra due configurazioni all'interno della stessa cella. È quindi possibile definire un grafo di connettività, chiamato connectivity graph, che rappresenta le relazioni di adiacenza tra le celle. I nodi, rappresentano le celle estratte dallo spazio libero e gli archi collegano i nodi corrispondenti alle celle adiacenti, l'unione delle celle rappresenta esattamente lo spazio libero.

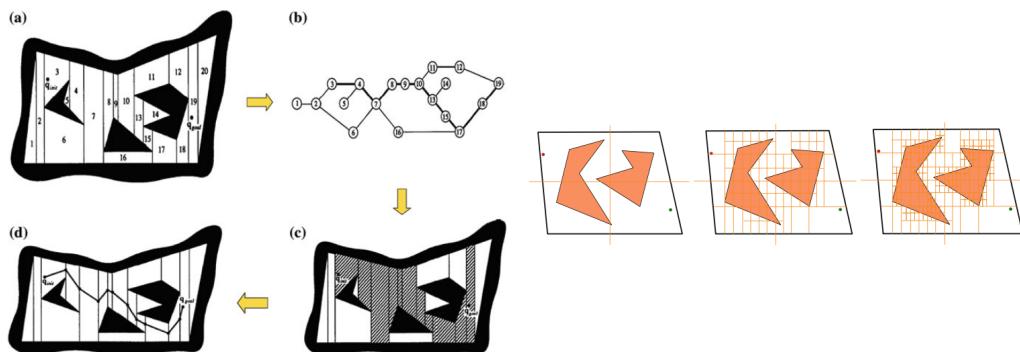


Figura 2.7: Le immagini di destra e sinistra mostrano due esempi di scomposizione attraverso i cell decomposition methods [18]

È importante precisare che in alcuni casi, un calcolo esatto dello spazio libero (*exact cell decomposition technique*) non è possibile o conveniente, devono quindi essere impiegati metodi di decomposizione cellulare approssimati (*approximate cell*

decomposition). Gli Artificial Potential Methods (metodi del potenziale artificiale) rappresentano un approccio alternativo per la pianificazione del percorso, si basano su concetti di potenziale e forze virtuali per guidare il movimento del robot in un ambiente. Nonostante richiedano una conoscenza accurata dell'ambiente, potrebbero richiedere un'iterazione manuale per ottenere risultati ottimali. Sono stati introdotti da Khatib [28], sono relativamente semplici da implementare e possono essere applicati a diversi tipi di robot e ambienti. L'idea di base è quella di definire delle “forze” o “potenziali” che agiscono su specifiche regioni o punti dell’immagine, in modo da guidare il processo di analisi o di estrazione delle informazioni desiderate. Questi metodi considerano il robot nello spazio delle configurazioni come un punto in movimento soggetto a un campo potenziale generato dalla configurazione di destinazione (goal) e dagli ostacoli presenti nel C-space. L’idea è quella di creare un campo di potenziale artificiale, nell’ambiente circostante il robot, progettato in modo tale che il robot sia attratto verso le posizioni desiderate come obiettivi o punti di interesse, in modo che sia respinto da ostacoli o aree pericolose. Nell’implementazione, vengono definiti due tipi di potenziali: attrattivo e repulsivo. Il primo viene applicato alle posizioni desiderate, creando un gradiente che spinge il robot verso tali posizioni, mentre il secondo viene applicato agli ostacoli, generando una forza che respinge il robot da tali ostacoli. Il potenziale totale è la somma di un potenziale attrattivo generato dalla configurazione di destinazione e di un potenziale repulsivo generato dagli ostacoli. Il robot, quindi, naviga nell’ambiente seguendo il gradiente di potenziale artificiale e muovendosi lungo questo gradiente, tende a evitare gli ostacoli e raggiungere le posizioni desiderate. Può essere utilizzato per vari scopi, come l’individuazione e l’estrazione di oggetti di interesse, il riconoscimento di pattern o la segmentazione dell’immagine.

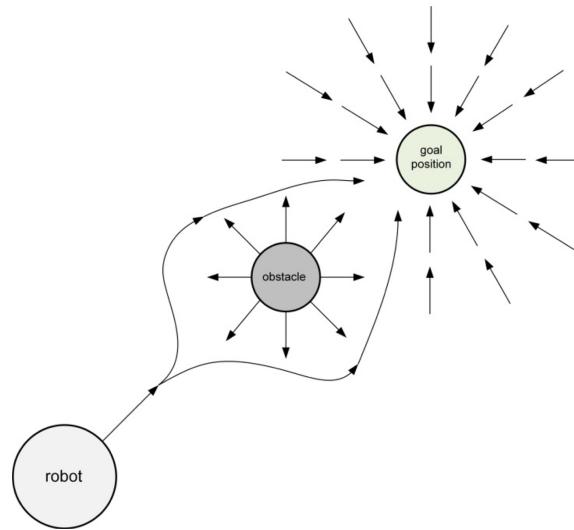


Figura 2.8: Esempio di simulazione di campo di potenziale artificiale [18].

Tuttavia, questi metodi possono essere soggetti a minimi locali, nei quali il robot può rimanere bloccato. Sono state proposte diverse soluzioni per affrontare questo problema, come l'utilizzo di funzioni potenziali di navigazione senza minimi locali o l'uso di pianificatori di percorsi casuali (RPP) per evitare minimi locali combinando concetti di campo potenziale artificiale con tecniche di ricerca casuale. Vari esempi di RPP sono stati trovati in [4] [6] [7] [8].

2.2 Algoritmi di navigazione basati sul learning

A differenza degli algoritmi di navigazione classici che si basano su regole euristiche definite “manualmente”, gli algoritmi di learning consentono al robot di apprendere automaticamente dai dati e dalle esperienze raccolte, in quanto sono progettati per consentire al robot di acquisire conoscenza e adattarsi all’ambiente circostante attraverso l’analisi dei dati. Una delle principali differenze con gli algoritmi classici risiede nel fatto che non richiedono una programmazione esplicita delle regole o delle procedure per affrontare un problema specifico: al contrario, un modello di machine learning, apprende autonomamente dai dati e adatta i propri parametri per migliorare le prestazioni nel tempo rendendo i modelli ottenuti più flessibili ed adatti a risolvere problemi complessi in cui le regole esatte o le soluzioni analitiche non sono facilmente determinabili. In generale, il processo di apprendimento coinvolge la raccolta di un insieme di dati di addestramento (training set), la costruzione di un modello o una rappresentazione del problema e l’ottimizzazione del modello attraverso l’uso di algoritmi di Machine Learning (ML). Uno dei metodi di ML più comuni è il *supervised learning* (learning supervisionato), che è anche il metodo più semplice dove un “supervisore” fornisce un insieme di dati di addestramento, in cui ogni sample (esempio) è associato a una corretta label (etichetta) o azione da compiere per addestrare il modello di apprendimento a comprendere il contesto e le azioni appropriate da compiere. Ad esempio, potrebbe essere fornito un insieme di dati di addestramento contenente immagini dell’ambiente circostante e le azioni corrispondenti da intraprendere (“vai avanti”, “ruota a sinistra”, etc.) ed il modello cercherà di apprendere una mappatura tra le immagini e le azioni desiderate. Nell’ambito dell’apprendimento di una policy (politica) di navigazione, ovvero di un insieme delle azioni da intraprendere per raggiungere la destinazione, il supervised learning viene denominato *Behavior Cloning* (BC, clonazione del comportamento) [16], il cui obiettivo è insegnare al modello a riprodurre il comportamento desiderato in base all’osservazione dei dati di addestramento forniti. Un agente, viene addestrato a imitare un comportamento o una sequenza di azioni basandosi su un insieme di dati di addestramento che contiene le coppie di input e output corrispondenti. In

sostanza, l'algoritmo viene addestrato per mappare gli stati o le osservazioni di input direttamente alle azioni o alle decisioni di output, un approccio utile quando si desidera che un agente imiti il comportamento specifico di un umano esperto. Questa strategia di apprendimento ha permesso di ottenere risultati convincenti su robot reali che generalizzano complessi comportamenti a nuovi scenari non strutturati [60] [17] [59].

In [52], gli autori si concentrano su un'evoluzione del BC chiamato *Behavioral Cloning from Observation* (BCO, clonazione del comportamento da osservazione), introducendo l'uso delle osservazioni visuali come input per il learning. Invece di basarsi solo sulle coppie di stato-azione come nel BC tradizionale, il BCO sfrutta le osservazioni dell'ambiente, come immagini o sequenze video, per apprendere un comportamento desiderato, permettendo all'agente di apprendere un comportamento complesso attraverso l'osservazione diretta, senza richiedere una conoscenza esplicita delle azioni corrispondenti. Il BCO differisce dalle altre tecniche per imitazione in quanto combina l'apprendimento del modello dinamico inverso con l'apprendimento di una policy di imitazione, consentendogli di fornire agli agenti una conoscenza preliminare di se stessi e del proprio ambiente, che può migliorare le prestazioni e ridurre la necessità di informazioni esplicite sull'azione. È particolarmente utile in scenari in cui le azioni non possono essere direttamente misurate o acquisite, come la guida autonoma, dove l'agente deve apprendere a navigare in un ambiente dinamico basandosi su input visivi. In questo contesto, l'agente impara a guidare in modo sicuro osservando esempi di comportamenti appropriati senza la necessità di esplicita programmazione delle azioni di guida. In base a come vengono utilizzate le label, si possono distinguere altri due tipi di tecniche per l'apprendimento: semi-supervised (semi-supervisionato) e unsupervised (non supervisionato). L'apprendimento semi-supervised si basa su un set di dati che contiene sia esempi etichettati che non, mirando a ridurre la quantità di dati etichettati necessari, sfruttando le informazioni contenute nei dati non etichettati. Una limitazione è che i dati di addestramento non etichettati potrebbero fornire informazioni irrilevanti o persino condurre a decisioni errate. Per ottenere prestazioni ottimali è necessario bilanciare dati etichettati e non. L'apprendimento unsupervised (non supervisio-

nato) è un approccio in cui l'agente apprende dalle caratteristiche interne dei dati di input senza l'uso di etichette, utile quando non sono disponibili dati etichettati. Le tecniche di clustering, riduzione della dimensionalità e reti generative sono spesso utilizzate in quest'ambito. Anche questo tipo di apprendimento ha alcune limiti, come l'incapacità di fornire informazioni accurate sull'ordinamento dei dati e la complessità computazionale. In Figura 2.9, sono presentati esempi di risultati ottenibili tramite tecniche di apprendimento unsupervised.

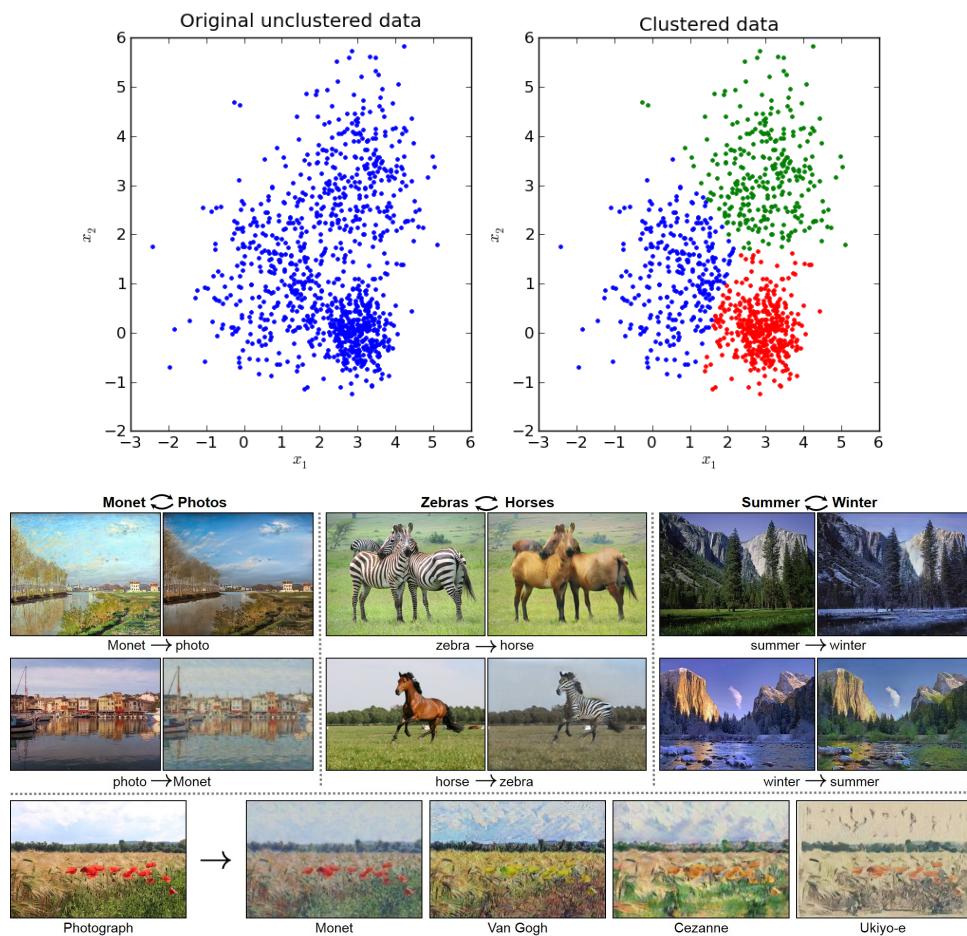


Figura 2.9: Esempi di risultati ottenuti da tecniche unsupervised. In alto un cluster [49], in basso un esempio di generazione di dati per mezzo di GAN [61].

In alto è mostrato il risultato di tecniche di *clustering* che raggruppano dati simili in gruppi, consentendo di identificare pattern o strutture nei dati senza la necessità di etichette. Nella parte inferiore dell'immagine, è rappresentato un esem-

pio di generazione di dati tramite *Generative Adversarial Network* (GAN). Le GAN sono un tipo di modello di machine learning che può generare dati nuovi, sintetici, imparando dalle caratteristiche dei dati di input. Queste reti sono utilizzate per la generazione di immagini, testo o altri tipi di dati e trovano applicazioni in settori come l'elaborazione delle immagini, la grafica computerizzata e la creazione di contenuti artificiali. È necessario precisare, però, che i metodi di semi-supervised learning e unsupervised learning non sono sempre adatti per l'implementazione di modelli per i sistemi di navigazione a causa della difficoltà ad ottenere performance accettabili in mancanza di etichette e di gestire l'incertezza degli ambienti di navigazione, oltre alla necessità di bilanciare l'esplorazione e lo sfruttamento delle azioni e ad adattarsi a nuovi scenari o ambienti complessi.

Invece, a differenza delle tecniche precedentemente discusse, il Reinforcement Learning (apprendimento per rinforzo) permette di ottenere risultati migliori nel contesto della navigazione autonoma rispetto agli approcci descritti precedentemente. Un agente apprende attraverso l'interazione con l'ambiente e la ricezione di feedback sotto forma di ricompense o punizioni, permettendo all'agente di esplorare l'ambiente e capire quali sono le azioni migliori da intraprendere in diverse situazioni. La Figura 2.10 aiuta la comprensione di questo concetto. Questa capacità di apprendimento dinamico ed adattativo è ciò che lo rende particolarmente adatto alla navigazione autonoma, in cui l'agente deve prendere decisioni in tempo reale sulla base delle informazioni provenienti dall'ambiente circostante.

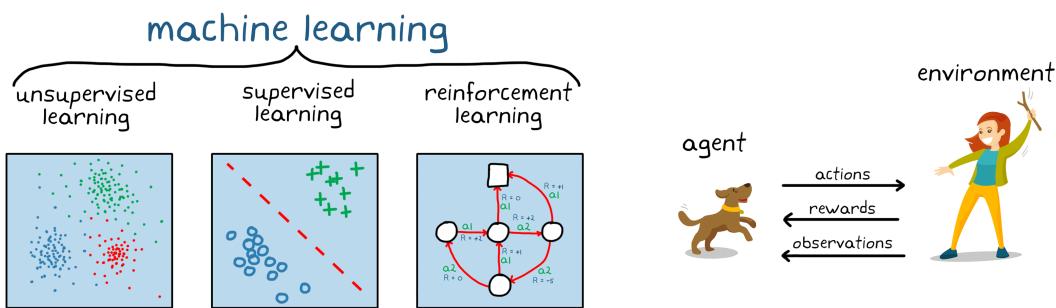


Figura 2.10: A sinistra, una classificazione dei vari metodi di machine learning [36]. A destra, un'immagine che aiuta a capire il funzionamento del reinforcement learning durante l'addestramento di un cane [41].

2.2.1 Breve storia del Machine Learning e del Deep Learning

Gli algoritmi di Machine Learning rappresentano una branca dell'intelligenza artificiale che si basano sull'idea che i modelli possano essere addestrati utilizzando dati di input e le relative labels, al fine di estrarre pattern e relazioni significative in modo da effettuare previsioni o prendere decisioni su nuovi dati non visti in fase di addestramento. Lo studio del Machine Learning risale agli anni '50 e '60, quando i primi algoritmi di apprendimento automatico furono sviluppati per risolvere problemi di *Pattern Recognition* [12] (riconoscimento dei pattern) e del linguaggio naturale nonostante la mancanza di dati di addestramento e le limitate risorse computazionali ne hanno limitato l'utilizzo pratico. Negli anni '80 e '90, con l'aumento della disponibilità di dati e l'avanzamento della capacità di calcolo, con l'avvento del *Deep Learning* (DL, apprendimento profondo), si è assistito a una evoluzione significativa in questo campo. I metodi di Deep Learning sono una sotto-categoria di tecniche utilizzate nell'ambito del Machine Learning che si basano sull'uso di reti neurali *Deep* (profonde) con più strati di neuroni (o percettroni, unità o nodi), gli elementi fondamentali e semplici di una rete neurale. Come si può evidenziare nella Figura 2.11, la struttura di un percettrone è ispirata al funzionamento dei neuroni biologici presenti nel cervello umano, nonostante siano un'astrazione matematica che opera su dati numerici.

