Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica per il Management

Analisi e validazione di tecniche di Reinforcement Learning su sensori low-power per applicazioni IoT di monitoraggio

Relatore:
Chiar.mo Prof.
Marco Di Felice

Presentata da:
Michele Greco

Correlatore:
Dott.
Angelo Trotta

Sessione III
Anno Accademico 2020/2021

Sommario

Lo studio di questa tesi è stato condotto in un periodo storico caratterizzato da un'elevata presenza di dispositivi connessi alla rete che sono in grado di percepire, elaborare, produrre e scambiare un elevato numero di dati con altri dispositivi. Tali dispositivi, connessi tra loro e in continua comunicazione, costituiscono l'Internet of Things, una rete che trova applicazione in diversi campi risolvendo problemi, creando opportunità e proponendo nuove sfide. In questo studio la sfida raccolta è stata quella di affiancare l'intelligenza artificiale a dispositivi IoT in modo da renderli capaci di apprendere, in autonomia, come svolgere il proprio lavoro in modo efficiente e cercando di risparmiare energia. Lo studio è stato condotto mediante simulazione di un sistema di monitoraggio ambientale costituito da un dispositivo governato da agente decisionale di Q-learning che trasmette informazioni ad attuatori simulati, che intervengono sui parametri ambientali, e a un centro di raccolta dati simulato. Le prestazioni del dispositivo smart sono state confrontate con quelle di altri due dispositivi di controllo semplici. Uno dei due dispositivi di controllo, detto diligent, ha avuto il compito di rappresentare il dispositivo più reattivo in merito agli interventi sui valori dei parametri ambientali. L'altro dispositivo semplice, detto energy saver, ha avuto il compito di risolvere il problema riducendo il numero di operazioni di sense utilizzate per entrare a conoscenza dei valori dei parametri ambientali. Lo studio ha dimostrato come il Reinforcement Learning sia effettivamente in grado di risolvere problemi descrivibili tramite modellazione di catene di decisioni per compiti a circuito chiuso dotati di stati che godono della proprietà di Markov. I risultati mostrano come il dispositivo smart sia stato in grado di apprendere autonomamente come comportarsi per svolgere il compito assegnato combinando i punti di forza dei dispositivi di controllo.

Indice

Elenco delle figure

Elenco delle tabelle

Listings

Introduzione

Lo studio di questa tesi è stato condotto in un periodo storico in cui sempre più oggetti possono connettersi alla rete e che, sempre più spesso, comprendono nel proprio nome la parola "smart". Telefoni, sistemi di gestione degli edifici e dispositivi in generale vengono impiegati ogni giorno per svolgere compiti che richiedono competenze che eguagliano, e talvolta superano, quelle di individui specializzati nell'esecuzione di un compito. La rete costituita da questi dispositivi è l'Internet of Things (IoT), la disciplina che invece consente ai dispositivi di acquisire caratteristiche tipiche dell'intelletto umano è l'Intelligenza Artificiale (I.A.).

Attualmente c'è sempre più bisogno di infrastrutture e servizi capaci di portare a termine compiti da remoto che, talvolta, necessitano di essere svolti senza l'intervento umano. Si pensi ad esempio al monitoraggio a distanza di pazienti sottoposti a trattamenti medici, al compito di monitoraggio condotto da robot intelligenti [3], all'automazione industriale, ai dispositivi utilizzati quotidianamente dagli sportivi per raccogliere dati relativi alle proprie performance oppure all'industria dell'intrattenimento che profila i propri utenti. Le effettive necessità appena accennate, che oggi più di ieri si fanno sentire a gran voce nella vita di tutti i giorni, e che domani più di oggi si potranno ribadire, hanno sottolineato l'importanza di condurre ricerche nei settori dell'IoT e dell'I.A.. Tali discipline sono infatti di ampio utilizzo in qualsiasi settore immaginabile e, già da tempo, sono in grado di fornire proposte risolutive alle situazioni introdotte, le discipline infatti consentendo di mettere in atto strategie effettive che conducono verso un miglioramento generale della qualità della vita. Per i motivi appena spiegati è stato deciso di intraprendere, con curiosità e dedizione, lo studio che ha condotto alla realizzazione di questo elaborato.

Lo scopo del lavoro di questa tesi è quello di progettare e implementare soluzioni software che consentono a dispositivi IoT di trarre vantaggio dall'I.A. al fine di svolgere il compito assegnato in modo efficiente riuscendo a risparmiare energia. Lo scenario preso in considerazione è quello in cui il dispositivo smart IoT progettato si occupa del monitoraggio ambientale comunicando informazioni utili ad attuatori che intervengono prontamente sui valori dei parametri che descrivono l'ambiente. Il dispositivo smart IoT che si intende progettare deve essere capace di capire in autonomia quale sia il lavoro che gli è stato assegnato e, di conseguenza, deve essere in grado di apprendere come svolgerlo

al meglio.

Per raggiungere gli obiettivi prefissati è stato necessario approfondire i temi relativi a IoT, Big Data e I.A.. I risultati del percorso di ricerca sono stati riportati nel Capitolo 1 dedicato allo stato dell'arte. La sezione 1 del Capitolo 1 tratta il tema dell'IoT. Questa sezione fa luce sul fatto che i dispositivi IoT, che fanno parte della rete, possono essere estremamente diversi gli uni dagli altri. Per esempio, si potrebbe avere a che fare con una rete IoT costituita da uno smartphone, uno smartwatch, un computer e un dispositivo dedicato all'assistenza domestica. I dispositivi dell'esempio devono poter comunicare tra loro senza che le differenze siano di ostacolo al raggiungimento dello scopo per la quale la rete è stata realizzata. I dispositivi utilizzano sensori e attuatori per poter interagire con l'ambiente che li circonda, grazie a questi componenti elettronici e alle computazioni effettuate, i dispositivi, producono un'elevata mole di dati che vengono scambiati attraverso la rete. Emerge dunque la necessità di utilizzare una piattaforma e dei protocolli di comunicazione, con e senza fili, che consentono ai diversi dispositivi di comunicare tra loro. Tra i numerosi protocolli utilizzati per realizzare reti di sensori è possibile fare un esempio di protocolli utilizzabili citando Zigbee e LoRa che consentono di mettere in atto comunicazioni machine to machine. Si nota inoltre il fatto che i dispositivi compiono continuamente operazioni e che tali operazioni per poter manifestarsi necessitano di consumare energia. In diverse applicazioni comuni i dispositivi IoT vengono alimentati attraverso l'utilizzo di batteria. Anche se la ricerca e lo sviluppo hanno consentito di realizzare batterie con prestazioni sempre migliori, risulta comunque necessario identificare strategie atte a preservarne la durata. Il focus di questo studio è rivolto al comportamento dei dispositivi e, di conseguenza, allo studio delle strategie adottabili per determinare le linee di condotta che determinano come le azioni vengano decise ed eseguite dai dispositivi.

Nella sezione 2 del Capitolo 1 si approfondisce il tema dei Big Data. Dagli approfondimenti è emerso che i set di dati tipicamente prodotti dai dispositivi IoT, in numerose situazioni, possono essere considerati appartenenti ai Big Data. Quello che ne consegue è che gli scenari in cui si ha a che fare con i Big Data sono caratterizzati da una rapida generazione di dati, che descrivono in modo accurato l'argomento a cui si riferiscono, che molto spesso necessitano di un'altrettanta rapida elaborazione. Dunque, lo scenario delineato dal sistema di monitoraggio in questione, dovrà essere caratterizzato dalla possibilità di poter descrivere continuativamente e in modo frequente lo stato dell'ambiente. Un'ulteriore caratteristica dovrà essere un'adeguata prontezza all'elaborazione dei dati e una consona reattività di intervento idonei a soddisfare le esigenze dei contesti in cui si lavora con i Big Data.

Nella sezione 3 e 4 del Capitolo 1 sono stati analizzati i temi relativi all'Intelligenza Artificiale e alle più comuni famiglie di metodologie del Machine Learning. Quello che è stato evidenziato è che per risolvere alcuni problemi, è molto vantaggioso, oppure inevitabile, disporre di calcolatori che riescono a riprodurre caratteristiche tipiche dell'intelletto umano. La facoltà di apprendere a svolgere un compito attraverso sperimentazioni auto-

nome e automatiche, oppure, la capacità di prendere decisioni contestuali basandosi su previsioni ed esperienza, sono esempi delle caratteristiche umane che si desidera vengano riprodotte dai calcolatori. Se non ci si avvalesse di calcolatori dotati di tali capacità, la risoluzione di alcuni problemi potrebbe essere ottenuta in maniera sub ottimale o, addirittura, potrebbe non verificarsi affatto. Le macchine, e i dispositivi intelligenti in generale, riescono a unire il vantaggio delle caratteristiche della mente umana alla potenza di calcolo del calcolatore. Per via delle considerazioni fatte, e alla luce delle analisi condotte, è stato ritenuto opportuno identificare nella tecnica del Q-earning la risorsa che consente di dotare il dispositivo del progetto di intelligenza. Tale tecnica, appartenente alla famiglia del Reinforcement Learning, consente al dispositivo di apprendere in autonomia come svolgere il compito assegnato. Il dispositivo riesce infatti a fronteggiare gli ostacoli tipicamente incontrati negli scenari influenzati dalla presenza dei Big Data e le limitazioni imposte dall'alimentazione tramite batteria. Le tecniche del Reinforcement Learning, oltre ad avvalersi di risultati ottenuti dalle ricerche condotte in ambito matematico, informatico e statistico, si avvale anche delle scoperte risultanti dagli studi condotti nel campo della psicologia comportamentale. La tecnica del Q-learning, infatti, consente di adottare strategie simili a quelle adottate dagli animali che esplorano ambienti sconosciuti con lo scopo di capire come poter raggiungere i propri obiettivi attraverso il proprio comportamento. Il processo di apprendimento, previsto dalla tecnica adottata, è improntato sulla strategia trial and error. Durante l'apprendimento si manifesta un rinforzo, quindi una predilezione, della scelta di quelle azioni che conducono a risultati positivi. Le azioni che una volte scelte hanno condotto a risultati sgradevoli subiscono un processo di estinzione della scelta, questo significa che col passare del tempo, tali azioni, verranno scelte sempre meno fino a quando non verranno mai più scelte.

Al termine di questa fase di ricerca è stato appurato che implementare tecniche di I.A. direttamente su dispositivo IoT, dotate di risorse limitate, consente di realizzare sistemi in grado di portare a termine compiti complessi in autonomia diventando abili nella gestione di una intera rete di sensori capendo allo stesso tempo come ottimizzare il consumo di risorse.

Dopo la fase di studio dello stato dell'arte è seguita la fase descritta nel Capitolo 2 dedicata alla progettazione e realizzazione del progetto. In questa fase sono stati affrontati concretamente gli ostacoli che hanno condotto al raggiungimento degli obiettivi prefissati. Nella sezione 1 del Capitolo 2 viene descritto il processo di raccolta e analisi dei requisiti del progetto di tesi. Nella sezione 2 del capitolo 2 sono state illustrate le scelte progettuali, le metodologie, le tecnologie e i materiali adottati per l'implementazione del progetto. Si è scelto di adottare un approccio simulato per condurre esperimenti relativi all'apprendimento dell'agente di Q-learning e un approccio pratico in cui sono stati condotti esperimenti con dispositivi reali in scenari reali.

Per quanto riguarda l'approccio sperimentale si è deciso di realizzare un sistema software di sperimentazione che studia il comportamento di 3 dispositivi di monitoraggio: un dispositivo smart, uno diligente e uno a risparmio energetico. Nel sistema l'ambiente,

i dispositivi e gli attuatori sono simulati. I dispositivi di controllo servono come mezzo di paragone per valutare l'operato del dispositivo smart che è l'oggetto di studio di questa tesi. Il dispositivo diligente rappresenta il sistema che riesce a identificare tutte le variazioni critiche dei parametri ambientali. Il dispositivo a risparmio energetico rappresenta invece il sistema che dovrebbe consentire di risparmiare energia attraverso il dosaggio delle azioni eseguite. Il dispositivo smart è il dispositivo che attraverso l'Intelligenza Artificiale, e le caratteristiche intrinseche della tecnica adottata, dovrebbe riuscire a combinare i punti di forza dei due dispositivi di controllo. Il sistema implementato è in grado di effettuare simulazioni che consentono di osservare l'evoluzione dell'ambiente, il comportamento dei dispositivi e gli effetti degli attuatori che intervengono sull'ambiente.

Per quanto riguarda l'approccio pratico, è stato realizzato un prototipo hardware del dispositivo di monitoraggio smart IoT attraverso schede di sviluppo per microcontrollori. Il prototipo utilizzato mostra, nell'esperimento 4 e nell'esperimento 5, come viene svolto in un ambiente reale il compito di monitoraggio da parte di un agent di Q-learning specializzato.

Nel capitolo 4, dedicato alle conclusioni, è emerso che il dispositivo smart IoT è riuscito ad imparare in autonomia la sequenza di azioni che consentono di svolgere in modo efficiente il compito assegnato. Il dispositivo smart che ne risulta riesce quindi a massimizzare le ricompense e risparmiare energia. Con questo studio è stato quindi dimostrata la possibilità di implementare un sistema capace di risparmiare energia durante lo svolgimento di un compito. Il risultato ottenuto è molto importante perché il risparmio energetico favorisce, da un punto di vista economico, una riduzione dei costi e, dal punto di vista ambientale, una riduzione degli sprechi e dell'inquinamento.

Capitolo 1

Stato dell'arte

1 Internet of Things

L'Internet of Things (IoT - internet delle cose) è la rete composta dagli oggetti intelligenti interconnessi attraverso internet. Secondo una stima del Cisco Internet Business Solutions Group, la nascita dell'Internet of Things è avvenuta tra il 2008 e il 2009 nel momento in cui in rete erano connessi più dispositivi che persone. Nel 2018 sono stati contati 22 miliardi di dispositivi intelligenti connessi [4].

L'IoT è un'architettura basata su layers. Un esempio generale e molto semplice si può pensare costituito da tre layers. Nel primo layer ci sono gli oggetti che mediante opportuno hardware e software riescono ad accedere alla rete. Il secondo layer è quello della connettività caratterizzata dai protocolli di trasmissione dell'informazione. Il terzo layer è costituito dall'IoT cloud contraddistinto dal software che consente ai dispositivi di comunicare e che consente la gestione delle interazioni tra le persone e i dispositivi della rete [29]. In alcuni studi viene indicata la presenza di due strati trasversali che forniscono funzionalità a tutti gli altri strati longitudinali dell'architettura, questi due strati trasversali sono costituiti dalla sicurezza e dal management che sono importanti in ogni layer [22]. Esistono diverse implementazioni dell'architettura dell'Internet of Things e per questo motivo essa può variare in base allo scopo e al contesto della rete stessa e dei compiti che i dispositivi devono svolgere [22]. Un esempio di architettura IoT viene mostrata nella figura 1.1.

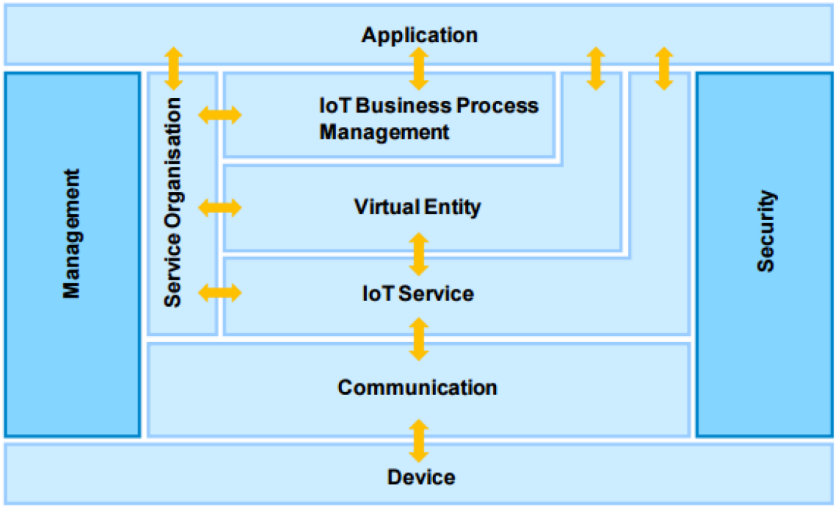


Figura 1.1: Esempio di architettura IoT [22].

Gli oggetti dell'IoT sono dispositivi dotati di diversi componenti elettronici tra cui sensori e attuatori. Sono inoltre provvisti dell'apposito software in grado di fornire le funzionalità per l'invio e la ricezione di messaggi. I sensori consentono di convertire misurazioni fisiche in dati numerici comprensibili ed utilizzabili dalla macchina, esempi di sensore sono quelli usati per percepire l'intensità della luminosità ambientale o determinare la prossimità di altri corpi. Gli attuatori sono le componenti che consentono al dispositivo di interagire con l'ambiente che lo circonda ed eventualmente cambiando lo stato in cui l'ambiente si trova. Un attuatore è ad esempio un led che si accende quando il dispositivo riceve un comando o quando un sensore percepisce la vicinanza di un corpo. Il led che illumina una stanza buia cambia lo stato della stanza che passa da buia a illuminata e per questo motivo viene considerato attuatore. In generale un attuatore converte l'energia elettrica in un'altra forma di energia.

Gli oggetti intelligenti, dunque, mediante i software e l'hardware di cui dispongono riescono ad inviare e ricevere informazioni trasmesse attraverso i protocolli di trasmissione dell'informazione. Per protocollo di trasferimento dell'informazione si intende un formato standardizzato utilizzato per la trasmissione di informazioni tra dispositivi. Le comunicazioni che avvengono nell'IoT sono Human to Machine (H2M: uomo - macchina) e Machine to Machine (M2M: macchina - macchina)[22]. I protocolli utilizzati nell'IoT dipendono dal livello dell'architettura in cui le informazioni devono essere trasferite. L'insieme delle caratteristiche dei dispositivi assieme ai protocolli dell'informazione consentono a tutti gli elementi dell'IoT di comunicare tra loro anche se sono diversissimi tra loro. Ad esempio i dispositivi comunicano con altri dispositivi e con i gateway e i gateway comunicano con i data center o con il cloud. I gateway vengono utilizzati perché fungono da intermediari tra i dispositivi, raccolgono i dati dei dispositivi e li inviano al cloud [1].

Una delle caratteristiche interessanti dell'IoT è il flusso di dati generato. I dispositivi che possono essere diversissimi tra loro sono univocamente identificabili. Entrando a far parte della rete contribuiscono alla creazione e modellazione dell'ambiente di cui fanno parte. Questo rende possibile avere informazioni sullo stato dell'ambiente dei dispositivi, sull'uso dei dispositivi e sulle persone che utilizzano o si trovano nell'ambiente dei dispositivi connessi.

Le reti IoT possono avere diverse finalità come accennato prima, una rete IoT può servire per l'intrattenimento, per l'industria automatizzata, per il settore medico e altro ancora. I dispositivi dell'IoT possono essere ad esempio smartphone e sistemi di sicurezza abitativi connessi tra loro, elettrodomestici intelligenti che preparano il cibo recependo comandi a distanza, veicoli che si guidano in autonomia, indumenti tecnici sportivi che, quando si usurano, avvisano il proprietario. Si può altrimenti pensare a dispositivi che monitorano le condizioni di salute dei pazienti o il corretto funzionamento delle macchine utilizzate dai pazienti.

L'IoT consente quindi di portare a termine compiti che altrimenti sarebbero difficili da svolgere con la stessa efficienza e precisione. È importante prestare particolare attenzione

alle questioni relative alla privacy e alla sicurezza degli utenti utilizzatori. Sia la gestione della sicurezza che l'analisi dei dati dell'IoT, dotati di molte delle caratteristiche utili ad essere definiti Big Data, sono compiti esemplari da poter affidare alle applicazioni dell'Intelligenza Artificiale [17].

2 Big Data

Si hanno tracce dei primi utilizzi del termine Big Data (grandi dati) a partire dalla metà degli anni 1990 da parte di John Mashey in riferimento alla gestione e analisi di dataset massivi. Dal 2001 in poi sono stati identificati i tratti caratteristici e le particolarità che hanno contraddistinto gli insiemi di dati etichettati come Big Data dai tradizionali small data. Doug Laney ha identificato nel 2001 "le tre v" che descrivono tre particolarità dei set di dati.

La prima v è il volume, questa caratteristica indica la dimensione massiva dei set di dati. Un esempio di questo tipo di set di dati sono quelli generati dal monitoraggio delle attività degli utenti di un social network. Il volume dei big data potrebbe raggiungere dimensioni talmente elevate da rendere impossibile l'archiviazione del set di dati su un singolo computer di fascia consumer. Una delle strategie utilizzate per il salvataggio di dati talmente massicci è il cloud.

La seconda v è la velocity che fa riferimento alla velocità di creazione ed eventualmente alla tempestività di elaborazione. Un dispositivo intelligente potrebbe generare in real-time dati per mezzo dei propri sensori e richiederne l'immediata elaborazione per poter fornire un servizio che ha la necessità di dare risposte immediate relativamente all'ambiente che circonda l'utente.

La terza v è la variety, essa fa riferimento al tipo e organizzazione dei dati ricevuti. I set di dati tradizionali sono strutturati, quindi seguono uno schema ben preciso. I dati strutturati vengono memorizzati in tabelle in cui le righe e le colonne rispettano determinate relazioni precise.

Nei Big Data i dati possono essere testi, immagini, audio e altro ancora. Questi dati differiscono dagli insiemi di dati tradizionali e in alcuni casi necessitano di essere pre-processati prima di poter essere elaborati dalla macchina per gli scopi desiderati. I dati incontrati nei Big Data possono quindi essere strutturati, semi-strutturati o non strutturati. Per poter etichettare in maniera precisa un set di dati come appartenente ai Big Data è conveniente affiancare anche altri aspetti e caratteristiche oltre a quanto descritto da "le tre v".

Con il passare del tempo sono stati associati ai Big Data numerosi altri termini con la v e con la p capaci di descrivere alcuni altri aspetti. Alcuni di questi termini descrivono le qualità dei Big Data, come ad esempio versatility e predictive, mentre altri fanno riflettere su alcune problematiche connesse, come ad esempio visionary e privacy. Nel 2013, Rob Kitchin, basandosi sulla revisione delle definizioni dei Big Data, realizzò una tabella che mette in luce le differenze tra gli small data e i Big Data mostrata nella tabella 1.1.

Nel 2015 venne creata da Kitchin e McArdle una seconda tabella estensiva della prima. Oltre alle caratteristiche essenziali vengono considerati anche aspetti come la gestione, l'acquisizione e la lavorazione dei dati. Tali informazioni sono osservabili consultando la tabella 1.2.

Tabella 1.1: Tabella 1 relativa al confronto tra big e small data. Kitchin, 2013, versione 2[16].

	Small data	Big Data
Volume	Limited to large	Very large
Velocity	Slow, freeze-framed/ bundled	Fast, continuous
Variety	Limited to wide	Wide
Exhaustivity	Samples	Entire populations
Resolution and indexicality	Course and weak to tight and strong	Tight and strong
Relationality	Weak to strong	Strong
Extensionality and scalability	Low to middling	High

Tabella 1.2: Tabella 2 relativa al confronto tra survey, administrative e big data, 2015 [16].

	Survey data	Administrative data	Big Data
Specification	Statistical products specified ex-ante	Statistical products specified ex-post	Statistical products specified ex-post
Purpose	Designed for statistical purposes	Designed to deliver/monitor a service or program	Organic (not designed) or designed for other purposes
Byproducts	Lower potential for by-products	Higher potential for by-products	Higher potential for by-products
Methods	Classical statistical methods available	Classical statistical methods available, usually depending on the specific data	Classical statistical methods not always available
Structure	Structured	A certain level of data structure, depending on the objective of data collection	A certain level of data structure, depending on the source of information
Comparability	Weaker comparability between countries	Weaker comparability between countries	Potentially greater comparability between countries
Representativeness	Representativeness and coverage known by design	Representativeness and coverage often known	Representativeness and coverage difficult to assess
Bias	Not biased	Possibly biased	Unknown and possibly biased
Error	Typical types of errors (sampling and non-sampling errors)	Typical types of errors (non-sampling errors, e.g., missing data, reporting errors and outliers)	Both sampling and non-sampling errors (e.g., missing data, reporting errors and outliers) although possibly less frequently occurring, and new types of errors
Persistence	Persistent	Possibly less persistent	Less persistent
Volume	Manageable volume	Manageable volume	Huge volume
Timeliness	Slower	Potentially faster	Potentially much faster
Cost	Expensive	Inexpensive	Potentially inexpensive
Burden	High burden	No incremental burden	No incremental burden
Geography	National, defined	National or extent of program and service	National, international, potentially spatially uneven
Demographics	All or targeted	Service users or program recipients	Consumers who use a service, pass a sensor, contribute to a project, etc.
Intellectual Property	State	State	State/Private sector/ User-created rights.

Source: Florescu et al. (2014: 2–3) and Kitchin (2015)

Esiste un approccio differente per la definizione dei Big Data che consiste nel prendere in considerazione principalmente le difficoltà riscontrate nella lavorazione dei set di dati come ad esempio l'elaborazione e la memorizzazione su di una singola macchina. Murthy nel 2014 ha realizzato una tassonomia basata su sei punti per poter categorizzare i Big Data. La tassonomia invece di basarsi sulle caratteristiche essenziali possedute dai Big Data fa riferimento alle metodologie di gestione e lavorazione utilizzate:

1. Dati :
    - latenza temporale per l'analisi: real-time, near real-time, batch.
    - struttura: structured, semi-structured, unstructured
2. Infrastruttura di calcolo: batch o streaming.
3. Infrastruttura di archiviazione: SQL, NoSQL, NewSQL
4. Analisi: supervised, semi-supervised, unsupervised o re-enforcement machine learning; data mining; statistical techniques.
5. Visualizzazione: maps, abstract, interactive, real-time.
6. Privacy e sicurezza: data privacy, management, security.

Quello che risulta è che i set di dati etichettati come Big Data possono essere molto diversi tra loro anche se presentano in comune alcune delle caratteristiche descritte. Questo significa che le caratteristiche essenziali e le difficoltà di elaborazione di un set di Big Data possano essere molto diverse da quelle di un altro set di Big Data. Considerando i dati generati annualmente da un sensore di un dispositivo intelligente si potrebbe osservare un basso volume e una bassa variety anche se dispongono di elevata velocity ed exhaustivity. Per exhaustivity si intende che l'intero sistema viene rappresentato dal dataset anziché essere rappresentato per mezzo di campioni parzialmente esaustivi del sistema.

A seconda del tipo di rilevazione e dalla frequenza delle misurazioni, per esempio durante un periodo di osservazione di un anno, tutte le misurazioni del sensore potrebbero occupare qualche GB di spazio risultando più leggero di un set di dati tradizionale etichettato come small data. Il volume si considera come il numero di records associate allo spazio necessario per memorizzare i risultati osservati, si intuisce subito che dataset diversi hanno caratteristiche molto differenti. Un ipotetico dataset composto dall'acquisizione periodica di immagini dispone di meno records che però hanno peso maggiore e natura diversa rispetto ai records delle misurazioni di un sensore appartenente a un dispositivo intelligente che misura la temperatura. Per poter etichettare un set di dati come Big Data è necessario che disponga della maggior parte delle caratteristiche descritte nella tabella di Kitchin del 2013, specialmente per quanto riguarda la velocity e la exhaustivity. Il volume e la variety non sono caratteristiche qualificanti se non vengono associate alla velocity e all'exhaustivity[16].

3 Intelligenza Artificiale

L'intelligenza artificiale (I.A.) è una branca dell'informatica che studia i metodi capaci di consentire a una macchina di affrontare compiti specifici che per essere svolti richiedono l'impiego di caratteristiche tipiche dell'intelletto umano.