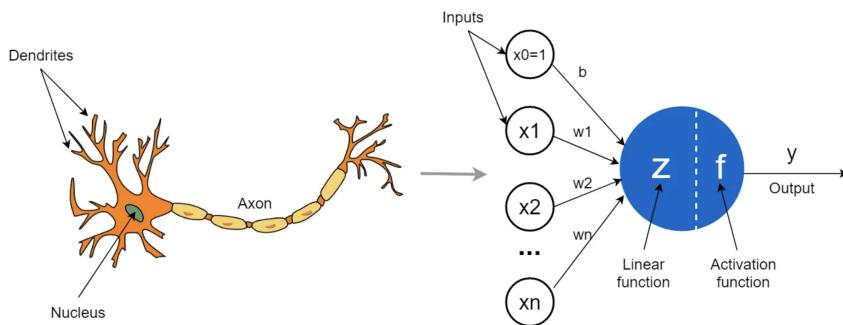


Figura 2.11: [39] Un neurone biologico a sinistra, un neurone artificiale a destra.

Le *Deep Neural Network* (DNN) permettono di apprendere rappresentazioni complesse e stratificate dei dati tramite il processo di retropropagazione dell'errore (back-propagation), offrendo vantaggi significativi rispetto alle reti neurali tradizionali, consentendo alle reti neurali profonde di migliorare le loro prestazioni e

apprendere in modo più efficace [43]. La principale differenza tra un algoritmo di deep learning e un algoritmo di machine learning convenzionale riguarda la complessità e la profondità della rete neurale utilizzata per l'apprendimento. Nel machine learning tradizionale si utilizzano spesso modelli *supervised* (supervisionati) come Supporting Vector Machine (SVM), Decision-Trees, Random Forest, KNN che si basano su funzioni di decisione relativamente semplici e si adattano bene a dati strutturati e a problemi di dimensioni ridotte. Il deep learning, invece, si distingue per l'utilizzo di modelli profondi, mostrati in [Figura 2.12](#) composti da molteplici strati di percettroni interconnessi.

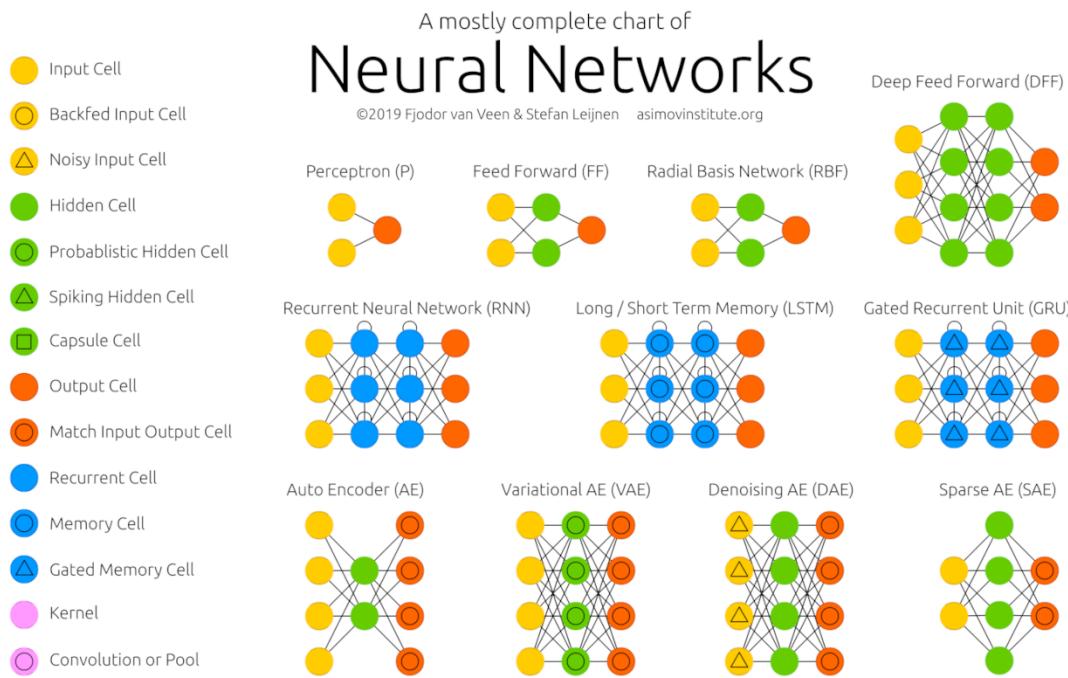


Figura 2.12: Una parte del *The Neural Network Zoo*, [31], l'immagine completa offre una panoramica generale delle architetture DNN presenti allo stato dell'arte.

Questi strati intermedi, noti come *hidden layers*, permettono al modello di imparare le rappresentazioni gerarchiche dei dati, riuscendo a cogliere informazioni complesse e astratte. La [Figura 2.13](#) evidenzia le analogie e le differenze. La *simple neural network* (rete neurale semplice) è formata da un numero limitato di hidden layers. Ogni neurone riceve un insieme di input, applica una funzione di attivazione e passa l'output ai neuroni successivi fino a raggiungere lo strato di output. La *deep*

learning neural network (rete neurale di apprendimento profondo) è rappresentata come una pila di numerosi strati di neuroni dove ogni strato ha una serie di neuroni collegati a quelli dello strato successivo. Questa architettura consente di creare DNN con molteplici strati nascosti, che possono variare da poche decine a centinaia o addirittura migliaia. Entrambe le reti utilizzano neuroni artificiali per elaborare informazioni attraverso le connessioni tra i neuroni e possono apprendere dai dati attraverso un processo di addestramento, regolando i pesi delle connessioni tra i neuroni per minimizzare una funzione di errore.

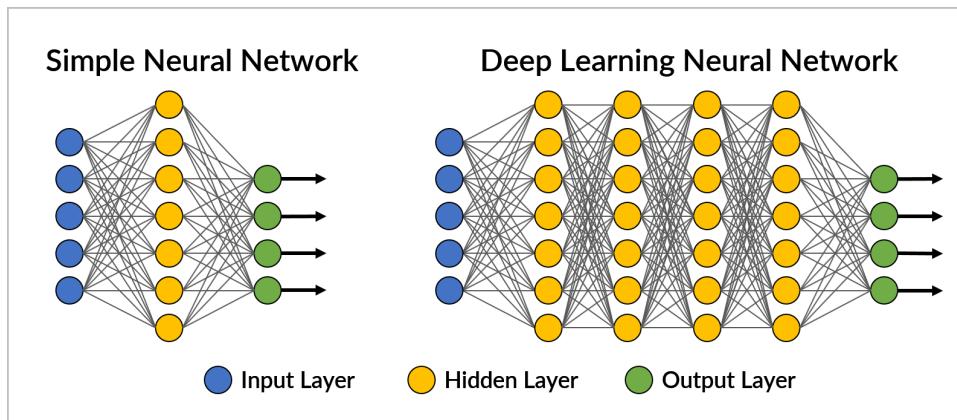


Figura 2.13: Una *Simple Neural Network* (rete neurale semplice) a sinistra e *Deep Neural Network* a destra [27]

Nonostante siano caratterizzate da un elevato numero di parametri e richiedano un notevole sforzo computazionale per l’addestramento, esse hanno dimostrato un’enorme efficacia in molte applicazioni, come l’*image recognition* (riconoscimento di immagini) [30], *speech recognition* (riconoscimento vocale) [20], *sign language recognition* (riconoscimento della lingua dei segni) [57].

2.2.2 Reinforcement Learning

Le origini del *Reinforcement Learning* (RL, apprendimento per rinforzo) risalgono agli anni '50 e '60, quando i ricercatori iniziarono a sviluppare algoritmi per far apprendere ai computer a prendere decisioni in modo autonomo attraverso l'interazione con l'ambiente [48] [55] [38]. Questa tecnica consiste nel fare imparare ad un agente cosa fare, cioè trovare un modo per far corrispondere a delle situazioni determinate azioni in modo da massimizzare la *reward* (ricompensa), come accade in un *markov decision process* (introdotto successivamente) [40]. A differenza di come si possa pensare, non si tratta di un metodo di ML unsupervised in quanto non punta a trovare una struttura nascosta nei dati. Il *learning agent* (agente apprendente) deve essere in grado di percepire lo stato dell' ambiente e deve essere in grado di intraprendere azioni che lo influenzano oltre a mantenere uno o più obiettivi relativi allo stato finale dell'ambiente. A quest'ultimo non viene detto quali azioni intraprendere, esso deve invece scoprire quali lo porteranno ad una maggiore ricompensa. Il RL si basa su due concetti fondamentali: *trial-and-error search* e *delayed reward*.

- *trial and error search*: per ottenere molte rewards, un agente deve trovare un buon compromesso tra scelte già effettuate e nuove: deve sfruttare ciò che ha già sperimentato per ottenere rewards “sicure”, ma deve anche esplorare per effettuare migliori scelte in futuro. Attraverso l’iterazione di molteplici tentativi ed errori, l’agente o il sistema impara gradualmente quali azioni o soluzioni sono più efficaci per raggiungere l’obiettivo;
- *delayed reward*: si riferisce alla situazione in cui la ricompensa o il feedback positivo o negativo per le azioni intraprese da un agente possono essere ritardati nel tempo. A differenza di altri tipi di apprendimento, dove il feedback è immediato e diretto, nel RL l’agente può dover aspettare un certo periodo di tempo prima di ricevere una ricompensa o un feedback sulla bontà delle sue azioni. Diverse strategie possono essere adottate per affrontare questo problema, come l’uso di algoritmi di apprendimento basati su sconti (discounting), che attribuiscono meno importanza alle ricompense ritardate nel tempo.

Inoltre, l'uso di funzioni di valore, come la Q-function, può aiutare l'agente a valutare le azioni sulla base delle ricompense future attese.

Al di là dell'agente e dell'ambiente, si possono identificare altri quattro sottoelementi principali: una *policy*, un *reward signal*, una *value function* e, facoltativamente, un *model of environment* [40]. La **policy** (politica, inteso come "modo di agire") definisce il modo di comportarsi del learning agent in un dato momento: è un'associazione tra gli stati percepiti dell'ambiente e le azioni da intraprendere quando ci si trova in quegli stati (in psicologia, un insieme di regole o associazioni stimolo-risposta). In alcuni casi, può essere una semplice funzione o una tabella di ricerca, mentre in altri può comportare calcoli estesi come per un processo di ricerca. Da sola è sufficiente a determinare il comportamento di un agente. Può essere stocastica, specificando le probabilità per ogni azione. Il *reward signal* (segnale di ricompensa) definisce quali sono gli eventi positivi e negativi per l'agente. L'ambiente invia all'agente periodicamente una *reward*, che fornisce una misura di valutazione del successo o della qualità delle azioni compiute da un agente intelligente. Viene assegnata dopo ogni azione eseguita dall'agente e rappresenta il feedback che il sistema riceve sull'efficacia delle sue scelte (in biologia si potrebbe pensare alle rewards come analoghe alle esperienze di piacere o dolore). Non indica come raggiungere l'obiettivo o quale azione specifica compiere, è compito dell'agente apprendere attraverso il processo di *trial-and-error*, esplorazione e sperimentazione, quali azioni siano più efficaci per massimizzarle. La **value function** (o *state value*, funzione valore) indica ciò che è buono a lungo termine per l'agente, a differenza del reward signal che indica ciò che è buono in senso immediato. Rappresenta l'importo totale della reward che un agente può aspettarsi di accumulare nel futuro (a partire da quello stato). I *values* indicano la desiderabilità a lungo termine degli stati dopo aver preso in considerazione gli stati che probabilmente seguiranno e le rewards associate. Ad esempio, uno stato potrebbe sempre produrre una ricompensa immediata bassa ma avere comunque un valore elevato perché è regolarmente seguito da altri stati che producono ricompense elevate, oppure il contrario. Solitamente possono essere utilizzate diverse value functions: la Q-function e la V-Function. La

Q-function associa uno stato e un’azione a un valore che rappresenta la somma delle ricompense attese nel corso del tempo. La V-function, invece, associa uno stato a un valore che rappresenta la somma delle ricompense attese prendendo le migliori decisioni possibili da quel punto in poi. Queste funzioni sono fondamentali poiché guidano le decisioni dell’agente nell’esplorazione e nell’ottimizzazione delle azioni da intraprendere per massimizzare le ricompense cumulative nel lungo termine.

Il model of environment (modello dell’ambiente) è qualcosa che “imita” il comportamento dell’environment o, più in generale, che consente di fare inferenze su come si comporterà. Ad esempio, dati uno stato e un’azione, il modello potrebbe prevedere lo stato successivo risultante e la ricompensa successiva. È utilizzato per pianificare le azioni future dell’agente, ovvero il modo di decidere su una linea di condotta considerando tutte le possibili situazioni future prima che vengano effettivamente vissute. Le metodologie che si avvalgono questi modelli sono chiamati *model based*, in contrasto con metodi più semplici privi di modelli che sono esplicitamente apprendenti per tentativi ed errori, è necessario raccogliere molti episodi di interazione con l’ambiente il che lo rende difficilmente utilizzabile direttamente nel mondo reale e l’addestramento dei modelli di DL in simulazione è una tappa forzata. Si dice quindi che il RL è “sample inefficient”. Il Reinforcement Learning (RL) ha aperto nuove possibilità entusiasmanti, non solo per i risultati finali ottenuti, che spesso sono di per sé molto interessanti, ma soprattutto perché i robot finali acquisiscono abilità attraverso l’apprendimento basato su prove ed errori. Tuttavia, un punto cruciale da considerare è il fatto che il processo di apprendimento in RL richiede un numero significativo di episodi di navigazione per elaborare e ottimizzare una policy efficace. Questo aspetto presenta una sfida considerevole quando si tratta di implementare il RL su robot fisici. La necessità di eseguire numerosi episodi di prova ed errori per apprendere una policy ottimale implica una richiesta di tempo che, nei contesti del mondo reale, risulta spesso proibitiva. Oltre alla questione temporale, emerge anche la fragilità dei robot fisici. L’esecuzione di innumerevoli tentativi ed errori potrebbe comportare un rischio significativo di danneggiare o usurare il robot, innalzando preoccupazioni sulla sua durata e sostenibilità a lungo termine. La necessità di preservare l’integrità fisica dei robot durante il processo di apprendi-

mento diventa quindi una sfida pratica da affrontare. Di conseguenza, l'approccio prevalente è l'utilizzo di simulazioni virtuali per la fase di apprendimento iniziale. Questo consente di eseguire gli svariati episodi necessari in un ambiente virtuale senza dover affrontare le limitazioni temporali e le preoccupazioni sulla resistenza fisica dei robot reali. Una volta che la policy è stata consolidata e ottimizzata attraverso l'apprendimento virtuale, può essere trasferita con maggiore sicurezza su un robot fisico. Questa metodologia di apprendimento in due fasi, con l'uso di simulazioni virtuali come banco di prova iniziale, ha dimostrato di essere un approccio pratico e prudente per garantire che i robot fisici possano acquisire competenze senza dover essere esposti a eccessivi rischi o a processi prolungati di apprendimento. L'integrazione sinergica di simulazioni virtuali e robot fisici rappresenta un passo importante verso la creazione di agenti intelligenti robusti e affidabili in scenari del mondo reale. Il framework su cui si basa è il **Markov Decision Process** (MDP), assumendo che l'agente possa osservare lo stato dell'ambiente, altrimenti si parlerebbe di MDP parzialmente osservabili.

2.2.2.1 Markov Decision Process

Il Markov Decision Process (MDP) è il framework matematico su cui si basa il reinforcement learning. In linea generale, definisce un *environment* (ambiente di decisione) stocastico in cui un agente può prendere decisioni in base allo stato corrente dell'ambiente. L'*environment* viene rappresentato come una serie di stati, azioni disponibili e transizioni di stato probabilistiche a cui è associata a una *reward* (ricompensa) che rappresenta il feedback fornito all'agente dopo l'esecuzione di un'azione in uno stato [40]. Il reinforcement learning utilizza i MDP per modellare problemi di decisione sequenziale (come ad esempio quello della navigazione) e ne fornisce una rappresentazione formale. Il goal (obiettivo) dell'agente è apprendere una policy decisionale che massimizzi le ricompense cumulative a lungo termine. La Figura 2.14 aiuta a comprendere meglio il concetto.

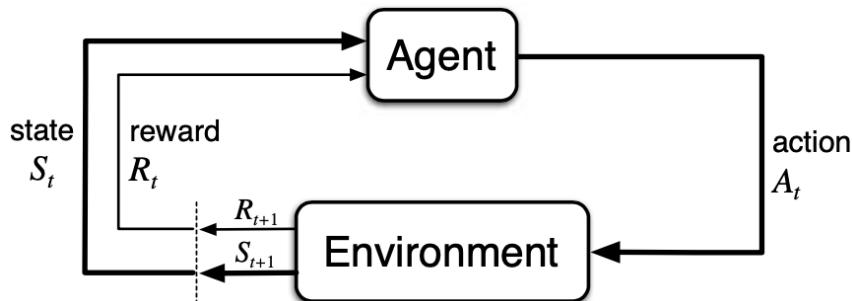


Figura 2.14: Figura che mostra l’interazione agente-ambiente in un MDP.

Seguendo lo schema, le *azioni* A_t sono le scelte fatte dall’agente; gli *stati* S_t sono la base per prendere le decisioni; le *ricompense* R_t sono la base per valutare le scelte da prendere. Tutto ciò che è interno all’agente è completamente noto e controllabile dall’agente stesso mentre tutto ciò che è esterno è parzialmente controllabile, e potrebbe non essere completamente noto. Agente ed ambiente interagiscono continuamente tra loro, l’agente seleziona le azioni e l’ambiente risponde. La proprietà fondamentale dei MDP recita: “*la probabilità di passare a uno stato successivo, data tutta la sequenza di stati passati, è uguale alla probabilità di passare allo stato successivo dato solo lo stato corrente.*” Formalmente:

$$[P(s_{t+1}|s_t, s_{t-1}, \dots, s_0) = P(s_{t+1}|s_t)] \quad (2.1)$$

La Formula 2.1 afferma che la probabilità condizionata di transizione da uno stato s_t a uno stato successivo s_{t+1} , tenendo conto di tutta la sequenza di stati passati s_{t-1}, \dots, s_0 , è uguale alla probabilità condizionata di transizione da s_t a s_{t+1} , dato solo lo stato corrente s_t . Dunque, per prendere una decisione, l’agente considera solo lo stato corrente e non deve tenere conto di tutta la “storia passata” dello stato, semplificando il problema e permettendo di utilizzare metodi di apprendimento efficienti per trovare policy decisionali ottime in modo da semplificare notevolmente la modellazione e la risoluzione di un MDP dato che non si presenta la necessità di memorizzare o considerare l’intera storia degli stati passati. La scelta delle reward è un aspetto cruciale e richiede una progettazione attenta, poiché influisce sul comportamento appreso dall’agente. Un altro parametro importante da considerare è il *discount factor* (fattore di sconto), utilizzato per bilanciare le *ricompense immediate*

rispetto alle *ricompense future* nel processo decisionale, esprimendo l'importanza relativa di queste rispetto alle immediate (vicino a 1 dà più importanza alle ricompense a lungo termine, mentre un valore più vicino a 0 dà più importanza alle ricompense immediate).

2.2.2.2 Deep Reinforcement Learning

Il *deep reinforcement learning* (DRL) è una tecnica che combina l'apprendimento *deep* con il Reinforcement Learning per risolvere problemi di decisione sequenziale in ambienti complessi, fa uso di DNN per approssimare la funzione di valore o la policy di un agente [32]. Come anticipato nella Sezione 2.2.1, questi modelli, sono in grado di apprendere rappresentazioni complesse dei dati, consentendo agli agenti di modellare l'ambiente e prendere decisioni in modo più accurato rispetto alla controparte *non deep*. L'adozione comporta alcune problematiche, in particolare:

- la necessità di grandi quantità di dati di addestramento: il training dei modelli effettuato mediante interazione con l'ambiente può richiedere un numero significativo di iterazioni e può essere computazionalmente costoso;
- il trade-off tra *exploration* (esplorazione) ed *exploitation* (sfruttamento): è necessario bilanciare il “desiderio” di scoprire nuove azioni e le relative ricompense per mezzo delle azioni già conosciute che hanno dimostrato di essere efficaci. Se l'agente si concentrassasse troppo sull'esplorazione (scegliere nuove azioni per valutarne l'efficacia), potrebbe perdere l'opportunità di sfruttare quelle già apprese per massimizzare le ricompense (utilizzare azioni già conosciute per ottenere ricompense ottenute in passato). D'altra parte, se l'agente si concentrassesse troppo sullo sfruttamento, potrebbe non scoprire nuove azioni che potrebbero a ricevere ricompense ancora più elevate;
- Stabilità dell'apprendimento: a causa dell'alta dimensionalità degli spazi di input è difficile andare a convergenza. Pertanto, sono necessarie tecniche avanzate per affrontare questi problemi e migliorare la stabilità dell'apprendimento, come l'utilizzo di value functions o policy approssimate [33], l'utilizzo di buf-

fer di riproduzione per memorizzare esperienze passate [34] [37] e tecniche di regolarizzazione.

Nonostante queste problematiche, il DRL ha dimostrato risultati notevoli in diverse applicazioni come il controllo di robot, i giochi, il controllo di processi industriali ed altro ancora [23]. La combinazione di queste due tecniche, nonostante le sfide implementative, offre la possibilità di affrontare problemi complessi in modo efficiente e di ottenere prestazioni superiori rispetto ad i modelli ottenuti con il machine learning [33] [2]. Rende più agevole gestire i problemi con spazi di stato e azione di grandi dimensioni [44] in quanto l'approccio permette all'agente di apprendere policy più sofisticate e adattive, migliorando le prestazioni complessive grazie al reinforcement learning. Nelle Sezioni 2.2.3.1, 2.2.3.2, 2.2.3.3 sono stati introdotti dei modelli ed una serie di algoritmi allo stato dell'arte che si fondano sui concetti appena discussi, ma prima è fondamentale introdurre gli aspetti cruciali per l'ottenimento di un modello di navigazione in grado di moversi in un contesto reale: le norme sociali.

2.2.3 Navigazione in presenza di persone

La navigazione in ambienti dinamici e l'adesione alle norme sociali sono aspetti fondamentali per un sistema di navigazione autonomo in grado di operare in contesti reali e interagire con altri agenti o persone. Le norme sociali definiscono i comportamenti accettati e le regole di interazione all'interno di una comunità o di un ambiente condiviso, oltre al fatto che possono variare a seconda del contesto culturale, di situazioni specifiche e dell'interpretazione individuale. Giocano un ruolo fondamentale per garantire la sicurezza, l'efficienza e il comfort delle interazioni tra il l'agente autonomo e gli altri utenti dell'ambiente. Possono includere regole di precedenza, segnaletica stradale, comportamenti per il sorpasso, regolamentazione delle velocità e così via. L'integrazione delle norme sociali all'interno di un sistema di navigazione autonoma comporta diverse sfide, una delle principali è la comprensione delle regole e delle dinamiche sociali all'interno di un contesto specifico: ciò richiede la capacità di riconoscere e interpretare il comportamento degli altri utenti,

comprendere le intenzioni e prevedere le loro azioni future. Sono individuate tramite l'analisi di contesti reali, ad esempio, usando la raccolta di dati sul comportamento degli utenti della strada attraverso sensori e l'analisi dei modelli comportamentali degli utenti in vari contesti (come si comportano in situazione di pericolo, in situazioni di sovraffollamento, come si effettua un sorpasso in maniera corretta). La differenza sostanziale tra un modello di navigazione che utilizza le norme sociali e uno che non lo fa, sta nella capacità di adattarsi e interagire in modo adeguato con gli altri utenti della strada. Un modello che le considera è in grado di prevedere le intenzioni degli altri agenti, rispettare le regole di precedenza, mantenere distanze di sicurezza e adottare comportamenti che siano accettati e comprensibili per gli altri utenti della strada contribuendo a rendere il movimento dell'agente sicuro, fluido e soprattutto prevedibile (in quanto si comporterà come un qualsiasi altro agente "reale", seguendo regole prefissate) per tutti gli altri agenti. D'altro canto, un modello che non tiene conto delle norme sociali, potrebbe comportarsi in modo non coerente con le aspettative degli altri utenti della strada, creando situazioni di pericolo (o conflitto), ad esempio, ignorando le regole di precedenza, tagliando la strada o non rispettando un'eventuale segnaletica. Infatti, non avendo un comportamento prevedibile sarà difficile per l'umano interagire con esso. Le immagini della Figura 2.15 mostrano un esperimento in un ambiente reale di un modello allenato a seguire le norme sociali e non.

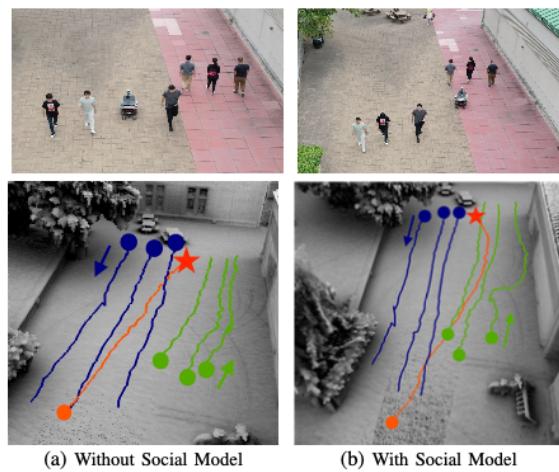


Figura 2.15: [58] Esperimento di navigazione in un ambiente reale in un'area di $10m \times 20m$ utilizzando modelli con norme sociali e non.

La prima riga mostra le foto di sei persone che camminano in due gruppi: un gruppo si muove lungo la direzione di navigazione del robot e l'altro gruppo si muove nella direzione opposta. La seconda riga mostra le traiettorie corrispondenti delle persone (blu e verde) e del robot (arancione). I punti indicano i punti di partenza e la stella indica il punto di arrivo del robot. In (a), quando non si applicano le norme sociali alla navigazione, il robot si dirige direttamente verso il punto obiettivo e taglia il gruppo a sinistra che si muove contro il robot. In (b), quando si applicano le norme sociali, il robot segue il gruppo sulla destra ed evita disturbi ai pedoni. Vari studi hanno tentato di risolvere il problema della navigazione autonoma seguendo le norme sociali con vari metodi, basati sulla predizione della traiettoria sui pedoni [1], [19], sui potential field [25] e sui modelli social force [22]. In [58] gli autori affermano che il sistema di navigazione ottenuto è in grado di operare in folle dense e di utilizzare le informazioni dei “gruppi sociali”, incorporando una deep neural network per tracciarli e unirsi al loro flusso, garantendo una navigazione sicura e naturale. La Figura 2.16 fornisce una panoramica del sistema di navigazione proposto.

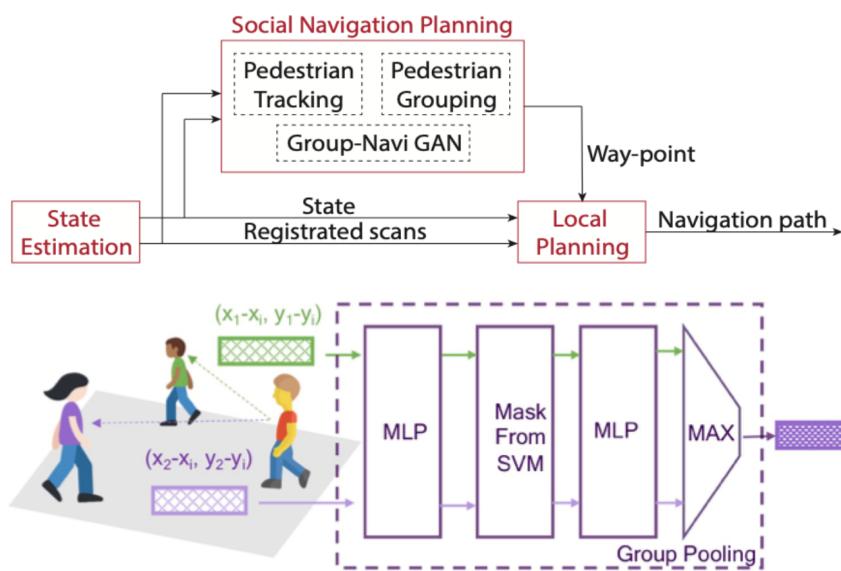


Figura 2.16: [58] In alto il diagramma del sistema di navigazione proposto composto da tre sottosistemi. In basso, la rete Group-Navi GAN

Come si evince dal diagramma in alto, il sottosistema è diviso in tre parti: stima dello stato (*state estimation*), pianificazione locale (*local planning*) e pianificazione

della navigazione sociale (*social navigation planning*). Il sottosistema di stima dello stato prevede una pipeline di elaborazione dati multistrato che sfrutta sensori LiDAR, visione e rilevamento inerziale, calcola la posizione del veicolo e registra i dati della scansione laser con la posa calcolata. Il sottosistema di pianificazione locale è un sottosistema di pianificazione di basso livello incaricato di evitare gli ostacoli nelle vicinanze del veicolo. L'algoritmo di pianificazione calcola una serie di traiettorie e percorsi privi di collisioni affinché il veicolo possa navigare mentre il sottosistema di pianificazione accetta osservazioni costituite solo da pedoni sottraendo la mappa precedente tracciando i pedoni nei dintorni del veicolo, quindi estrae le informazioni di raggruppamento dagli schemi di camminata dei pedoni, con le quali il sottosistema genera punti di passaggio (come input del sottosistema di pianificazione locale). L'input del modulo sono gli spostamenti relativi dei pedoni circostanti rispetto all'agente target. Il modulo associa l'agente target a un gruppo in base alle informazioni sul movimento e genera regolazioni del percorso affinché il robot segua il gruppo. Il modello risultante tiene conto delle regole sociali, del comportamento umano e dei comportamenti collettivi in folle dense. Secondo i risultati riportati, il sistema è in grado di navigare in sicurezza in un'area densamente popolata (oltre 10 persone in un'area di $10m \times 20m$) seguendo il flusso della folla per raggiungere l'obiettivo. Nel paper non vengono menzionate esplicitamente le norme sociali ma il sistema impara a tracciare e ad unirsi ai gruppi di persone, garantendo una navigazione conforme e naturale, causando meno collisioni e disturbi ai pedoni circostanti.

2.2.3.1 LSTM-RL (Long Short-Term Memory)

I modelli *Long Short-Term Memory* (LSTM) [24] sono un tipo di architettura ricorrente che consentono di catturare dipendenze a lungo termine all'interno di una sequenza di dati. A differenza delle reti neurali tradizionali, che possono avere difficoltà nel memorizzare informazioni a lungo termine, sono progettate per mantenere la memoria delle informazioni rilevanti nel corso del tempo: ciò è particolarmente utile in situazioni in cui la decisione corrente dipende da eventi passati lontani nel tempo. Successivamente, la combinazione del RL con LSTM ha portato a numerosi progressi [3]. LSTM-RL può risolvere compiti non markoviani con dipendenze a lun-

go termine tra eventi rilevanti, cosa difficile per gli algoritmi RL tradizionali, oltre ad essere capace di apprendere da input sensoriali grezzi senza necessità di elaborazione precedente, apre nuove possibilità per l'utilizzo del RL nelle applicazioni del mondo reale in cui sono presenti hidden layers e dipendenze a lungo termine. Sebbene questa architettura abbia mostrato risultati promettenti nella risoluzione di alcune casistiche, non è una soluzione valida per tutti i casi e potrebbe avere dei limiti se applicata ad ambienti o compiti eccessivamente complessi. Infatti, oltre a richiedere un grande sforzo computazionale, rispetto ad altri modelli di RL, la scelta dei valori dei parametri, come la dimensione della memoria LSTM o la lunghezza delle sequenze, può influire sulle prestazioni del modello e richiedere esperimenti iterativi per trovare le migliori configurazioni. In aggiunta, le reti LSTM sono modelli complessi e possono mancare di interpretabilità rispetto ad altri algoritmi di RL più semplici in quanto la complessità del modello stesso può influire sulla comprensione delle decisioni prese dal modello.

2.2.3.2 CADRL

Proposto da [11], CADRL (Collision Avoidance Deep Reinforcement Learning) è un algoritmo di prevenzione delle collisioni multiagente decentralizzato basato sul RL e DL. Utilizza una Deep Neural Network per approssimare la Q-function, utilizzata per effettuare una stima sulle azioni ottimali per evitare le collisioni. In linea generale, la Q-function, assegna ad ogni coppia (stato, azione) un valore che rappresenta il ritorno totale atteso per l'esecuzione di un'azione nello stato corrispondente. Formalmente, può essere scritta come $Q(s, a)$, dove s rappresenta lo stato ed a rappresenta l'azione. La rete neurale riceve in input lo stato dell'agente, inclusa la propria posizione e velocità in aggiunta a quella relativa agli altri agenti, restituendo i *Q-values* per le diverse scelte di azione che sono i valori specifici assegnati dalla Q-function alle diverse coppie (stato, azione). Dunque, CADRL mira a minimizzare il rischio di collisioni considerando fattori come raggiungere la destinazione desiderata o mantenere una certa velocità per consentire agli agenti di evitare le collisioni pur considerando sempre altre esigenze o priorità di navigazione. I risultati hanno dimostrato che può apprendere in modo efficace policy per evitare le collisioni su-

rando gli approcci tradizionali basati su regole in termini di sicurezza ed efficienza. Durante il processo di addestramento, gli agenti interagiscono con un ambiente di simulazione in cui vengono generate situazioni di potenziali collisioni ed imparano a prendere decisioni di azioni continue, come regolare la velocità e la direzione. La rete neurale viene addestrata utilizzando un algoritmo di deep Q-learning modificato per gestire azioni continue. Rispetto lavori precedenti, il metodo adatta una procedura di apprendimento offline rendendolo efficiente dal punto di vista computazionale, ad esempio per il controllo decentralizzato su dieci agenti, si osserva che ciascuna iterazione di CADRL impiega 62 ms, utilizzando una CPU i7-5820K CPU. I risultati della simulazione di [11] mostrano che CADRL si comporta in modo leggermente migliore di ORCA [53] (usato come baseline o termine di paragone) nei casi di test più semplici e di oltre il 26% meglio nei casi di test difficili (> 75 percentile). La differenza di prestazioni è più evidente nei casi di test con più agenti.

2.2.3.3 SA-CADRL

In [10] gli autori estendono CADRL incorporando le norme sociali nel processo di apprendimento, introducendo SA-CADRL (Socially Aware Collision Avoidance with Deep Reinforcement Learning). L'integrazione di queste in una policy di navigazione permette al robot di comprendere e rispettare le regole e le aspettative sociali, migliorando la sua interazione con gli esseri umani e riducendo il rischio di comportamenti indesiderati o pericolosi. Nell'articolo [10] ne sono state individuate alcune, mostrate in Figura 2.17.

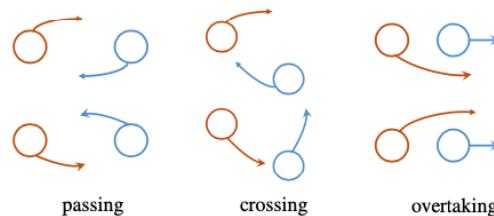


Figura 2.17: Rappresentazione grafica di norme sociali relative alle azioni di *passing* (superamento), *crossing* (attraversamento) e *overtaking* (sorpasso) tra agenti o individui in un contesto sociale. Il cerchio rappresenta l'agente e la freccia la direzione che vuole percorrere [10].

Il modello ottenuto apprende policy che rispettano diverse norme sociali come le regole di navigazione per mancini o destrimani, e può generalizzare bene tra diversi individui e situazioni, penalizzando le azioni che violano le norme sociali come passare dalla parte sbagliata o tagliare fuori altri agenti. Gli autori hanno strutturato la reward in modo da specificare “cosa non fare” (violazioni delle norme sociali) piuttosto che specificare direttamente i dettagli di “cosa fare”. L’utilizzo del DRL permette di ottenere risultati migliori dato che il modello non si concentra sulle feature estratte della osservazioni, ma apprende policy che rispettano le norme e consente al sistema di generalizzare bene in contesti svariati. Nello specifico, gli autori hanno sanzionato le azioni che violavano le seguenti norme sociali:

- Passaggio dalla parte sbagliata: quando due agenti si avvicinano frontalmente devono passare su lati opposti per evitare la collisione, penalizzando le azioni che avrebbero comportato il passaggio dalla stessa parte dell’altro agente;
- Interruzione: quando un agente si muove più velocemente di un altro e lo sorpassa, deve lasciare il posto all’agente più lento per evitare la collisione. Per incoraggiare questo comportamento, SA-CADRL penalizza le azioni che avrebbero comportato l’interruzione dell’altro agente.

SA-CADRL è stato addestrato utilizzando una variante del Q-learning con ripetizione dell’esperienza (come in [37]). È stata utilizzata una rete neurale per rappresentare la value function e la policy, opportunamente addestrata utilizzando dati simulati. La penalizzazione per la violazione delle norme sociali è stata aggiunta alla funzione di rewards durante l’addestramento ed è stata gradualmente aumentata nel tempo per incoraggiare gli agenti ad apprendere comportamenti socialmente consapevoli. Il modello ottenuto è stato valutato con esperimenti nel mondo reale con veicoli robotici che navigano in ambienti affollati. Le traiettorie generate della policy, in Figura 2.18, sono efficienti in termini di tempo e concordano con le caratteristiche qualitative mostrate nella Figura 2.17.