Sebbene per macchina si intende una creazione dell'uomo capace di compiere operazioni predeterminate e addetta a uno scopo [5], esistono delle distinzioni che sono state fatte da Alan Turing in riferimento all'idea di macchina intelligente [25]. Turing si era posto la domanda: "Le macchine possono pensare?", dopo diverse considerazioni ha reputato opportuno semplificare la domanda e le condizioni utili a valutare se una macchina fosse intelligente oppure no. Per Turing una macchina potrebbe essere considerata intelligente qualora prendendo parte, per almeno cinque minuti, a una sessione del gioco dell'imitazione fosse stata capace di ingannare un giocatore umano. Inoltre sembra che Turing avesse previsto l'eventuale clonazione umana, egli affermava che nel caso in cui fosse stato possibile creare artificialmente un umano, quest'ultimo, sarebbe stato automaticamente escluso dall'esperimento. Dunque, Turing, avrebbe accettato esclusivamente calcolatori digitali diversi dall'uomo per il suo test. Potrebbe sembrare strano e curioso paragonare l'essere umano a un calcolatore digitale, eppure esistono tracce scritte risalenti al 1613, attribuite a Richard Braithwaite [10], che mostrano come il termine computer facesse riferimento a persone, che seguendo rigide regole e direttive, svolgevano calcoli aritmetici. Le regole e le direttive che i computer dovevano eseguire alla lettera venivano fornite da altre persone dotate di una più approfondita conoscenza della matematica. Uno dei lavori dei computer era quello di contribuire alla creazione di tavole numeriche utilizzate per l'astronomia, la navigazione e l'ingegneria.

Un particolare esempio di macchina creata dall'uomo è il sillogismo di Aristotele che, seppur intangibile in quanto strumento della mente, consente di affrontare e portare a termine problemi tipici del ragionamento deduttivo. L'utilizzo del sillogismo prevede che i problemi vengano descritti e formalizzati secondo certe regole proprie della struttura di tale macchina.

La macchina a cui Turing faceva riferimento è il computer digitale che, durante il corso della storia, è stata soggetta a diverse implementazioni.

La Macchina Analitica di Charles Babbage [15] per esempio è un computer digitale programmabile che venne realizzato con ingranaggi e meccanismi, si tratta quindi di un artefatto meccanico.

Turing si riferiva alle implementazioni che rispettano le caratteristiche della macchina universale di Turing e che dispongono di un hardware adeguato, tali macchine riescono a calcolare tutto ciò che è calcolabile secondo la grammatica e le regole di Turing e dispongono della potenza computazionale necessaria a fronteggiare gli ostacoli dell'intelligenza artificiale.

Allo stato attuale quando si parla di computer digitale, molto probabilmente, si pensa al calcolatore elettronico, effettivamente il computer è la macchina attualmente utilizzata

per implementare le metodologie proprie dell'intelligenza artificiale.

La programmazione delle macchine attualmente a disposizione consente la creazione sistemi di I.A. che sono capaci di imitare alcune delle caratteristiche tipiche dell'intelligenza umana, per questo motivo tali sistemi sono in grado di portare a termine un compito assegnato mostrando un'efficienza che è almeno identica a quella individuabile nella prestazione di un umano specializzato nel compito in questione.

L'intelligenza artificiale è una realtà che è già ampiamente integrata nella vita di tutti i giorni. I dispositivi portatili sono in grado di riconoscere il viso dei rispettivi proprietari. Le applicazioni software riescono ad apprendere i comportamenti e le preferenze degli utenti. Le macchine sono in grado di riconoscere caratteristiche specifiche all'interno di un set di immagini fornendo un supporto indispensabile ai medici che devono fare diagnosi complesse.

Quanto descritto fino a ora fa luce sulla sempre esistita necessità umana di risolvere problemi e di dover effettuare calcoli che diventano, col tempo, sempre più dispendiosi e complessi. Per questi motivi l'informatica e le discipline interconnesse sono in continua evoluzione e alla costante ricerca di nuove tecnologie e metodologie utili ad affrontare le nuove sfide. L'intelligenza artificiale mediante i suoi diversi approcci riesce a risolvere, con ottimi risultati, i problemi accennati. I Big Data che caratterizzano questo periodo storico fungono sia da risorsa che da incentivo per lo sviluppo dell'Intelligenza Artificiale.

4 Machine Learning

Il machine learning (apprendimento automatico) è una disciplina dell'Intelligenza Artificiale che studia l'insieme di metodologie, ideologie e tecniche che consentono ad una macchina di attuare meccanismi di apprendimento e utilizzo della conoscenza necessaria allo svolgimento di un compito.

Il termine machine learning venne introdotto da Arthur Samuel nell'articolo in cui studiò due algoritmi di machine learning che avevano lo scopo di rendere la macchina più brava del programmatore nel gioco degli scacchi [21].

La macchina, per poter apprendere e beneficiare della conoscenza acquisita, deve utilizzare opportuni algoritmi di apprendimento che elaborano i dati di cui entra in possesso. Esistono diversi approcci al machine learning, ognuno di essi si occupa di un particolare dominio di problemi e per questo motivo le tecniche impiegate traggono vantaggio dalla conoscenza di numerose altre discipline tra cui per esempio la statistica, l'analisi dei dati e le neuroscienze.

Tom Mitchell ha dato la seguente definizione di programma che apprende:

> *"A computer program is said to learn from experience (E) with respect to some class of tasks (T) and performance measure (P), if its performance at tasks in T , as measured by P, improves with experience E." [Machine Learning - Tom M. Mitchell]*
>
> Traduzione: *"Un programma di computer si dice che apprende dall'esperienza (E) in relazione a una classe di task (T) e alla misure di performance (P), se le sue performance nello svolgere il task in T, come misurato da P, migliorano con l'esperienza E."*

L'utilizzo del machine learning si è ampiamente diffuso per poter far fronte a tutti quei problemi che per essere affrontati, in modo autonomo ed efficiente, richiedono la potenza e precisione di una macchina in combinazione alle capacità cognitive degli esseri senzienti [5]. La macchina diventa quindi in grado di imitare caratteristiche come per esempio la facoltà di prendere decisioni in base alla situazione in funzione dell'esperienza, la capacità di acquisire nuova conoscenza quando necessario, il poter effettuare riconoscimenti attraverso l'identificazione di caratteristiche distintive e il poter adottare comportamenti orientati al problem solving e al miglioramento dei risultati. In alcuni casi i problemi del machine learning sono caratterizzati da una massiccia mole di dati su cui la macchina deve, in qualche modo, trovare dei collegamenti o effettuare identificazioni che un operatore umano non riuscirebbe o che terminerebbe in un tempo troppo elevato. In altri casi il problema è sprovvisto di data set e la macchina deve capire come interagire con l'ambiente che lo circonda in modo da portare a termine il suo scopo in modo sicuro ed efficiente. Una macchina che riesce a riconoscere e raggruppare in base al soggetto rappresentato una massiccia mole di immagini o un robot che scansiona,

preleva, trasporta e deposita oggetti in un magazzino sono concretizzazioni degli esempi appena descritti.

Le caratteristiche degli algoritmi del machine learning vengono rese effettive, all'interno della macchina, mediante la programmazione di algoritmi che vengono definiti *soft coded*. Questi algoritmi conferiscono alla macchina la facoltà di decidere quali azioni, previste dal programmatore, sia meglio eseguire in base al contesto e all'esperienza acquisita.

Le scelte che inizialmente la macchina intraprende possono essere discutibili, per questo motivo il machine learning prevede una fase, detta training (addestramento), in cui la macchina acquisisce e affina la conoscenza di cui ha bisogno. In alcuni paradigmi del machine learning il training può essere identificato in un periodo di tempo preciso e limitato, in altri paradigmi invece il sistema apprende costantemente per tutta la sua intera esistenza. Ogni approccio del machine learning ha metodologie proprie di apprendimento che possono differire di molto l'una dall'altra.

Tra i vari settori in cui viene impiegato il machine learning troviamo l'intrattenimento, la finanza e la medicina. Un esempio di impiego nell'intrattenimento consiste nella profilazione e previsione dei comportamenti degli utenti di un'applicazione. Per quanto riguarda le applicazioni del machine learning nella finanza si può pensare alla previsione dell'andamento dei titoli azionari. Una delle applicazioni del machine learning in campo medico è il monitoraggio della quantità di radiazioni assunte dai pazienti sottoposti a radioterapia al fine di rendere il trattamento più sicuro [12].

Nelle sezioni seguenti verranno illustrati i tre paradigmi principali del machine learning che hanno dato origine a tutti gli altri approcci. Supervised e unsupervised learning verranno trattati in modo generale e non approfondito. Il reinforcement learning è il paradigma che verrà utilizzato nel progetto di tesi e quindi trattato in dettaglio.

4.1 Supervised Learning

Il supervised learning (apprendimento supervisionato) è uno dei paradigmi del machine learning. In questo approccio il supervisore fornisce alla macchina un training set di cui si conosce la descrizione della situazione e la sua label che specifica quale azione corretta dovrebbe intraprendere il sistema nella situazione in questione [13]. In sostanza il set di dati fornisce oltre agli input anche i relativi output corretti che la macchina dovrebbe generare. Grazie a queste informazioni l'algoritmo viene addestrato fino alla corretta esecuzione del suo compito. La conoscenza dell'azione giusta che il sistema avrebbe dovuto intraprendere consente all'algoritmo di apprendere dagli errori e di migliorare le elaborazioni.

Le tecniche del supervised learning forniscono alla macchina la capacità di acquisire la conoscenza necessaria a comportarsi correttamente in scenari che non sono stati previsti dall'insieme di dati utilizzato per l'addestramento.

Una delle tecniche utilizzate in questo paradigma è quella che prende il nome di rete neurale [27], l'architettura di questa tecnica si ispira alle reti neurali degli organismi biologici. Esaminando in modo generico una basilare rete neurale, utilizzata ad esempio per catalogare delle immagini, è possibile osservare la seguente composizione architetturale. La rete risulta composta da elementi chiamati neuroni. I neuroni vengono organizzati in strutture chiamate layers che raggruppano i neuroni in base alle loro funzionalità. I layers sono tre: l'input layer che si occupa della ricezione dei dati di input, l'hidden layer che svolge diverse computazioni e l'output layer che fornisce il risultato del compito della macchina. Ogni neurone è collegato ai neuroni dello strato successivo grazie a canali dotati di un peso che consiste in un valore numerico assegnato al canale. I pixel delle immagini che la rete neurale deve elaborare devono essere convertiti in valori numerici. Dopo la conversione ogni neurone dell'input layer prende in input un pixel dell'immagine convertita. Per ogni neurone esiste un pixel dell'immagine, ogni pixel viene associato ad un particolare neurone e quel neurone è associato esclusivamente a quel particolare pixel. I neuroni dispongono di una funzione che elabora le informazioni numeriche a disposizione. La funzione determina se il neurone deve attivarsi oppure no. Se un neurone si attiva allora trasmette il dato in avanti all'idoneo neurone successivo. La propagazione dell'informazione procede in avanti in questo modo fino all'output layer dove ad attivarsi sarà esclusivamente il neurone che rappresenta l'output dell'algoritmo. Nell'output layer c'è un numero di neuroni pari al numero di categorie in cui le immagini devono essere catalogate. Il neurone che si attiva rappresenta la categoria associata all'immagine elaborata. Se la categoria ottenuta come output dall'algoritmo differisce da quella reale allora è necessario intervenire sulla rete neurale. Mediante la propagazione all'indietro si aggiornano i parametri usati dalle funzioni dei neuroni in modo da poter ottenere il giusto output associato all'input esaminato. Una volta che la rete neurale è stata addestrata genera un modello che rappresenta il dominio del problema del set di dati di addestramento. Mediante l'utilizzo del modello la rete sarà in grado di svolgere

il suo compito in set di dati diversi da quello di addestramento che sono appartenenti al dominio del problema rappresentato dal modello. I tempi di addestramento della rete neurale dipendono dalla complessità del compito e dal tipo di dati che deve analizzare, può richiedere ad esempio alcuni minuti, alcune ore o diversi giorni. Reti neurali più sofisticate di quella descritta, come ad esempio le convolutional neural network (CNN) [8], vengono utilizzate per elaborare le immagini ottenute dagli esami clinici che supportano il lavoro dei medici. La rete neurale riesce ad identificare nelle immagini dei pazienti la presenza di caratteristiche tipiche di una determinata malattia aiutando il medico a formulare diagnosi complesse.

4.2 Unsupervised Learning

L'unsupervised learning (apprendimento non supervisionato) è un paradigma del machine learning [14]. Nell'unsupervised learning il dataset fornito alla macchina dispone della descrizione della situazione ma è sprovvisto della label che specifica la corretta azione che il sistema deve intraprendere in quella determinata situazione. In sostanza si conosce l'input ma si è sprovvisti del relativo output corretto. Le caratteristiche dell'unsupervised learning, a differenza del supervised learning, consentono un impiego del paradigma in contesti in cui i dati vengono acquisiti in real time. Lo scopo dell'unsupervised learning è quello di trovare pattern nascosti all'interno del set di dati elaborati. I dati presi in input sono tipicamente campioni indipendenti che appartengono ad una distribuzione di probabilità sconosciuta. L'identificazione dei pattern consente di trovare associazioni o anomalie nelle informazioni rappresentate dai dati. Un primo esempio di compito svolto dagli algoritmi di unsupervised learning è l'associazione. Per associazione si può pensare a una macchina che prende in input i dati delle performance delle vendite dei prodotti di un'azienda e restituisce, come output, informazioni che fanno luce sulle vendite complementari che potrebbero interessare ai clienti. Per esempio si potrebbe scoprire che un cliente che acquista il prodotto A, con forte probabilità, è incline all'acquisto del prodotto B e a questo punto potrebbe, probabilmente, essere interessato a comprare anche il prodotto C. Con queste informazioni si possono operare azioni mirate a migliorare le vendite, eventualmente si potrebbe agire esponendo i prodotti A, B e C insieme e creando un'opportuna offerta *pacchetto completo*.

Uno degli algoritmi utilizzati per portare a termine questo tipo di compito è l'Apriori algorithm [7]. L'algoritmo a priori utilizza regole di associazione che ad un oggetto antecedente associa un probabile oggetto conseguente. La struttura è del tipo "se adesso ho A, allora ho una certa probabilità di avere in seguito B".

Un altro esempio di compito svolto dagli algoritmi di unsupervised learning è il clustering [11]. Per clustering si intende il compito svolto da una macchina che identifica in autonomia pattern distintivi nascosti in un set di dati preso in input. Supponiamo che il set di dati rappresenti un insieme di frutti, la macchina diventa in grado di raggruppare tutti i frutti che dispongono di caratteristiche in comune. La macchina riesce a raggruppare tutti i frutti in base alla tipologia creando il gruppo dell'uva, delle zucche, delle castagne e così via. La macchina non sa esattamente che tipo di frutto sta raggruppando, non ne conosce il nome e nessuno le ha fornito indicazioni a riguardo. Il ragionamento alla base è simile a quello di un bambino piccolo. Immaginiamo che il bambino preso in considerazione viva in una famiglia con un cane. Quello che il bambino capisce è che vive con un animale a quattro zampe, peloso e che abbaia. Se il bambino dovesse incontrare un cane sconosciuto, di un'altra razza, lo assocerebbe direttamente a un qualcosa di simile all'animale con cui gioca solitamente a casa, cioè al suo cane. Se il bambino dovesse invece vedere un gatto potrebbe inizialmente pensare che si tratti di un cane ma sentendolo miagolare lo assocerebbe a un animale diverso dal suo cane.

4.3 Reinforcement Learning

Una delle prime volte in cui venne utilizzato il termine reinforcement learning, nella letteratura ingegneristica, risale al 1965 nell'articolo di Waltz e Fu in cui descrivevano un sistema di controllo che apprendeva da solo mediante l'utilizzo di tecniche di rinforzo [26]. Il reinforcement learning (apprendimento per rinforzo) è uno dei paradigmi del machine learning che più si avvicina all'apprendimento umano e animale. I primi contributi alla disciplina, che risalgono al 1952, sono da attribuire a Bellman in riferimento alle sue pubblicazioni relative agli studi della programmazione dinamica [9]. Una delle differenze sostanziali di questo paradigma rispetto alle tecniche di supervised e unsupervised learning è la metodologia con cui la macchina acquisisce i dati. La macchina acquisisce conoscenza attraverso l'interazione con l'ambiente seguendo una filosofia trial and error (prova ed errore) che premia le decisioni che portano al raggiungimento dello scopo desiderato.

Gli scenari tipici del reinforcement learning prevedono sequenze di decisioni che un sistema decisionale deve compiere per poter raggiungere uno scopo. Un esempio tipico è uno scenario in cui un robot deve imparare a svolgere in autonomia e in modo efficiente un lavoro che richiede capacità decisionali. Un robot magazziniere che preleva e deposita oggetti evitando collisioni con oggetti e umani è un esempio di scenario di reinforcement learning. I problemi del reinforcement learning possono essere rielaborati in modo da essere considerati scenari in cui la macchina interagisce con l'ambiente cercando di massimizzare il risultato che potrebbe ottenere prendendo le giuste decisioni durante lo svolgimento del compito assegnato. Secondo questo principio è possibile descrivere modelli accurati del problema utilizzando tecniche matematiche che verranno poi risolte utilizzando opportuni algoritmi. Per poter approfondire il reinforcement learning è necessario elencare e comprendere alcuni componenti fondamentali che sono caratteristici degli scenari in cui viene applicato il reinforcement learning:

- Agent: rappresenta l'agente che è il sistema decisionale destinato all'apprendimento di una linea di condotta che consente di svolgere in modo intelligente il lavoro di cui è stato incaricato.

- Environment: rappresenta l'ambiente che costituisce tutto ciò che differisce dal sistema decisionale. L'ambiente dispone di stati e rappresenta il mondo che ospita l'agent. L'agente interagisce con l'ambiente continuamente.

- Reward: rappresenta la ricompensa associata a determinate interazioni particolari dette azioni che l'agent compie nei confronti dell'ambiente.

- Time step: rappresenta l'unità di tempo passo temporale utilizzata per organizzare e determinare la progressione dei problemi affrontati.

Caratteristiche essenziali

L'essenza del reinforcement learning risiede nell'apprendimento da parte di un agente attraverso l'interazione con l'ambiente. L'agente ha la necessità di apprendere perché vuole raggiungere uno scopo, per farlo ha a disposizione un insieme di azioni che gli consentono di interagire costantemente con l'ambiente che inizialmente potrebbe essere del tutto sconosciuto. Uno dei principi che viene adottato nel reinforcement learning per poter mettere in atto tali processi di apprendimento è quello dell'associazione stimolo-risposta studiato in psicologia nella branca del comportamentismo. Diversi studiosi hanno condotto ricerche in questo campo, di seguito vengono riportate alcune delle nozioni che sono utili al reinforcement learning:

- Pavlov ha studiato gli stimoli condizionanti dimostrando che uno stimolo può indurre al compimento di una determinata azione.[20]
- Thorndike ha studiato il condizionamento operativo attraverso esperimenti che prevedevano l'apprendimento per trial and error. Thorndike ha identificando, tra le altre cose, l'importanza del rinforzo, ovvero quel premio che l'agent ottiene al raggiungimento del suo obiettivo dopo aver compiuto la giusta azione.[24]
- Skinner ha identificato che il rinforzo rappresenta la condizione necessaria e sufficiente affinché l'apprendimento abbia luogo e che l'assenza del rinforzo causa l'estinzione di un comportamento.[18]
- Premack ha dimostrato che l'apprendimento avviene a prescindere dal rinforzo. Quello che il rinforzo consente è il miglioramento delle prestazioni durante la fase di apprendimento e di conseguenza una maggior velocità di apprendimento in vista del raggiungimento di un obiettivo.[6]

I concetti introdotti possono essere illustrati attraverso un esempio che prende in considerazione un animale che ha bisogno di nutrirsi.

Esempio 1 *Un animale si trova in un territorio sconosciuto e pieno di ostacoli, il suo obiettivo è quello di cercare del cibo per potersi sfamare. L'animale, cercando di raggiungere il suo obiettivo, viene costantemente esposto a degli stimoli. Per avere successo deve rispondere agli stimoli percepiti compiendo le giuste azioni. Ad esempio possiamo immaginare che l'animale durante i suoi spostamenti sia in grado di avvistare un arbusto ricco di frutti invitanti che chiameremo* $cibo_1$*, esso però è situato oltre un ostacolo che per essere superato dall'animale richiede un notevole dispendio di energie fisiche. L'animale nota inoltre che vicino a sé c'è un'altra fonte di sostentamento chiamata* $cibo_2$ *che può raggiungere senza nessuno sforzo. L'animale decide di mangiare* $cibo_2$ *apprendendo che risulta essere per nulla gustosa e anche scarsamente nutriente. L'animale in seguito alle sue azioni ha sperimentato una sensazione spiacevole determinata dal cattivo sapore*

e dalla scarsa commestibilità di cibo$_2$. L'animale percepisce che il suo livello di fame è rimasto invariato e di conseguenza non ha ancora raggiunto il suo obiettivo. Decide a questo punto di affrontare l'ostacolo per poter raggiungere il cibo$_1$ che per istinto gli sembra più invitante. Mentre affronta l'ostacolo prova le spiacevoli sensazioni di affaticamento e stanchezza. L'animale si trova ora in prossimità di cibo$_1$, lo mangia e percepisce una sensazione piacevole determinata dal sapore gustoso e dal fatto che sente la fame attenuarsi.

Analizzando quello che è successo nell'esempio possiamo affermare dunque che l'animale, attraverso i suoi tentativi, si è avvicinato al raggiungimento del suo obiettivo perché ha scoperto cosa deve fare per combattere la fame, per potersi sfamare dovrà ripetere sequenze di azioni fino a quando sperimenterà la sazietà. L'animale ha ricevuto degli stimoli e ha preso delle decisioni, ogni volta che la decisione intrapresa ha causato una sensazione positiva ha guadagnato un rinforzo positivo e ogni volta che le sue decisioni hanno causato sensazioni spiacevoli ha ottenuto un rinforzo negativo. L'animale, che ha quindi fatto esperienza attraverso l'esplorazione, probabilmente ha capito per quale tipo di cibo conviene sforzarsi di più e per quale di meno. Magari con il passare del tempo imparerà quali punti del territorio sono più ricchi di cibo. A questo punto Se considerassimo i rinforzi positivi come punteggi numerici positivi e i rinforzi negativi come punteggi numerici negativi potremmo azzardare che l'animale abbia l'obiettivo di ottenere la maggior quantità di punti numerici positivi possibile. Questo obiettivo di massimizzazione può essere raggiunto cercando di apprendere il comportamento che gli garantisce il maggior numero di punteggi numerici positivi. Questo comportamento prevederà che l'animale intraprenda un numero strettamente necessario di azioni con punteggio numerico negativo in grado di garantire sul lungo periodo la possibilità di compiere azioni che generano punteggi numerici positivi. Se l'animale continuerà a vivere in questo territorio avrà molto spesso l'obiettivo di sfamarsi e quindi probabilmente con il passare del tempo imparerà, attraverso gli stessi meccanismi, a svolgere il suo lavoro in modo sempre più efficiente.

I sistemi di reinforcement learning diventano in grado di apprendere mediante le strategie studiate dalla psicologia e dalle neuroscienze avvalendosi di algoritmi e strumenti matematici che consentano l'implementazione su di un calcolatore dei principi illustrati. Proprio come l'animale dell'esempio, il calcolatore diventa in grado di svolgere il suo lavoro in autonomia cercando di imparare il comportamento migliore. Nel capitolo del Markov decision process verrà approfondito il modello matematico utilizzato dai metodi di reinforcement learning per rappresentare problemi simili a quelli descritti nell'esempio di questo capitolo. Quello che dovrebbe risultare chiaro è quindi che il reinforcement learning si occupa di problemi a circuito chiuso che dunque si ripetono per poter risolvere un problema più grande. I problemi a circuito chiuso sono caratterizzati da sequenze di decisioni che prevedono il compimento di azioni che producono degli effetti che possono manifestarsi sul lungo periodo.

Bilanciamento tra esplorazione e sfruttamento

Nel reinforcement learning si esclude l'eventualità in cui l'agent sia fornito di istruzioni preliminari riguardo a quali azioni sia giusto intraprendere quando l'ambiente si trova in un determinato stato. Quello che invece succede è che l'agent stesso mediante l'apprendimento passa da una situazione di incertezza ad una situazione migliore in cui conosce qualcosa in più dell'ambiente e del comportamento che è meglio adottare. I problemi che gli algoritmi di reinforcement learning affrontano sono di diversa natura. Risulta possibile identificare alcune categorie di problemi tipicamente studiati dalla disciplina:

- Problemi associativi: un'azione riceve una ricompensa che varia a seconda dallo stato in cui l'ambiente si trova.
- Problemi non associativi: un'azione riceve la stessa ricompensa a prescindere dallo stato in cui l'ambiente si trova.
- Stazionari: dopo ogni scelta si riceve una ricompensa numerica estratta da una distribuzione di probabilità stazionaria che dipende dall'azione selezionata. La distribuzione di probabilità non varia nel tempo.
- Non stazionari: il vero valore dell'azione cambia nel tempo.

Negli scenari reali del reinforcement learning capita molto spesso di dover affrontare problemi associativi. In letteratura si dice infatti che i problemi di reinforcement learning completi siano quei problemi che:

- Prevedono la presenza di un agente che apprende un comportamento mediante meccanismi di prova ed errore.
- Il comportamento appreso dall'agente consente di identificare l'azione migliore da compiere quando l'ambiente si trova in un determinato stato.
- Le azioni compiute dall'agente influenzano in parte le transizioni dell'ambiente da uno stato all'altro e di conseguenza hanno in parte effetto sulle ricompense.