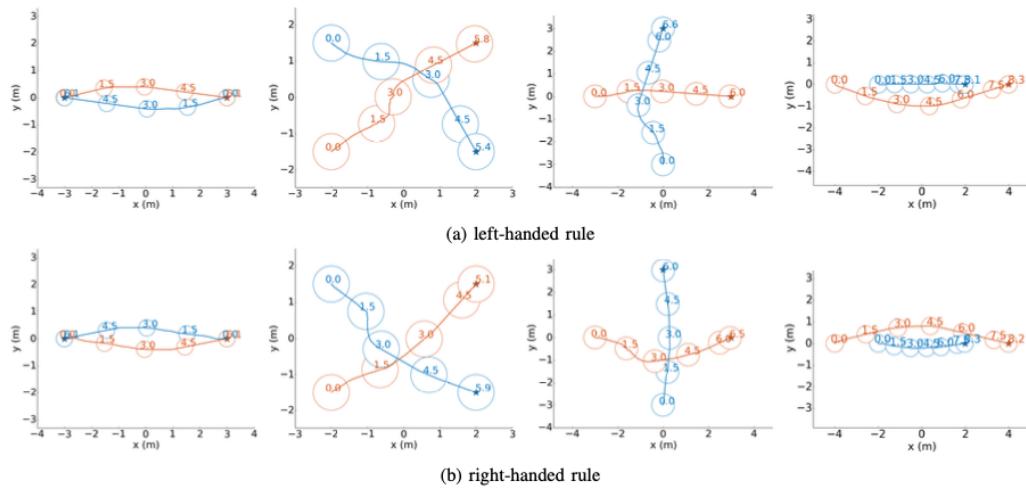


Figura 2.18: Policy SA-CADRL che esibiscono comportamenti socialmente consapevoli. I cerchi mostrano la posizione di ciascun agente all'ora indicata e le stelle indicano gli obiettivi. (a) e (b) mostrano le traiettorie generate dalle policy addestrate ad apprendere rispettivamente regole per mancini e destrimani [10].

In particolare, in un test con due agenti, tutti i metodi basati su RL (incluso SA-CADRL) hanno prodotto percorsi più efficienti, in termini di tempo, rispetto ORCA. Inoltre, il modello ottenuto con SA-CADRL ha mostrato una leggera preferenza a destra, coerente con le norme di navigazione per destrorsi. Infine, nelle simulazioni nel mondo reale con veicoli robotici, SA-CADRL è stato in grado di evitare con successo le collisioni rispettando le norme sociali implementate.

3

CrowdSim-Real: un simulatore avanzato per Social Navigation in scenari realistici

Per utilizzare le tecniche di apprendimento automatico precedentemente citate nel Capitolo 2 ed ottenere un modello di navigazione è necessario un simulatore che consenta di addestrare un modello e visualizzarne i risultati. È stato preso in rassegna lo stato dell’arte dei simulatori open source disponibili e dopo un’attenta valutazione, è stato scelto **CrowdNav**, sviluppato presso l’EPFL (Ecole polytechnique fédérale de Lausanne) e impiegato nel campo della navigazione autonoma in ambienti affollati [9]. La capacità di simulare la presenza di pedoni in movimento e la possibilità di apportare modifiche al codice esistente stesso (data la natura open source), sono state le caratteristiche fondamentali per la scelta del simulatore come punto di partenza di questo studio. Il simulatore è stato implementato utilizzando Python 3.6.15, più nel dettaglio, sono state utilizzate la libreria PyTorch per lo sviluppo e l’implementazione delle reti neurali ed OpenAI Gym, utilizzata nel campo dell’apprendimento automatico e della robotica per l’addestramento e la valutazione di algoritmi di ML, in particolare quelli legati al RL. Gym fornisce un’ampia gamma di ambienti di simulazione, detti *environment*, che rappresentano problemi specifici o scenari in cui gli algoritmi di apprendimento possono essere addestrati e valutati. È stato necessario anche utilizzare la libreria RV02 “Reciprocal Velocity Obstacles” (Vincoli di Velocità Reciprocii), utilizzata principalmente per la simulazione del movimento dei pedoni.

3.1 Il simulatore originale e i modelli di DRL disponibili

Gli autori [9] considerano un task di navigazione di un robot all'interno di un ambiente, caratterizzato dalla presenza di un gruppo di n persone formulato come un *sequential decision making problem in a reinforcement learning framework* (Capitolo 2.2.2.1). L'environment è totalmente osservabile e ogni agente (robot o umano) possiede delle proprietà; come la posizione $p = [px, py]$, la velocità $v = [vx, vy]$ e il raggio r (ogni agente i è rappresentato da una circonferenza). Il robot è consapevole della sua posizione rispetto all'ambiente, inclusa la posizione dell'obiettivo (o goal) pg e la velocità preferita f e si assume che la velocità del robot vt possa essere raggiunta immediatamente dopo aver effettuato un'azione. Il modello estrae features per le interazioni a coppie tra il robot e ciascun umano e cattura le interazioni tra gli umani tramite mappe locali che vengono poi aggregate con un meccanismo di *self attention* [42] che deduce l'importanza di ciascuna persona per i successivi step di navigazione. Il sistema permette di tenere conto di un numero arbitrario di agenti, fornendo una buona comprensione del comportamento della folla per la pianificazione del percorso. Questo meccanismo consente a una rete neurale di assegnare pesi diversi a diverse parti di un'input (ad esempio, una frase, sequenza di parole, una lista di persone all'interno di un ambiente) in base alle relazioni tra gli elementi della sequenza stessa. In altre parole, il modello può dare più importanza a certi elementi in input rispetto ad altre in base al contesto e alla struttura della sequenza. Come accennato nel Capitolo 2, il problema della navigazione robotica può essere visto come un problema di ottimizzazione dove l'obiettivo dell'agente (robot) è massimizzare le *rewards* e lo fa evitando la collisione e raggiungendo il *goal*. La policy degli umani, ORCA [53] invece non è coinvolta in alcuna forma di learning e necessita la posizione degli umani in tempo reale per funzionare. La traiettoria degli umani è generata secondo due regole per la gestione del comportamento: `circle_crossing` o `square_crossing`. Nella prima gli umani vengono posizionati casualmente su un cerchio di raggio 4 metri (personalizzabile da configurazione) e la loro posizione *target* è in direzione diametralmente opposta sullo stesso cerchio. Una

perturbazione casuale può essere aggiunta alle coordinate (x, y) sia della posizione iniziale che di quella finale degli esseri umani. Nel secondo tipo di configurazione, gli umani possono essere posizionati in un quadrato/rettango dove il goal è dall'altra parte del quadrilatero `square_crossing`. La posizione di partenza e destinazione del robot è fissa e non può essere modificata. Le funzionalità presenti sono un punto di partenza buono ma non sufficiente per il raggiungimento nostro obiettivo, soprattutto perché non è presente la possibilità di inserire ostacoli o planimetrie che porterebbero il modello a funzionare in contesti reali e nei casi d'uso specifici. Si aggiunge anche che di default la posizione di partenza e di arrivo è sempre la stessa, cosa non applicabile al caso reale in cui il robot si troverà sempre in una posizione diversa. A differenza del setup originale, nel nostro caso l'agente robotico non avrà la conoscenza delle coordinate di ogni agente od ostacolo ma dovrà prendere le decisioni sulla solo base della percezione proveniente da un sensore LiDAR, rendendo necessario apportare delle modifiche anche all'input del modello. Inizialmente sono stati eseguiti dei training con la configurazione originale del simulatore in modalità `unicycle` (un modello di movimento in cui un robot si muove utilizzando una sola ruota o un singolo punto di contatto con la superficie) utilizzando i tre modelli già implementati: LSTM-RL, CADRL, SARL.

In Figura 3.1 dei grafici che descrivono le prestazioni dei modelli dopo 10000 episodi di esecuzione.

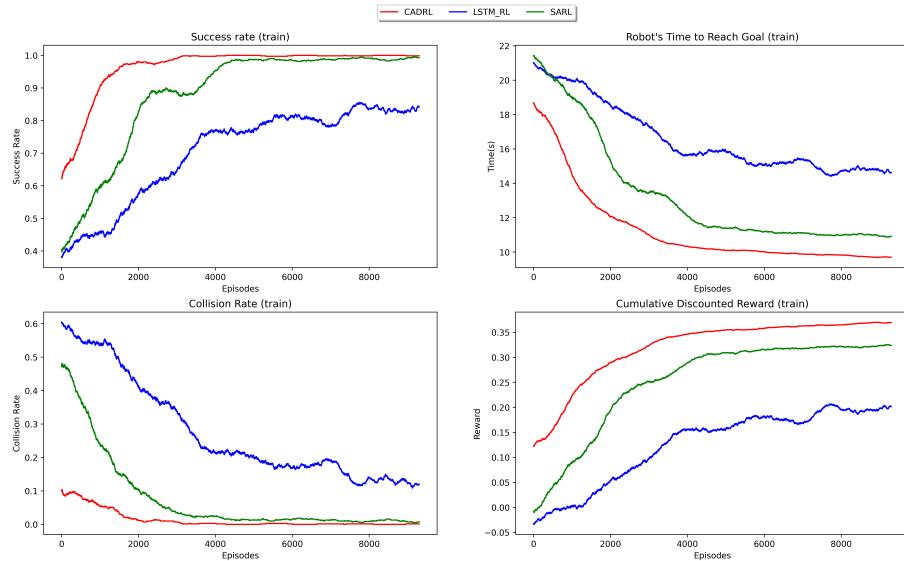


Figura 3.1: Collision Rate, Collision Discounted Reward, Success Rate e Robot's time to reach Goal dei tre modelli, dopo 10000 episodi.

Nel dettaglio:

- **Collision Rate** (Figura 3.1 in alto a sx): la percentuale di collisioni che si verificano durante l'esecuzione della navigazione, calcolato come il numero di collisioni totali diviso il numero di episodi di test. Una collisione tra due agenti si verifica quando la distanza tra il robot e un altro agente è inferiore a zero;
- **Cumulative Discounted Reward** (Figura 3.1 in alto a dx): la ricompensa totale media ottenuta dal modello durante l'esecuzione, calcolata come la somma delle ricompense istantanee ottenute durante l'esecuzione di un episodio;
- **Success rate** (Figura 3.1 in basso a sx): la percentuale di episodi in cui il robot raggiunge il goal senza collisioni, calcolato come il numero di episodi in cui il robot raggiunge il goal senza collisioni diviso il numero di episodi di test.
- **Robot's time to reach Goal** (Figura 3.1 in basso a dx): il tempo medio necessario al robot per raggiungere l'obiettivo, è calcolato come la somma del

tempo necessario al robot per raggiungere il goal in ogni episodio, diviso il numero di episodi di test.

Questi valori sono fondamentali per valutare le prestazioni del modello e per capire se le modifiche successive apportate portano ad un miglioramento o meno. La metrica più importante è la “Cumulative Discounted Reward”, necessaria per valutare le prestazioni di un agente o di un sistema di intelligenza artificiale che apprende attraverso l’interazione con un ambiente. In un contesto di apprendimento per rinforzo, un agente compie azioni in un ambiente e riceve ricompense in risposta a tali azioni rappresenta la somma delle ricompense ricevute dall’agente nel corso del tempo, dove ogni ricompensa è *discounted*, in modo che le ricompense future abbiano un peso inferiore rispetto a quelle immediate. Il discount factor è impostato a 0.9 di default. In aggiunta, sono stati realizzati dei grafici per mostrare il tempo di training richiesto dai diversi modelli di navigazione ed il tempo richiesto medio per il completamento di un episodio, vedi Figura 3.2.

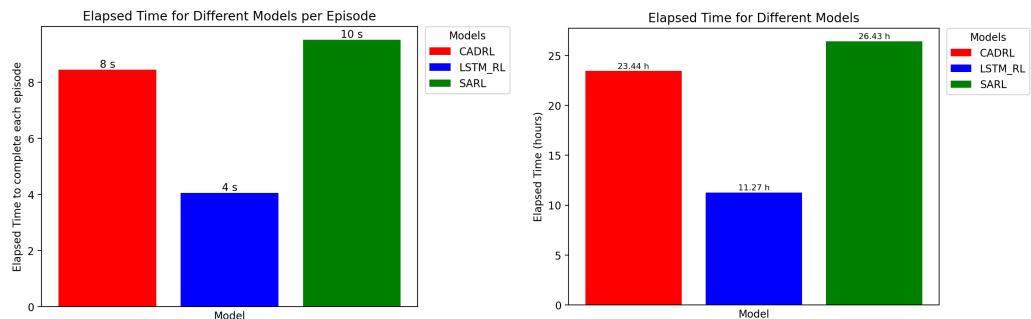


Figura 3.2: Tempo di esecuzione necessario per completare il training di 10000 episodi con i tre modelli già implementati nel progetto originale.

È importante precisare che quest’ultima misura non fornisce una descrizione veritiera del modello, in quanto un episodio può terminare immediatamente se va a collisione subito, il tempo di esecuzione richiesto sarà inferiore di un episodio andato in timeout o arriva prima. Dai dati ottenuti, il più efficiente in termini di tempo è stato CADRL. Questo sarà utilizzato come baseline per effettuare le modifiche e le valutazioni successive sulle prestazioni del modello. Si nota che il simulatore presenta già una funzionalità per effettuare il rendering di un episodio.

3. CrowdSim-Real: un simulatore avanzato per Social Navigation in scenari realistici 51

Questa funzionalità è stata fondamentale per verificare il corretto funzionamento del simulatore e successivamente per assicurarsi che le modifiche apportate non abbiano causato malfunzionamenti. Un esempio in Figura 3.3.

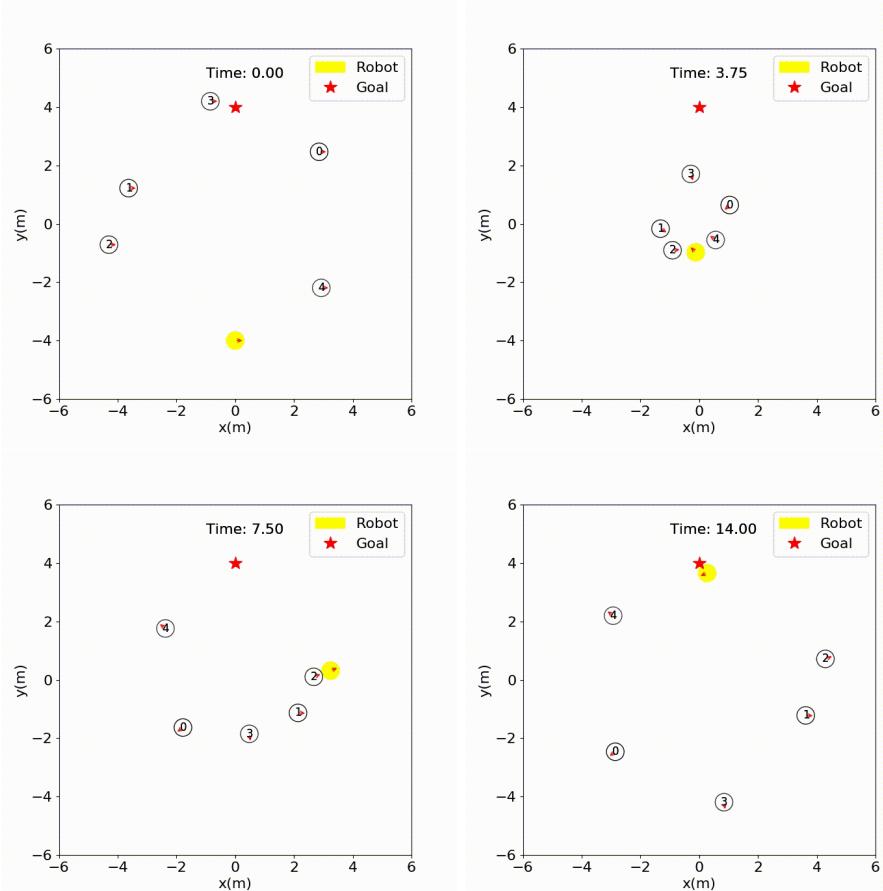


Figura 3.3: Snapshot di un episodio di navigazione usando CADRL. L'agente è rappresentato dal cerchio in giallo, la sua direzione dalla freccia rossa, l'obiettivo dalla stella in rosso, gli umani sono rappresentati dai cerchi numerati.

3.1.1 L'ambiente di simulazione

L'ambiente simulato è rappresentato su un piano cartesiano con origine fissa in (-6, -6), le dimensioni sono già indicate in metri (larghezza e lunghezza totale di 12 metri). La posizione iniziale e il goal del robot sono predefiniti e fissi, definiti da una coppia di punti $(px, py), (gx, gy)$. Un agente è identificato dalla posizione corrente ed il proprio raggio occupante (in origine 0.30(m), cambiato in 0.15(m)). Prima di provvedere alla gestione degli ostacoli è stato necessario cambiare il punto di origine dell'ambiente in (0, 0) e rendere dinamica sia la dimensione dell'ambiente che le posizioni di partenza e destinazione dell'agente, in modo da rendere più agevole la gestione delle planimetrie e permettere la creazioni di ambienti dinamici non fornendo nessun tipo di limitazioni al modello. In uno spazio 2D un ostacolo può essere rappresentato da un poligono con quattro punti (lb, lt, rt, rb) (rispettivamente: left bottom, left top, right top, right bottom) e la collisione tra questo e l'agente è calcolata dalla distanza tra il centro dell'agente (meno il raggio) e il lato ad esso più vicino. È stata implementata una classe per la gestione di quest'ultimo, sfruttando la libreria `shapely` per la gestione delle collisioni ed il calcolo delle distanze nel poligono, dato che presenta delle funzioni ottimizzate che permettono di calcolare la distanza tra un punto ed un poligono. Un agente è quindi considerato in collisione con un ostacolo se presenta la distanza tra questi è uguale a zero come mostrato in Figura 3.4.

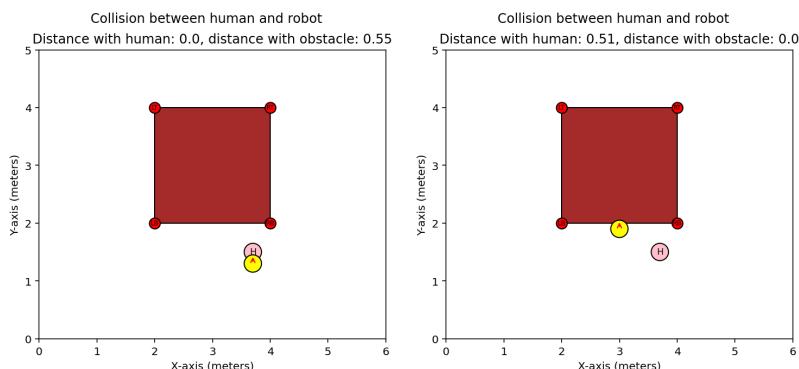


Figura 3.4: Rendering di un ambiente in cui il robot è rappresentato dal cerchio giallo, gli umani dal cerchio rosa e gli ostacoli da un poligono marrone. A sinistra un esempio di collisione con un umano, a destra un esempio di collisione con un ostacolo.

Dai vari test effettuati, si nota che ORCA non è ottimale nell'esecuzione di un percorso in presenza di ostacoli fissi in quanto tende semplicemente a diminuire la velocità in prossimità di quest'ultimo, fino a fermarsi senza raggiungere la destinazione. Dopo aver implementato queste feature sono state individuate due tipologie di configurazioni dell'ambiente da usare per l'addestramento e la valutazione del modello:

1. configurazione con planimetrie di ambienti reali;
2. configurazione con ambienti di dimensione variabile e ostacoli randomici (che chiameremo per comodità *dinamici*).

Si assume che per entrambe le configurazioni che la posizione di partenza e di arrivo sia del robot che degli umani cambi ad ogni episodio e che questi non collidano con gli ostacoli e che siano distanti da questi di almeno 0.2 (m).

Planimetrie da ambienti reali

Allenare l'agente in un ambiente complesso offre diversi vantaggi. Infatti, simulare un ambiente con caratteristiche più vicine al reale permette al modello di operare in ambienti mai visti prima. Di contro, il modello potrebbe richiedere tempi di esecuzione più elevati e reward più basse, data la maggiore complessità delle traiettorie da percorrere. Le planimetrie sono state estratte dal dataset chiamato Gibson [56], che consiste in un'insieme di ambienti 3D reali acquisiti tramite un apposito scanner. Le planimetrie 2D utilizzate sono state estratte a partire dai modelli degli ambienti 3D, importati all'interno del simulatore Habitat [50]. È stato applicato lo split "medium" suggerito nell'articolo del dataset e sono state eliminate le planimetrie non utilizzabili corrotte o contenente spazi indipendenti non connessi tra loro. Degli operatori morfologici sono stati applicati per ottenere degli ambienti navigabili, nonostante la perdita di alcuni dettagli. Un esempio in Figura 3.5.

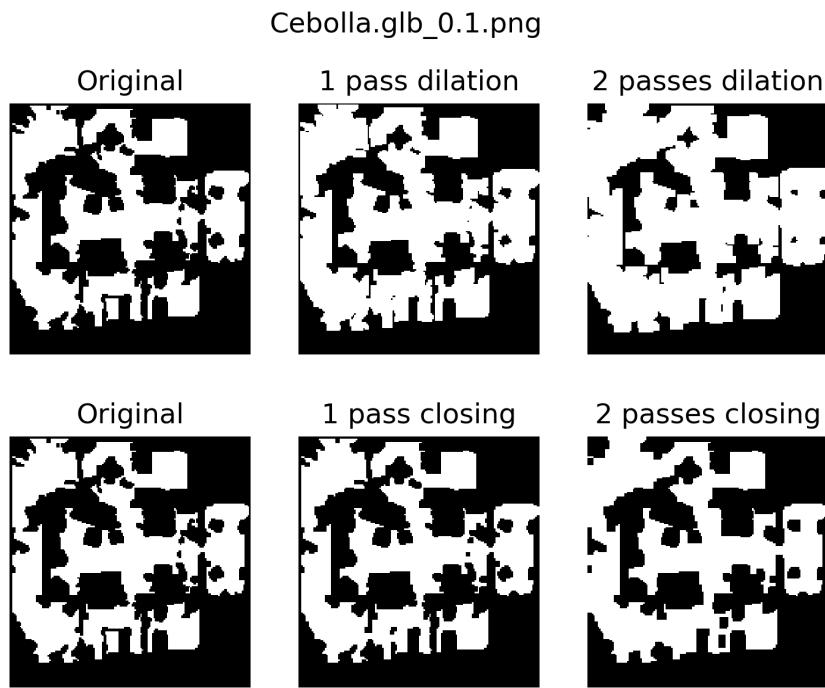


Figura 3.5: Esempi di applicazioni degli operatori morfologici, chiusura e dilatazione su immagini rappresentati planimetrie, in bianco le aree percorribili, in nero gli spazi occupati.

Gli operatori morfologici rivestono un ruolo fondamentale nel pre-processing delle immagini e contribuiscono in modo significativo alla qualità dei dati di input. Le immagini iniziali (*Figura 3.5* a sinistra) rappresentano le planimetrie originali che contengono artefatti, dovuti a imperfezioni dei modelli 3D degli ambienti o a imprecisioni del processo di estrazione, e che è desiderabile rimuovere o minimizzare. Nella dilatazione (*Figura*, prima riga, *3.5 1 pass dilation, 2 passes dilation*), viene applicato un operatore di dilatazione che mira a espandere le regioni bianche nell’immagine in modo da connettere o ampliare le aree attraversabili presenti nell’immagine originale. Nei passi di chiusura (*Figura*, seconda riga, *3.5 1 pass closing, 2 passes closing*), viene applicato l’operatore di chiusura, che combina la dilatazione con l’erosione (restringimento delle regioni scure). Questa operazione ha lo scopo di unire le regioni bianche pur preservando le zone nere, che risultano ben definite. Tale processo è fondamentale per garantire che i dati di input siano di alta qualità e pronti per le fasi successive di analisi e elaborazione. Le planimetrie sono state trasformate in un insieme di rettangoli tramite l’uso di un algoritmo di decompo-

sizione dello spazio occupato (la zona nera nelle immagini in Figura 3.5) che ha il compito di massimizzare la copertura dell'area considerata, minimizzando al contempo il numero di poligoni (rettangoli) utilizzati. Questa operazione è necessaria per semplificare la gestione delle collisioni e della percezione con il sensore LiDAR. Prima di passare alla fase di training è stato necessario fare una cernita degli ambienti utilizzati in quanto alcuni risultavano navigabili, altri presentano spazi non connessi tra loro o corridoi troppo stretti per permettere la navigazione. Il dataset conta un totale di 81 ambienti di cui 71 usati per il training e 10 per la fase di test. Alcuni esempi mostrati in Figura 3.6.

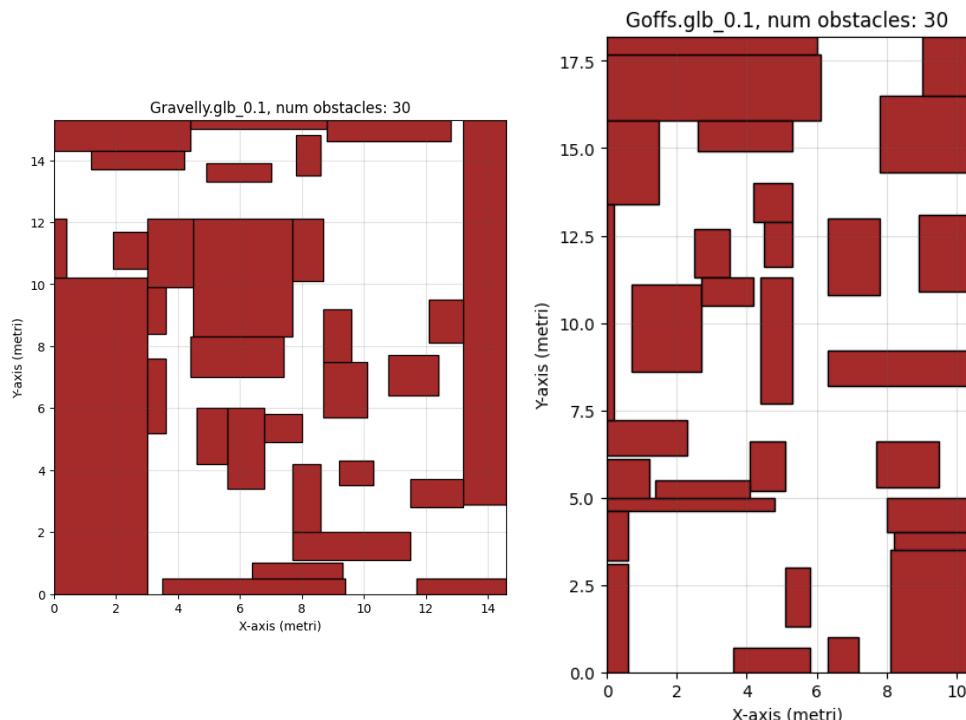


Figura 3.6: Due planimetrie estratte da due ambienti del dataset Gibson, ottenute con l'esecuzione dello script `plot_planimetries_env_from_csv.py`

Le planimetrie sono quindi saranno caricate a partire da un path definito a livello di configurazione. Per diminuire il tempo di attesa durante la generazione di punti di start e goal, della navigazione, è stata implementata una logica che carica da file .csv tutte le posizioni, precedentemente elaborato, previa l'esecuzione dell'apposito script chiamato `plot_planimetries_env_from_csv.py`.

Ambienti dinamici

Nel caso di training in ambienti dinamici la dimensione dei limiti dell'ambiente viene generata dinamicamente entro dei limiti impostabili da configurazione (nei nostri esperimenti è stato utilizzando il range (8, 15)). Allo stesso modo, il punto di partenza e l'obiettivo dell'agente sono generati randomicamente. Nello specifico vengono generate una coppia di coordinate per la partenza una coppia per il goal. Vengono prima generati i punti e successivamente vengono generati n ostacoli randomici che mantengono una distanza minima dai punti di partenza. Così facendo si rende possibile un percorso percorribile e si è sicuri che la posizione di partenza e di arrivo non venga ostruita dagli ostacoli, come si può osservare in Figura 3.7. Si aggiunge che sono stati implementati diversi controlli per assicurarsi che gli ostacoli non si sovrappongano tra loro.

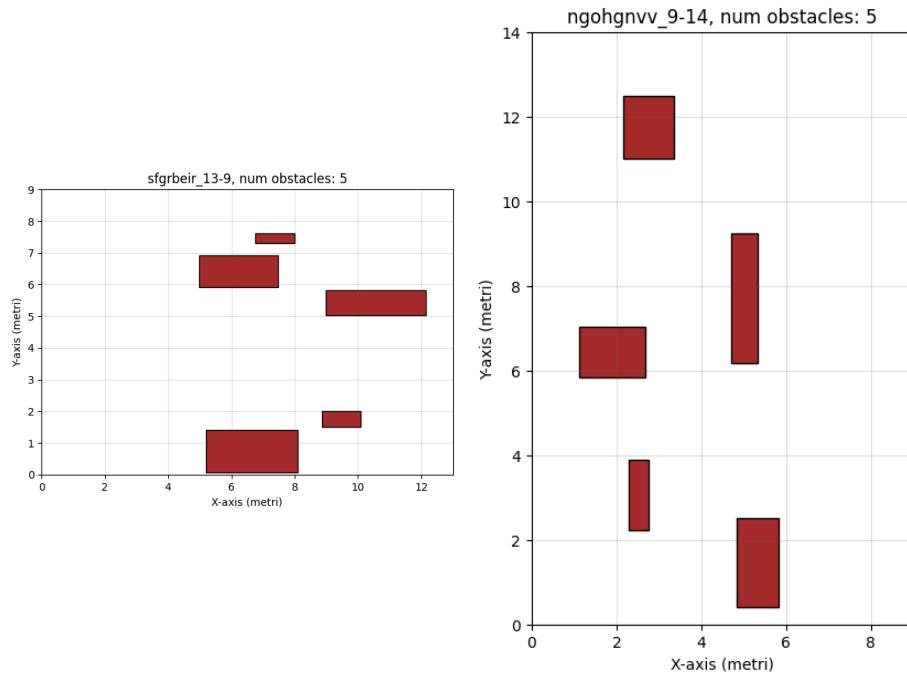


Figura 3.7: Rendering di due environment generati usando la funzione `generate_and_create_sg_dynamic_env.py`.

Inoltre, la distanza tra la partenza e l'arrivo è pari ad almeno un terzo del lato più lungo dell'ambiente. Questi accorgimenti sono necessari per evitare di allenare l'agente seguendo episodi non validi. In fase di training, è difficile e dispendioso

in termini di tempo verificare che ogni singolo episodio sia valido e di conseguenza bisogna applicare tutti i suddetti accorgimenti per minimizzare questi casi. Come detto precedentemente, per velocizzare il training della planimetria è stata prevista, la generazione dei punti di partenza e di destinazione in un file .csv ed il relativo caricamento automatico, previa l'esecuzione dell' script

`create_sg_for_planimetries_env.py`. La stessa funzionalità è stata utilizzata per gli ambienti dinamici, ma attualmente solo nella di test. Per mezzo dell'esecuzioni di questi script è possibile visualizzare le posizioni di partenza e di arrivo generate dal .csv ed avere uno strumento più affidabile per verificare che siano valide, permettendo di individuare eventuali vicoli ciechi, corridoi troppo stretti o stanze chiuse nella planimetria che renderebbero impossibile la navigazione. In Figura 3.8 è possibile osservare dei punti di partenza e arrivo, campionati e salvati nel file .csv.

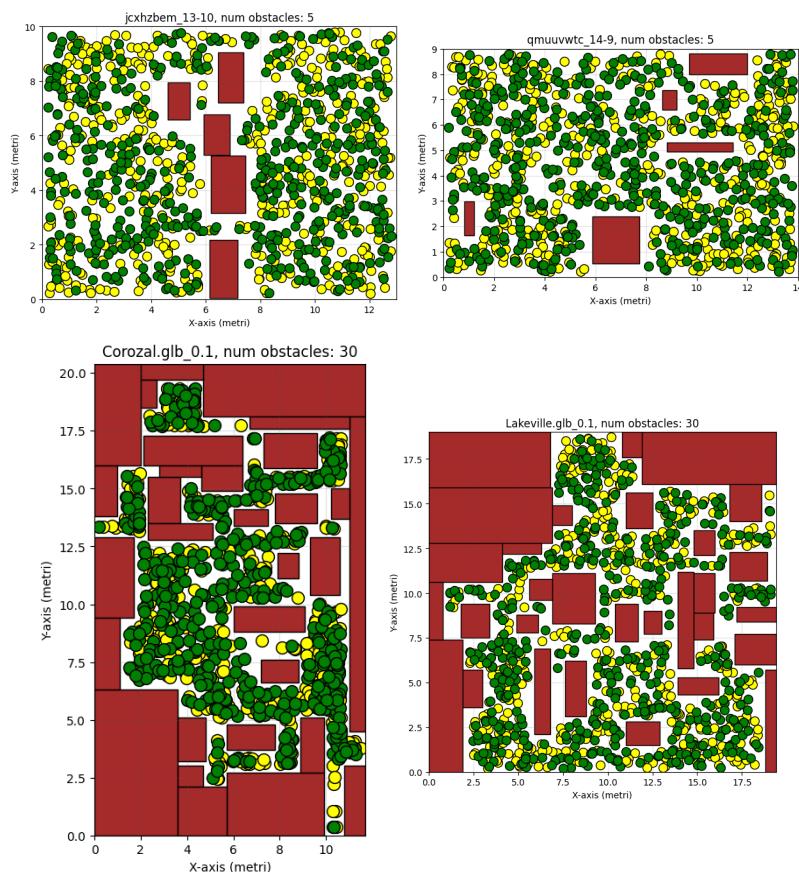


Figura 3.8: Punti di partenza e arrivo, generati e salvati nel file .csv. La terza planimetria è inutilizzabile in quanto vi sono dei punti isolati non navigabili.

L'utilizzo degli ostacoli negli ambienti di training permette di creare scenari più complessi e variegati all'interno dei quali il robot deve affrontare situazioni più realistiche, rendendo necessaria una pianificazione del percorso in ambienti più strutturati, migliorando la capacità dei modelli di apprendimento a navigare attraverso corridoi, vicoli ciechi, ostacoli isolati e altre caratteristiche complesse che non sarebbero facilmente riprodotte usando ostacoli semplici. In particolare, l'utilizzo di planimetrie consente di creare scenari di addestramento e test più complessi e realistici.

3.1.2 Modifica delle policy degli umani

In questa fase è stato necessario modificare la policy di movimento degli umani. `square_crossing` e `circle_crossing` sono state pensate in un ambiente libero e senza ostacoli fissi, in cui l'agente è in grado di muoversi liberamente senza tenere in considerazione gli ostacoli fissi. Una novità introdotta in questo studio consiste nell'implementazione di una nuova policy denominata `random_goal`. Questa permette di guidare l'agente nel raggiungimento di una posizione di destinazione predefinita, partendo dalla generazione casuale della posizione iniziale, seguendo gli stessi criteri utilizzati per la creazione dei punti di partenza del robot il cui uso è stato motivato dalla necessità di migliorare il funzionamento degli agenti umani all'interno di spazi chiusi. Le policy precedenti erano pensate per contesti aperti, caratterizzati dalla presenza di ostacoli in movimento. Tuttavia, queste configurazioni non erano completamente adatte per simulare scenari più complessi. La strategia adottata è stata quella di assegnare un punto di destinazione e fermarsi in prossimità di esso una volta raggiunto: questa semplice, ma efficace, simulazione del movimento si adatta meglio agli ambienti chiusi e contribuisce a generare dinamiche più realistiche. Gli snapshot degli episodi hanno mostrato un aspetto rilevante: nel caso in cui più agenti abbiano condiviso una parte del percorso di navigazione, è stato possibile ottenere la simulazione di una folla, contribuendo alla creazione di scenari più complessi e adatti alla valutazione delle capacità dell'agente in contesti realistici durante la fase di training e test. È stato previsto un valore soglia per assicurarsi una distanza

minima tra i goal degli umani, nel nostro caso settata ad 1 (m). Se non viene trovata una posizione valida, dopo 10 tentativi il valore soglia verrà decrementato, per evitare che in fase di training si impieghi troppo tempo per la generazione. Nella fase di rendering è stata resa disponibile un'opzione per mostrare l'obiettivo degli umani, come mostrato Figura 3.9.