Si considera che tutti i problemi di reinforcement learning sono organizzati in passi temporali che si susseguono e che sono univocamente identificabili, il simbolo usato per la rappresentazione di un generico passo temporale è t. I problemi di reinforcement learning completi verranno affrontati più avanti, in questo capitolo viene trattato il problema noto come "exploration and exploitation dilemma" o "conflict between exploration and exploitation" (dilemma/conflitto tra esplorazione e sfruttamento). Facendo riferimento all'Esempio 1 in cui un animale si muove in un ambiente sconosciuto e ricco di ostacoli per trovare del cibo, possiamo identificare dei criteri con cui vengono compiute le scelte dell'agent. Quando l'animale compie scelte che gli fanno conoscere qualcosa di nuovo riguardo l'ambiente in cui si trova, si dice che ha computo scelte di esplorazione. Quando

l'animale sceglie di compiere azioni che per certo gli garantiscono sensazioni piacevoli, allora sta effettuando scelte di sfruttamento. Affidarsi esclusivamente a una dei due tipi di scelta potrebbe molto probabilmente condurre al non raggiungimento dell'obiettivo dell'agent. Quello che serve è un equilibrio tra le due tipologie di azioni capace di consentire all'agent di apprendere e svolgere in modo intelligente il suo lavoro raggiungendone la conclusione. Per affrontare tale dilemma si fa riferimento ad un problema noto come il k-armed bandit (Slot machine a k-leve), questo tipico e particolare problema del reinforcement learning è non associativo, dunque dotato di un solo stato, e nella versione più semplice è anche stazionario. Il problema k-armed bandit consiste in una sequenza di decisioni che l'agente deve compiere scegliendo una tra le k azioni disponibili. L'approccio più semplice proposto dalla letteratura per affrontare questo dilemma consiste nello stimare la ricompensa attesa dal compimento di un'azione (estimate of the expected value) e nell'adozione di una strategia di selezione dell'azione da compiere. Questa metodologia prevede che ogni azione disponga di un vero valore atteso $q_*(a)$ e di un valore atteso stimato $Q_t(a)$. Un metodo semplice per stimare il valore atteso di un'azione è detto sample-avarage method (metodo della media del campione):

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}} \tag{1.1}$$

Questo metodo prevede che per ogni azione a intrapresa dall'agente si tenga traccia delle ricompense ottenute. In questo modo è semplice calcolare la media delle ricompense ottenute. Questo valore $Q_t(a)$ fornisce una stima del valore ottenibile dal compimento della relativa azione a. La stessa formula può essere ottimizzata per rendere più efficienti le computazioni svolte con il calcolatore mediante l'adozione della seguente formula aggiornata:

$$Q_{n+1} \quad = \quad Q_n + \frac{1}{n}\Big[R_n - Q_n\Big] \tag{1.2}$$

Ora che si dispone di una stima del valore atteso dal compimento di un'azione si è finalmente in possesso di una base su cui compiere delle scelte. Ogni azione è adesso dotata del relativo valore atteso, almeno una delle azioni dispone di valore atteso maggiore o uguale al valore atteso di tutte le altre azioni, le azioni con il valore atteso maggiore si chiamano greedy-action (azione avida). Una strategia di selezione potrebbe essere quindi quella mostrata dalla formula che segue:

$$A_t \doteq \arg\max_a Q_t(a) \tag{1.3}$$

L'agent, che segue quello che viene formalizzato dalla formula, decide a ogni turno l'azione con il valore atteso più alto. Questo comportamento si dice essere greedy (avido). Le scelte fatte in modo greedy sono le scelte dette di sfruttamento. Potrebbe capitare che più azioni dispongano della stessa $Q_t(a)$, in questo caso si applica un criterio arbitrario per selezionare una tra tutte le greedy actions a disposizione. Una delle strategie arbitrarie da poter adottare in questo caso è la scelta di una delle azioni in modo casuale. Le scelte greedy sono buone per ottenere elevate ricompense nell'immediato, precludono però la massimizzazione della somma delle ricompense sul lungo periodo. Quando in letteratura si considerano problemi non stazionari, in cui le ricompense di un'azione cambiano nel tempo, si fa utilizzo di uno step size come indicato da α.

$$Q_{n+1} \quad = \quad (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i \tag{1.4}$$

Lo step size costante α utilizzato nella formula serve ad attribuire un peso maggiore alle ricompense ottenute più di recente. In alcuni casi è conveniente che lo step size α sia variabile e ridotto durante il processo di apprendimento dell'agent.

Come detto precedentemente, le scelte greedy non garantiscono sul lungo periodo la massimizzazione della somma delle ricompense, per questo motivo necessitano di un miglioramento. Un approccio di miglioramento utilizzato consiste nell'introduzione all'interno delle strategie greedy di una probabilità $1-\epsilon$ in cui l'agent compie scelte in modo casuale. Approcci del genere che si comportano per la maggior parte del tempo in modo greedy e ogni tanto in modo casuale prendono il nome di approcci $\epsilon - greedy$. Le scelte compiute in modo casuale sono scelte di esplorazione che permettono all'agente di acquisire conoscenza sull'ambiente. Grazie all'esplorazione l'agente diventa in grado di identificare il comportamento che sul lungo periodo consente di massimizzare la somma delle ricompense totali. In letteratura il valore ϵ può essere progressivamente ridotto durante lo svolgimento del lavoro, in questo modo vengono garantiti: un'esplorazione iniziale, la focalizzazione sulle ricompense nei passi successivi e un comportamento complessivo che evita di abbandonare l'esplorazione. Quando il problema è non stazionario le strategie $\epsilon - greedy$ si comportano meglio delle strategie puramente greedy, questo perché il problema ha per natura bisogno di esplorazione continua per verificare se le azioni un tempo da evitare siano ora diventate ottimizzanti. In problemi dove la ricompensa non varia, e non ci sono cambi di stato, conviene utilizzare il metodo greedy perché una volta provata un'azione si conosce il suo valore reale e si può continuare scegliendo sempre la migliore. Nei problemi con ricompense variabili si ha che il metodo $\epsilon - greedy$ faccia convergere $Q_t(a)$ a $q_*(t)$ se il problema dispone di infiniti passi temporali.

Un trucco utilizzato per incentivare l'esplorazione iniziale prende il nome optimistic initial values (valori iniziali ottimistici). Questo approccio fornisce all'agente una stima iniziale, del valore atteso da un'azione, che in realtà è eccessivamente alta. L'agente che

agisce in modo greedy sceglie queste azioni apparentemente profittevoli, una volta provate si accorge che il valore atteso inizialmente fornito era sovrastimato e, di conseguenza, decide di ridimensionarla. L'agent in questo modo prova tutte le azioni migliorando la sua conoscenza iniziale del problema. Gli effetti di questo trucco però si esauriscono una volta che l'agente ha provato almeno una volta tutte le azioni.

Seppur la letteratura dimostri che l'approccio $\epsilon - greedy$ sia migliore dell'approccio greedy in problemi non stazionari, resta il fatto che le scelte casuali potrebbero comunque ricadere su azioni greedy. A prescindere dalla stazionarietà o non stazonarietà del problema si ha sempre a che fare con il fatto che una stima potrebbe non coincidere con il valore reale. Per poter avvicinare la stima al valore reale è necessario adottare un comportamento orientato all'esplorazione continua. Il metodo Upper-Confidence-Bound (UCB) fornisce un meccanismo di selezione delle azioni a favore di quelle azioni che potrebbero potenzialmente essere ottimali, fornisce inoltre una soluzione al rischio di selezione casuale di un'azione greedy. La formula del meccanismo UCB è la seguente:

$$A_t \doteq \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right] \tag{1.5}$$

$N_t(a)$ è il numero di volte che l'azione a è stata scelta prima dell'attuale passo temporale t, se $N_t(a)$ dovesse essere 0 allora l'azione viene considerata massimizzante e quindi verrebbe selezionata. $c > 0$ controlla il grado di esplorazione. Tutto il termine sotto radice è una misura di incertezza e varianza della stima del valore atteso dell'azione a. In sostanza il termine sotto radice che viene sommato a $Q_t(a)$ (stima del valore dell'azione) rappresenta un limite superiore al quale la stima del valore dell'azione potrebbe tendere, proprio per questo motivo si dice che con questo metodo si effettuano scelte basate sulla potenziale ottimalità dell'azione a. Ogni volta che l'azione a viene selezionata, l'incertezza dell'azione a viene ridotta. Ogni volta che un'azione diversa da a viene selezionata si ha la crescita dell'incertezza di a. In questo approccio le azioni scelte più frequentemente e quelle con stime inferiori vengono, progressivamente, scelte meno di frequente. Questo approccio di selezione risulta di difficile estensione ai problemi non stazionari e con insieme degli stati composto da elevato numero di elementi. Si tratta di un metodo di selezione che ha performance migliori di $\epsilon - greedy$.

In letteratura viene inoltre proposto un metodo che invece di stimare il valore atteso dell'azione si occupa della stima di un valore $H_t(a)$ che indica la preferenza dell'azione a rispetto a un'altra azione. La frequenza di selezione di un'azione aumenta all'aumentare di $H_t(a)$. Il valore di preferenza non ha nulla a che vedere con il valore atteso dell'azione, quindi bisogna evitare di pensare che un'azione con elevato $H_t(a)$ abbia sempre un elevato

$Q_t(a)$. La formula per stimare la preferenza di un'azione è la seguente:

$$\Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a) \tag{1.6}$$

$\pi_t(a)$ è la probabilità di selezionare l'azione a quando ci si trova al passo temporale t. Inizialmente i valori di preferenza delle azioni vengono assegnati arbitrariamente, dunque per ogni azione $H_1(a)$ potrebbe valere ad esempio 0. Ogni volta che viene selezionata un'azione accade che il relativo valore di preferenza viene aggiornato secondo la formula che segue:

$$\begin{aligned} H_{t+1}(A_t) &\doteq H_t(A_t) + \alpha\big(R_t - \bar{R}_t\big)\big(1 - \pi_t(A_t)\big), && \text{and} \\ H_{t+1}(a) &\doteq H_t(a) - \alpha\big(R_t - \bar{R}_t\big)\pi_t(a), && \text{for all } a \neq A_t \end{aligned} \tag{1.7}$$

Ci sono ulteriori approfondimenti relativi a questo approccio che vengono trattati in letteratura.[23] Come detto in precedenza i problemi di reinforcement learning prevedono che gli agenti imparino delle politiche o sfruttino dei metodi capaci di far fronte alla mutevolezza delle ricompense e degli stati dell'ambiente.

Ora che il dilemma è stato formalizzato è possibile esprimerne l'importanza ed i concetti fondamentali facendo riferimento all'Esempio 1. Immaginiamo che l'animale si trovi di fronte a uno scenario del tutto nuovo, ha davanti a sé $cibo_1$ e $cibo_2$ che conosce bene e poi ha anche $cibo_3$ che non conosce affatto. L'animale che vuole massimizzare la somma delle ricompense totali dovrebbe esplorare decidendo di mangiare $cibo_3$, in questo modo capirebbe quanta ricompensa $cibo_3$ sia in grado di fornire. In effetti $cibo_3$ è quello migliore, se l'animale avesse deciso di andare sul sicuro mangiando $cibo_1$ avrebbe compiuto una scelta non ottimale. Quindi i principi di psicologia e le formulazioni matematiche concordano nel fatto che un agente che vuole massimizzare la somma delle ricompense deve evitare l'estinzione di alcuni comportamenti, deve dunque mantenere un certo grado di esplorazione. Per evitare l'estinzione di comportamenti è necessario trovare il giusto bilanciamento tra lo sfruttamento e l'esplorazione per evitare di svolgere il proprio lavoro in modo fallimentare.

Markov Decision Process

Il Markov Decision Process (MDP) è un modello matematico utilizzato per descrivere e risolvere i problemi che prevedono sequenze decisionali finalizzate all'ottimizzazione di un risultato. Il modello è caratterizzato da tre elementi fondamentali: l'environment (ambiente), l'agent (agente decisionale) e la reward (ricompensa).

L'environment è tutto ciò che si trova al di fuori dell'agent ed è caratterizzato da stati che l'agent è in grado di percepire. L'insieme di tutti gli stati di un environment viene indicato con il simbolo S.

L'agent è un sistema decisionale dotato di un insieme di azioni A di cui si avvale per consentire al sistema più ampio di cui fa parte di svolgere in modo intelligente il lavoro per la quale quest'ultimo è stato progettato. Ne risulta che il compito dell'agent è quello di decidere la sequenza migliore di azioni da intraprendere al fine di raggiungere il suo obiettivo di massimizzazione. L'agent interagisce continuamente con l'environment e decide quale azione intraprendere tenendo in considerazione lo stato in cui l'ambiente si trova, intraprendendo le azioni è in parte in grado di influenzare lo stato dell'ambiente avendo una certa probabilità di portarlo da uno stato S_1 ad un altro stato S_2.

Quando l'agent intraprende un'azione ottiene una reward calcolata dall'environment attraverso una funzione arbitraria che prende in input lo stato in cui l'agent ha effettuato la decisione e l'azione che è stata intrapresa.

Quindi lo scopo dell'agent è quello di massimizzare sul lungo periodo la somma della sequenza di ricompense ottenute.

Gli MDP sono scanditi da passi temporali discreti indicati con la lettera t che partono da 0 e vengono incrementati in modo unitario. Ad ogni passo temporale l'environment si trova in un determinato stato S_t e l'agent deve decidere quale azione A_t sia meglio intraprendere per raggiungere il suo scopo. Dopo che l'agent intraprende l'azione accade che il passo temporale t viene incrementato di una unità diventando $t+1$, a questo punto l'agent ottiene una ricompensa R_{t+1} relativa allo stato del passo temporale precedente e all'azione che ha intrapreso.

Quello che si ottiene è detta traiettoria che consiste in una sequenza di mattoncini costituiti da: uno stato di partenza S_t, un'azione intrapresa A_t, uno stato raggiunto S_{t+1} e una ricompensa ottenuta R_{t+1}.

La somma di tutte le reward che l'agent ha ottenuto attraverso una traiettoria viene chiamata return e viene indicata con la lettera G, l'agent massimizza la return attraverso la miglior traiettoria possibile che per l'appunto restituisce sul lungo periodo una return con il massimo valore possibile.

Nel reinforcement learning si assume che gli stati degli MDP siano dotati della *Markov property*, questa proprietà prevede che lo stato disponga delle informazioni necessarie a identificare le interazioni agent-environment significative anche senza un sistema che tenga traccia della storia di ogni singolo avvenimento.

Come detto in precedenza, l'agent deve decidere quali azioni deve compiere un sistema più grande per poter svolgere il suo lavoro. Alla base del problema si ha quindi un sistema che deve svolgere un lavoro e che per farlo deve completare dei task che consistono in una sequenza di azioni intraprese dopo esser state opportunamente decise e che portano ad un obiettivo concreto. Molto probabilmente i task da completare sono ripetitivi, hanno un inizio e in alcuni casi una fine ben identificabile, in altri casi non terminano mai o non è semplice identificarne la fine naturale. Questi task sono la porzione del problema che viene modellata mediante MDP, per questo motivo un MDP può avere un numero di passi temporali t finito o infinito. Nella trattazione di questa tesi vengono presi in considerazione esclusivamente gli MDP che dispongono di un numero finito di passi temporali t e che quindi prevedono una fine naturale e ben identificabile rappresentata con il passo temporale terminale T. Gli MDP di questo tipo si chiamano MDP finiti e sono caratterizzati da insiemi A, S e R contenenti un finito numero di elementi. I problemi di reinforcement learning che rispettano tutte le caratteristiche indicate fino ad ora sono detti task episodici e vengono rappresentati per mezzo degli MDP finiti. I task episodici sono dotati sia dell'insieme S che dell'insieme $S+$, il primo racchiude tutti gli stati non terminali assunti dall'ambiente nei passi temporali t non terminali, il secondo oltre a tutti gli elementi dell'insieme S contiene anche lo speciale stato terminale assunto dall'ambiente nel passo terminale T dell'episodio. Quando l'environmnt raggiunge lo stato terminale l'episodio finisce. L'episodio ricomincia a prescindere dal motivo che ne ha causato la fine, lo stato iniziale assunto dall'environment viene estratto secondo una determinata probabilità da uno degli stati iniziali previsti dall'insieme S. Il susseguirsi di episodi continua fintanto che le condizioni del problema esaminato lo consentono e il sistema ha ancora del lavoro da svolgere. Si considera quindi che gli episodi terminino tutti nello stesso stato terminale che l'environment può assumere in diversi modi e per diversi motivi.

Come accennato in precedenza negli MDP si utilizza il concetto di state action pair (coppia stato-azione) che rappresenta la situazione in cui l'agent decide di intraprendere un'azione quando l'ambiente si trova in un determinato stato, questo concetto viene espresso con la seguente scrittura (S_t, A_t). La state action pair costituisce l'input della funzione che genera la reward che l'agent ottiene dopo aver intrapreso l'azione decisa. Il sistema di ricompense è fondamentale perché deve incentivare l'agent nel portare a termine con successo il task in modo da poter consentire al sistema di svolgere il proprio lavoro in modo intelligente. Risulta fondamentale evitare di utilizzare la ricompensa come mezzo per dire esplicitamente all'agent come vogliamo che affronti il suo task. Il sistema di ricompensa serve per dire all'egent cosa vogliamo che faccia e non come vogliamo che lo faccia. Ad esempio un robot che deve scappare da un labirinto potrebbe ricevere ricompense pari a 0 nel caso in cui intraprendesse un'azione che lo farebbe spostare su di una casella del percorso all'interno del labirinto, potrebbe invece ricevere ricompensa negativa qualora scegliesse un'azione che lo farebbe sbattere contro un muro e, infine, potrebbe ricevere una ricompensa positiva nel caso in cui intraprendesse un'azione che

lo condurrebbe su di una casella del percorso al di fuori del labirinto. La ricompensa positiva serve a dire cosa vogliamo che l'agent faccia e cioè scappare dal labirinto, la ricompensa negativa serve a dire all'agent che non deve far sbattere il robot contro ai muri.

Nel caso di task episodici è possibile evitare distinzioni tra un episodio e l'altro perché la modellazione mediante MDP fa in modo che le caratteristiche descritte siano vere per tutti gli episodi, questo aspetto fornisce un vantaggio alla notazione utilizzata per rappresentare gli episodi. In teoria quando si costruisce una traiettoria oltre ad indicare l'elemento dell'insieme sarebbe previsto indicare anche il numero progressivo che identifica l'episodio considerato. Il fatto che non sia necessario fare distinzioni tra un episodio e l'altro, per via di quanto appena detto, consente l'utilizzo di un abuso di notazione. Questa notazione facilita la rappresentazione delle traiettorie permettendo di omettere l'indice numerico progressivo utilizzato per identificare gli episodi.

A questo punto è possibile rappresentare una traiettoria con la seguente scrittura: $S_0, A_0, R_1, S_1, A_1, R_2, ...S_T$. Se fosse stato necessario indicare anche l'indice dell'episodio, facendo riferimento ad un generico episodio con indice i, sarebbe stato necessario utilizzare la seguente scrittura: $S_{0,i}, A_{0,i}, R_{1,i}, S_{1,i}, A_{1,i}, R_{2,i}, ...S_{T,i}$. La Figura 1.2 riassume tutti i concetti visti fino ad ora.

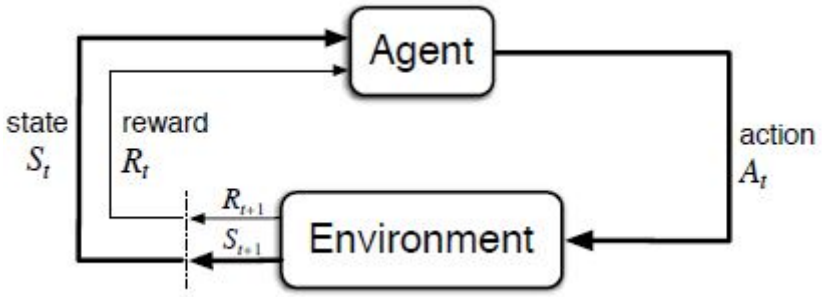


Figura 1.2: Schema di interazione agente-ambiente. [23]

Le azioni che un agente compie devono essere decise secondo un certo criterio, serve quindi un comportamento che l'agent deve adottare durante la sperimentazione del suo task. Questa linea di condotta prende il nome di policy. Più nel dettaglio, per policy si intende la mappatura della probabilità di scegliere una determinata azione quando l'ambiente si trova in un determinato stato. Se l'agent sta seguendo la policy π allora la scrittura $\pi(a|s)$ indica, considerato il passo temporale t, la probabilità che l'azione $A_t = a$ se lo stato assunto dall'ambiente $S_t = s$. In sostanza è una mappatura della probabilità di scegliere una determinata azione in relazione a un determinato stato. La policy dunque descrive e rende effettivo il comportamento di un agent nelle implementazioni di reinforcement learning.

Negli algoritmi utilizzati per risolvere i problemi tipici del reinforcement learning vengono stimate e utilizzate le value fanction: $v_\pi(s)$ nota come *state value function* e $q_\pi(s,a)$ nota come *action-value function*. La state value function mostrata in (1.8) calcola il valore di uno stato quando l'agent segue una determinata policy. In sostanza indica quale return un agent può attendersi se partendo dallo stato s si comporta come descritto dalla policy π seguita.

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t{=}s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t \models s\right], \text{ for all } s \in \mathcal{S} \tag{1.8}$$

La action-value function mostrata in (1.9) restituisce il valore della coppia stato-azione. In sostanza indica quale return l'agent può attendersi se intraprende l'azione a quando l'environment assume lo stato s e continua a seguire il comportamento descritto dalla policy π di cui dispone.

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t{=}s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t{=}s, A_t{=}a\right] \tag{1.9}$$

La value function indica all'agent quanto sia profittevole in termini di reward future condurre l'ambiente dallo stato attuale verso un determinato stato intraprendendo un'opportuna azione. Dunque in base al tipo di value function viene indicato quanto sia vantaggioso trovarsi in uno stato o compiere una determinata azione in un determinato stato per poter raggiungere una determinata return. La funzione valore utilizzata dall'agent viene stimata attraverso l'esperienza ottenuta attraverso gli episodi.

A questo punto sono stati espressi tutti i concetti fondamentali necessari ai metodi di reinforcement learning. L'idea che sorge spontanea è che potrebbe esserci qualche metodologia che, una volta adottata, consente di modificare il comportamento dell'agent nel tempo al fine di raggiungere il suo obiettivo di massimizzazione. Esistono metodi del reinforcement learning che stimano ed aggiornano la value function e successivamente effettuano miglioramenti alla policy. Mediante questo procedimento l'agent conosce sempre di più l'ambiente del problema affrontato e capisce quali azioni è meglio intraprendere.

In reinforcement learning gli agent apprendono sperimentando numerose volte gli episodi e avvalendosi degli algoritmi tipici della disciplina. L'agent stima il valore di uno stato mediante la state value function. Se l'agent segue la policy π, contemporaneamente sperimenta infinite volte uno stato e tiene, altresì, traccia della media delle return che quello stato gli ha fornito, allora sarebbe in grado di stimare l'effettivo valore di quello stato. Se invece un agent segue la policy π, tiene traccia delle return ottenute per ogni coppia stato-azione e, in contemporanea, sperimenta infinite volte lo stato e l'azione in questione, allora è in grado di stimare il vero valore della coppia presa in considerazione. Dunque in uno scenario di reinforcement learning risolvere un task significa trovare quella

policy capace di consentire all'agent di massimizzare sul lungo periodo la return. Esiste sempre una policy ottimale perché per esserlo deve avere la return attesa maggiore o uguale a tutte le altre policy possibili. Le policy ottimali vengono indicate dalla scrittura π_* e condividono la stessa value function ottimale, sia essa v_* o q_*. L'idea che viene naturalmente alla mente è che potrebbe essere utile trovare un meccanismo in cui vengano aggiornate sia le value function che le policy in modo da poter scegliere le azioni in base a un valore che tende a quello reale e seguendo comportamenti che garantiscono il raggiungimento dell'obiettivo.

Considerando gli scenari di reinforcement learning osservabili in pratica si può osservare che raramente un agent apprenda value functions e policy ottimali, infatti ancora oggi i computer non riescono a calcolare tutte le mosse ottimali da effettuare nei vari scenari che si presentano durante una partita del gioco degli scacchi. Quindi negli scenari reali si fa utilizzo di un'approssimazione che consente di raggiungere risultati efficienti e comunque migliori di quelli che si otterrebbero facendo svolgere il lavoro a un operatore umano. Il reinforcement learning rende possibile approssimare la policy ottimale permettendo all'agent di determinare la miglior sequenza di azioni in riferimento agli stati che l'ambiente assume più frequentemente.

Programmazione Dinamica

L'insieme degli algoritmi di Programmazione Dinamica (PD) vengono utilizzati per calcolare una policy ottimale mediante diverse metodologie. Per poter calcolare la policy ottimale attraverso la PD è necessario disporre di un modello perfetto dell'ambiente rappresentato mediante Markov Decision Process. Una delle strategie per calcolare la policy ottimale avviene in due parti, attraverso la policy evaluation (valutazione della linea di condotta) e attraverso la policy improvement (miglioramento della linea di condotta). Per policy evaluation si intende il processo di stima del valore atteso di un determinato stato quando l'agent segue una determinata policy.

$$v_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid s, \pi\right] \tag{1.10}$$

Si considerano k passi, il primo passo è il passo 0 in cui si inizializza la prima stima della funzione valore v_0 con il valore 0 associato a tutti i valori attesi di ogni stato della funzione valore. A questo punto si prosegue incrementando i passi, si passa quindi da una generica funzione valore v_k a una generica funzione valore v_{k+1}. Per ogni v_k si effettua un'iterazione per ogni stato della funzione valore e si utilizza la formula che segue

$$\forall s: \quad v_{k+1}(s) = \mathbb{E}\left[R_{t+1} + \gamma v_k(S_{t+1}) \mid s, \pi\right] \tag{1.11}$$

Durante l'iterazione ci si basa su delle stime precedenti per fare delle nuove stime, questo procedimento viene definito *fare bootstrap*. Se, dopo l'esecuzione della regola di aggiornamento appena descritta, accade che $v_{k+1} = v_k$ significa per definizione che sono stati trovati i veri valori degli stati. Se il fattore di sconto γ è minore di 1 e si considerano passi infiniti allora si ha sempre la convergenza della stima al valore reale. Se il discount factor è grande si intende che il problema trattato è più difficile da risolvere rispetto ad altri con discount factor inferiore, di conseguenza si ha bisogno di guardare molto in là nel futuro e la convergenza ai valori reali diventa più lenta. In caso di task episodici dove i passi temporali sono limitati è ancora possibile che si verifichi la convergenza ma è generalmente più difficile.

La policy improvement consiste nell'intervenire sulla policy in modo da cambiarla per renderla ottimale. Si utilizza un'iterazione che prende in considerazione una nuova policy ottenuta associando allo stato s la greedy action che è quella con valore $q_\pi(s, a)$ maggiore. L'iterazione avviene utilizzando la seguente formula

$$\forall s: \quad \pi_{\text{new}}(s) = \operatorname*{argmax}_a q_\pi(s, a) = \operatorname*{argmax}_a \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a\right] \tag{1.12}$$

Se accade che a un certo punto dell'applicazione ripetuta della formula si ottiene $v_{\pi_{new}}(s) = v_\pi(s)$ per tutti gli stati s, significa che la nuova policy ottenuta risulta essere migliore della vecchia oppure ottimale.

Negli esempi proposti in letteratura si utilizzano iterazioni dette policy iteration che coinvolgono alternativamente sia la policy evaluation che la policy improvement, lo scopo è quello di convergere progressivamente verso la funzione valore e la linea di condotta ottimali.

In riferimento alla Figura 1.3, le frecce verso l'alto mostrano la necessità di ridefinire la funzione valore, le frecce verso il basso indicano la necessità di defnire una nuova policy in funzione della nuova funzione valore.

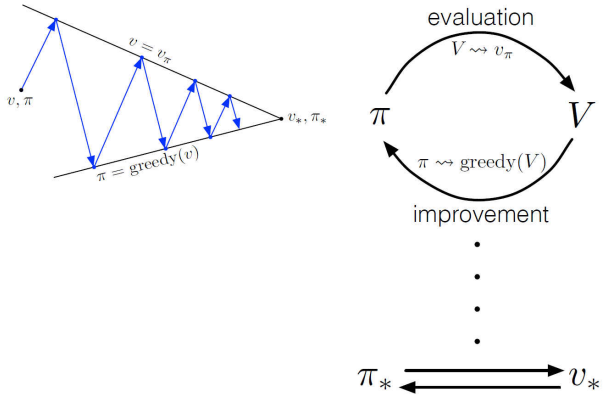


Figura 1.3: Immagine raffigurante le dinamiche della policy iteration [23]

Il principio della policy iteration è fondamentale per gli algoritmi che si occupano di problemi completi di reinforcement learning. Dal momento che nelle applicazioni reali è possibile che non si raggiunga la policy ottimale si può, per evitare di effettuare computazioni dispendiose inutilmente, arrestare il processo iterativo quando la policy inizia a migliorare in modo trascurabile. Per ottenere l'interruzione preventiva della policy iteration si possono utilizzare due metodi: presa in considerazione di un valore di soglia che misura la quantità di miglioramento della policy o l'adozione di un'interruzione forzata dopo un numero preciso di iterazioni.

Uno dei metodi utilizzato in PD per l'ottimizzazione della policy è la value iteration. Questa metodologia segue la logica iterativa della policy evaluation ma utilizza la Bellman optimality equation come regola di aggiornamento dalla funzione valore $v_k(s)$ alla funzione valore $v_{k+1}(s)$:

$$\forall s: \quad v_{k+1}(s) = \mathbb{E}\left[R_{t+1} + \gamma v_k(S_{t+1}) \mid s, \pi\right] \tag{1.13}$$

Come visto in precedenza quando questa iterazione della regola di aggiornamento smette di produrre miglioramenti significa che si è arrivati alla scoperta dei valori ottimali. Quello che la formula mostra è equivalente a quello fatto per la policy iteration con la differenza che qui si effettua un passo di value estimation e poi si effettua lo step di greedify e cioè miglioramento della policy. Come negli altri casi la convergenza all'ottimalità

potrebbe richiedere illimitati passi iterativi e nelle applicazioni reali si possono utilizzare efficacemente approssimazioni dell'ottimalità.