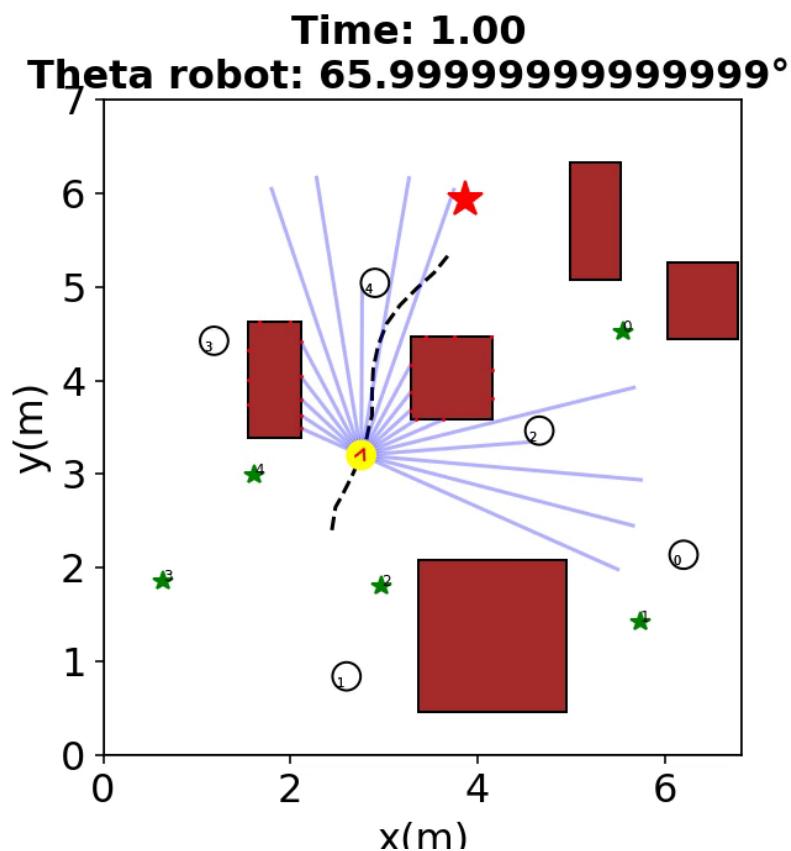


Figura 3.9: Frame di un episodio di navigazione dove gli obiettivi degli umani sono evidenziati con le stelle in verde, l'obiettivo del robot in rosso.

3.1.3 Simulazione del LiDAR

Il sensore LiDAR (Light Detection and Ranging) è una tecnologia attiva di *remote sensing* che consente di determinare la distanza di un oggetto o di una superficie utilizzando un impulso laser, (vedi Figura 3.10) dove la distanza dell'oggetto è determinata misurando il tempo trascorso fra l'emissione dell'impulso e la ricezione del segnale retrodiffuso. La sorgente del sistema è un laser, ovvero un fascio coerente di luce a una precisa lunghezza d'onda, inviato verso il sistema da osservare. È definita *tecnologia attiva* in quanto, a differenza di altre tecnologie di telerilevamento passivo che sfruttano l'energia emessa dal sole (es. sensori ottici) emette una certa energia, sotto forma di un raggio laser, per rilevare la forma di oggetti. Nella fattispecie il LiDAR, a differenza di tecnologie simili quali il radar o il sonar, usa poca energia emettendo un laser con lunghezze d'onda ultraviolette, nel visibile o nel vicino infrarosso.

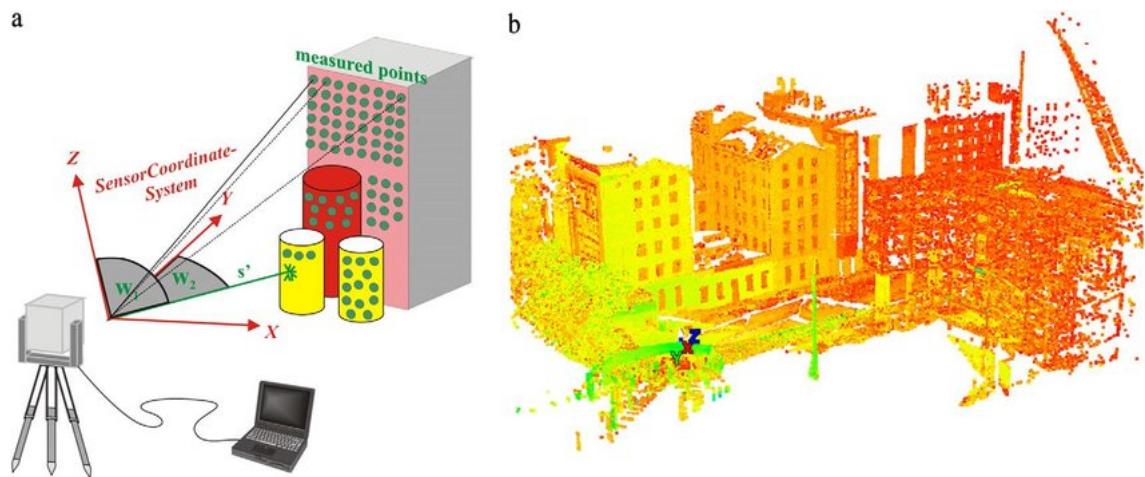


Figura 3.10: (a) il processo di scansione laser per la misurazione di punti 3D [46]. (b) Un esempio di dati scansionati al laser di un edificio in costruzione [51]e

Date queste proprietà, in un piano 2D, un raggio LiDAR può essere simulato da fasci di rette che partono da un punto (px, py) con campo visivo θ e lunghezza l . In questo modo, è possibile ottenere un fascio LiDAR con n raggi, o meglio, un vettore di lunghezza n in cui ogni valore indica la distanza di un ostacolo rispetto a (px, py) :

$$v = [x_1, x_2, x_3, \dots, x_n] \quad (3.1)$$

Se alla posizione i del vettore vi è la distanza massima significa che il raggio non si sarà intersecato con nessun oggetto e di conseguenza lo spazio sarà libero. È possibile impostare da configurazione il numero di raggi, il campo visivo e la distanza massima percepita. Per ogni raggio del fascio, viene calcolata l'eventuale intersezione con uno degli ostacoli (poligono per ostacolo fisso, cerchio per altro agente). Se ciò avviene significa che il raggio è stato interrotto e la distanza viene decrementata fino al punto in cui vi è l'effettiva intersezione dal raggio e l'ostacolo. Un esempio del risultato in **Figura 3.11**.

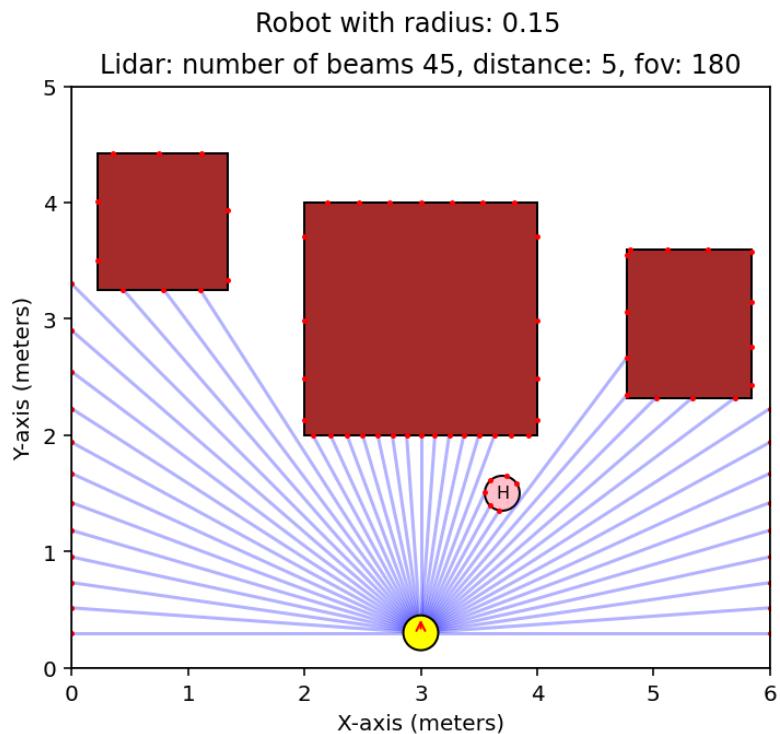


Figura 3.11: L'immagine mostra la simulazione di un sensore LiDAR di 45 raggi e campo visivo di 180 gradi, rivolto verso la direzione indicata dalla freccia rossa.

Infine per simolare i limiti dell'environment sono stati aggiunti 4 ostacoli all'esterno, le cui dimensioni sono uguali alle dimensioni dell'ambiente. Se questo controllo non fosse stato aggiunto l'agente puntando verso i limiti dell'ambiente avrebbe percepito uno spazio libero e quindi avrebbe provato provare ad aggirare l'ostacolo uscendo dai limiti dell'ambiente. La computazione del LiDAR richiede un certo sforzo di elaborazione e nelle prime fasi di esecuzione il training per 1000

3. CrowdSim-Real: un simulatore avanzato per Social Navigation in scenari realistici ---

episodi ha impiegato circa 2 giorni, usando un LiDAR con 180 raggi, campo visivo di 180° e lunghezza 1(m). Questo perché nelle prime versioni di implementazione ogni raggio doveva verificare l'intersezione con ogni ostacolo presente nell'ambiente. Per diminuire il tempo di calcolo del vettore, invece di verificare l'intersezione di ogni raggio con ogni ostacolo, viene memorizzata una lista di agenti e ostacoli che si intersecano con un cerchio di dimensioni poco maggiori alla lunghezza del LiDAR (+0.3) centrato alla posizione del robot. Successivamente, sulla base di questa lista ridotta verranno calcolate le intersezioni di prossimità. Questo accorgimento permette di diminuire notevolmente il tempo di calcolo del vettore in quanto il numero di possibili intersezioni da calcolare è notevolmente ridotto. Il vettore, infine, viene normalizzato per contenere valori nello spazio compresi tra zero ed uno: $v \in [0, 1]^n$. L'introduzione di queste funzionalità rappresenta un importante passo in avanti nella creazione di un sistema di navigazione robotica più realistico dell'iniziale, in quanto il vettore di prossimità del LiDAR fornisce un'informazione fondamentale per percepire l'ambiente circostante.

3.1.4 Modifiche del modello

Come detto nel Capitolo 2.2.3.2 il modello CADRL riceve in input l’osservazione corrente e restituisce l’azione da compiere. L’osservazione è rappresentata dall’oggetto `JointState` che contiene un vettore di lunghezza 13: i primi 6 valori rappresentano il `SelfState`, le informazioni correnti dell’agente, mentre i restanti 7 rappresentano le informazioni degli umani (classe `ObservableState`).

La nostra modifica consiste nel sostituire l’input precedentemente descritto, ottenuto dall’unione delle due classi, con il vettore del LiDAR, che permette di osservare la configurazione totale dell’ambiente circostante, il `SelfState` è stato mantenuto in modo tale da avere il riferimento sulla posizione dell’environment. A differenza del modello originale [11], non è necessario effettuare una trasformazione sulle coordinate in quanto la policy ottenuta sarà invariante per ogni coordinata. Le modalità di training della ValueNetwork è rimasta invariata. Si nota che nella fase di imitation learning, che precede quella di reinforcement learning, l’agente opera in un ambiente dinamico senza ostacoli. Dato che in questa fase viene usata la policy ORCA, è stata effettuata una modifica che consiste nel far sì che durante l’imitation learning, la policy ORCA restituisca il vettore del LiDAR come stato, consentendo così al modello CADRL di apprendere da questa policy. Di conseguenza la dimensione dell’input è stata resa dinamica e variabile e dipende dalla lunghezza del vettore del LiDAR. La dimensione del MLP è stata lasciata uguale alla versione originale ((150, 100, 100, 1)). Come funzione di attivazione viene usata la ReLU.

3.1.5 Esperimenti

La configurazione originale prevede 10000 episodi di training con una evaluation di 100 episodi ogni 1000 di training seguito da 500 episodi di test. Dai primi esperimenti, ci si è accorti che il completamento del training richiedeva dei tempi eccessivamente lunghi, come può essere osservato in **Figura 3.12**, per cui, per ottimizzare i tempi di esecuzione, è stato ridotto il numero degli episodi di evaluation a 10 e la fase di test eseguita alla fine è stata ridotta ad un solo episodio. I modelli saranno poi valutati alla fine con dei test case appositi.

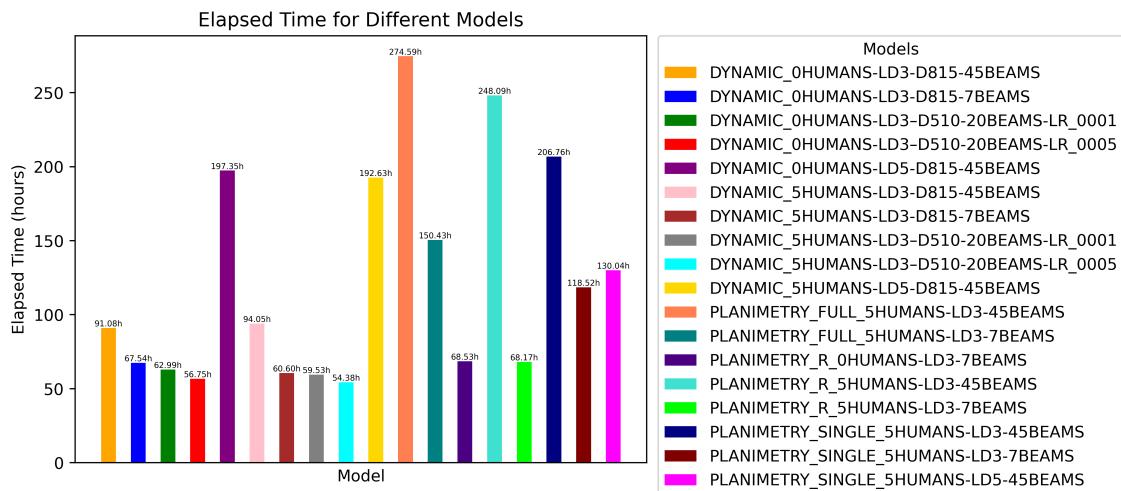


Figura 3.12: Tempo di esecuzione del training dei differenti modelli, al variare delle configurazioni usate.

La denominazione dei modelli, come specificato nella legenda del grafico, segue la seguente struttura:

- Il primo parametro indica il tipo di ambiente utilizzato: “dynamic” si riferisce all’ambiente generato randomicamente, mentre “planimetry” indica l’utilizzo di ambienti caricati da file .csv;
- Il secondo parametro, applicato solo nel caso delle planimetrie, indica la quantità di ambienti considerati. Con “full” si fa riferimento all’utilizzo dell’intero dataset di planimetrie, “R” indica l’uso di un dataset ridotto (10 ambienti), e “single” implica l’utilizzo di una singola planimetria;

- Il numero precedente a “humans” indica la quantità di umani generati durante l’esecuzione degli episodi;
- Il numero successivo a “LD” (LiDAR Distance) indica la massima distanza di ogni raggio del LiDAR;
- Il numero precedente a “beams” indica il numero di raggi utilizzati dal LiDAR.

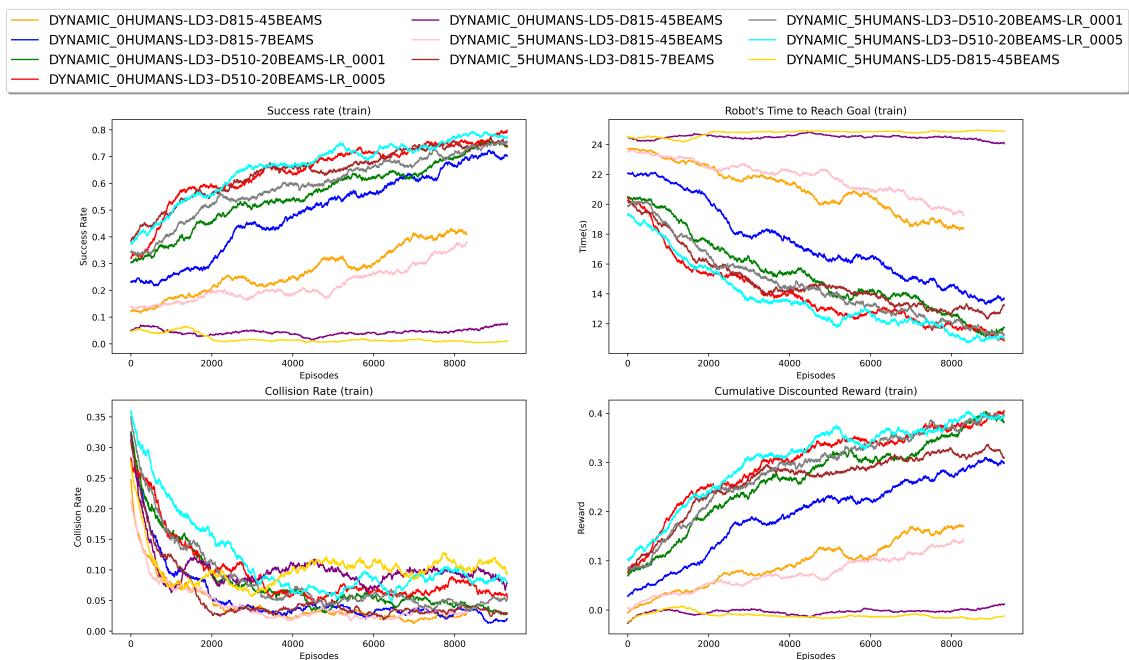
Training in ambienti dinamici

Ad ogni episodio viene generato un ambiente con dimensioni randomiche, selezionate dall’intervallo specificato nella configurazione. Il punto di partenza e di arrivo è generato casualmente con un numero di ostacoli la cui dimensione minima e massima è definita in configurazione. È stato scelto un numero di ostacoli pari a 5, i quali hanno assicurato una buona copertura dell’ambiente. Le configurazioni provate sono riassunte nella **Tabella 3.1**

beams	humans	LiDAR distance	range env.	training ep.	lr
45	5	3	(8,15)	8000	0.0001
45	0	3	(8,15)	8000	0.0001
45	5	5	(8,15)	10000	0.0001
45	0	5	(8,15)	10000	0.0001
7	5	3	(8,15)	10000	0.0001
7	0	3	(8,15)	10000	0.0001
20	5	3	(5,10)	10000	0.0001
20	5	3	(5,10)	10000	0.0005
20	0	3	(5,10)	10000	0.0001
20	0	3	(5,10)	10000	0.0005

Tabella 3.1: Configurazioni dei modelli dinamici allenati nello scenario “dinamico”

Osservando i grafici dei risultati ottenuti in Figura 3.13, notiamo che i modelli con LiDAR distance 5 e 45 raggi (curve in arancione, viola, rosa e giallo), nonostante siano quelli con le informazioni più dettagliate e precise, ottengono risultati peggiori rispetto agli altri. In particolare, in questo caso, combinando i risultati del grafico del “Success Rate” e del “Robot’s Time to Reach Goal”, si evince che il robot tende a stare fermo piuttosto che muoversi per non rischiare di accumulare delle reward negative. A conferma di ciò sono state avviate delle simulazioni con un numero di raggi pari a 7 (curve blu e marrone) che hanno confermato l’ipotesi.



Probabilmente, ci troviamo davanti ad un problema di dimensionalità di input del modello. Questo accade perché fornire input con dimensionalità elevata può portare a un numero eccessivo di dati da elaborare, rendendo il training più lento e difficoltoso. In aggiunta, si nota che la disposizione degli ostacoli non sempre è ottimale, infatti, in alcuni episodi nell’ambiente generato, si verifica che gli ostacoli non si collocano lungo la traiettoria del robot, compromettendo la validità di tali episodi di navigazione. Idealmente sarebbe opportuno che tra la partenza e l’arrivo ci sia almeno un ostacolo in mezzo, ma questa condizione non è assicurata. Sulla base

di queste problematiche sono stati avviati altri training riducendo l'intervallo per la generazione randomica dei limiti dell'ambiente a (5,10), avvicinando di conseguenza gli ostacoli tra loro, ed utilizzando un numero di raggi pari a 20. I risultati ottenuti hanno avuto dei miglioramenti significativi come si nota dalle curve (verde, rosso, grigio, turchese). A differenza dei casi con un numero di raggi pari a 45, l'andamento delle curve del grafico del “Robot's Time to Reach Goal” è proporzionale all'ascesa delle curve del “Success Rate”, suggerendo che, a differenza dei primi esperimenti, l'agente impiega sempre meno tempo per raggiungere l'obiettivo, confermando l'ipotesi che il modello è in grado di muoversi più efficientemente nello spazio. Nonostante un fascio di 7 raggi sia poco realistico, è stato deciso di effettuare comunque delle prove attenendosi alla dimensione del vettore di input originale di CADRL, dal momento che bisogna tenere in considerazione che quest'ultimo (il vettore del LiDAR) debba essere unito al vettore del **SelfState**, contenente le informazioni sullo stato dell'agente, prima di essere passato al modello. Grazie a queste modifiche è stato possibile ottenere un miglioramento sostanziale in termini di success rate, collision rate e di tempo necessario per raggiungere il goal. Infine, stati avviati dei test di 500 episodi per ogni modello, vedesi **Tabella 3.2**

Model	success rate	collision rate	avg time	cumulative reward	freq in danger	avg min dist
dynamic_0humans-ld3-d510-20beams-lr_0001	0.502	0.002	14.801	0.319	0.0	0.0
dynamic_5humans-ld3-d815-7beams	0.658	0.078	13.748	0.278	0.061	0.045
dynamic_5humans-ld3-d510-20beams-lr_0005	0.602	0.17	13.786	0.273	0.115	0.059
dynamic_5humans-ld5-d815-45beams	0.02	0.132	24.688	-0.038	0.056	0.038
dynamic_0humans-ld3-d815-7beams	0.584	0.004	14.496	0.294	0.0	0.0
dynamic_5humans-ld3-d815-45beams	0.41	0.056	17.9995	0.163	0.052	0.045
dynamic_0humans-ld3-d510-20beams-lr_0005	0.562	0.004	13.5405	0.359	0.0	0.0
dynamic_0humans-ld5-d815-45beams	0.022	0.0	24.6155	0.011	0.0	0.0
dynamic_5humans-ld3-d510-20beams-lr_0001	0.63	0.098	13.069	0.309	0.111	0.058
dynamic_0humans-ld3-d815-45beams	0.364	0.0	18.2835	0.191	0.0	0.0

Tabella 3.2: Risultati ottenuti dopo l'esecuzione di 500 test case.

Gli esperimenti condotti sono stati essenziali per ottenere una prima panoramica sulle limitazioni del modello. Dall'analisi dei risultati e dei grafici, emergono chiaramente situazioni in cui un numero eccessivo di raggi si traduce in un apprendimento poco efficace. Al contrario, quando forniamo in input un numero di raggi che formano un vettore di lunghezza simile all'originale (come nel caso di 7 raggi) o vicino ad esso (come nel caso di 20 raggi), si nota un apprendimento più efficace. È probabile

che sia necessario migliorare l'architettura del modello di navigazione, considerando opzioni come l'incremento delle dimensioni, il numero di livelli nascosti o l'introduzione di operazioni di convoluzione 1D per catturare le relazioni locali presenti nel segnale del LiDAR. In Figura 3.14, sono riportati degli snapshot di un episodio di navigazione in ambiente dinamico che ha appreso correttamente come raggiungere la destinazione evitando ostacoli e umani.

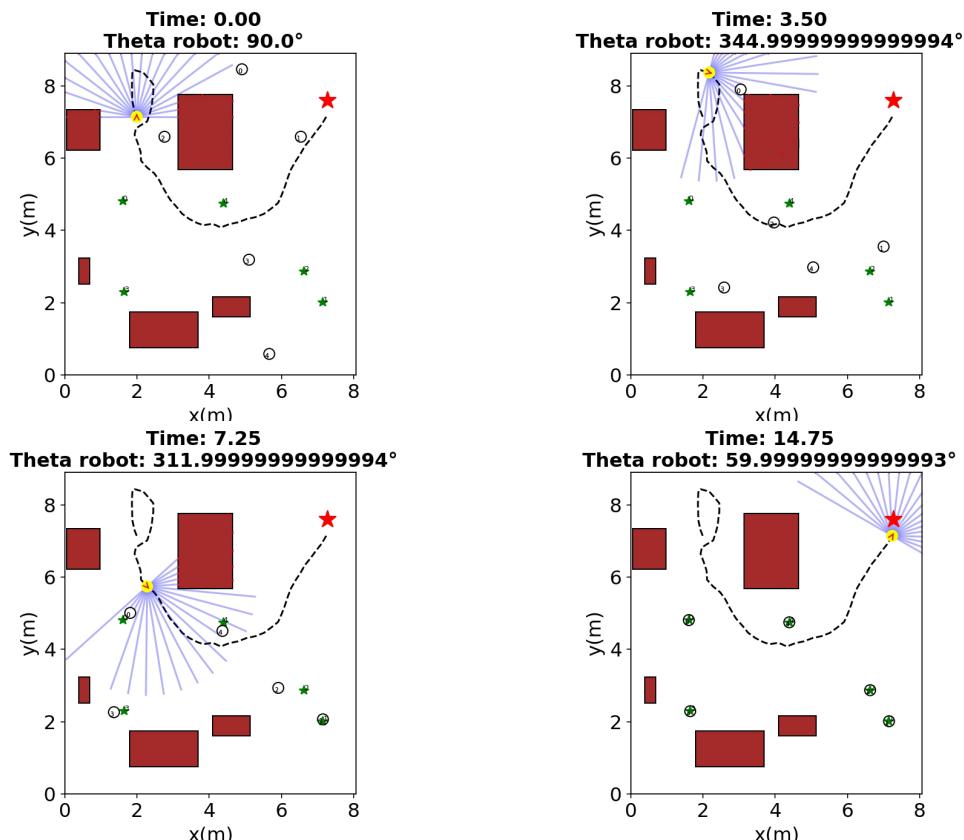


Figura 3.14: Evoluzioni di un episodio di navigazione su ambiente dinamico. L'agente in giallo è equipaggiato con un LiDAR di 20 raggi e deve raggiungere la sua destinazione (stella rossa) evitando di scontrarsi con gli oggetti fissi (rettangoli marroni) e con gli umani (cerchi bianchi), che a loro volta navigano verso il loro goal (stelle verdi)

Training in ambienti con planimetrie

In questi esperimenti l'agente viene immesso in planimetrie di ambienti reali, come precedentemente descritto nel Capitolo 3.1.1. Per mezzo di tali test si vuole misurare la capacità di apprendimento e adattamento in contesti più complessi rispetto alla semplice disposizione randomica degli ostacoli. Per garantire al modello di navigazione di osservare una grande varietà di ambienti, è stato ritenuto opportuno cambiare la planimetria di allenamento ad intervalli regolari, pari alla divisione tra il numero totale di episodi e il numero di planimetrie disponibili nel training set. Utilizzando tutte le 71 planimetrie con 10000 episodi, viene effettuato un cambio di ambiente ogni 117 episodi. In particolare si vuole cercare di quantificare il numero di episodi necessari affinché l'agente apprenda efficacemente la struttura di ciascuna planimetria. Cambiare la planimetria troppo presto potrebbe impedire al modello di apprendere adeguatamente o, al contrario, una variazione poco frequente limiterebbe la sua capacità di adattamento, per questo motivo sono stati avviati vari training utilizzando un diverso numero di planimetrie. Attraverso l'esposizione a una varietà di ambienti, l'agente è stimolato a sviluppare strategie più generali, in grado di affrontare con successo una gamma più ampia di situazioni. Ciò è particolarmente utile per evitare il fenomeno dell'overfitting, cosicché il modello non si adatti eccessivamente a uno specifico ambiente, bensì impari a generalizzare su scenari diversi. Inoltre, tali cambiamenti di scenario possono fungere da test critici per identificare eventuali lacune nelle capacità del modello, rivelando se vi sono aree in cui potrebbe essere necessario un miglioramento. Se il modello riesce a mantenere prestazioni consistenti nonostante le variazioni ambientali, ciò aumenta la fiducia nella sua capacità di gestire scenari reali complessi. Al contrario, eventuali diminuzioni di prestazioni durante i cambi di planimetria potrebbero indicare la necessità di ottimizzazioni o di ulteriori adattamenti nell'architettura del modello. Le configurazioni dei training effettuati sono riassunte nella Tabella 3.3.

beams	humans	LiDAR distance	planimetries	training ep.	lr
45	5	5	1	5000	0.0001
45	5	3	71	1000	0.0001
45	5	3	1	9000	0.0001
45	5	3	10	9000	0.0001
7	5	3	1	10000	0.0001
7	5	3	71	10000	0.0001
7	0	3	10	10000	0.0001
7	5	3	10	50000	0.0001

Tabella 3.3: Configurazioni di addestramento utilizzate per gli esperimenti.

Si aggiunge che in alcune configurazioni è stato necessario fermare preventivamente l'addestramento data la quantità esigente di tempo richiesto per il completamento. I grafici dei risultati ottenuti sono in Figura 3.15.

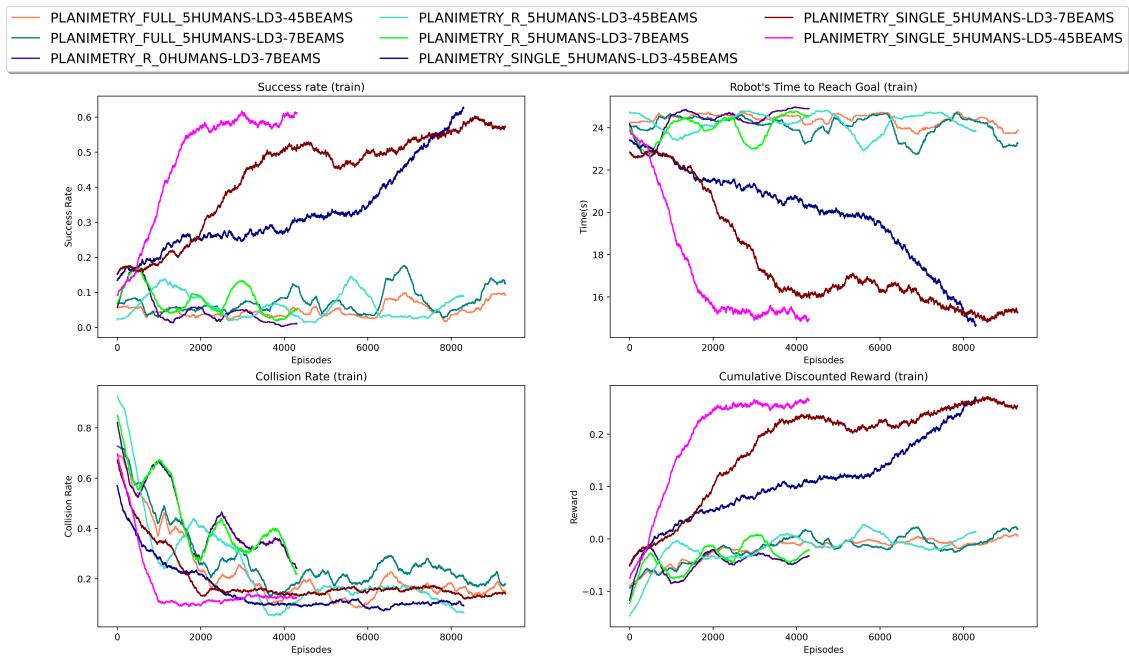


Figura 3.15: Risultati del training negli ambienti con planimetria ottenuti con le configurazioni in Tabella 3.1

I risultati più promettenti emergono chiaramente dal training effettuato sul dataset ridotto o su una singola planimetria (curve fucsia, marrone e blu scuro). È interessante notare che, a differenza degli esperimenti in cui sono stati usati ambienti dinamici, le configurazioni con 45 raggi e distanza 5 producono risultati soddisfacen-

ti probabilmente perché trovandoci in un ambiente più limitato del caso dinamico, il modello ha bisogno di più informazioni per percepire l'ambiente. D'altra parte, l'impiego del dataset completo di planimetrie (curve in verde, turchese, arancio, viola, verde acqua) non ha portato a miglioramenti significativi nelle prestazioni. L'oscillazione continua tra 1 e 0 nella curva della "Success Rate", sia nel caso dell'uso della planimetria ridotta che in quella completa, suggerisce che l'agente sta incontrando difficoltà nella generalizzazione. È probabile che, ad ogni cambiamento della planimetria, l'agente perda la capacità di orientarsi in modo efficace. Questa ipotesi trova supporto nell'analisi combinata del "Robot's Time to Reach Goal" e del "Collision Rate", che indica che il robot sembra propendere per rimanere fermo piuttosto che affrontare una penalizzazione nella reward. Questo risultato suggerisce che, per apprendere in modo più efficace da una vasta gamma di ambienti complessi, il modello necessita di un numero significativamente maggiore di episodi. Nell'ambito di questi esperimenti, è stato impiegato un numero di episodi identico a quello del progetto originale, è importante considerare che gli ambienti originali non presentavano ostacoli e risultavano meno complessi. Una stima approssimativa indica la necessità di un numero di episodi almeno 10 volte superiore, questa conclusione è supportata anche dall'analisi visiva dei grafici. Si aggiunge che le istanze con 20 raggi e learning rate incrementato non sono state avviate per mancanza di tempo. Infine, sono stati avviati dei test di 500 episodi su ogni modello, vedi Tabella 3.4

Model	success rate	collision rate	avg time	cumulative reward	freq in danger	avg min dist
planimetry_r_0humans-ld3-7beams	0.026	0.006	13.453	0.015	0.0	0.0
planimetry_r_5humans-ld3-45beams	0.084	0.162	14.296	-0.023	0.084	0.099
planimetry_full_5humans-ld3-45beams	0.082	0.15	12.373	-0.023	0.091	0.103
planimetry_r_5humans-ld3-7beams	0.066	0.198	16.874	-0.04	0.09	0.094
planimetry_single_5humans-ld3-45beams	0.152	0.164	12.382	0.017	0.096	0.101
planimetry_full_5humans-ld3-7beams	0.1	0.202	11.197	-0.015	0.082	0.108
planimetry_single_5humans-ld5-45beams	0.166	0.214	9.119	0.006	0.093	0.088
planimetry_single_5humans-ld3-7beams	0.132	0.276	12.748	-0.024	0.097	0.095

Tabella 3.4: Risultati ottenuti dall'esecuzione di 500 test case per ogni modello.

I risultati ottenuti, sebbene non possano essere classificati come soddisfacenti, costituiscono comunque una base valida su cui costruire per futuri studi. In alcuni casi, il modello ha comunque registrato esiti positivi, come evidenziato nella Figura 3.16, che mostra uno snapshot di un episodio di navigazione in cui l'agente è riuscito a raggiungere con successo la destinazione.

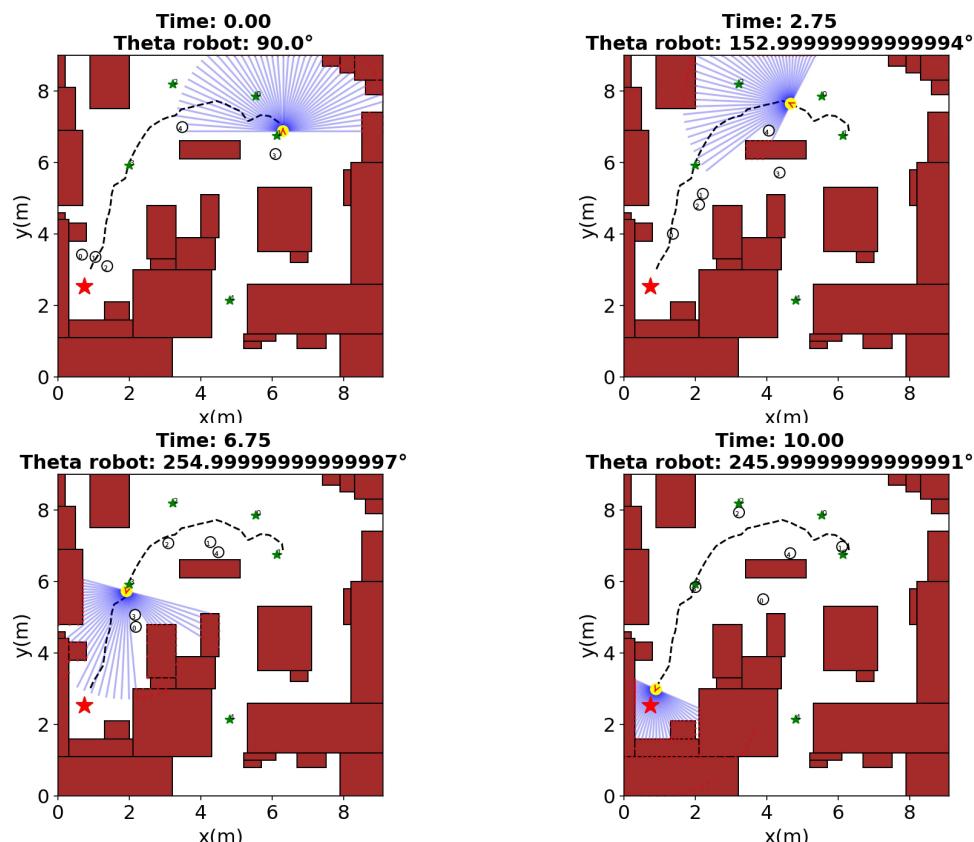


Figura 3.16: Evoluzione di un episodi di navigazione su ambiente caricato da planimetria. L'insieme degli ostacoli (rettangoli marroni) è piazzato in modo da replicare una planimetria estratta da un ambiente reale.