Tutti i metodi visti fino ad ora vengono detti synchronous Dynamic Programming algorithms, la loro caratteristica distintiva è che vengono effettuati aggiornamenti sincroni che coinvolgono tutti gli stati dell'ambiente.

Gli Asynchronous Dynamic Programming Algorithms hanno la particolarità di aggiornare gli stati uno alla volta, questo consente di ridurre le computazioni necessarie per gli aggiornamenti. In questi metodi la convergenza alla policy ottimale avviene se ogni stato viene selezionato ed aggiornato un numero di volte che tende ad infinito. Questo approccio si contraddistingue grazie a tre idee:

- Utilizzo di algoritmi in place: l'aggiornamento dei valori avviene con la sostituzione dei vecchi valori con quelli nuovi.

- Prioritized sweeping: l'aggiornamento viene effettuato sugli stati che risultano aver ricevuto la stima con maggior errore rispetto al valore aggiornato. Se l'aggiornamento di uno stato ha prodotto una differenza elevata tra il vecchio valore e quello aggiornato significa che, relativamente a quello stato, c'è molto da imparare. Se uno stato non mostra segni di aggiornamento allora possiamo supporre che la stima sia sufficientemente accurata. In questo modo si stabilisce la priorità di aggiornamento degli stati.

- Real-time dynamic programming: secondo questa idea viene data la priorità agli stati dell'ambiente che sono vicini. Questo principio come il prioritized sweeping serve a ridurre le computazioni perché metodi simili vengono adottati in contesti in cui l'agente ha poco tempo per "pensare".

In letteratura si afferma che, ignorando alcuni dettagli tecnici, si ha che nel caso peggiore il tempo impiegato dai metodi di PD per trovare una linea di condotta ottimale è polinomiale. I metodi di PD si dice che soffrano di curse of dimensionality (maledizione della dimensionalità). Secondo tale "maledizione" la dimensione dell'insieme degli spazi di un problema cresce di una quantità che corrisponde al prodotto del numero di valori assunti dalle variabili da cui dipende la determinazione degli stati dell'ambiente. Anche un elevato numero di azioni può contribuire insieme agli stati ad influenzare negativamente la dimensione del problema. Risulta quindi che i metodi di PD, in combinazione con la potenza di calcolo dei calcolatori odierni, sono tutto sommato buoni per affrontare e risolvere MDP con milioni di stati. Negli scenari reali, solitamente, questi metodi convergono più velocemente degli scenari teorici che illustrano il caso pessimo. In problemi dove l'insieme degli stati è dotato di numerosi elementi è preferibile utilizzare Asynchronous Dynamic Programming Methods perché i Synchronous Dynamic Programming Methods richiedono potenza di calcolo e memoria elevate a causa dei numerosi stati che caratterizzano il problema.[23]

Monte Carlo

I metodi Monte Carlo (MC) vengono applicati in contesti in cui si è sprovvisti di completa conoscenza dell'ambiente. Ai metodi MC è sufficiente affidarsi all'esperienza. Ciò significa che gli agent di MC, mediante esperienza diretta o simulata con l'ambiente, effettuano campionamenti della sequenza degli stati, delle azioni e delle ricompense. Questi metodi in base alle caratteristiche illustrate si dice che apprendano dall'esperienza diretta o simulata perché non sono a conoscenza delle dinamiche dell'ambiente ma le imparano durante il processo di apprendimento. Questi metodi sono in grado di fornire all'agent il comportamento ottimale per lo svolgimento del lavoro assegnato. Il modello utilizzato in questi metodi è diverso da quello della PD. Infatti nei metodi MC il modello è sprovvisto delle probabilità che descrivono le transizioni di stato effettuate dall'ambiente, di conseguenza il modello dispone esclusivamente della descrizione delle transizioni campione. I metodi MC si basano sulla tecnica di averaging sample returns descritta in precedenza. Si tratta di metodologie che sono applicabili a task episodici. Una delle caratteristiche di MC è che la stima della funzione valore e l'aggiornamento della policy avvengono esclusivamente quando l'episodio raggiunge lo stato terminale e si conclude. Dunque i metodi MC differiscono dai metodi online che si aggiornano step-by-step, in questo caso infatti l'aggiornamento avviene episode-by-episode. In MC la funzione valore viene calcolata utilizzando campionamenti della somma delle ricompense totali ottenute dall'esperienza dei vari episodi (return). Il calcolo della policy ottimale in MC sfrutta i principi di policy iteration forniti dalla PD, questo significa che esiste un procedimento di predizione, un procedimento di miglioramento della policy e un procedimento finale che combina i primi due.

La predizione della funzione valore in MC avviene mediante il campionamento ed il calcolo della media dei valori delle return osservati dopo aver visitato uno stato. Più return vegono osservate e campionate e più la media converge al valore atteso reale. In questi metodi quando si visita uno stato s durante un episodio si dice che viene effettuata una *visit* (visita) allo stato s. Esistono due metodologie per la stima della funzione valore che vengono adottate in MC, la prima si chiama *first-visit MC method* e la seconda si chiama *every-visit MC method*. Entrambe i metodi, per la legge dei grandi numeri, riescono a convergere al valore atteso reale. In *first-visit MC method* si effettua l'aggiornamento del valore dello stato s esclusivamente quando si effettua la prima visita che lo coinvolge durante la sperimentazione di un episodio. Dal momento che durante un MDP è possibile tornare a visitare diverse volte uno stato s, quello che succede in *every-visit MC method* è l'effettuazione dell'aggiornamento del valore dello stato s ogni volta che viene visitato durante la sperimentazione dell'episodio.

Di seguito viene riportato lo pseudocodice dell'algoritmo *first-visit MC method*:

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated
Initialize:
    $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$
    $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):
    Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:
            Append G to $Returns(S_t)$
            $V(S_t) \leftarrow$ average$(Returns(S_t))$

Figura 1.4: Pseudocodice dell'algoritmo *first-visit MC method*. [23]

Nei metodi MC la stima di ogni stato è indipendente dagli altri e per questo motivo questi metodi *non fanno bootstrap* come accadeva per i metodi di PD. Per via di questa indipendenza tra gli stati, il costo computazionale della stima del valore di un singolo stato non viene influenzato dal numero complessivo degli stati dell'ambiente del problema esaminato. Questi metodi diventano attraenti in contesti in cui si desidera stimare il valore di un sottoinsieme degli stati di un ambiente.

Uno degli obiettivi primari per i metodi MC è la stima di q_* che rappresenta il valore delle coppie stato-azione. Questo è necessario perché a differenza dei metodi di PD qui non si ha un modello dettagliato dell'ambiente. Quindi per poter stimare q_* si utilizza la tecnica di policy evaluation utile a stimare $q_\pi(s, a)$. Per la stima di $q_\pi(s, a)$ si utilizzano le tecniche MC utilizzate per stimare la funzione valore. In questo contesto per visita si intende il raggiungimento di uno stato s in cui viene selezionata l'azione a. A questo punto identificato il concetto di visita si applica *first-visit MC method* o *every-visit MC method*. Il funzionamento è identico al caso della stima della funzione valore con la differenza che adesso si tiene traccia delle return ottenute dalle coppie stato-azione. La convergenza di questi metodi ai valori reali avviene al tendere ad infinito del numero di visite. Per consentire un grado di esplorazione necessario a fronteggiare policy deterministiche si fa l'assunzione detta *exploring starts* che consiste nel dare per scontato che gli episodi possano partire in modo da consentire la visita di tutte le coppie stato-azione.

I metodi MC gestiscono il problema del controllo, che consiste nell'approssimazione della policy ottimale, seguendo gli stessi principi usati dai metodi PD attraverso la policy iteration. La Figura 1.5 seguente riporta lo schema utilizzato durante il controllo.

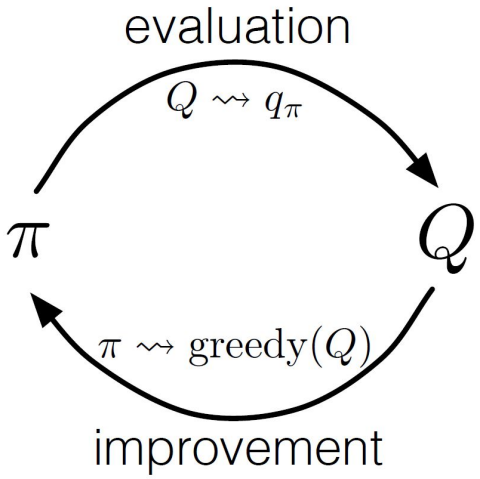


Figura 1.5: Schema della policy iteration nei metodi MC. [23]

In questo contesto il miglioramento della policy avviene selezionando la greedy action in funzione della funzione valore che considera la coppia stato-azione.

$$\pi(s) \doteq \arg\max_a q(s,a) \tag{1.14}$$

In base a quanto già detto per la PD se accade che $\pi_{k+1} = \pi_k$ significa allora che l'iterazione ha consentito il calcolo della policy ottimale.

In base a quanto detto fino ad ora per il MC si assume che la policy ottimale viene trovata avvalendosi del campionamento delle return osservate e quindi senza la perfetta conoscenza del problema come anticipato all'inizio di questa sezione. Le considerazioni fatte fino ad ora coinvolgono l'assunzione exploring starts e la disponibilità di un infinito numero di episodi.

La Figura 1.6 mostra l'algoritmo utilizzato per la stima della policy ottimale con l'assunzione dell'exploring starts.

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:
$\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$
$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):
Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0
Generate an episode from S_0, A_0, following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
$G \leftarrow 0$
Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
$G \leftarrow \gamma G + R_{t+1}$
Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
Append G to $Returns(S_t, A_t)$
$Q(S_t, A_t) \leftarrow$ average$(Returns(S_t, A_t))$
$\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

Figura 1.6: Pseudocodice policy iteration in MC con exploring starts. [23]

Negli scenari reali purtroppo l'assunzione exploring starts non è considerabile, per far fronte a questa condizione ci si avvale di metodi off-policy e metodi on-polict. I metodi on-policy cercano di valutare e migliorare la policy usata per compiere le decisioni, il metodo Monte Carlo ES descritto in Figura 1.6 ne è un esempio che però deve essere modificato per non avvalersi dell'assunzione exploring starts. Il miglioramento proposto dalla letteratura è l'utilizzo di una policy epsilon greedy che consenta un'esplorazione continua durante tutta la fase di apprendimento dell'agente. L'algoritmo viene riportato nella Figura 1.7

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-soft policy

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

Figura 1.7: Pseudocodice policy iteration MC $\epsilon - greedy$. [23]

I metodi off-policy utilizzano policy diverse, ad esempio è possibile utilizzarne due. La prima policy (target policy) è quella appresa e che tende alla policy ottimale mentre la seconda (behavior policy) è quella più esplorativa e viene utilizzata per la generazione del comportamento. I metodi off-policy convergono meno velocemente dei metodi on-policy anche se sono più potenti e generali. In alcuni casi speciali i metodi off-policy vengono considerati on-policy quando la target policy e la behaviour policy sono le stesse. Si considera π essere la target policy e b essere la behaviour policy. Si fa l'assunzione di convergenza tra le due politiche e cioè che $\pi(a|s) > 0$ implica $b(a|s) > 0$. Secondo questa assunzione possiamo utilizzare gli episodi e la policy b per stimare la policy π perché si richiede che ogni azione presa sotto la policy π sia presa, almeno occasionalmente, sotto la policy b. Secondo questa assunzione b è stocastica negli stati in cui non è identica a π, mentre la politica π dovrebbe essere deterministica. La b dunque segue politiche $\epsilon - greedy$ mentre la π segue una politica greedy.

Quasi tutte le metodologie off-policy utilizzano l'importance sampling che consiste in una tecnica generale che serve a stimare i valori attesi sotto una certa distribuzione dato un campionamento di un'altra distribuzione. Questa tecnica la si impiega applicando dei pesi alle return in base alla probabilità, detta importance-sampling ratio, di incontrarle lungo la traiettoria che viene a costituirsi utilizzando le due policy. Partendo da un ipotetico stato S_t, considerando la policy π, la probabilità di avere una successiva

traiettoria stato-azione *At, St+1,At+1, . . . ,ST* si calcola con la formula:

$$\begin{aligned}
&\Pr\{A_t, S_{t+1}, A_{t+1}, \ldots, S_T \mid S_t, A_{t:T-1} \sim \pi\} \\
&\quad = \pi(A_t|S_t)p(S_{t+1}|S_t, A_t)\pi(A_{t+1}|S_{t+1}) \cdots p(S_T|S_{T-1}, A_{T-1}) \\
&\quad = \prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)
\end{aligned} \tag{1.15}$$

Per calcolare importance-sampling ratio si utilizza la formula che segue:

$$\rho_{t:T-1} \doteq \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)p(S_{k+1}|S_k, A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)} \tag{1.16}$$

Per stimare invece $v_\pi(s)$ si possono utilizzare le seguenti due formule:

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{|\mathcal{T}(s)|} \tag{1.17}$$

La 1.17 utilizza la media semplice e al denominatore viene mostrato l'insieme di passi temporali in cui lo stato s viene visitato. Questa tecnica si chiama ordinary importance sampling.

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1}} \tag{1.18}$$

La 1.18 utilizza la media pesata e prende il nome di weighted importance sampling. Questo metodo dispone di una varianza inferiore e per questo è preferito.

Entrambe i metodi ordinary importance sampling e weighted importance sampling esistono nella forma first-visit e every-visit. I metodi every-visit eliminano la necessità di tenere traccia di quale stato è già stato visitato e sono più facili da estendere alle approssimazioni.

Metodi Temporal Difference

L'insieme di metodi Temporale Difference (TD) sono l'idea centrale e più nuova che caratterizza il reinforcement learning. I Metodi TD combinano le idee degli approcci MC e di PD. I metodi TD riescono ad imparare dall'esperienza diretta come i metodi MC e fanno bootstrap come i metodi di PD.

Il problema della predizione nei metodi MC viene affrontato step-by-step, ogni volta che si riceve la ricompensa viene aggiornata la funzione valore. Il metodo più semplice di TD effettua l'aggiornamento seguendo la seguente regola:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big] \tag{1.19}$$

Questo metodo di aggiornamento chiamato TD(0), oppure one-step TD, rappresenta un caso speciale di metodi più avanzati, non trattati in questa tesi, che corrispondono a TD(λ) e n-step TD. Lo pseudocodice del metodo TD(0) viene mostrato nella Figura 1.8.

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
    $S \leftarrow S'$
  until S is terminal

Figura 1.8: Pseudocodice del metodo TD(0) relativo al problema della predizione. [23]

In questa fase i metodi TD combinano il campionamento dei metodi MC e il bootstrap dei metodi PD. La stima del valore dello stato attualmente visitato si basa sulla stima dello stato che viene raggiunto alla selezione dell'azione. Gli aggiornamenti di questo tipo prendono il nome di sample updates perché guardano alle stime degli stati, o coppie stato-valore, futuri. A differenza del PD non si deve iterare per tutti gli stati, l'aggiornamento avviene solo per lo stato di interesse.

Il vantaggio dei metodi TD risiede in due ragioni: la prima è che non necessitano di un modello accurato dell'ambiente del problema, la seconda è che gli aggiornamenti avvengono step-by-step e per questo motivo sono adatti ad applicazioni note in reinforcement

learning con il termine *online*. Si ricorda che i problemi di natura online necessitano di decisioni immediate. È stato provato che per qualsiasi policy fissata π, il metodo TD(0) converge a v_π se il parametro step size *alpha* viene decrementato nel tempo secondo quanto descritto in (1.4).

Considerando il metodo TD(0) e ipotizzando che venga usato il batch update, accade che la convergenza all'ottimalità avviene più velocemente rispetto ai metodi MC. Questo accade perché il metodo TD(0) in questo contesto calcola la certainty-equivalence estimate. Con batch update si intende che tutti gli aggiornamenti che avvengono ad ogni passo temporale di un episodio vengono considerati come un lotto che racchiude questi aggiornamenti e rappresentano un'informazione di miglioramento unitaria. La certainty-equivalence estimate assume che dato un modello del problema, allora, è possibile calcolare una value function corretta se il modello rappresentativo del problema risulta corretto.

Q Learning

Il Q learning [28] è la tecnica di reinforcement learning utilizzata nel progetto di tesi. Si tratta di un metodo di controllo, inventato da Watkins nel 1989, appartenente all'insieme dei metodi TD. Si basa sulla programmazione dinamica ed è utilizzato nella risoluzione di problemi online relativi a task episodici. Si tratta di un metodo off-policy perché viene utilizzata una policy per la selezione delle azioni ($\epsilon - greedy$) che consente di calcolare una policy ottimale *greedy*. L'agent si avvale di una tabella che prende il nome di Q-table le cui righe rappresentano gli stati incontrati durante lo svolgimento del task e le colonne rappresentano le azioni di cui l'agent dispone in ogni stato assunto dall'ambiente. Ogni cella della Q-table contiene un punteggio numerico di qualità che indica quanto conveniente sia, allo scopo di massimizzare la return, intraprendere una specifica azione in uno specifico stato. Il valore in questione è il risultato della value function $q(a, s)$. La Q-table viene inizializzata in modo arbitrario al momento della creazione dell'agent, solitamente è composta da soli 0, e viene aggiornata durante il processo di apprendimento mediante il meccanismo di *bootstrap* che in inglese significa "uscire dalle situazioni sfavorevoli con le proprie forze". Il metodo è online perché gli aggiornamenti avvengono step-by-step durante ogni passo temporale degli episodi.

Il meccanismo di *bootstrap* avviene attraverso l'utilizzo da parte dell'agent della Q-function (1.20):

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right] \tag{1.20}$$

Lo scopo della Q-function è quello di determinare la qualità della scelta di intraprendere una specifica azione quando l'ambiente si trova in uno specifico stato.

I parametri utilizzati della funzione sono:

- S_t: stato in cui l'ambiente si trova quando l'agent effettua la decisione dell'azione a.
- A_t: azione decisa dall'agent che si trova in un ambiente che ha stato s.
- $Q(S_t, A_t)$: Qualità della coppia stato-azione presa in considerazione dall'agent.
- α: learning rate compreso tra 0 e 1. Determina quanto velocemente l'agent sia in grado di apprendere. Più alto è il numero tanto più rapidamente l'agent potrà apprendere.
- R_{t+1}: ricompensa ottenuta intraprendendo l'azione a quando l'ambiente è nello stato s.
- γ: tasso di sconto compreso tra 0 e 1. Determina quanta influenza abbiano le decisioni prese nel passato rispetto a quelle prese più di recente. Valori alti di gamma attribuiscono maggiore peso alle scelte compiute più di recente. Serve a bilanciare l'importanza delle ricompense immediate rispetto a quelle future.
- S_{t+1}: stato raggiunto dall'environment dopo che l'agent intraprende l'azione a_t.
- $\mathbf{max}_a Q(S_{t+1}, a)$: identifica la cella della Q-table con punteggio massimo considerando la colonna che corrisponde allo S_{t+1} che l'environment assume dopo che l'agent intraprende l'azione A_t.

La Q-function determina la qualità di una coppia stato-azione prendendo in considerazione la ricompensa immediata ricevuta e il valore di qualità più alto presente nella riga della Q-table corrispondente allo stato raggiunto dall'ambiente attraverso l'azione intrapresa dall'agent. Per questo motivo il Q-learning fa *bootstrap*.

Con il susseguirsi degli episodi, e per merito dell'apprendimento, ci si aspetta che a un certo punto si verifichi la propagazione della qualità di maggior valore presente in una cella della Q-table attraverso l'intera tabella. Questa propagazione si manifesta per via del funzionamento della Q-function che aggiorna in place il valore della coppia stato-azione considerata prendendo in considerazione il valore della coppia stato-azione futura. Il punteggio della coppia stato-azione futuro influenza il punteggio della coppia stato azione attuale. La propagazione consente di identificare in ogni riga della Q-table l'azione di maggior qualità. Dunque considerando l'intera Q-table è possibile identificare l'insieme di azioni migliori che costituisce la sequenza di azioni che una volta intrapresa conduce alla massimizzazione della return ottenuta dall'agent durante lo svolgimento del suo task all'interno dell'ambiente di appartenenza. Il metodo converge alla policy ottimale se tutte le coppie stato-azione vengono provate un numero di volte che tende ad infinito. Negli scenari reali la convergenza può manifestarsi prima, questo accade in

base a tutto quello che è stato esaminato fino ad ora nella sezione dello stato dell'arte di questa tesi.

Le R_{t+1} attribuite alle coppie stato-azione vengono generalmente memorizzate in una tabella, nominata R-table, che viene adeguatamente inizializzata dal programmatore con lo scopo di incentivare l'agent al raggiungimento dell'obiettivo del task. Viene disincentivata la focalizzazione da parte dell'agent relativamente ai sotto-obiettivi del task.

Come già detto in precedenza, un aspetto importante del reinforcement learning è la capacità del sistema di poter determinazione degli stati in cui l'ambiente si trova. Se si considera uno scenario smart IoT, come quello trattato nel progetto di tesi, il dispositivo deve essere dotato di sensori capaci di percepire l'ambiente circostante e di strutture software in grado di rilevare parametri significativi del dispositivo. L'insieme delle informazioni raccolte devono consentire al sistema di identificare e rappresentare lo stato ambientale sotto forma di dato utile all'agent per le sue computazioni.

Uno scenario tipico del Q-learning è quello in cui un robot ha come task quello di attraversare un territorio inizialmente sconosciuto, e pieno di ostacoli, con l'intento di raggiungere uno specifico punto obiettivo. Semplificando questo scenario è possibile identificare il robot con l'agent, il terreno con l'ambiente e i movimenti verso nord, sud, ovest ed est con le azioni a disposizione dell'agent. L'ambiente viene rappresentato sotto forma di tabella in cui ogni cella rappresenta l'area occupata dal robot. A questo punto è necessario realizzare la R-table che per essere adeguata deve prevedere ricompensa positiva, ad esempio +100, solo quando l'agent decide l'azione che lo conduce da una casella limitrofa a quella con il punto obiettivo verso l'effettiva casella contenente il punto obiettivo. Ogni passo che conduce l'agent in una casella priva di ostacoli ha ricompensa -1, mentre ogni azione che condurrebbe ad una casella occupata da un ostacolo produce una ricompensa di -10 e il robot non combie alcun passo. Una situazione da evitare è l'assegnazione di reward positiva alle azioni che spostano l'agent in caselle prive di ostacoli che sono diverse da quella che contiene il punto obiettivo. Se l'agent ricevesse ricompense ad ogni raggiungimento di una casella non terminale, allora potrebbe succedere che provi a massimizzare la return spostandosi tra caselle vicine senza mai raggiungere l'obiettivo del task.

Il task si sviluppa attraverso episodi che per consentire all'agent di apprendere devono poter essere sperimentati diverse volte. Nello scenario preso in considerazione gli stati considerati non terminali sono rappresentati da tutte le caselle diverse da quella che contiene il punto obiettivo, mentre, lo stato finale è rappresentato dalla casella che contiene il punto obiettivo. Come detto anche nel capitolo del MDP accade che ogni volta che l'agent conduce l'ambiente nello stato terminale, l'episodio termina e un suo task è concluso. Se il lavoro complessivo dell'agent non è terminato ed è in condizione di continuare allora si proseguirà con il portare a termine nuove istanze del suo task attraverso nuovi episodi. Dopo un certo numero di episodi l'agent apprende come svolgere il suo lavoro in autonomia.

Due problemi del Q-learning sono la dimensione della Q-table e la velocità di convergenza della policy comportamento (b) alla policy obiettivo (π) ottimale. Un ambiente influenzato da molte variabili ha come conseguenza la presenza di una Q-table molto grande, essa in alcuni casi potrebbe eccedere la memoria a disposizione del dispositivo. Per questo motivo è necessario disporre di strategie idonee a consentire l'ottimizzare della rappresentazione degli stati. La velocità di convergenza viene influenzata negativamente dai comportamenti casuali finalizzati a incentivare l'esplorazione da parte dell'agent che però è fondamentale, una minore esplorazione causerebbe infatti un agent meno efficiente che potrebbe non riuscire ad identificare la policy ottimale.

Capitolo 2

Progetto

Il progetto di tesi consiste nello sperimentare l'applicabilità del Q-learning nella risoluzione di tutti quei problemi dotati di stati ben identificabili e ben modellabili come sequenze di decisioni che prevedono l'esecuzione di azioni. Gli stati del problema devono disporre della proprietà di Markov, in questo modo ogni stato risulta in grado di illustrare cosa sia successo nei passi temporali precedenti di un episodio anche senza che venga tenuta traccia di una cronologia degli avvenimenti. L'obiettivo del progetto è quindi quello di adottare metodologie e tecnologie che consentono di realizzare sistemi decisionali governati da agenti di Reinforcement Learning, utilizzando la tecnica del Q-learning, capaci di apprendere a svolgere un lavoro in modo efficace ed efficiente.

Il Q-learning viene tipicamente utilizzato in scenari in cui un sistema deve spostare il suo corpo, reale o virtuale, all'interno di un ambiente. Tipici esempi sono i robot industriali che svolgono il loro lavoro in modo intelligente e automatico attraverso il movimento parziale o totale del corpo che li costituisce [19]. Altri esempi che rappresentano uno scenario interessante e idoneo per il Q-learning sono gli ambienti virtuali come i videogiochi in cui una entità deve effettuare spostamenti o azioni per poter raggiungere un obiettivo [2]. In ogni applicazione tipica del Q-learning, dunque, i temi ricorrenti sono: lo spostamento e il passaggio da uno stato all'altro causato dal compimento di un'azione. Quindi, l'agente può intervenire direttamente su un sottoinsieme delle caratteristiche che costituiscono l'intero insieme delle caratteristiche ambientali. Questo concetto di spostamento è stato tenuto in considerazione e reso più astratto per poter essere utilizzato nel progetto di tesi.

1 Specifiche del progetto

Lo scenario preso in considerazione è quello in cui un dispositivo smart IoT effettua il monitoraggio costante del valore dei parametri che caratterizzano un ambiente. Nell'ambiente sperimentale i parametri presi in considerazione sono la temperatura, l'umidità e la presenza di connettività. L'ambiente monitorato è soggetto allo scorrere del tempo. I valori associati ai parametri ambientali descrivono lo stato generale dell'ambiente. La necessità del controllo da parte del dispositivo smart IoT è data dal fatto che i valori assunti dai parametri ambientali variano nel tempo. I valori, che tutti i parametri ambientali assumono al tempo x, creano una configurazione che identificano lo stato dell'ambiente considerato. Il dispositivo smart IoT viene istruito in merito ai valori che i parametri ambientali devono mantenere affinché lo stato generale dell'ambiente possa essere considerato in una condizione ideale. Lo scopo del dispositivo smart IoT è quello di effettuare opportune azioni volte a contrastare le variazioni di quei valori dei parametri ambientali che coincidono o superano i valori limite imposti dall'utente. Per esempio, prendendo in considerazione il parametro temperatura, il compito del dispositivo smart IoT è quello di inviare informazioni agli opportuni attuatori capaci di riscaldare o raffreddare l'ambiente in caso di temperature al di fuori dei limiti prefissati in modo da riportare il valore del parametro entro i limiti stabiliti. Nel momento in cui la temperatura ha raggiunto opportuni valori, il dispositivo deve comunicare nuovamente con gli attuatori al fine di interrompere l'azione di riscaldamento o raffreddamento precedentemente intrapresa dall'attuatore. Come spiegato, nella sezione dedicata allo stato dell'arte, per ambiente si intende tutto ciò che non è l'agente. L'ambiente è pertanto parte del dispositivo oltre che essere il luogo fisico o virtuale che ospita il dispositivo stesso. La figura 2.1 mostra l'architettura del sistema software di sperimentazione in grado di rappresentare tutte le entità dello scenario considerato al fine di condurre le sperimentazioni e le analisi necessarie al raggiungimento degli obiettivi di tesi.