4

Conclusioni

Nel corso di questo lavoro, sono state esaminate le piattaforme disponibili in commercio, evidenziando le sfide e le opportunità presenti nell’ambito della navigazione autonoma. Il focus iniziale è stato rivolto agli algoritmi di navigazione classici, analizzando approfonditamente l’acquisizione dell’ambiente, la creazione delle mappe e il path planning. Questa fase ha permesso di comprendere a fondo i vantaggi e gli svantaggi di tali approcci consolidati, sottolineando la necessità di soluzioni più flessibili e adattabili ai contesti reali. Successivamente, lo studio si è indirizzato verso gli algoritmi basati sul learning, esplorando la breve storia del machine learning e del deep learning. Il focus si è poi concentrato sul reinforcement learning, approfondendo concetti come il Markov Decision Process e il Deep Reinforcement Learning. In particolare, sono state esaminate soluzioni di navigazione in presenza di persone, tra cui LSTM-RL, CADRL e SA-CADRL. Il contributo più significativo di questo lavoro è stato lo sviluppo di CrowdSim-Real, che estende il simulatore originale permettendo training in scenari realistici. Attraverso la modifica delle policy degli umani, la simulazione del LiDAR e altre modifiche al modello, abbiamo cercato di affrontare le problematiche e migliorare la rappresentazione degli ambienti complessi in cui i robot devono navigare. Durante la fase di training, è stato necessario implementare strategie mirate a limitare i calcoli al fine di ottenere una simulazione più efficiente e risultati più rapidi. Nonostante i risultati iniziali degli esperimenti non siano stati particolarmente positivi, è stato possibile apportare miglioramenti sostanziali attraverso la variazione della configurazione di training, ottenuta attraverso diverse prove, che hanno coinvolto la modifica del numero di raggi del LiDAR e l’adattamento delle dimensioni dell’ambiente, specialmente nel contesto di scenari dinamici. Nonostante i progressi ottenuti negli ultimi esperimenti,

ti, emergono alcune limitazioni che aprono la strada a possibili aree di miglioramento e future ricerche. Una delle sfide principali riguarda il fenomeno noto come “stuck problem”. La preferenza del robot di rimanere fermo anziché esplorare o rischiare di eseguire azioni che potrebbero portare a penalizzazioni richiede un’attenzione particolare. Una possibile soluzione potrebbe derivare dalla modifica delle reward, introducendo penalità più severe per il comportamento “stuck” e incoraggiando così il modello a muoversi in modo più proattivo. Un’altra limitazione chiave riguarda la dimensione dell’input fornito all’attuale modello di navigazione (numero di raggi del LiDAR). Infatti, un input più ricco di informazioni può migliorare notevolmente la precisione delle letture sensoriali e, di conseguenza, la capacità dell’agente di comprendere e reagire in modo più accurato nell’ambiente circostante e di reagire correttamente. Tuttavia, è importante considerare che l’aumento della complessità dell’input implica la necessità di un modello di rete neurale più grande e con più parametri per elaborare efficacemente un maggior numero di dati, ciò comporta inevitabilmente una richiesta di un numero maggiore di episodi di allenamento, traducendosi in un prolungamento del tempo necessario per completare il processo di addestramento. Pertanto, l’esplorazione di nuove architetture più complesse e l’ottimizzazione del processo di simulazione emergono come priorità chiave per possibili sviluppi futuri del sistema. L’ottimizzazione delle policy di navigazione degli umani simulati rappresenta un’altra direzione di sviluppo. Abbiamo osservato che logiche più complesse, come il ritorno indietro una volta raggiunto l’obiettivo, potrebbe contribuire a una simulazione più realistica per avere umani in movimento durante l’intera durata. Si aggiunge la limitazione riscontrata con l’algoritmo ORCA nel navigare verso l’obiettivo in presenza di ostacoli o quando il goal non è direttamente visibile agli umani simulati. Miglioramenti significativi in questo contesto sono necessari per garantire una rappresentazione più fedele del comportamento umano, contribuendo in modo significativo alla precisione e alla completezza della simulazione. Ridurre la complessità computazionale e ottimizzare i tempi di addestramento rappresentano sfide cruciali da affrontare in futuro, potrebbe dunque essere vantaggioso esplorare approcci di simulazione parallela su GPU, l’implementazione di questa funzionalità potrebbe accelerare notevolmente il processo di simulazione de-

gli scenari di simulazione e quello di training, consentendo esperimenti più rapidi e una maggiore flessibilità nella modifica e nella sperimentazione di modelli diversi e complessi. In conclusione, questo lavoro rappresenta un notevole passo avanti nel campo dell'apprendimento autonomo, ma riconosce apertamente che il cammino verso la maturità del modello è ancora in corso. Le limitazioni identificate non solo offrono spunti per miglioramenti, ma stimolano riflessioni profonde che saranno fondamentali per la realizzazione di agenti intelligenti sempre più competenti e versatili in ambienti complessi e dinamici.

Ringraziamenti

Ringrazio il prof. Farinella per avermi permesso di intraprendere questo studio ed il dott. Rosano per avermi supportato durante lo sviluppo del progetto. Nonostante le numerose notti insonni ad implementare funzionalità e a monitorare i training, sono profondamente orgoglioso di aver concluso questo percorso con uno studio così complesso e di aver contribuito, almeno in parte, allo sviluppo futuro di questi sistemi.

Ringrazio le persone per cui è stato possibile tutto questo: i miei genitori che mi hanno insegnato puntare sempre in alto e a non accontentarmi mai senza farmi mai mancare nulla, nonostante le avversità. A voi dedico questo percorso, consapevole che il vostro desiderio era che completassi gli studi. Sappiate che non avrei mai accettato di fermarmi solo perché ho avuto la necessità di lavorare ed è solo grazie ai vostri insegnamenti che sono giunto a questo traguardo. Un grazie va anche alla mia sorellina per essermi stata accanto durante questo periodo di impegni, per esserti accollata tu le tante cose che avrei dovuto fare io, per essere la mia compagnia preferita di shopping e perfetta ridimensionatrice dei miei drammi, facendomi realizzare che a volte (spesso) i problemi esistono solo nella mia testa. Una ringraziamento particolare va ad un membro della famiglia che purtroppo non mi aspetterà a casa, Pluto. Ti ringrazio per avermi fatto compagnia durante il primo anno di questo percorso, quando ero costretto a restare a casa tra lavoro, Covid e studio, non facendomi pesare tutte le ore trascorse in postazione. Ogni volta che pensavo “*ma chi mel’ha fatto fare?*” guardavo i tuoi occhioni pieni di felicità e realizzavo che sarebbe stato solo un periodo in cui potevo godermi di più te e la mia famiglia.

Ringrazio la mia ragazza, Marta. Ti ringrazio per la profonda stima che hai nei miei confronti e nelle mie potenzialità, per esserti accollata di stare a casa infiniti weekend a causa della mia scaramanzia, “perché prossima settimana c’è l’esame”, per essere stata paziente, comprensiva, per avermi fatto ragionare nei momenti di ira, per esserti preoccupata più tu di me della mia salute mentale e fisica e, soprattutto, per essere stata semplicemente insieme a me. Aver trascorso questo periodo, a tratti infernale, con te a fianco è stata la cosa migliore che potesse capitarmi e sono felice di aver concluso questo percorso con te. Sei sempre stata pronta a starmi vicino con un sorriso nonostante a volte avessi la stessa voglia di vivere di un ameba e per questo non ti sarò mai grato abbastanza.

Ringrazio la mia comitiva, i miei amici di una vita e quelli nuovi: Antonio, Barbara, (Zio) Carmelo, Carmen, Chiara, Ciccio, Dario, Elia, Marta (di nuovo), Mimmo, Nico, Silvio, Salvo Bascetta, Tore Russo, Tizzone, Vincenzo. Nonostante la mia poca presenza, i miei scleri e le mie prese a male ci siete sempre stati, accettandomi ed essendo sempre pronti ad esserci in ogni occasione. Vi ringrazio per tutte le volte che mi avete fatto passare via i brutti pensieri, essere stati una valvola di sfogo e soprattutto per essere sempre una costante della mia quotidianità. Lavorando ho notato che spesso che, col passare degli anni, le comitive si sfaldano, le persone cambiano ed è difficile mantenere rapporti soldi con una cerchia di amici. Sebbene sia abbastanza complicato, il più delle volte, ragionare con 12 *buzzurri* e 4 *streghe* devo dire di essere orgoglioso di poter fare affidamento su di voi e spero che malgrado le incomprensioni ed i difetti di ognuno di noi resti sempre salda la voglia di restare insieme. Vi voglio bene.

Desidero esprimere un ringraziamento speciale a tutta la restante parte della mia famiglia e a ogni amico che non ho menzionato, sono convinto che non ci sia mai stato un momento in cui non abbiate creduto in me. Vorrei includere in questo gruppo anche i ragazzi delle partite *dal Signorello* che, nonostante le mie limitate capacità, mi hanno sempre incluso ed incoraggiato costantemente a migliorare e a concentrarmi sulla crescita anziché sui fallimenti, filosofia che cerco di applicare ogni giorno nella mia vita.

Desidero esprimere la mia gratitudine anche a tutti i colleghi, con i quali ho collaborato, la cui compagnia ha reso le giornate di lavoro più piacevoli. Desidero ringraziare anche la disponibilità dei titolari (alcuni), i quali si sono dimostrati sempre disponibili e comprensibili a modificare i turni e hanno costantemente dato la priorità alla mia formazione rispetto alle esigenze lavorative.

In conclusione, ringrazio me stesso per essere riuscito a completare questo lavoro nei tempi prefissati fregandomene della stanchezza, delle troppe ore trascorse davanti uno schermo, della solitudine e dei problemi. È stato un percorso tortuoso, difficile e continuamente passato a “testa bassa a lavorare” ma adesso finalmente posso rialzarla e cogliere i frutti del mio lavoro. Vorrei rivolgermi al “me” del futuro con un consiglio: non arrenderti mai, continua a credere in te stesso e ricorda che con il duro lavoro nulla è impossibile.

“*Non è nelle stelle che è conservato il nostro destino, ma in noi stessi.*”

Danilo

Bibliografia

- [1] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [2] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and et al. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.
- [3] B. Bakker. Reinforcement learning with long short-term memory. In *Advances in Neural Information Processing Systems*, 2001.
- [4] J. Barraquand and J. Latombe. Robot motion planning: A distributed representation approach. *International Journal of Robotic Research - IJRR*, 10:628–649, 12 1991.
- [5] Igor Borovikov. https://www.gamedev.net/tutorials/_/technical/artificial-intelligence/navigation-graph-generation-r2805/, 2011.
- [6] S. Caselli and M. Reggiani. Erpp: An experience-based randomized path planner. pages 1002 – 1008 vol.2, 2000.
- [7] S. Caselli, M. Reggiani, and R. Rocchi. Heuristic methods for randomized path planning in potential fields. In *Proceedings 2001 IEEE International Symposium on Computational Intelligence in Robotics and Automation (Cat. No.01EX515)*, pages 426–431, 2001.
- [8] S. Caselli, M. Reggiani, and R. Sbravati. Parallel path planning with multiple evasion strategies. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, volume 1, pages 260–266 vol.1, 2002.

- [9] C. Chen, Y. Liu, S. Kreiss, and A. Alahi. Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning.
- [10] Y. F. Chen, M. Everett, M. Liu, and J. P. How. Socially aware motion planning with deep reinforcement learning, 2018.
- [11] Y. F. Chen, M. L., M. Everett, and J. P. How. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning, 2016.
- [12] T. M. Cover. Learning in pattern recognition. In S. Watanabe, editor, *Methodologies of Pattern Recognition*, pages 111–132. Academic Press, 1969.
- [13] C. Demetrescu, I. Finocchi, and G.F. Italiano. *Algoritmi e strutture dati*. Collana di istruzione scientifica. McGraw-Hill Companies, 2004.
- [14] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1959.
- [15] Stanford Artificial Intelligence Laboratory et al. Robotic operating system.
- [16] P. Florence, C. Lynch, A. Zeng, O. Ramirez, A. Wahid, L. Downs, A. Wong, J. Lee, I. Mordatch, and J. Tompson. Implicit behavioral cloning. 2021.
- [17] P. Florence, L. Manuelli, and R. Tedrake. Self-supervised correspondence in visuomotor policy learning, 2019.
- [18] F. Gomez-Barvo (eds.) G. Carbone. *Motion and Operation Planning of Robotic Systems: Background and Practical Approaches*. Springer.
- [19] A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi. Social gan: Socially acceptable trajectories with generative adversarial networks, 2018.
- [20] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

- [21] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. 1968.
- [22] D. Helbing and P. Molnár. Social force model for pedestrian dynamics. *Physical Review E*, 51(5):4282–4286, May 1995.
- [23] P. Henderson, R. Islam, P. Bachman, and et al. Deep reinforcement learning that matters. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 2285–2294. PMLR, 2018.
- [24] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [25] F. Hoeller, D. Schulz, M. Moors, and F.E. Schneider. Accompanying persons with a mobile robot using motion prediction and probabilistic roadmaps. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007.
- [26] A. Javaid. Understanding dijkstra algorithm. *SSRN Electronic Journal*, 2013.
- [27] Stylianos Kampakis. What deep learning is and isn't. <https://thedatascientist.com/what-deep-learning-is-and-isnt/>, 2021.
- [28] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 500–505, 1985.
- [29] K. Kimura, J. Reichert, A. Olson, O. Ranjbar Pouya, X. Wang, Z. Moussavi, and D. Kelly. Orientation in virtual reality does not fully measure up to the real-world. *Scientific Reports*, 7, 2017.
- [30] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 2012.
- [31] S. Leijnen and F. Veen. The neural network zoo. 47:9, 2020.

- [32] Y. Li. Deep reinforcement learning: An overview. 2017.
- [33] T. P Lillicrap, J. J Hunt, A. Pritzel, and et al. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.
- [34] L. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.
- [35] MathWorks. <https://it.mathworks.com/help/reinforcement-learning/ug/create-custom-grid-world-environments.html>.
- [36] MathWorks. Che cos’è il reinforcement learning? <https://it.mathworks.com/discovery/reinforcement-learning.html>.
- [37] V. Mnih, K. Kavukcuoglu, D. Silver, and et al. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
- [38] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.
- [39] Rukshan Pramoditha. The concept of artificial neurons (perceptrons) in neural networks. <https://towardsdatascience.com/the-concept-of-artificial-neurons-perceptrons-in-neural-networks-fab22249cbfc>, 2021.
- [40] A. G. Barto R. S. Sutton. *Reinforcement Learning: An Introduction, 2nd Edition*. Bradford Books, 2011.
- [41] Pratik Randad. Reinforcement learning- basics. <https://medium.com/mathematics-vidhya/a-beginners-guide-to-reinforcement-learning-88a330d8d94e>, 2020.
- [42] J. Ruiz-Serra, J. White, S. Petrie, T. Kameneva, and C. McCarthy. Towards self-attention based navigation in the real world. 09 2022.
- [43] D. E Rumelhart, G. E Hinton, and R. J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

- [44] J. Schulman, P. Moritz, S. Levine, and et al. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations*, 2016.
- [45] A. Signifredi, L. Bombini, A. Coati, J. S. Medina, and D. Molinari. A general purpose approach for global and local path planning combination. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 996–1001, 2015.
- [46] R. Staiger. Terrestrial laser scanning. 2003.
- [47] J. Sundram, H. Nguyen Duong, G. Soh, R. Bouffanais, and K. Wood. Development of a miniature robot for multi-robot occupancy grid mapping. 07 2018.
- [48] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.
- [49] Sybernix. K-means data clustering, 2017.
- [50] A. Szot, A. Clegg, E. Undersander, E. Wijmans, Y. Zhao, J. Turner, N. Maestre, M. Mukadam, D. Chaplot, and O. Maksymets et al. Habitat 2.0: Training home assistants to rearrange their habitat. *Advances in Neural Information Processing Systems*, 34:251–266, 2021.
- [51] P. Tang, D. Huber, B. Akinci, R. Lipman, and A. Lytle. Automatic reconstruction of as-built building information models from laser-scanned point clouds: A review of related techniques. *Automation in Construction*, 19(7):829–843, 2010.
- [52] F. Torabi, G. Warnell, and P. Stone. Behavioral cloning from observation. 2018.
- [53] J. van den Berg, S. Guy, M. Lin, and D. Manocha. *Reciprocal n-Body Collision Avoidance*. 2011.

- [54] Kevin Wang. Path finder. <https://github.com/kevinwang1975/PathFinder/>.
- [55] C. JCH Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.
- [56] F. Xia, A. Zamir, Z. He, A. Sax, J. Malik, and S. Savarese. Gibson env: Real-world perception for embodied agents. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9068–9079, 2018.
- [57] L. Yao, N. Ballas, João P. Costa, S. Gülçehre, and A. C. Courville. Deepsign: Deep learning for automatic sign language recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 20–28. IEEE, 2015.
- [58] X. Yao, J. Zhang, and J. Oh. Following social groups: Socially compliant autonomous navigation in dense crowds. 2019.
- [59] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics, 2020.
- [60] T. Zhang, Z. McCarthy, O. Jow, D. Lee, X. Chen, K. Goldberg, and P. Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation, 2018.
- [61] J. Zhu, T. Park, P. Isola, and A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.