Il dispositivo smart IoT è in grado di compiere le sue decisioni grazie all'ausilio di un agente di Q-learning. Durante lo svolgimento del lavoro l'agente consente al dispositivo di apprendere quali siano le decisioni migliori da compiere in relazione agli stati in cui l'ambiente si trova in quel momento. Una prima parte del progetto consiste dunque nella realizzazione di un ambiente simulato che consenta di osservare e testare il processo di apprendimento dell'agente. Come mostrato in figura 2.1 il sistema è composto da diversi entità simulate, sistemi e sotto-sistemi. Attraverso il codice scritto è possibile seguire il processo di apprendimento partendo dalla modellazione dei dati, passando per le simulazioni e arrivando infine allo studio dei dati prodotti. Durante la simulazione, si potrà osservare l'agente partire da una condizione di completa impreparazione fino ad arrivare a una condizione di specializzazione nel lavoro da svolgere. Il progetto è dunque un sistema software di sperimentazione che può essere considerato il laboratorio in cui condurre gli esperimenti di apprendimento del progetto di tesi. Il sistema realizzato mette a disposizione le seguenti funzionalità:

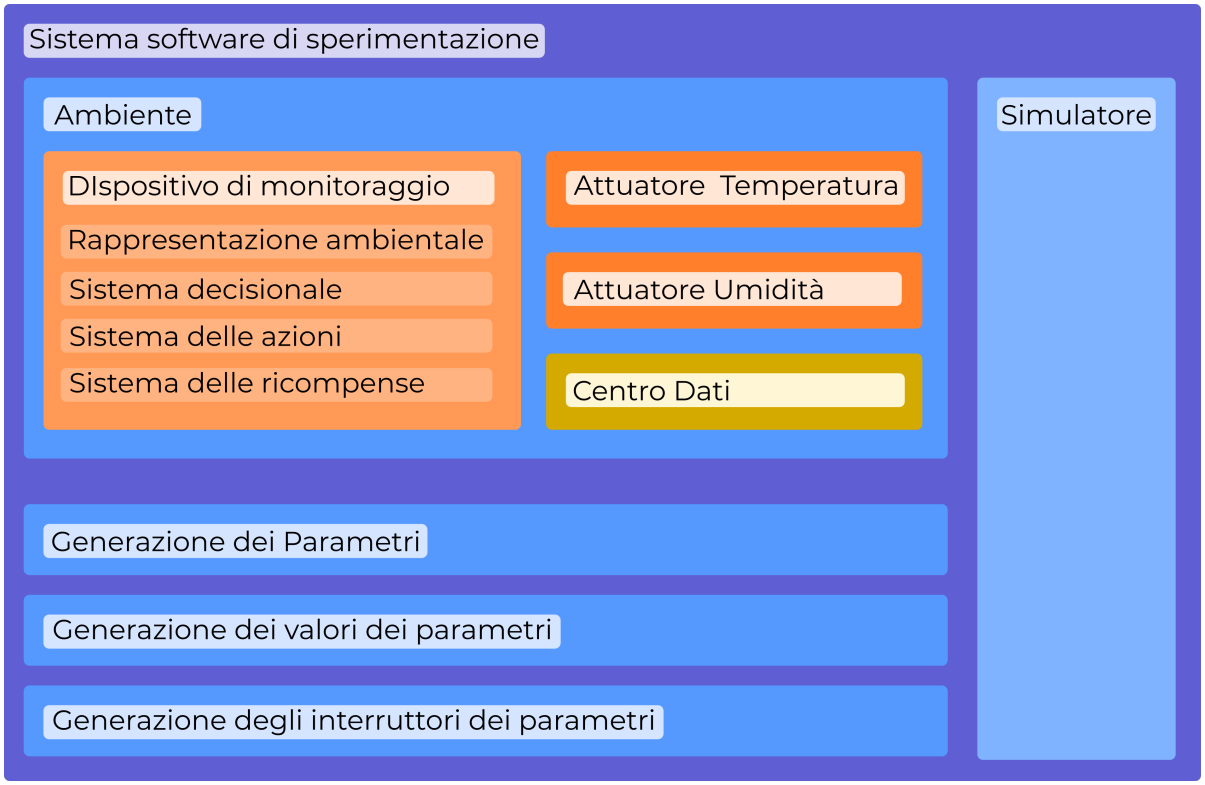


Figura 2.1: Grafico dell'architettura del sistema software di sperimentazione.

- Conduzione di simulazione sperimentale in cui l'ambiente viene sottoposto allo scorrere del tempo e alla variazione dei parametri ambientali. Durante la simulazione le variazioni ambientali vengono sottoposte al monitoraggio di tre dispositivi: uno smart, uno che svolge il lavoro senza mai andare in risparmio energetico e uno che se possibile entra in modalità di risparmio energetico.

- Salvataggio della Q-table ottenuta a fine sperimentazione. Essa rappresenta l'esperienza che il dispositivo ha acquisito attraverso il processo di apprendimento.

- Studio della convergenza della Q-table alla policy ottimale. La funzionalità consente di determinare in quale momento l'agent ha appreso come svolgere in modo efficiente il lavoro assegnato. La condizione sperimentale più soddisfacente si verifica quando il problema sottoposto consente all'agent di apprendere il comportamento migliore in relazione a tutti gli stati previsti dall'ambiente. In questo caso si può osservare una coincidenza dei punteggi di qualità di tutte le celle della Q-table con i punteggi di qualità di tutte le celle della policy ottimale. Potrebbe capitare che un agent sia sottoposto ad un problema parziale durante la simulazione, in questo caso visiterebbe un sottoinsieme di tutti i possibili stati dell'ambiente. In tal caso il

dispositivo si considera comunque specializzato se le celle degli stati visitati hanno punteggi di qualità che coincidono con i punteggi di qualità delle celle della policy ottimale. Questa funzionalità produce un documento che mostra il momento in cui l'agent si è specializzato mediante la rappresentazione della Q-table di specializzazione, il numero di episodi e il numero di passi temporali totali che hanno condotto alla Q-table di specializzazione.

- Studio e rappresentazione dei risultati. I dati prodotti dalla simulazione vengono rappresentati mediante grafici che descrivono statisticamente i dati e che consentono di mostrare le fasi di apprendimento e le differenze tra gli approcci utilizzati.

Terminati gli esperimenti di simulazione è necessario procedere verso esperimenti che coinvolgono dispositivi e ambienti reali. Si desidera infatti realizzare un prototipo hardware del dispositivo di monitoraggio smart IoT in grado di percepire realmente l'ambiente che lo circonda. Il prototipo deve essere capace di comportarsi come il dispositivo simulato, deve quindi poter identificare i momenti in cui è meglio eseguire azioni di intervento e i momenti in cui è meglio eseguire le azioni di gestione e di risparmio energetico. Il prototipo deve essere inoltre dotato della facoltà di fare logging che consente di tenere traccia degli episodi sperimentati. I dati prodotti dal dispositivo reale devono poter essere estratti da uno script di analisi e rappresentazione dei dati al fine di ottenere informazioni relative al modo in cui il lavoro di monitoraggio è stato svolto.

2 Tecnologie, materiali e metodi

2.1 Metodo

Per poter realizzare un sistema decisionale di Reinforcement Learning, utile alla risoluzione del problema presentato nelle specifiche del progetto, è stato necessario realizzare un sistema di sperimentazione software capace di simulare tutti gli eventi tipici di un ambiente adatti a fornire tutti gli stimoli di cui un agent ha bisogno per poter intraprendere il processo di apprendimento. Per questo motivo è stato deciso di realizzare un sistema di sperimentazione software capace di simulare dispositivi di monitoraggio, attuatori e ambienti che ospitano le entità appena citate. Oltre al dispositivo smart, che rappresenta l'oggetto dello studio di tesi, sono stati simulati anche due dispositivi semplici di controllo. I dispositivi di controllo sono stati utilizzati come mezzo di paragone per le prestazioni del dispositivo smart. Oltre ai dati simulati che il sistema software è in grado di generare è stato realizzato un dispositivo di misurazione hardware mediante l'utilizzo di schede di sviluppo per microcontrollori che ha consentito di raccogliere un set di dati reale. Il set di dati reale è stato poi utilizzato dal simulatore per testare il comportamento dei dispositivi simulati. Durante la simulazione il software di sperimentazione produce tutti i dati necessari a dare risposta ai quesiti posti dallo studio condotto. Al termine della simulazione il sistema di sperimentazione effettua l'analisi dei dati raccolti durante il processo e produce file di output che verranno discussi nella sezione dedicata all'analisi dei risultati.

Dopo aver studiato la tecnica e aver condotto gli esperimenti di apprendimento è stato pensato di realizzare un prototipo hardware del dispositivo smart IoT di monitoraggio. Il prototipo è stato realizzato utilizzando una developement board ELEGOO, lo scopo di questo prototipo è stato quello di fornire la prova dell'efficacia reale di un agente di Q-learning alle prese con lo scenario reale descritto in dettaglio nella sezione dedicata al prototipo.

2.2 Tecnologie e materiali

Python

Lo sviluppo del sistema di sperimentazione software utilizzato per condurre gli esperimenti è avvenuto all'interno del IDE PyCharm 2020.3.3 (Community Edition) mediante l'utilizzo del linguaggio di programmazione Python nella sua versione 3.9.0.

Oltre alle librerie standard di Python sono state utilizzate le seguenti librerie:

- NumPy: si tratta di una libreria utilizzata per il calcolo statistico che tra le altre cose fornisce un oggetto array multidimensionale. L'array multidimensionale e i metodi della libreria sono stati utilizzati per la gestione della R-table, della Q-table e della rappresentazione tabellare della policy ottimale.

- SciPy: si tratta di una libreria realizzata con lo scopo di risolvere problemi scientifici e matematici. Essa è costruita su NumPy e consente di manipolare e visualizzare i dati attraverso un'ampia scelta di comandi ad alto livello. La libreria è stata utilizzata nel progetto per effettuare calcoli statistici sui dati prodotti dal sistema software di sperimentazione.

- tabulate: libreria utilizzata per formattare e mostrare all'interno di file TXT le informazioni relative alla Q-table risultante e di convergenza.

- matplotlib: libreria che consente la rappresentazione grafica dei dati scientifici. Nel progetto questa libreria è stata utilizzata per presentare i dati dei risultati degli esperimenti condotti.

C++

C++ è un linguaggio realizzato da Bjarne Stroustrup come estensione del linguaggio C. Questo linguaggio consente un alto livello di controllo sulle risorse di sistema e in merito alla memoria. Si tratta di uno dei linguaggio di programmazione più popolari al mondo e può essere facilmente ritrovato in applicazioni di tutti i giorni come sistemi operativi e interfacce grafiche. Si tratta di un linguaggio di programmazione orientato agli oggetti e consente di realizzare applicazioni adattabili a diverse piattaforme. Uno dei vantaggi di questo linguaggio è la possibilità di realizzare applicazioni molto veloci per via del fatto che si tratta di un linguaggio di programmazione mid level e quindi più vicino all'hardware rispetto ad altri liguaggi di programmazione.

In questo progetto il linguaggio C++ è stato utilizzato per la programmazione delle schede di sviluppo per microcontrollori impiegate nella la realizzazione del dispositivo hardware di misurazione dei parametri ambientali e del prototipo hardware del dispositivo smart IoT, la programmazione è avvenuta attraverso l'utilizzo dell'ambiente di sviluppo Arduino IDE.

Microcontrollori

I microcontrollori sono piccoli circuiti integrati che vengono impiegati per portare a termine funzionalità specifiche utilizzando poca alimentazione. Sono dotati internamente di processore, bus, ram e storage. Dispongono quindi di tutto il necessario per l'esecuzione del codice che descrive il compito da svolgere. La potenza computazionale di queste componenti elettroniche non necessita di essere elevata perché i compiti svolti sono semplici e poco impegnativi computazionalmente. Possono essere programmati con linguaggi di alto livello come ad esempio C o di basso livello come ad esempio assembly. Il codice sorgente programmato per consentire lo svolgimento del compito deve essere convertito in codice binario e inviato dal computer al microcontrollore per poter essere eseguito. Tradizionalmente il codice veniva trasferito dal computer alla memoria interna del microcontrollore per mezzo di un hardware chiamato programmer. Attualmente la programmazione dei microcontrollori è diventata più semplice grazie all'utilizzo delle microcontroller development boards. Queste boards sono schede di sviluppo che ospitano il microcontrollore e numerosi altri circuiti e componenti elettronici utili alla semplificazione dell'utilizzo del microcontrollore. Queste schede possono essere collegate facilmente al computer mediante connessione USB, consentendo un agevole trasferimento del codice da computer a microcontrollore.

Uno degli elementi importanti del microcontrollore è l'ADC che converte il voltaggio analogico in voltaggio digitale in modo da poter essere memorizzato sotto forma di dato e utilizzato dal microcontrollore per le computazioni. In alcuni casi è utile trasformare il segnale digitale utilizzato dalle computazioni del microcontrollore in segnale analogico, questa conversione avviene mediante la DAC.

I microcontrollori si presentano in diversi formati detti packaging e vengono integrati in un circuito per mezzo dei pin di cui dispongono mediante saldatura o semplice alloggiamento in appositi spazi dedicati. A livello industriale i microcontrollori vengono utilizzati nelle macchine automatiche per conferirgli la capacità di interazione con l'ambiente. Un microcontrollore è in grado di interagire con l'ambiente mediante sensori che convertono l'energia elettrica in informazione ed attuatori che trasformano l'informazione in energia elettrica che a sua volta viene trasformata in altre forme di energia.

Quando si utilizzano microcontrollori e si realizzano dei prototipi, o quando si svolgono degli studi relativi all'elettronica, molto probabilmente si implementano i circuiti utilizzando dei supporti come quello mostrato in figura 2.2 che prendono il nome di breadboard.

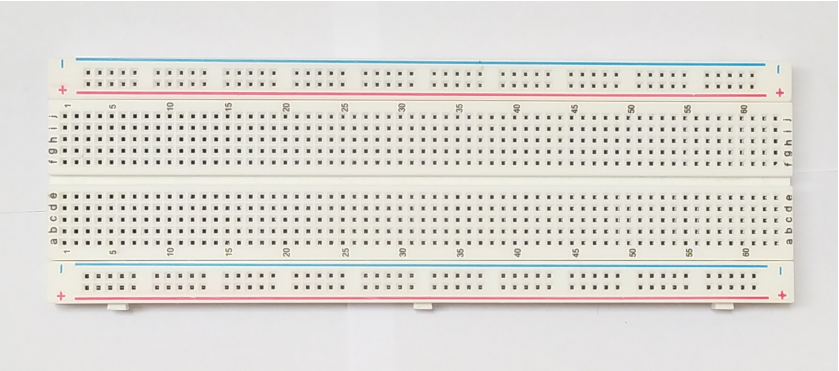


Figura 2.2: Breadboard - Supporto tipicamente utilizzato per lo studio dell'elettronica e per la realizzazione di semplici prototipi.

Queste basette consentono di connettere tra loro i componenti elettronici mediante gli appositi alloggiamenti forati a forma di piccolo quadrato, evitando saldature, che è possibile osservare lungo l'intero supporto. Nella figura 2.3 viene mostrata una sezione di breadboard per mostrare come gli alloggiamenti siano collegati tra loro. Le linee rosse e blu, in sovraimpressione, mostrano le connessioni tra gli alloggiamenti di alimentazione e massa, le linee gialle mostrano le connessioni tra gli alloggiamenti dedicati ai componenti e, per finire, la linea verde mostra la separazione tra le due sezioni della breadboard.

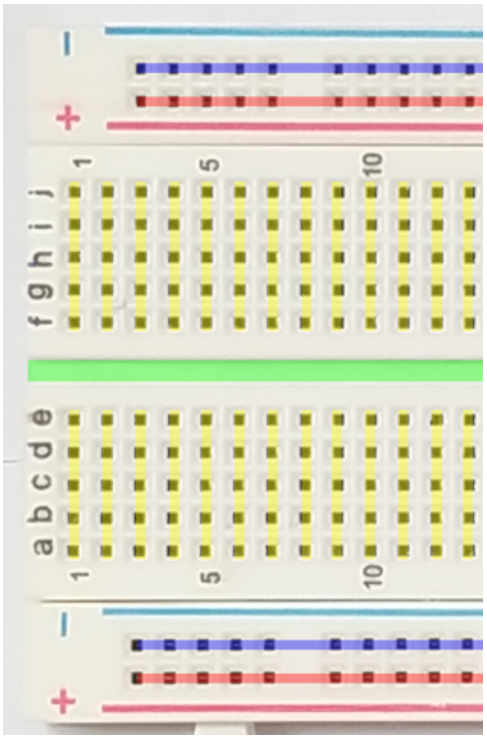


Figura 2.3: Breadboard - Connessioni tra gli alloggiamenti.

Le bradboard sono state utilizzate in questo progetto, assieme alle schede di sviluppo per microcontrollori, durante la realizzazione dei diversi dispositivi hardware impiegati.

Elegoo Arduino board

In questo progetto sono state utilizzate due schede di sviluppo per microcontrollori diverse, una per il dispositivo hardware di misurazione e una per il prototipo hardware del dispositivo smart.

La scheda di sviluppo utilizzata per la misurazione dei valori di temperatura e umidità, che popolano il set di dati impiegato nel progetto, è una ELEGOO Uno R3 realizzata seguendo gli schemi open source forniti da Arduino relativamente alla scheda di sviluppo Arduino Uno. Il dispositivo di misurazione è stato equipaggiato con il sensore di rilevazione di umidità e temperatura DHT11 e con l'orologio RTC DS3231 utilizzato per tenere traccia delle misurazioni al fine di realizzare un set di dati di 24 ore che va dalle ore 00:00 e termina alle ore 23:59.

I primi esperimenti relativi alla realizzazione del dispositivo di misurazione sono stati condotti su AUTODESK TINKERCAD in cui è stato possibile utilizzare la funzionalità di realizzazione di circuiti per sperimentare la circuitazione e la programmazione del dispositivo.

La programmazione del dispositivo è avvenuta utilizzando l'ambiente di sviluppo Arduino IDE 1.8.13. Il dispositivo di misurazione viene mostrato nella figura 2.4.

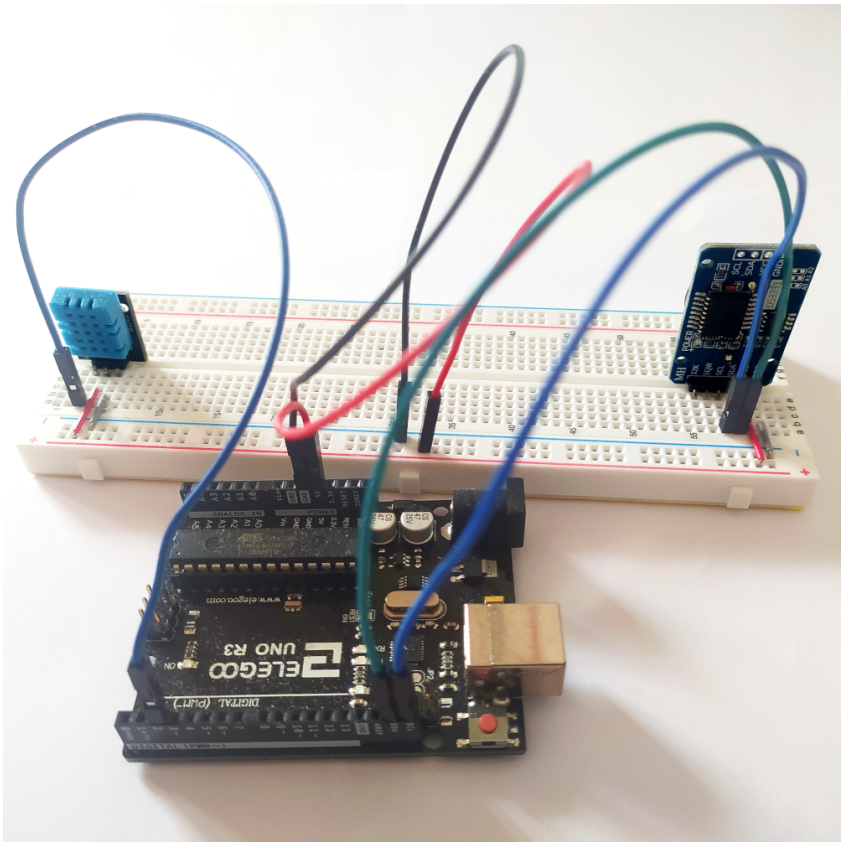


Figura 2.4: Dispositivo di misurazione utilizzato per la realizzazione del dataset dei valori reali.

La scheda di sviluppo utilizzata per realizzare il prototipo hardware del dispositivo smart è una ELEGOO MEGA 2560 R3. Per realizzare il dispositivo è stato realizzato un circuito attraverso l'impiego di LED, resistori e sensore DHT11, il circuito risultante viene mostrato nella figura 2.5.

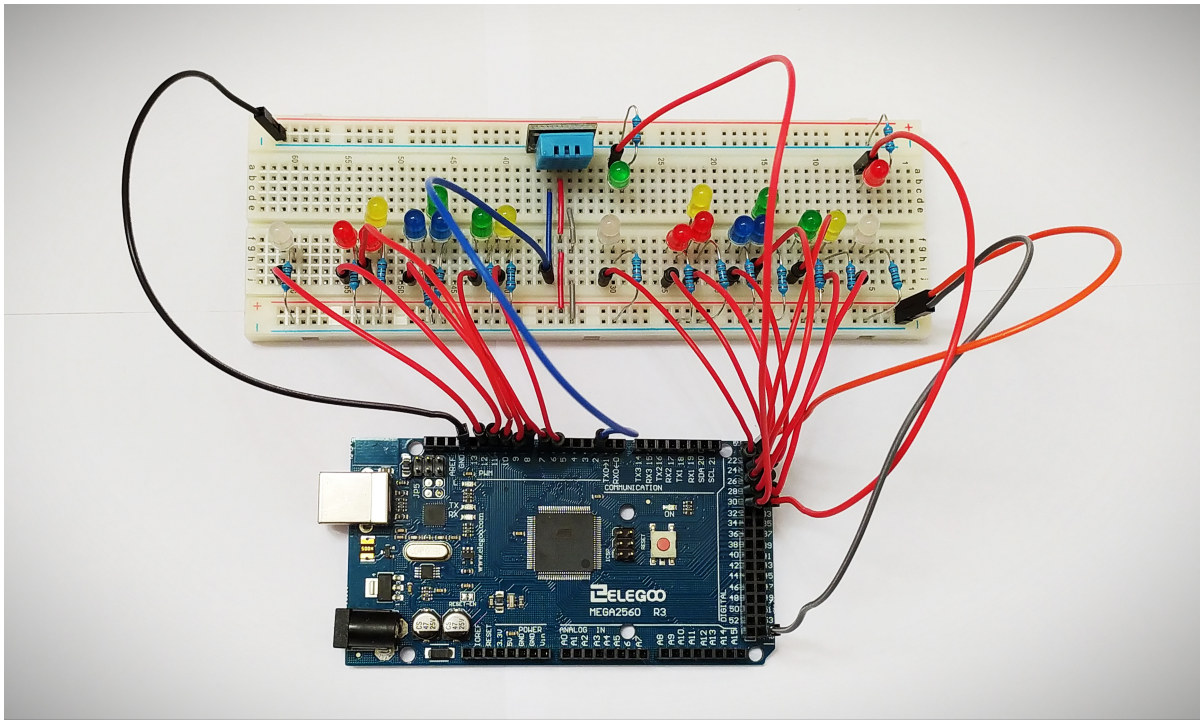


Figura 2.5: Prototipo del dispositivo smart IoT.

3 Sistema software di sperimentazione

Come da specifiche il passo fondamentale da compiere risiede nella realizzazione del sistema software di sperimentazione. Esso rappresenta infatti il laboratorio dell'esperimento di tesi consentendo di realizzare, testare e osservare l'intero processo di apprendimento. Il sistema è formato da diverse componenti e sottosistemi. Le componenti sono le classi che appartengono sia al dominio del problema che della soluzione. I sottosistemi sono l'insieme di funzioni e classi che consentono di realizzare e mettere a disposizione del sistema le funzionalità necessarie alla sperimentazione. Tutte le componenti del sistema software si considerano simulate e quindi composte da solo software e non realizzate con dispositivi hardware. Gli aspetti fondamentali del sistema di sperimentazione software sono descritte in dettaglio nelle relative sottosezioni seguenti.

Dispositivo smart IoT

La rappresentazione del dispositivo smart IoT all'interno del sistema software di sperimentazione è possibile grazie all'oggetto device della classe SmartIoTDevice il cui codice è disponibile nella sezione listing A.1.

L'oggetto device è il prototipo virtuale del dispositivo smart IoT descritto dalle specifiche del progetto. L'oggetto device è in grado di percepire l'ambiente, di rappresentare gli stati ambientali e di compiere le azioni decise dall'agente di Q-learning che lo governa. La maggior parte dell'attenzione dell'esperimento è rivolta alle attività di questo oggetto. Una volta che il sistema software di sperimentazione avrà generato tutti i dati necessari alla descrizione dell'ambiente dell'esperimento sarà possibile avviare una simulazione che coinvolge il dispositivo intelligente utilizzando il sottosistema di simulazione del sistema software di sperimentazione. Durante la simulazione l'oggetto device sarà in grado di simulare tutti i comportamenti di un dispositivo reale progettato per svolgere il lavoro descritto nelle specifiche del progetto. I dati generati verranno raccolti ed inviati ad un apposito oggetto di analisi dei dati che li utilizzerà per produrre i risultati.

L'architettura del dispositivo può essere descritta dalla seguente lista di sistemi:

- Sistema di rappresentazione dello stato ambientale: il sistema consente al dispositivo di capire in che stato l'ambiente si trovi in quel momento. Il sistema crea una rappresentazione dello stato ambientale utile allo svolgimento del lavoro. La rappresentazione è possibile effettuando l'azione di sensing dell'ambiente che restituisce i valori che i parametri ambientali assumono in quel preciso momento. L'insieme dei valori dei parametri ambientali percepiti consentono di creare una configurazione degli interruttori ambientali. Le configurazioni ambientali vengono rappresentate tramite numero in base mista per mezzo di un sistema di rappresentazione numerica in base mista. Le configurazioni rappresentate in base mista vengono convertite in numeri in base dieci per poter essere utilizzate come indici delle righe della Q-table.

- Sistema decisionale: il sistema consente all'agente che governa il dispositivo di decidere le azioni da compiere quando lo stato ambientale si trova in un determinato stato. Il sistema decisionale si basa sulla tecnica di Reinforcement Learning nota come Q-learning. Mediante questa tecnica il dispositivo passa da una situazione di completa impreparazione a una situazione di specializzazione nel lavoro da svolgere. L'apprendimento avviene durante lo svolgimento del lavoro e quindi in real time. Le decisioni vengono prese facendo affidamento ad una Q-table che effettua una mappatura del valore delle coppie stato-azione. Le scelte compiute da questo sistema vengono inviate al sistema delle azioni che le esegue. Una volta avvenuta l'esecuzione il sistema delle ricompense fornisce una ricompensa positiva o negativa in base alla coppia stato-azione generata dal passo temporale di riferimento. In questo modo il dispositivo impara mediante i principi di trial and error descritti nella sezione dello stato dell'arte.

- Sistema di ricompense: il sistema consente al dispositivo di assegnare delle ricompense positive o negative alle azioni intraprese dal sistema delle azioni. Le ricompense vengono assegnate avvalendosi della R-table mostrata nella figura 2.11 che consiste in una matrice che mappa le ricompense delle coppie stato-azione del problema esaminato.

L'oggetto device apprende qualcosa di nuovo fino a quando non ha provato più volte tutte le azioni in tutti gli stati previsti dall'ambiente. Il dispositivo smart IoT si considera specializzato in senso generale nel lavoro che deve svolgere nel momento in cui valgono contemporaneamente le seguenti condizioni:

1. Tutti gli stati ambientali possibili sono stati visitati, cioè tutte le celle della tabella sono state aggiornate.

2. Per ogni riga della Q-table è facile identificare le azioni di maggior qualità diversa da 0 che coincidono con le azioni di maggior qualità della relativa riga presente nella tabella della policy ottimale. Come detto in precedenza se il dispositivo affronta un problema che per l'intero svolgimento prevede transizioni in un sottoinsieme della totalità degli stati del problema allora si può considerare che il dispositivo si sia specializzato quando valgono le stesse condizioni ma in riferimento ai soli stati sperimentati.

Dispositivi di controllo

I dispositivi di controllo sono stati implementati per poter avere un mezzo di paragone utile a valutare le performance del dispositivo smart. Questi dispositivi sono stati implementati creando oggetti della classe SimpleDevice del modulo simle_device.py il cui codice è mostrato nella sezione del listing A.2.

Uno dei dispositivi di controllo è il dispositivo diligent, esso viene sottoposto agli stessi dati ambientali utilizzati nella simulazione con il dispositivo smart. Questo dispositivo utilizza una logica prestabilita che ne definisce il comportamento. Nello specifico ad ogni secondo simulato della giornata si effettua un sensing, in base al risultato se il dispositivo nota parametri fuori limite effettua la comunicazione agli attuatori ed al centro di raccolta dati. La rappresentazione dello stato ambientale e la decisione delle azioni viene mediante semplice logica if then else. Il dispositivo non entra mai in risparmio energetico.

L'altro dispositivo di controllo, detto energy saver, funziona allo stesso modo del dispositivo diligente con la differenza di entrare in risparmio energetico nel caso in cui i valori osservati risultano essere entro i limiti consentiti e qualora non ci fossero interventi in atto.

Generazione dei parametri ambientali

Il sottosistema di generazione dei parametri ambientali consente di generare i parametri che descrivono l'ambiente che racchiude l'agente. I parametri ambientali utilizzati nell'esperimento sono i gradi della temperatura, la percentuale di umidità e lo stato della connessione del dispositivo alla rete. Il sottosistema è predisposto per due tipi di parametri: parametri discreti e parametri continui. I parametri continui sono quelli che in questo dominio assumono valori decimali come per esempio la temperatura, e l'umidità. I parametri discreti sono quelli che in questo dominio vengono espressi con numeri interi che rappresentano gli stati che descrivono il parametro, per esempio la presenza di connessione viene indicata con il numero 1 mentre l'assenza di connessione viene indicata con il numero 0.

Le informazioni relative alla creazione dei parametri sono contenute all'interno di file CSV di inizializzazione che il sistema consulta a execution time. I file CSV utilizzati per l'inizializzazione dei parametri continui contengono le seguenti colonne per ogni riga del documento:

- *name*: indica il nome del parametro.
- *lower_limit_value*: valore limite che se raggiunto o superato indica che la temperatura è troppo bassa rispetto alla condizione ideale.
- *desired_value*: il valore ideale relativo al problema considerato.
- *upper_limit_value*: valore limite che se raggiunto o superato dal valore del parametro indica che la temperatura è troppo alta rispetto alla condizione ideale.

Tabella 2.1: Esempio di file di inizializzazione CSV di parametri continui.

name	*lower_limit_value*	*desired_value*	*upper_limit_value*
temperature	21.50	22.00	22.50
humidity	42.00	44.00	46.00

I file CSV utilizzati per l'inizializzazione dei parametri discreti contengono le seguenti colonne per ogni riga del documento:

- *name*: indica il nome del parametro.
- *value_names*: indica la lista di nomi dei valori numerici che il parametro può assumere.
- *values*: indica la lista di possibili valori numerici che il parametro può assumere.

- *lower_limit_value*: valore limite che se raggiunto o superato indica che la temperatura è troppo bassa rispetto alla condizione ideale.
- *desired_value*: indica il valore ideale del parametro.
- *upper_limit_value*: valore limite che se raggiunto o superato dal valore del parametro indica che la temperatura è troppo alta rispetto alla condizione ideale.

Tabella 2.2: Esempio di file di inizializzazione CSV di parametri discreti.

name	*value_names*	*values*	*lower_limit_value*	*desired_value*	*upper_limit_value*
wifi	offline#online	0#1	0	1	float("inf")

Il codice riportato nel listing A.3 mostra la funzione di inizializzazione invocata nel modulo main.py utilizzata per la generazione dei parametri continui. Il codice mostrato nel listing A.4 mostra la funzione utilizzata per inizializzare i parametri discreti.

I parametri di temperatura e di umidità servono a poter simulare l'azione di sensing ispirata al sensore DHT11. Di conseguenza il parametro della connessione serve per poter simulare il comportamento di controllo dello stato della connessione tipico delle librerie realizzate per essere usate con moduli elettronici come ad esempio l'ESP8266.

Ogni volta che il dispositivo fa sense percepisce i valori dei parametri generati dal sistema di generazione dei valori dei parametri ambientali. I valori attribuiti ai parametri di temperatura e umidità possono essere generati mediante modellazione o mediante estrazione da set di dati di misurazioni reali. Il parametro wifi invece viene sempre modellato.

Quindi quando il dispositivo fa sense entra a conoscenza del valore di temperatura, umidità e wifi. I valori vengono utilizzati per configurare gli interruttori, associati ai parametri, che sono in grado di descrivere esaustivamente lo stato ambientale. Entrando a conoscenza del valore dei parametri ambientali, il dispositivo si accorge se l'ambiente è in uno stato in cui è necessario intervenire oppure entrare in modalità di risparmio energetico. Proprio mediante il sense il dispositivo è in grado di capire se è connesso alla rete oppure no. In caso il dispositivo non fosse connesso, perché il wifi ha valore 0, viene effettuato un tentativo di connessione. Se nel momento in cui il dispositivo prova a riconnettersi lo stato del parametro wifi è 1, il dispositivo setta una variabile di controllo a True e sa di essere connesso. Altrimenti il dispositivo tenterà a ristabilire la connessione nel passo temporale successivo.

Generazione dei valori dei parametri ambientali

Questo sottosistema consente di generare una lista che in ogni posizione contiene a sua volta una lista di valori associati a uno specifico parametro ambientale. La lista dei valori detta parameter_values è composta da 86400 osservazioni. L'osservazione in posizione 0 rappresenta il valore che il parametro assume durante il secondo della giornata che corrisponde alle 00:00:00. L'osservazione che invece si trova in posizione 86399 corrisponde al valore assunto dal parametro durante il secondo della giornata che corrisponde alle ore 23:59:59. Dunque le 86400 osservazioni rappresentano la suddivisione di una giornata di 24 ore nei risultanti secondi. I secondi della suddivisione rappresentano i passi temporali di avanzamento della simulazione. La lista dei valori viene utilizzata durante la simulazione per determinare i valori assunti dai parametri ambientali nei secondi simulati della giornata. L'interrogazione avviene in modo sequenziale partendo da un secondo di partenza fino ad un secondo di arrivo. La lista dei valori dei parametri viene inizializzata utilizzando le informazioni contenuti nel relativo file di inizializzazione mostrato nella tabella 2.3

Tabella 2.3: Esempio di file di istruzione CSV per la lista dei valori dei parametri.

$name$	$default_value$
temperature	22.00

Una volta inizializzata la lista dei parametri con i valori di default è possibile apportare modifiche in due modi: utilizzando il sistema di modellazione dei dati ambientali oppure importando dati estratti da un dataset di misurazioni reali. Nella sperimentazione sono stati utilizzati entrambe le possibilità di modellazione della lista dei valori.

La generazione mediante sistema di modellazione dei dati consente di realizzare scenari arbitrari utili alla sperimentazione. Mediante gli scenari modellati infatti è possibile condurre l'agent in tutti gli stati previsti verificando se l'agent risulta in grado di apprendere oppure no. Questa strategia è un'alternativa alla tecnica utilizzata in letteratura in cui si svolgono esperimenti episodici utilizzando estrazioni casuali degli stati di partenza[23]. Lo scenario modellato, anche se non rappresentativo di una condizione realmente osservabile nella realtà, risulta molto utile ed efficace dal punto di vista sperimentale, ai fini dei test e per l'osservazione del processo di apprendimento. Ulteriori dettagli su questa scelta saranno riportati nella sezione di analisi dei risultati e delle conclusioni.

Le funzioni di modellazione dei dati sono due:

- Funzione di modellazione dei valori dei parametri continui.
- Funzione di modellazione dei valori dei parametri discreti.

Entrambe le funzione eseguono la modellazione sulla lista dei valori del parametro considerato utilizzando le istruzioni contenute nei file CSV di modellazione. I file CSV di istruzione per la modellazione contengono le seguenti informazioni:

- *destination_value*: rappresenta il valore di destinazione della modellazione.
- *start*: indica a quale secondo del giorno (posizione della lista) deve iniziare il processo di modellazione.
- *gradual*: indica la durata in secondi del periodo di modellazione graduale del valore del parametro. In questo periodo di tempo il valore del parametro viene modificato utilizzando una quantità costante a partire dal valore attuale fino al raggiungimento del *destination_value*.
- *duration*: indica la durata del periodo in cui il parametro mantiene il valore disturbato.
- *recovery*: indica la durata del periodo di recupero in cui il valore del parametro viene modificato di una quantità costante a partire dal valore disturbato fino al raggiungimento del valore di default.

Nella tabella 2.4 vengono illustrate le istruzioni di modellazione per il parametro temperatura. Le istruzioni indicano che il valore della temperatura deve passare gradualmente, nell'arco di 10 secondi, dal valore attuale al valore do 26.00 C°. Raggiunta la temperatura di 26.00 C° il parametro deve continuare a mantenere tale valore per 30 secondi. Terminati i 30 secondi il valore passa gradualmente da 26.00 C° al valore che il parametro stava assumendo prima della modellazione.

Tabella 2.4: Esempio di file di istruzione CSV per il disturbo dei valori dei parametri continui.

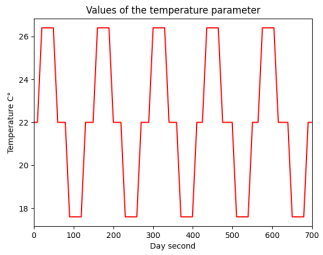
$destination_value$	$start$	$gradual$	$duration$	$recover$
26.00	10	10	30	10

Se consideriamo invece la funzione di modellazione dei valori dei parametri discreti è necessario utilizzare un file di istruzioni CSV formattato come mostrato nella tabella 2.5. Le informazioni che il file CSV descrive sono le seguenti:

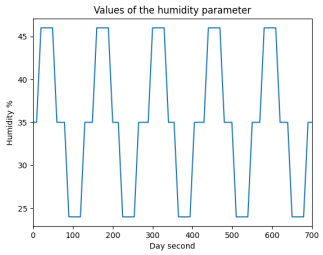
- *destination_value*: rappresenta il valore dello stato del parametro.
- *start*: secondo della giornata simulata in cui iniziare la modellazione.
- *duration*: indica la durata di mantenimento del destination value.

Tabella 2.5: Esempio di file di istruzione CSV per il disturbo dei valori dei parametri discreti.

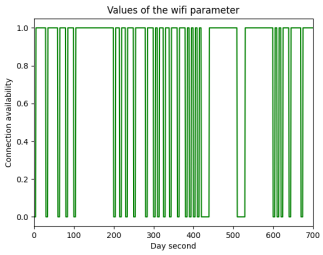
destination_value	*start*	*duration*
0	0	5



(a) Modellazione dei valori del parametro temperature.



(b) Modellazione dei valori del parametro humidity.



(c) Modellazione dei valori del parametro wifi.

Figura 2.6: Esempio di modellazione dei valori dei parametri.

I dati risultanti dal processo di modellazione, e utilizzati per l'esperimento, sono mostrati nella figura 2.6a in merito alla temperatura, nella figura 2.6b per quanto riguarda l'umidità e nella figura 2.6c in riferimento alla connettività Wi-Fi. I valori dei parametri generati mediante estrapolazione dei dati da dataset sono invece mostrati nella figura 2.7 per quanto riguarda la temperatura e nella figura 2.8 per quanto riguarda l'umidità.

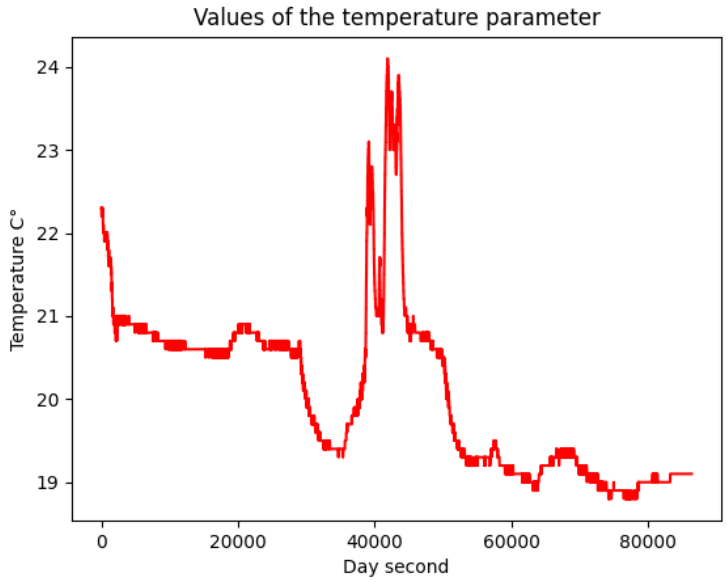


Figura 2.7: Plot dei valori di temperatura contenuti nel dataset.

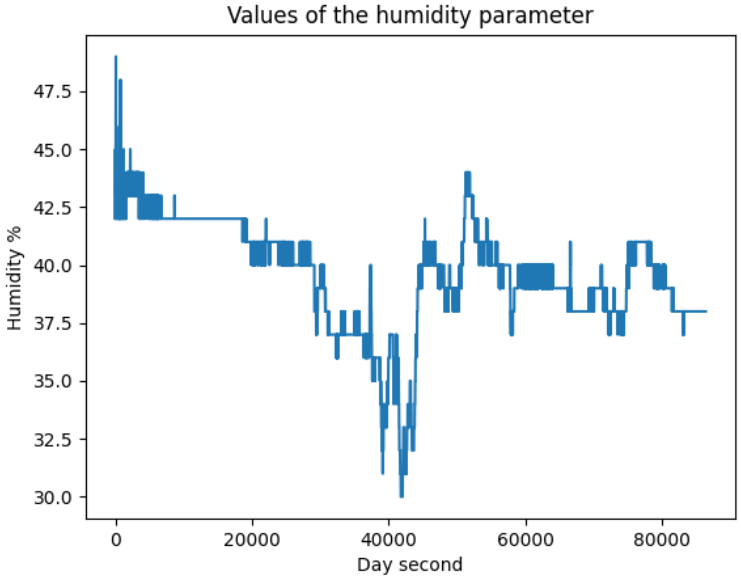


Figura 2.8: Plot dei valori di umidità contenuti nel dataset.

Il dataset utilizzato per l'estrapolazione dei valori dei parametri è stato acquisito mediante un dispositivo hardware di misurazione realizzato tramite scheda di sviluppo per microcontrollori UNO R3 Controller Board del produttore ELEGOO. Il dispositivo di misurazione è stato dotato di sensore DHT11 e di orologio RTC DS3231. Tramite il codice mostrato nella sezione listing A.5, è stato possibile raccogliere per 24 ore un'osservazione al secondo relativamente a valore di temperatura, percentuale di umidità, data e ora dell'ambiente reale osservato. Il dispositivo di misurazione ha quindi misurato gli 86400 valori delle relative posizioni della lista dei valori dei parametri. Le misurazioni raccolte sono state salvate in un file CSV opportunamente formattato nel modo mostrato dalla tabella di esempio 2.6. Il file CSV è stato realizzato per mezzo della comunicazione seriale USB tra il dispositivo di misurazione e un monitor seriale gestito dal software PuTTY. La formattazione delle righe del file csv è avvenuta mediante la programmazione del dispositivo di misurazione utilizzando la funzione Serial.print() di Arduino IDE mostrata nella sezione listing A.6. L'effettivo salvataggio delle righe all'interno del documento è avvenuto mediante la funzione di logging su file TXT di PuTTY. In sostanza ogni istruzione Serial.print() invia al monitor seriale informazioni che, una dopo l'altra, compongono una riga del file di log che sarà poi utilizzato come file CSV. Le informazioni raccolte nel modo descritto sono state successivamente arricchite mediante l'utilizzo dei fogli di calcolo di Google in modo da poter inserire arbitrariamente i valori del parametro wifi. La struttura risultante del dataset CSV è la seguente:

- *date*: indica la data in cui il processo di misurazione ha avuto luogo.
- *time*: indica l'orario di misurazione.
- *temperature*: indica la temperatura dell'ambiente reale osservato relativa a una specifica data e ora.
- *humidity*: indica la percentuale di umidità dell'ambiente reale osservato relativa a una specifica data e ora.
- $wifi$: rappresenta lo stato di connessione simulata del dispositivio smart IoT alla rete wifi relativamente a una specifica data e ora.

Tabella 2.6: Esempio di dataset utilizzato per la realizzazione della lista dei valori dei parametri.

date	*time*	*temperature*	*humidity*	$wifi$
2021/01/16	00:00:00	22.30	42.00	1

Il codice riportato nella sezione del listing A.7 mostra le funzioni utilizzate per inizializzare i valori dei parametri utilizzando i valori di default.

Generazione degli interruttori dei parametri ambientali

Gli interruttori sono un aspetto fondamentale del sistema ed appartengono esclusivamente al dispositivo che utilizza il Q-learning. Un interruttore fornisce al dispositivo le informazioni in merito alla necessità di intervento sul parametro a cui l'interruttore è associato. Per esempio se l'ambiente è descritto dal parametro temperatura sarà necessario realizzare un interruttore del parametro temperatura che in base allo stato assunto, in modo contestuale, riesce a istruire il dispositivo sull'eventuale necessità di riscaldare o raffreddare l'ambiente. Nella sezione listing A.8 viene mostrato il codice dell'oggetto interruttore.

La generazione degli interruttori necessita delle informazioni contenute negli oggetti che descrivono i parametri e in opportuni file CSV di inizializzazione come quello mostrato in tabella 2.7.

Tabella 2.7: Esempio di file di inizializzazione CSV degli interruttori

switch_name	type	number_of_possible_states
wifi	boolean	2
humidity	range	3
temperature	range	3

Gli interruttori creati vengono inseriti all'interno della lista degli interruttori. La lista viene ordinata mediante algoritmo quick sort in modo da posizionare gli interruttori in ordine crescente di numero dei possibili stati assumibili. Una volta che la lista degli interruttori è stata ordinata è necessario ordinare anche la lista dei parametri e la lista dei valori dei parametri. Per esempio, se dopo l'ordinamento gli oggetti della lista degli interruttori risultano fare riferimento in prima posizione al parametro wifi, in seconda posizione al parametro temperature e in terza posizione al parametro humidity, allora, anche le altre liste dovranno seguire un ordinamento tale da disporre i propri oggetti in modo che in prima posizione facciano riferimento a wifi, in second aposizione facciano riferimento a temperature e in terza posizione facciano riferimento a humidity. In questo modo l'interruttore, il parametro e la lista di valori del parametro, estratti ad esempio dalla posizione arbitraria 1, faranno tutti riferimento allo stesso parametro ambientale. Come risultato dunque si deve ottenere una configurazione delle liste simile a quella mostrata nella tabella 2.8.

La configurazione degli interruttori viene generata dal sottosistema di rappresentazione dello stato ambientale. Il funzionamento della configurazione degli interruttori è spiegato in dettaglio nell'apposita sezione dedicata al sottosistema di rappresentazione dello stato ambientale. La lista dei parametri e la lista degli interruttori sono le informazioni che consentono al sottosistema di rappresentazione dello stato ambientale di rappresentare l'ambiente dopo che il dispositivo effettua il sensing.

Tabella 2.8: Esempio di ordinamento delle liste di interruttori, parametri e valore dei parametri.

	Position 0	Position 1	Position 2
switches	wifi	temperature	humidity
parameters	wifi	temperature	humidity
parameters_value	wifi	temperature	humidity

L'implementazione degli interruttori ha consentito di rappresentare il problema di Reinforcement Learning considerato nell'esperimento mettendo in atto un meccanismo analogo allo spostamento che gli agenti di Q-learning compiono per poter risolvere i task episodici illustrati negli esempi presentati in letteratura [23] e nei problemi proposti in gym [2]. Per comprendere l'analogia con lo spostamento è utile pensare ad un esempio in cui il dispositivo rileva uno stato ambientale di intervento in cui il valore di temperatura risulta essere al di fuori dei limiti consentiti. A questo punto l'interruttore della temperatura viene impostato nello stato 1 che rappresenta la necessità di intervento sul parametro. Il dispositivo quindi interviene riscaldando o raffreddando opportunamente l'ambiente. L'interruttore passa dunque da 1 a 0 indicando che l'intervento è stato effettuato. In questo modo l'agente si è spostato da uno stato all'altro tramite un'azione. Si può immaginare la situazione come una console di comando con m leve, rappresentanti gli interruttori del sistema, che assumere in continuazione una tra le n posizioni che coincidono con il numero dei possibili stati che un interruttore può assumere. Si pensi ora all'agente come a un robot con m braccia che riporta in continuazione le leve dalla posizione attuale alla posizione ottimale. Seguendo questo ragionamento è stato più agevole trovare una semplice interpretazione e soluzione alla transizione tra gli stati dell'ambiente.

Rappresentazione dello stato ambientale

Questo sotto sistema si occupa della rappresentazione dello stato ambientale mediante la configurazione degli interruttori che sono associati ai parametri ambientali. La configurazione degli interruttori avviene dopo l'acquisizione del valore dei parametri ambientali mediante l'azione di sensing e dopo aver verificato le condizioni di impostazione dello stato degli interruttori. Come detto in precedenza nel sistema software di sperimentazione sono stati predisposti due tipi di interruttori: quelli che possono assumere 2 stati e quelli che ne possono assumere 3. Questi due tipi di interruttori vengono configurati mediante funzioni diverse.

Nell'esperimento si ha un solo interruttore a 2 stati che viene associato al parametro wifi che viene utilizzato per descrivere lo stato di connettività Wi-Fi. Il parametro wifi quando assume il valore 0 indica l'assenza di connessione, quando assume invece il valore 1 indica la presenza di connessione. L'interruttore che ne deriva è quindi un interruttore booleano che indica con lo stato 0 la condizione ideale in cui il dispositivo non deve fare nulla perché il valore del parametro di riferimento è 1. Invece, quando l'interruttore si trova nello stato 1 indica al dispositivo la necessità di tentare ad effettuare nuovamente una connessione perché il valore del parametro di riferimento è 0 che sta ad indicare assenza di connessione.

Gli interruttori a 3 stati utilizzati nell'esperimento sono quelli associati ai parametri temperatura e umidità. Questi parametri variano nel tempo attraversando diversi altri valori. Per esempio, se la temperatura attuale è 20° e l'ambiente si sta riscaldando quello che succede è che la temperatura assume i valori ipotetici 20.01°, 20.02°, 20.03° fino a quando non ha raggiunto un valore stabile o, in alcuni casi, continuando ad oscillare. Quindi questo tipo di parametro presenta un periodo di variazione graduale, un eventuale periodo di mantenimento di un valore stabile raggiunto e, per finire, un periodo graduale di ripristino del valore ideale. Un dispositivo che interviene sulla temperatura deve riuscire a capire quando l'intervento ha sortito gli effetti desiderati in modo da poter comunicare la volontà di arrestare gli effetti scatenati dall'intervento. Per via di quanto appena, gli interruttori ternari, indicano con lo stato 0 la condizione ideale in cui non è necessario fare nulla, con lo stato 1 la condizione in cui è necessario intervenire e con lo stato 2 la condizione in cui è necessario arrestare gli effetti dell'intervento precedentemente fatto.

Per configurazione degli interruttori si intende dunque una rappresentazione costituita da una lista di numeri interi ordinata in ordine crescente di numero dei possibili stati assumibili. Ad esempio se si dispone di una lista di interruttori come quella mostrata in tabella 2.9 una possibile configurazione potrebbe essere quella mostrata dalla rappresentazione di una lista di Python mostrata nella tabella 2.10. La configurazione della 2.10 si traduce in: connessione wifi disponibile, necessità di intervento sulla temperatura e necessità di arresto dell'intervento sull'umidità.

Tabella 2.9: Lista degli interruttori dei parametri ambientali.

Posizoine 0	Posizoine 1	Posizoine 2
wifi	humidity	temperature

Tabella 2.10: Esempio di configurazione per gli interruttori wifi, voltage, temperature e humidity.

Posizoine 0	Posizoine 1	Posizoine 2
0	1	2

Il dispositivo smart IoT è in grado di sperimentare stati di intervento, identificati dalla configurazione degli interruttori, e stati di gestione che sono rappresentati da un numero intero. L'esperimento dispone di 3 stati di gestione e di n stati di intervento che sono determinati dalle n combinazioni possibili ottenibili tramite le varie configurazioni degli interruttori. Generalizzando ad un unico stato di intervento, e caratterizzando lo stato di intervento in cui è necessario effettuare il rapporto, si possono riassumere tutti gli stati sperimentati dall'agent durante un episodio mediante la lista seguente:

- WAKE_UP: stato di gestione di partenza dell'episodio raggiunto dal dispositivo dopo essersi svegliato dallo sleep.
- STANDARD: stato di gestione determinato dopo il sense in cui tutti i valori dei parametri ambientali sono considerati accettabili.
- TERMINAL: stato di gestione in cui l'episodio sperimentato termina. Questo stato viene raggiunto quando il dispositivo compie l'azione di sleep o quando il dispositivo compie l'azione di report nello stato di intervento REPORT.
- INTERVENTION: stato di intervento determinato dopo l'azione di sense. Viene identificato univocamente dalla configurazione degli interruttori associato ai parametri. In questo l'agent comunica le informazioni dei parametri fuori standard ai relativi attuatori che li gestiscono.
- REPORT: stato in cui tutte le trasmissione per attivare gli attuatori sono state effettuate. Risulta a tal punto necessario comunicare al data center i dati dell'episodio.

Dunque alla luce di quanto detto fino ad ora l'intero insieme degli stati del problema sono: 1 stato WAKE_UP, 1 stato STANDARD, 17 stati di INTERVENTION in cui

almeno un parametro ha stato diverso da 0, 1 stato di REPORT in cui tutti gli interruttori hanno stato 0 dopo aver effettuato gli opportuni interventi e uno stato TERMINAL. In totale, quindi, l'agent può sperimentare 21 stati. Considerato il numero di stati ed il numero di azioni effettuabili sarebbe poco chiaro realizzare una rappresentazione grafica delle transizioni degli stati mediante le azioni. Per fronteggiare il problema si considera un caso più semplice costituito dal solo parametro temperatura e dalla sola azione di intervento correlata a questo parametro. Le transizioni dello scenario semplificato sono illustrate nel diagramma delle transizioni della figura 2.9. In tale scenario semplificato le azioni disponibili sono quindi:

- sense: consente di ottenere il valore della temperatura attuale.
- temperature intervention: comunica il valore della temperatura all'attuatore che interviene sulla temperatura.
- report: invia i dati dell'episodio al data center.
- sleep: entra in modalità risparmio energetico.

L'insieme di tutti gli stati possibili nello scenario semplificato è rappresentato invece dalla seguente lista:

- WAKE_UP: rappresenta lo stato di partenza che viene raggiunto al momento del risveglio del dispositivo.
- STANDARD: stato raggiunto se l'operazione di sense ha restituito una configurazione della temperatura pari a 0 in cui non c'è bisogno di dover fare nulla.
- OUT_OF_RANGE_T: stato raggiunto se l'operazione di sense restituisce una configurazione diversa da 0 in cui è necessario intervenire sulla temperatura.
- REPORT: stato in cui l'intervento sulla temperatura è stato effettuato ed è necessario fare rapporto dei risultati al data center.
- TERMINAL: stato raggiunto dopo aver effettuato lo sleep o dopo aver effettuato l'azione di report nell'opportuno stato di report.

La rappresentazione degli stati è fondamentale perché il dispositivo smart IoT, per poter prendere le sue decisioni, si affida ad un agente di Q-learning che necessita di conoscere in che stato si trovi l'ambiente che lo ospita. La mappatura delle azioni che è meglio compiere negli stati di interesse risiede nella Q-table utilizzata dall'agent. Nel sistema software di sperimentazione la Q-table viene implementata tramite un array bidimensionale numpy. Le righe della matrice rappresentano gli stati dell'ambiente e vengono indicizzate, come per le liste, utilizzando indici interi.

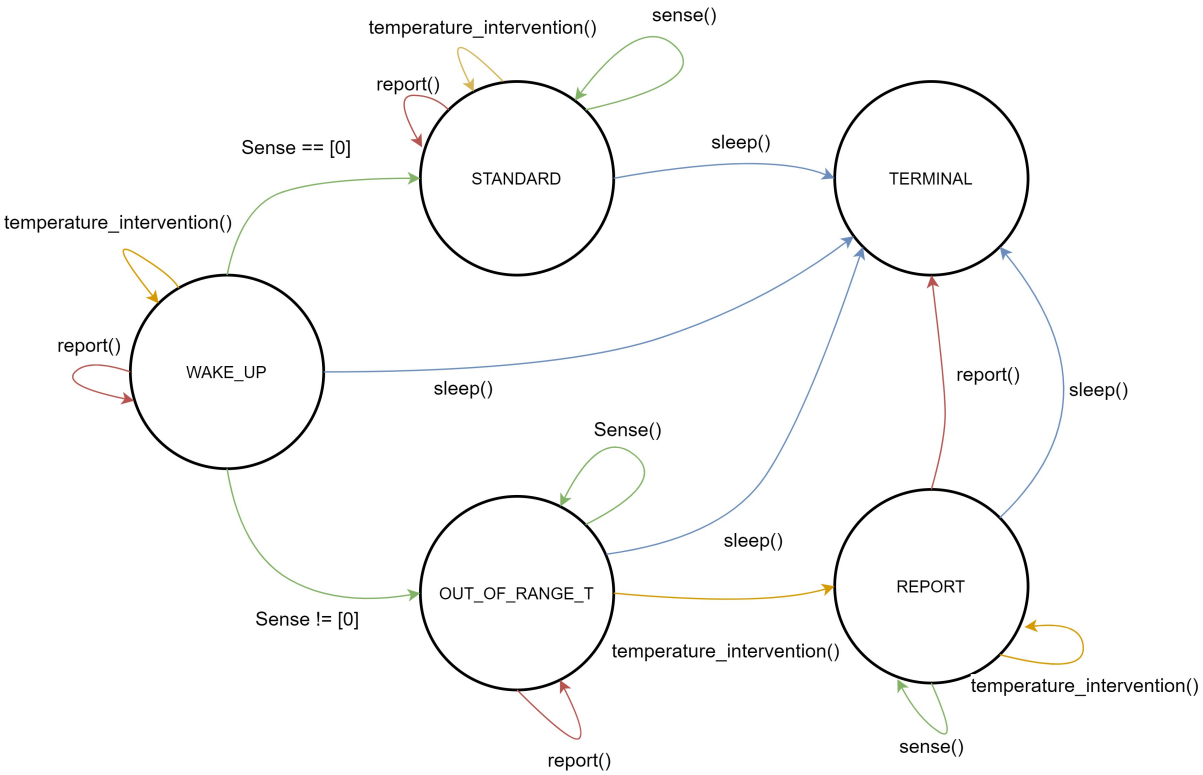


Figura 2.9: Transizioni tra gli stati dello scenario semplificato.

A questo punto sorge una questione da tenere in considerazione, le configurazioni degli interruttori sono liste di interi che devono essere convertiti in qualche modo in un indice intero utilizzato per accedere alle righe della Q-table. Una prima soluzione, non utilizzata nell'esperimento, consiste nel ricavare un numero in base dieci utilizzando in modo ordinato i numeri degli stati degli interruttori. In questo modo si genera una lista di n posizioni dove n è il più grande numero ottenuto dall'estrazione ordinata dei numeri della lista della configurazione. La lista generata contiene però molti spazi vuoti perché solo i numeri estratti dalle configurazioni degli interruttori rappresentano stati reali. Per esempio se la configurazione degli interruttori è 1,2,2 allora si ottiene 233 applicando l'estrazione. A questo punto si genera una lista di 122 posizioni in cui solo 18 posizioni sono utilizzate e 104 no. Le 18 posizioni utilizzate sono le 18 combinazioni possibili degli interruttori: 0,0,0 = 0, 1,0,0 = 1, 1,1,0 = 11, 1,1,1 = 111 e così via fino ad ottenere tutte e 18 le combinazioni. Una Q-table generata in questo modo risulta essere però poco efficiente dal punto di vista del consumo dello spazio del dispositivo.

Nell'esperimento la questione è stata risolta implementando un sottosistema di rappresentazione numerica in base mista. Il sottosistema si occupa infatti di trasformare la configurazione degli interruttori in un numero a base mista che viene successivamente

convertito in numero in base dieci all'occorrenza. Il numero risultante in base dieci che si ottiene dalla conversione è quello che viene utilizzato per interrogare la Q-table. Il numero in base mista viene utilizzato invece per fornire al dispositivo informazioni in merito allo stato di intervento incontrato. Ulteriori dettagli sul sottosistema di rappresentazione numerica in base mista sono riportati nell'apposita sottosezione.

Per concludere la sezione, possiamo riassumere i concetti affermando che l'agent sperimenta due circuiti decisionali. Un circuito è caratterizzato dagli stati di intervento, l'altro circuito è caratterizzato dagli stati di gestione. Per esempio, è possibile immaginare che il dispositivo abbia determinato una configurazione degli interruttori che coincide con 1,0,0, questo stato indica che è necessario tentare di effettuare nuovamente la connessione senza intervenire sugli altri parametri perché assumono valori entro i limiti stabiliti. Il dispositivo compie la giusta azione quando effettivamente compie l'azione reconnect. A prescindere dal successo o dal fallimento dello stabilimento della connessione il dispositivo è intervenuto e quindi l'interruttore relativo al parametro wifi deve essere cambiato e quindi viene modificato da 1 a 0. La configurazione degli interruttori è diventata quindi 0,0,0, ciò significa che il dispositivo ha raggiunto lo stato di REPORT e può concludere il suo task effettuando l'azione di report al data center. La configurazione 0,0,0 è una condizione particolare, essa infatti significa due cose diverse in base al contesto in cui viene incontrato:

- Caso 1: Il dispositivo si trova nello stato WAKE_UP, effettua l'azione di sense e percepisce una configurazione 0,0,0. In questo caso la configurazione 0,0,0 significa che l'ambiente si trova in una condizione ideale è non c'è nulla da fare. Dunque in questo caso aver compiuto l'azione di sense produce la transizione dallo stato WAKE_up allo stato STANDARD. Da questo stato il dispositivo ha convenienza ad entrare in sleep e se lo fa si verifica la transizione dallo stato STANDARD verso lo stato TERMINAL, l'episodio si conclude.

- Caso 2: Il dispositivo si trova nello stato WAKE_UP, effettua l'azione di sense e percepisce una configurazione diversa da 0,0,0 come per esempio 1,0,0. In questo caso il dispositivo si sposta nella sezione di intervento perché si trova in uno stato di intervento. Come mostrato in precedenza la configurazione 1,0,0 significa che è necessario tentare nuovamente di connettersi alla rete Wi-Fi. Il dispositivo effettua dunque l'azione reconnect e la configurazione passa da 1,0,0 a 0,0,0. Il dispositivo si sposta dallo stato 1,0,0 allo stato 0,0,0 che corrisponde allo stato REPORT. In questo stato di intervento il dispositivo trasmette i dati al data center. Una volta compiuta l'azione di rapporto avviene la transizione verso lo stato TERMINAL e l'episodio si conclude.

Rappresentazione numerica in base mista

Il *sottosistema di rappresentazione numerica a base mista* mette a disposizione una funzionalità molto importante per l'intero sistema di sperimentazione. Questa funzionalità consente di rappresentare gli stati particolari dell'ambiente che prendono il nome di stati di intervento. Uno stato di intervento è uno stato molto importante che richiede al dispositivo smart IoT di intervenire per modificare il valore dei parametri fuori standard. La rappresentazione di questi stati particolari avviene mediante una scrittura che consente di descrivere la configurazione degli stati assunti dagli interruttori associati ai parametri ambientali. Consideriamo quindi un interruttore A che può assumere gli stati a_1 e a_2 e un interruttore B che può assumere gli stati b_1 e b_2. Quando l'interruttore A si trova nello stato a_2 mente B si trova nello stato b_2 ne risulta che gli interruttori sono in una precisa configurazione che descrive uno stato di intervento. Il sottosistema si basa sul numero di stati che possono essere assunti dagli interruttori associati ai parametri ambientali. Per capire meglio il meccanismo si considera un ambiente ipotetico descritto dai parametri: temperatura, wifi, umidità e voltaggio. Gli interruttori associati ai parametri dell'ambiente ipotetico hanno le caratteristiche descritte nella tabella 2.11: Le possibili combinazioni che gli interruttori associati ai parametri ambientali possono

Tabella 2.11: Informazioni sugli interruttori associati ai parametri che caratterizzando l'ambiente dell'esperimento.

Nome interruttore/parametro	*Possibili stati*	*Nome degli stati possibili*
temperatura	3	Ideale, intervento e stop intervento.
umidità	3	Ideale, intervento e stop intervento.
voltaggio	2	ideale e intervento.
wifi	2	ideale e intervento.

generare si ottengono moltiplicando il numero dei possibili stati assumibili. In questo caso dunque si consulta la tabella 2.11 e si calcola che le possibili combinazioni sono $3 * 3 * 2 * 2 = 36$. Una configurazione è quindi una lista di interi rappresentanti ciascuno lo stato in cui si trova uno specifico interruttore. Per esempio 1,1,1,1 indica che gli interruttori di temperatura, umidità, voltaggio e wifi sono nello stato 1 che significa intervento richiesto su ogni parametro. Quella appena descritto è una configurazione degli interruttori e quindi uno stato di intervento. Ogni combinazione possibile necessita di un numero che la identifichi univocamente, questo numero è la configurazione stessa che per essere utilizzata va regolamentata da regole ben precise e sempre valide. Il sottosistema è stato implementato per risolvere il problema dell'identificazione univoca degli stati di intervento tramite l'utilizzo di un sistema numerico a base mista. Oltre alla rappresentazione è stato necessario di conseguenza fornire un metodo bidirezionale

di conversione dalla base mista alla base decimale del numero che rappresenta lo stato di intervento.

Il sottosistema di rappresentazione entra in azione sin dal momento in cui viene creata la lista degli interruttori associati ai parametri. Infatti, dopo la creazione, la lista degli interruttori viene riordinata in ordine crescente di numero degli stati possibili come spiegato nella sezione della generazione degli interruttori. Prendendo in considerazione lo scenario ipotetico descritto in questa sezione è possibile osservare uno dei possibili ordinamenti validi della lista degli interruttori osservando la tabella 2.12. Durante

Tabella 2.12: Esempio di lista degli interruttori ordinata in modo valido per la il sottosistema di rappresentazione numerica a base mista.

Posizione 0	*Posizione 1*	*Posizione 2*	*Posizione 3*
2 (stati di wifi)	2 (stati di voltaggio)	3(stati di temperatura)	3 (stati di umidità)

l'ordinamento, purché si rispetti l'ordine ascendente del numero dei possibili stati dell'interruttore, non ha importanza quale interruttore sia posizionato prima degli altri interruttori dotati dello stesso numero di possibili stati. La lista illustrata nella tabella 2.12 rappresenta quindi il sistema di numerazione a base mista utilizzato negli esempi di questa sezione.

Considerando il caso preso in esame potremmo utilizzare la scrittura 1,1,0,2 per descrivere una configurazione di esempio. La configurazione descritta significa:

- Necessità di effettuare la connessione alla rete wifi.
- Necessità di intervenire sul voltaggio di input.
- Assenza di necessità di intervento sulla temperatura.
- Necessità di arrestare gli effetti ancora in atto sul valore di umidità causati del precedente intervento.

La stessa configurazione appena descritta potrebbe essere rappresentata dunque mediante un numero in base mista che risulterebbe essere 2011. Il numero 2011 è il risultato della rimozione delle virgole e dell'inversione dei numeri utilizzati per descrivere la configurazione degli stati degli interruttori. Per esempio, il numero 2011 nella base mista 3322 corrisponde al numero decimale 27. Il numero in forma decimale viene utilizzato per accedere alla riga 27 della Q-table corrispondente allo stato di intervento 2011.

Nella prima posizione della lista mostrata dalla tabella 2.12 si ha la posizione del sistema di misurazione di peso minore. La base di tale posizione risulta essere 2, ciò significa che sono ammessi esclusivamente i due simboli 0 e 1. In sostanza i simboli

ammessi nella posizione che ha una determinata base di valore b sono i numeri che vanno da 0 fino a $b-1$. Lo stesso ragionamento vale quindi per la seconda posizione della lista che è anch'essa è in base 2. In terza e quarta posizione sono consentiti i tre simboli 0,1 e 2 perché la base della posizione è 3. Quindi il numero più grande rappresentabile è 1122 che per essere letto va invertito diventando il numero 2211 in base mista che in base dieci corrisponde al numero 35. Quindi come anticipato si possono rappresentare 36 combinazioni che partono dalla 0-esima ed arrivano alla 35-esima combinazione.

La conversione di un numero dalla base mista alla base decimale avviene mediante il seguente algoritmo:

1. Necessario prendere in input una lista di interruttori ordinata in ordine crescente di numero dei possibili stati. Questa lista indica il sistema di numerazione a base mista utilizzato.

2. Identificare le basi b del sistema di numerazione a base mista scelto. Identificate le basi vanno estratte prendendole una volta sola. In questo caso le base sono 2, 2, 3 e 3 ma prendendole una volta sola si ottiene $b_1 = 2$ e $b_2 = 3$.

3. Determinare il peso p_n di ogni posizione prevista dal sistema di numerazione. Le posizioni del sistema di numerazione corrispondono alle posizioni della lista e possono essere considerati come gli alloggiamenti delle cifre che compongono il numero in base mista. Il peso di una posizione consiste nel prodotto di m potenze dove m è il numero di basi b identificate al passo 2. Ogni potenza ha come base le basi b identificate al passo 2. Le potenze hanno per esponente un numero che viene incrementato progressivamente in base alle 4 casistiche indicate di seguito:

   (a) La prima posizione in assoluto ha tutte le potenze con base pari a 0. In questo caso la prima posizione ha base $\beta_1 = 2$ e seguendo la regola appena descritta il si ha che $p_1 = 2^0 \cdot 3^0 = 1 \cdot 1 = 1$

   (b) Raggiungere una posizione che ha base $\beta_n = \beta_{n-1}$ impone l'incremento dell'esponente della potenza che ha per base β il numero che corrisponde con la base trovata nella posizione raggiunta. In sostanza se $\beta_{n-1} = \beta_n = 2$ allora si incrementa l'esponente della potenza che ha come base il 2, dunque $p_2 = 2^1 \cdot 3^0 = 2 \cdot 1 = 2$.

   (c) Raggiungere una posizione con una base β_n diversa da quella della posizione precedente β_{n-1} comporta l'incremento della potenza che ha per base il numero della base della posizione precedente e cioè la potenza con base uguale a β_{n-1}. In questo caso dunque la posizione 3 del nostro esempio ha base $\beta_n = 3$ mentre la posizione precedente aveva base $\beta_{n-1} = 2$, quindi si ha che $p_3 = 2^2 \cdot 3^0 = 4 \cdot 1 = 4$. Questo passaggio si chiama passaggio di transizione.

(d) Se la posizione precedente ha sperimentato il passaggio di transizione e la posizione attuale ha base identica alla posizione del passaggio di transizione, cioè se succede che $\beta_{n-1} = \beta_n$, allora si verifica l'arresto dell'incremento dell'esponente della potenza che ha come base il numero della base ormai non più presente nella posizione attuale, cioè quella con $\beta_{n-2} \neq \beta_{n-1}$ e di conseguenza $\beta_{n-2} \neq \beta_n$ dal momento che $\beta_{n-1} = \beta_n$. In più viene incrementato l'esponente della potenza che ha per base β_n che in questo caso è 3. Nel caso considerato dunque, arrivati alla posizione quattro si nota che la base è di nuovo 3 come è successo per la posizione 3 e si nota anche che la posizione 3 ha sperimentato il passaggio di transizione. Si ha dunque come risultato che $p_4 = 2^2 \cdot 3^1 = 4 \cdot 3 = 12$.

Il procedimento viene iterato fino al termine delle posizione della lista rappresentante il sistema di rappresentazione numerica in base mista.

4. A questo punto, conosciuti tutti i pesi, è necessario conoscere il valore della posizioni del numero in base mista. La formula per calcolare il valore è $v_n = c_n \cdot p_n$ dove c_n rappresenta il simbolo utilizzato per la cifra presente nella posizione n esaminata. Invece p_n è il peso della posizione n esaminata. Per esempio prendendo in considerazione il numero 1000 e facendo riferimento alla prima posizione si ha che $v_1 = c_1 \cdot p_1 = 1 \cdot 1 = 1$. Si procede nell'identificazione de valori di tutte le posizioni del numero che in questo caso risulta essere 0 in tutte le posizioni diverse dalla prima per via del simbolo 0 associato alle cifre.

5. Conosciuti tutti i valori di tutte le posizioni il numero convertito dalla base mista alla base decimale lo si ottiene effettuando la somma di tutti i valori trovati. La formula di conversione è dunque la seguente $n = v_1 + ... + v_n$.

Un esempio completo di conversione da base mista a base decimale verrà illustrato di seguito utilizzando il sistema di rappresentazione numerica in base mista mostrato nella tabella 2.12. Si considera la configurazione degli interruttori 1102. Sappiamo che la rappresentazione viene espressa sotto forma di numero invertendo la scrittura, quindi nell'esempio si effettuerà la conversione del numero in base mista 2011. Si inizia il processo di conversione partendo dalla cifra di peso minore che è quella all'estrema destra. Si procede, passo dopo passo, verso la cifra di peso maggiore che si trova all'estrema sinistra.

1. Conversione posizione 1: $c_1 = 1$, $p_1 = 2^0 \cdot 3^0 = 1$, $v_1 = c_1 \cdot p_1 = 1 \cdot 1 = 1$.

2. Conversione posizione 2: $c_2 = 1$, $p_2 = 2^1 \cdot 3^0 = 2$, $v_2 = c_2 \cdot p_2 = 1 \cdot 2 = 2$.

3. Conversione posizione 3: $c_3 = 0$, $p_3 = 2^2 \cdot 3^0 = 4$, $v_3 = c_3 \cdot p_3 = 0 \cdot 4 = 0$.

4. Conversione posizione 4: $c_4 = 2$, $p_4 = 2^2 \cdot 3^1 = 12$, $v_4 = c_4 \cdot p_4 = 2 \cdot 12 = 24$.

5. Risultato della conversione: $n = v_1 + v_2 + v_3 + v_4 = 1 + 2 + 0 + 24 = 27$.

La conversione da base decimale a base mista avviene utilizzando il metodo della sottrazione. Come già detto in precedenza ogni posizione può assumere un limitato numero di simboli dettato dalla base della posizione. Quindi ogni posizione ha in totale i simboli dove i è dettato dalla base della posizione. Le posizioni del sistema di numerazione sono al massimo m. Il numero in base dieci da convertire in numero in base mista viene indicato con n.

1. Partendo dalla posizione di peso maggiore, che risulta essere la $m - esima$, e considerando il simbolo più grande che quella posizione può assumere, che risulta essere il simbolo $i - esimo$, si effettua la sottrazione $n - c_{m,i} \cdot p_m$ che restituisce la differenza d_m. Il risultato d_m ricade in 3 casistiche.

   (a) $d_m > 0$. In questo caso la posizione è valida per la conversione e quindi si mette da parte il simbolo utilizzato per la cifra di questa posizione. Il numero m viene aggiornato ogni volta che si verifica questo caso nel modo che segue $n = d_m$. A questo punto si considera una nuova posizione e si effettua l'operazione mostrata al passo 1.

   (b) $d_m = 0$. In questo caso si verifica il termine del processo di conversione. Anche in questo caso $n = d_m$ e di conseguenza quando $n = 0$.

   (c) $d_m < 0$. In questo caso la posizione ha un valore troppo grande e quindi eccede il numero da convertire e la cosa non va bene. Risulta necessario ritentare la sottrazione selezionando un cifra più bassa di una unità fino a quando non si raggiunge lo 0. Questa iterazione garantisce di ottenere una $d_m \geq 0$. Il caso in cui la differenza sia maggiore di 0 si verifica sempre a patto che $n > 0$, questo perché se si effettua la differenza $n - 0 = n$ si ottiene che $d_m = n$ e dato che $n > 0$ allora anche $d_m > 0$. Il risultato di questo caso riconduce al caso a oppure al caso b.

Si illustra di seguito un esempio di conversione di un numero dalla base dieci alla base mista. Dal momento che le cifre delle posizioni del numero in base mista sono inizialmente sconosciute, si utilizzerà il simbolo "c_n" come segnaposto per la cifra di tale numero che si trova in posizione n. Dunque, quello che avviene ad ogni passo del metodo illustrato di seguito, è una costruzione cifra dopo cifra del numero in base mista. Ad ogni passo della tecnica illustrata si verificherà la sostituzione dell'n-esimo segnaposto con la n-esima cifra che compone il numero in base mista. Il numero in questione è $n = 27$ che verrà convertito nel numero $mixed = \ c_4c_3c_2c_1$ rappresentato secondo il sistema di rappresentazione mostrato nella tabella 2.12.

1. Si parte dalla posizione di peso maggiore che risulta essere la quarta con $p_4 = 2^2 \cdot 3^1 = 12$. Si calcola il valore più alto possibile per questa posizione che si ottiene

quando $c_4 = 2$ restituendo $v_4 = p_4 \cdot c_4 = 12 \cdot 2 = 24$. A questo punto si calcola $d_4 = n - v_4 = 27 - 24 = 3$. In questo caso $d > 0$ che corrisponde al caso a. Si tiene da parte il simbolo c_4 della posizione esaminata e si inizia a trascrivere il risultato $mixed = 2c_3c_2c_1$. A questo punto si aggiorna il numero da convertire e di conseguenza si ha che $n = d_4 = 3$.

2. Si procede verso la posizione 3 dove $p_3 = 2^2 \cdot 3^0 = 4$ e il simbolo più alto possibile è $c_3 = 2$. Si ottiene $v_3 = p_3 \cdot c_3 = 4 \cdot 2 = 8$. A questo punto si calcola $d_3 = n - v_3 = 3 - 8 = -5$ ricadendo nel caso c in cui $d < 0$. Risulta dunque necessario riprovare con un simbolo della posizione più basso. Si prova con $c_3 = 1$ ottenendo $v_3 = 4$ e una differenza $d_3 = n - v_3 = 3 - 4 = -1$. Non va ancora bene perché si è ancora nel caso c. La posizione non può essere utilizzata e quindi la si segna come 0 perché se $c_3 = 0$ allora $v_3 = 4 * 0$ e di conseguenza $d_3 = 3 - 0 = 3$ che corrisponde al caso a. A questo punto siamo nel caso b e si tiene traccia del simbolo della cifra di questa posizione ottenendo $mixed = 20c_2c_1$ ed $n = n - 0$ che quindi resta invariato.

3. In questo step si raggiunge la posizione 2, qui la base è 2 e quindi sono ammessi solo i simboli 0 e 1. Andando più velocemente e seguendo quanto illustrato fin'ora ci si accorge che per questa posizione è necessario usare $c_2 = 1$ per poter ottenere $d_2 = 3 - 2 = 1$. Siamo nel caso a e quindi teniamo traccia del simbolo e lo aggiungiamo al risultato ottenendo $mixed = 201c_1$. A questo punto si ha che $n = 1$.

4. In questo step si raggiunge la posizione 1 che è l'ultima. La base qui è 2 e l'algoritmo deve terminare ottenendo una $d_1 = 0$. Per farlo ci si accorge che è necessario usare una $c_1 = 1$ che restituisce appunto $d_1 = 1 - 1 = 0$ e quindi $n = 0$. Ci si ritrova nel caso a, si tiene traccia del simbolo della cifra di questa posizione e si ottiene che $mixed = 2011$. Tutte le posizioni sono state identificate, $d_1 = 0$ e dunque il procedimento è terminato con successo.

Il risultato dell'esempio mostra che il sistema di conversione bidirezionale funziona. Mediante i due esempi è stato possibile passare dal numero 2011 in base mista al numero 27 in base dieci per poi tornare nuovamente al numero 2011 in base mista.

Il sottosistema di rappresentazione numerica in base mista consente di suddividere la Q-table in sezioni. Nell'esperimento infatti la Q-table è dotata di n righe ottenute dalla somma di m righe e $k = 3$ righe che generano appunto una tabella organizzata nel modo seguente:

- Le prime m righe della Q-table rappresentano le combinazioni degli interruttori che corrispondono agli stati di intervento. Questa porzione di Q-table viene chiamata porzione di intervento.
- Le restanti $k = 3$ righe della Q-table rappresentano gli stati di gestione del dispositivo stesso. Questa sezione si chiama porzione di gestione.

La suddivisione della Q-table utilizzata nell'esperimento di tesi è quella mostrata nella figura 2.10. Gli indici delle colonne corrispondono alle azioni descritte nella lista:

- 0: trasmissione valore di temperatura all'attuatore che interviene sulla temperatura.
- 1: trasmissione valore di umidità all'attuatore che interviene sull'umidità.
- 2: procedura di connessione alla rete.
- 3: sensing del valore dei parametri ambientali.
- 4: trasmissione delle informazioni di intervento al data center.

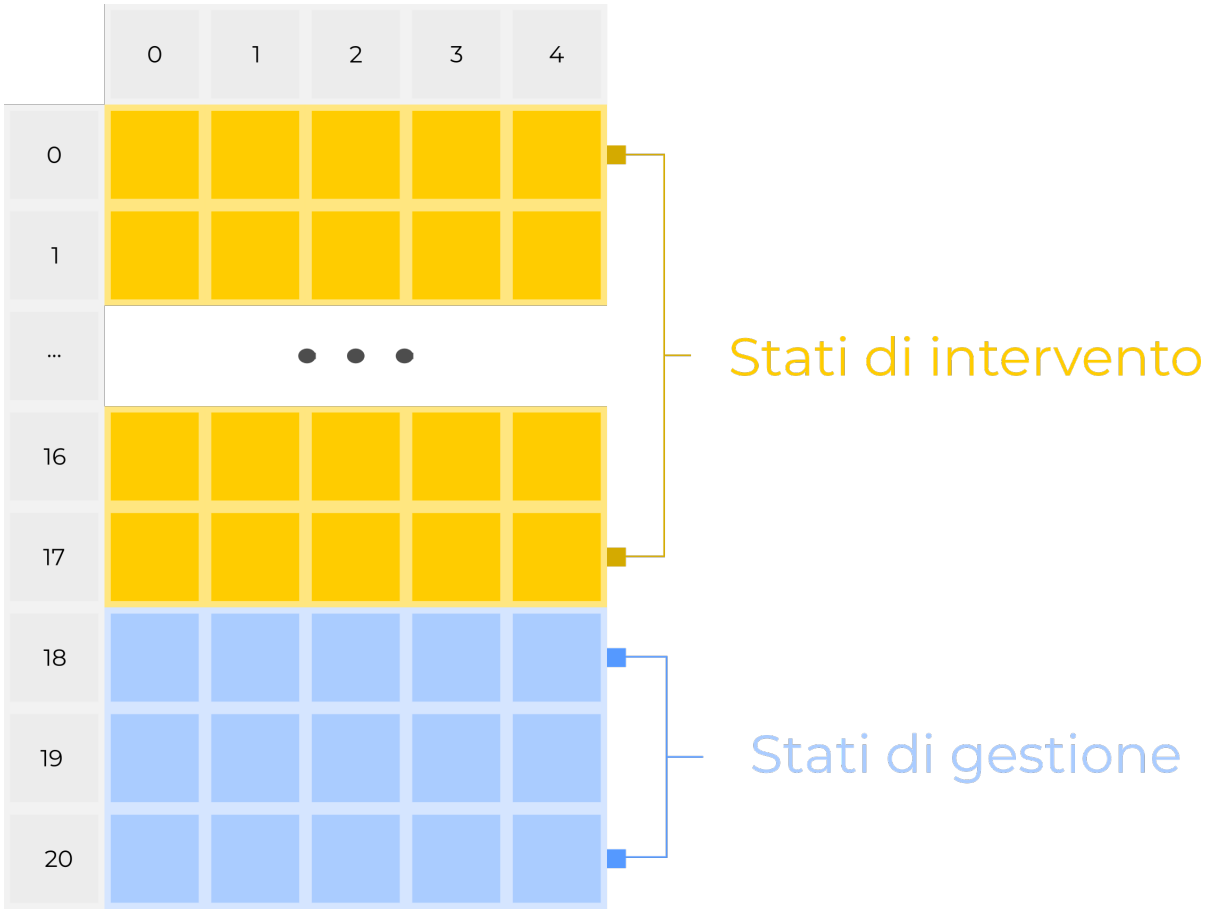


Figura 2.10: Suddivisione della Q-table in stati di intervento e stati di gestione.

Il funzionamento di questo sottosistema viene riassunto dall'esempio che segue. Il dispositivo smart IoT percepisce l'ambiente mediante l'azione sense, in questo modo viene

a conoscenza dei valori che i parametri ambientali stanno assumendo in quel momento. Il sottosistema di rappresentazione dello stato ambientale riesce di conseguenza a ricostruire la configurazione degli stati degli interruttori associati ai parametri. Il sistema di rappresentazione numerica in base mista entra in gioco fornendo, all'intero sistema software di sperimentazione, un meccanismo di corrispondenza univoca e diretta tra il numero che indica la configurazione degli interruttori e il numero che indica la relativa riga della Q-teble.

La strategia di numerazione adottata e la suddivisione della Q-table in sezioni hanno consentito la modellazione del particolare tipo di problema esaminato nell'esperimento di tesi. La metodologia seguita ha consentito inoltre di implementare una doppio circuito di decisioni che il dispositivo si trova a sperimentare in base al risultato dell'azione sense. Un circuito è quello che il dispositivo sperimenta quando si occupa della risoluzione di un problema di monitoraggio ambientale, ad esempio quando il dispositivo si accorge di dover riscaldare o raffreddare l'ambiente. Un secondo circuito è quello che il dispositivo sperimenta quando si risveglia e valuta la convenienza di entrare in risparmio energetico.

Il codice utilizzato per la conversione da base mista a base dieci viene mostrato nel listing A.9.

Sistema decisionale

Questo sottosistema consente all'agente che governa il dispositivo smart IoT di prendere decisione in merito alle azioni che è meglio compiere in base allo stato ambientale incontrato. Il sottosistema è stato implementato nel sistema attraverso il modulo agent.py il cui codice viene illustrato nella sezione listing A.10.

Come mostrato dalla sezione del listing A.10, l'agent compie scelte epsilon greedy. Il valore epsilon viene definito al momento della creazione dell'agent e la casualità delle scelte viene decrementata all'aumentare del numero di scelte compiute. A volte può capitare che in una riga della Q-table tutte o alcune colonne abbiano lo stesso valore e che tale valore sia il massimo di quella riga. In tal caso la scelta avviene in modo casuale tra le azioni di pari valore. Gli aggiornamenti ai valori di qualità della Q-table avvengono mediante la funzione di bootstrap il cui funzionamento è stato descritto nel capitolo dedicato allo stato dell'arte.

La Q-table come detto nelle altre sezioni della tesi è suddivisa in due porzioni: la sezione di intervento e la sezione di gestione. La sezione di intervento è composta da 18 righe che si ottengono calcolando il numero di combinazioni possibili che gli interruttori dei parametri possono generare. Dunque dalla posizione 0 alla posizione 17 sono presenti gli stati di intervento rappresentati mediante il procedimento spiegato nella sezione dedicata al sottosistema di rappresentazione numerica in base mobile. Dalla posizione 18 si ha invece l'inizio della sezione di gestione costituita dai seguenti stati:

- WAKE_UP che si trova in posizione 18.
- STANDARD che si trova in posizione 19.
- TERMINAL che si trova in posizione 20

Il sistema è in grado di generare la divisione tra le due sezioni in autonomia calcolando il numero di possibili combinazioni degli interruttori e predisponendo nelle ultime 3 posizioni della Q-table gli stati di gestione.

Quando l'agent compie una decisione il dispositivo ottiene come risultato il numero della colonna della Q-table che corrisponde all'identificativo di una delle azioni a disposizione del dispositivo che sono memorizzate in una lista del sistema delle azioni. Il procedimento di identificazione delle azioni è stato spiegato nella sezione dedicata al sottosistema delle azioni. Quindi, l'identificativo numerico dell'azione serve a determinare la coppia stato azione utile alla somministrazione della ricompensa all'agent e serve a consentire l'esecuzione delle azioni decise.

Sistema delle ricompense

Questo sottosistema serve a ricompensare positivamente o negativamente le azioni decise dall'agent e compiute dal dispositivo. Il meccanismo si basa sull'utilizzo di una R-table. La R-table è dimensionalmente identica alla Q-table. All'interno delle celle della R-table ci sono le ricompense delle coppie stato-azione previste dal problema sperimentale. Nella R-table il valore di default è -1 per tutte le celle. Per poter definire il lavoro è stata modificata la R-table attribuendo un punteggio di 100 alla coppia stato-azione che descrive la condizione in cui il dispositivo effettua l'azione di report quando si trova nello stato REPORT. I comportamenti sbagliati sono stati evitati attribuendo -100 punti alle azioni di sleep durante gli stati di intervento. Le colonne della riga dello stato finale, cioè le colonne dell'ultima riga della R-table, hanno ricompensa pari a 0. La R-table risultante che è stata utilizzata nell'esperimento è rapresentata in figura 2.11.

```
R-table
[[  -1.   -1.   -1.   -1.  100. -100.]
 [  -1.   -1.   -1.   -1.   -1. -100.]
 [  -1.   -1.   -1.   -1.   -1. -100.]
 [  -1.   -1.   -1.   -1.   -1. -100.]
 [  -1.   -1.   -1.   -1.   -1. -100.]
 [  -1.   -1.   -1.   -1.   -1. -100.]
 [  -1.   -1.   -1.   -1.   -1. -100.]
 [  -1.   -1.   -1.   -1.   -1. -100.]
 [  -1.   -1.   -1.   -1.   -1. -100.]
 [  -1.   -1.   -1.   -1.   -1. -100.]
 [  -1.   -1.   -1.   -1.   -1. -100.]
 [  -1.   -1.   -1.   -1.   -1. -100.]
 [  -1.   -1.   -1.   -1.   -1. -100.]
 [  -1.   -1.   -1.   -1.   -1. -100.]
 [  -1.   -1.   -1.   -1.   -1. -100.]
 [  -1.   -1.   -1.   -1.   -1. -100.]
 [  -1.   -1.   -1.   -1.   -1. -100.]
 [  -1.   -1.   -1.   -1.   -1. -100.]
 [  -1.   -1.   -1.   -1.   -1. -100.]
 [  -1.   -1.   -1.   -1.   -1.   -1.]
 [   0.    0.    0.    0.    0.    0.]]
```

Figura 2.11: R-table utilizzata nell'esperimento di tesi

Il compito della R-table è quello di definire il lavoro del dispositivo intelligente, per raggiungere l'obiettivo è necessario inserire nelle celle giuste l'adeguata ricompensa positiva. L'unica ricompensa positiva inserita fa capire all'agente che il suo lavoro è gestire tutti i problemi di monitoraggio e, in fine, comunicare le informazioni dell'episodio al data center. La modifica dei valori di default della R-table al fine di definire il lavoro del dispositivo avviene mediante l'iterazione di una lista popolata dalle istruzioni relative alle celle della R-table da modificare.

Il sottosistema è stato implementato nel sistema software di sperimentazione mediante il modulo r_table.py il cui codice viene riportato nella sezione listing A.11 presente nella pagina successiva.

Sistema delle azioni

Come previsto dal Reinforcement Learning, anche all'interno del sistema software di sperimentazione l'agente deve sceglie quali azioni il dispositivo smart IoT deve compiere. Come detto in precedenza la scelta avviene mediante la tecnica di Q-learning con scelta $\epsilon - greedy$. L'agent restituisce al dispositivo l'identificativo dell'azione scelta. Il sottosistema delle azioni è composto da una lista di azioni ordinate seguendo l'ordine delle colonne della Q-table. L'ordinamento descritto consente di utilizzare l'indice delle colonne della Q-table come indice che estrae dalla lista delle azioni gli oggetti capaci di eseguire il codice rappresentativo delle azioni a disposizione del dispositivo. Le azioni sono oggetti implementati come specializzazione della classe Action mostrata nella sezione listing A.12. Nella sezione listing A.13 viene mostrata l'implementazione dell'azione specializzata di sleep.

Una volta che tutte le azioni sono state inserite nella lista delle azioni, viene creata una doppia mappatura che associa la posizione dell'azione nella lista al nome dell'azione. La mappatura viene realizzata utilizzando i dizionari di Python. In questo modo è possibile recuperare ed eseguire l'azione nel caso in cui si conosca il nome dell'azione o nel caso in cui se ne conosca l'identificativo. Il commitment time delle azioni serve a definire quanto tempo il dispositivo è impegnato all'esecuzione dell'azione. Si assume che tutte le azioni abbiano durata istantanea a differenza dell'azione di sleep. Il comportamento delle azioni implementate nel sistema è simulato e serve solo a poter studiare e analizzare il processo di apprendimento dell'agent. Per esempio l'azione sense non utilizza sensori elettronici per acquisire dati ma consente al dispositivo simulato di entrare a conoscenza dei valori che stanno attualmente assumendo i parametri ambientali.

Le azioni sono dotate della funzione execute che viene specializzata in base all'oggetto azione rappresentato e serve ad eseguirne il comportamento.

Inserire un nuovo parametro ambientale richiede l'implementazione di una nuova azione che gestisca il parametro e di conseguenza servirebbe un interruttore e una lista di valori dei parametri. In questo modo il sistema di sperimentazione diventerebbe in grado di allenare dispositivi smart IoT in grado di fronteggiare problemi più complessi. Possono esistere azioni che non sono associate a nessun parametro ambientale come ad esempio l'azione di sleep.

Dal momento che il sistema di sperimentazione simula il comportamento di un agente che apprende in uno scenario on-line, e quindi sperimenta l'apprendimento in real time, è normale che sbagli compiendo azioni che in determinati stati vengano considerate inopportune. Nelle conclusioni verranno espresse considerazioni a riguardo di tale condizione.

Simulatore

Il simulatore è il cuore del sistema di sperimentazione software. Viene implementato attraverso la programmazione dell'oggetto simulator della classe Simulator. Il simulatore, mediante le informazioni e i metodi di cui dispone, riesce a mettere in atto una simulazione ambientale durante la quale vengono riprodotte le dinamiche fisiche osservabili in un ambiente allo scorrere del tempo. Lo scorrere del tempo viene rappresentato con una linea temporale implementata tramite la variabile day_second. Ogni secondo della giornata simulata rappresenta un passo temporale della simulazione. Ad ogni passo temporale i parametri ambientali assumono il corrispondente valore contenuto nella lista dei valori dei parametri ambientali relativi al secondo della giornata raggiunto dalla variabile day_second durante la simulazione. La simulazione ambientale può essere condotta per testare il comportamento del dispositivo diligente, del dispositivo a risparmio energetico e del dispositivo intelligente. Ad ogni passo temporale il simulatore consente al dispositivo simulato di eseguire il suo codice e di svolgere quindi il suo lavoro. I dispositivi semplici eseguono delle semplici istruzioni hard coded if then.

Per quanto riguarda il dispositivo smart IoT la situazione è più complessa. Infatti durante la simulazione è possibile osservare l'intero processo di apprendimento attraverso la tecnica di Q-learning. Durante il processo l'agent che governa il dispositivo impara passo dopo passo a svolgere in modo sempre più efficiente il proprio lavoro. Ad ogni secondo del giorno simulato durante la simulazione, se il dispositivo non si trova nella condizione di sleep, sperimenta un episodio in cui, idealmente, avvengono i seguenti eventi:

- Transizione nello stato WAKE_UP.
- Esecuzione dell'azione sense utilizzata per verificare le condizioni ambientali. In questo modo il dispositivo viene a conoscenza dei valori che la simulazione ha attribuito ai parametri ambientali in riferimento al secondo raggiunto della giornata. A questo punto:
  - Se il dispositivo rileva problemi relativi al valore dei parametri allora si sposta nello stato INTERVENTION idoneo. A questo punto esegue le azioni di intervento raggiungendo lo stato di REPORT ed in fine conclude l'episodio dopo aver effettuato l'azione di report giungendo allo stato TERMINAL.
  - Se il dispositivo non rileva problemi entra nello stato STANDARD in cui esegue l'azione sleep per poter terminare l'episodio nello stato TERMINAL.

Lo scorrere del tempo durante la simulazione è determinato da un ciclo while dei relativi metodi di simulazione dei tre dispositivi programmati. Il tempo della simulazione avanza tenendo in considerazione del commitment time delle azioni se è superiore a 0 oppure avanza arbitrariamente di un secondo per simulare la frequenza massima delle

misurazioni possibili mediante sensore DHT11. La simulazione si svolge entro i limiti stabiliti dal secondo di partenza e dal secondo di fine simulazione che sono compresi tra 0, che rappresenta le ore 00:00:00, e il secondo 86399 che rappresenta l'ora 23:59:59 di un giorno simulato.

La figura 2.12 mostra la sequenza dei messaggi scambiati durante la simulazione che coinvolge il dispositivo smart IoT.

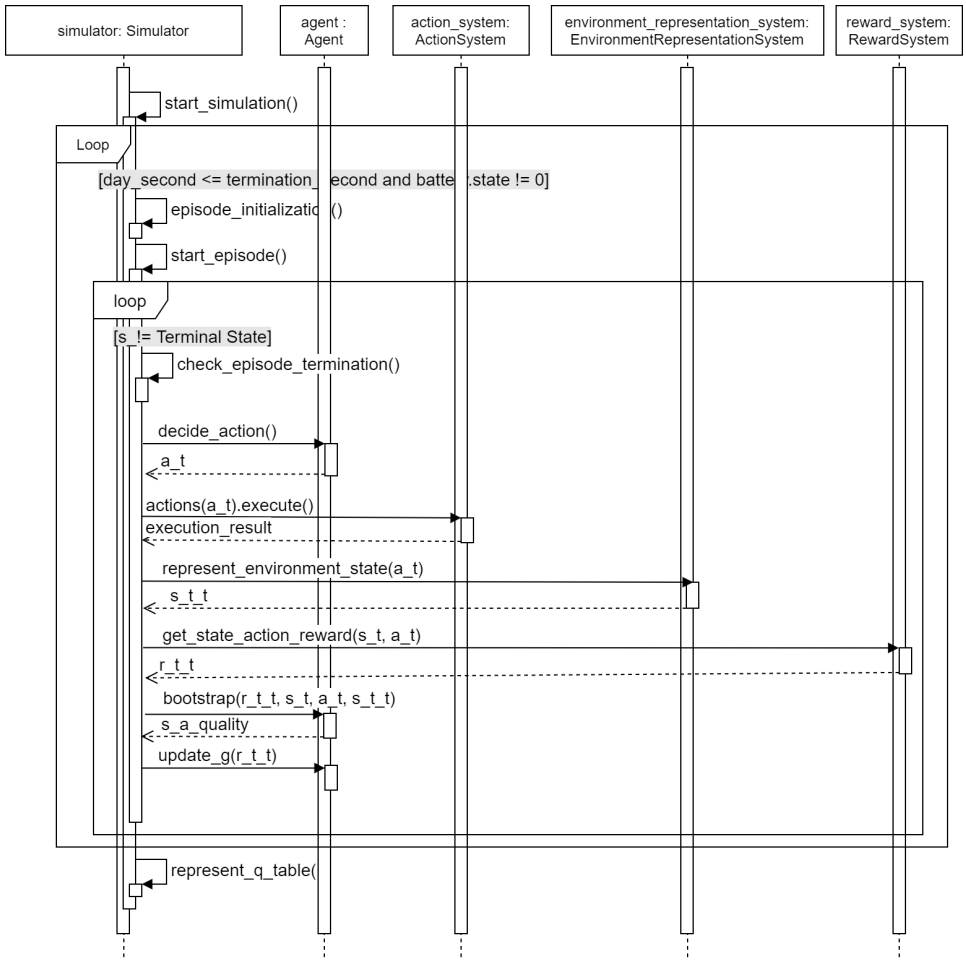


Figura 2.12: Diagramma di sequenza del simulatore.

Le informazioni prodotte durante le simulazioni vengono memorizzate nell'oggetto data_analyst della classe DataAnalyst per essere utilizzate durante la produzione dei grafici e dei file che costituiscono i risultati dell'esperimento di tesi.

Dunque, la simulazione serve a sottoporre ai dispositivi del progetto i valori che idealmente un ambiente assumerebbe se non ci fosse un sistema di monitoraggio ambientale. I vari dispositivi durante la simulazione compiono il loro lavoro di monitoraggio facendo attenzione ai valori dei parametri che la simulazione gli sottopone. In base ai risultati dell'azione di sense e mediante le strategie adottate dai singoli dispositivi, si verifica un'influenza dei parametri ambientali ai quali vengono fatti assumere valori accettabili ottenuti mediante l'intervento degli attuatori simulati. L'influenza dei parametri avviene quindi mediante la simulazione delle comunicazioni tra i dispositivi di monitoraggio e i dispositivi attuatori di influenza dei parametri. A seguito della comunicazione simulata, il sistema simula anche l'intervento degli attuatori che si manifesta con l'effettiva modifica dei valori dei parametri ambientali. I valori vengono modificati in base al comportamento dell'attuatore che si sta simulando in quel momento. Per esempio se il valore della temperatura ambientale è 20° ed eccede il limite superiore consentito allora il dispositivo di monitoraggio comunica la temperatura all'attuatore della temperatura che deciderà di attivare il sistema di refrigerazione. La temperatura misurata che risultava essere 20° diventa invece un numero calcolato come 20° - delta, il risultato rappresenta il valore influenzato dall'intervento dell'attuatore. Il sistema software di sperimentazione è dunque in grado di sperimentare il riscaldamento, il raffreddamento, l'umidificazione e la de-umidificazione dell'ambiente. Al termine della simulazione e dopo aver inviato i dati al sistema di analisi dei dati è possibile vedere cosa sia successo ai parametri ambientali guardando al grafico della simulazione riportato in figura 2.13

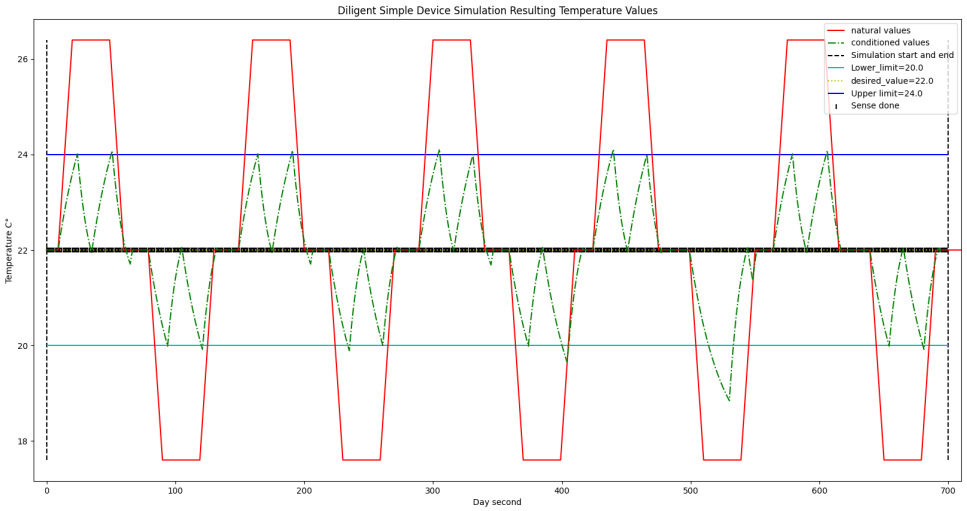


Figura 2.13: Grafico del risultato della simulazione condotta con un dispositivo semplice diligente.

Analisi dei dati

Il sottosistema di analisi dei dati si occupa dell'analisi e della rappresentazione dei dati prodotti dal sistema di sperimentazione software durante gli esperimenti simulati condotti per ogni dispositivo del progetto. I dati in questione vengono memorizzati nelle variabili istanza dell'oggetto data_analyst della classe DataAnalyst ed utilizzati dai vari metodi della stessa classe al termine della simulazione.

Una delle funzionalità fondamentali di questo sottosistema consiste nello studio della convergenza della Q-table appresa dal dispositivo smart alla policy ottimale. Infatti, per poter capire in modo semplice e preciso se il dispositivo smart ha appreso come svolgere in modo corretto ed efficace il proprio lavoro è necessario utilizzare un termine di paragone che consiste in una rappresentazione della policy ottimale sotto forma di tabella mostrata in figura 2.14. Durante ogni passo temporale delle traiettorie, che si generano durante gli episodi che il dispositivo smart sperimenta, vengono salvati tutti i dati utilizzati dall'agent nel processo di apprendimento. Tra i dati salvati c'è anche la copia della Q-table di ogni passo temporale ricavata mediante il procedimento di bootstrap effettuato dall'agent. Tutte le informazioni vengono salvate in oggetti della classe Episode che vengono poi inviati all'oggetto data_analust. Il codice relativo al modulo episode.py utilizzato per memorizzare i dati dell'apprendimento viene mostrato nella sezione listing A.14. Durante lo studio della convergenza l'oggetto data_analyst esamina tutti gli oggetti episode in sui possesso. Per ogni oggetto episode ripercorre i passi temporali memorizzati negli oggetti della classe Trajectory ed estrae i cloni della Q-table. La procedura termina quando la Q-table rispetta i criteri di convergenza o quando tutti i passi temporali vengono esaminati senza raggiungere la convergenza.

s index	s name	temperature_intervention	humidity_intervention	reconnect	sense	report	sleep
state 0	INTERVENTION [0, 0, 0]	0	0	0	0	1	0
state 1	INTERVENTION [1, 0, 0]	0	0	1	0	0	0
state 2	INTERVENTION [0, 1, 0]	0	1	0	0	0	0
state 3	INTERVENTION [1, 1, 0]	0	0	1	0	0	0
state 4	INTERVENTION [0, 2, 0]	0	1	0	0	0	0
state 5	INTERVENTION [1, 2, 0]	0	0	1	0	0	0
state 6	INTERVENTION [0, 0, 1]	1	0	0	0	0	0
state 7	INTERVENTION [1, 0, 1]	0	0	1	0	0	0
state 8	INTERVENTION [0, 1, 1]	1	1	0	0	0	0
state 9	INTERVENTION [1, 1, 1]	0	0	1	0	0	0
state 10	INTERVENTION [0, 2, 1]	1	1	0	0	0	0
state 11	INTERVENTION [1, 2, 1]	0	0	1	0	0	0
state 12	INTERVENTION [0, 0, 2]	1	0	0	0	0	0
state 13	INTERVENTION [1, 0, 2]	0	0	1	0	0	0
state 14	INTERVENTION [0, 1, 2]	1	1	0	0	0	0
state 15	INTERVENTION [1, 1, 2]	0	0	1	0	0	0
state 16	INTERVENTION [0, 2, 2]	1	1	0	0	0	0
state 17	INTERVENTION [1, 2, 2]	0	0	1	0	0	0
state 18	WAKE_UP 18	0	0	0	1	0	0
state 19	STANDARD 19	0	0	0	0	0	1
state 20	TERMINAL 20	0	0	0	0	0	0

Figura 2.14: Rappresentazione della policy ottimale sotto forma di tabella.

I file di output prodotti dalle funzioni del sottosistema dedicato all'analisi dei dati sono i seguenti:

- Plot del valore dei dati utilizzati: il file mostra una rappresentazione grafica dei parametri ambientali che vengono utilizzati durante l'esperimento. I valori mostrati sono i valori sottoposti ai dispositivi del progetto che se rilevati essere fuori standard devono essere contrastati per garantire un ambiente in condizioni ottimali.

- Log della simulazione del dispositivo smart IoT: il file TXT descrive in dettaglio gli eventi avvenuti durante la simulazione che coinvolgeva il dispositivo smart IoT.

- Descrizione della Q-table risultante: il file TXT descrive la Q-table ottenuta al termine dell'esperimento condotto coinvolgendo il dispositivo smart IoT.

- Grafico delle simulazioni: il grafico in questione mostra i valori dei parametri ambientali sottoposti al dispositivo e i valori influenzati risultanti derivanti dal lavoro di monitoraggio del dispositivo coinvolto nella simulazione. Il sistema produce un grafico per le temperature e uno per l'umidità di ogni dispositivo coinvolto nelle simulazioni.

- Istogramma dei valori risultanti: il grafico mostra il conteggio dei valori dei parametri ottenuti come risultato del lavoro di monitoraggio dei dispositivi.

- Grafico a barre periodo fuori limite: il grafico mostra il conteggio dei secondi in cui i parametri ambientali hanno assunto valori oltre ai limiti consentiti. Il grafico mette in relazione le informazioni relative a tutti i dispositivi del progetto.

- Grafico a barre periodo nei limiti: il grafico mostra il conteggio dei secondi in cui i parametri ambientali hanno assunto valori entro i limiti consentiti. Il grafico mette in relazione le informazioni relative a tutti i dispositivi del progetto.

- Grafico a torta percentuale fuori limite: il grafico mostra la percentuale di secondi in cui i parametri influenzati dal dispositivo coinvolto nella simulazione hanno assunto valori oltre i limiti consentiti in relazione ai secondi totali della durata della simulazione.

- Grafico a torta percentuale entro ai limite: il grafico mostra la percentuale di secondi in cui i parametri influenzati dal dispositivo coinvolto nella simulazione hanno assunto valori entro i limiti consentiti in relazione ai secondi totali della durata della simulazione.

- Plot dell'evoluzione della return: il grafico mostra l'andamento della somma di tutte le ricompense ottenute dal dispositivo smart IoT ad ogni passo temporale.

- Plot delle ricompense ottenute: il grafico mostra quale ricompensa il dispositivo smart IoT ha ottenuto ad ogni passo temporale.
- Histogramma del numero di azioni per episodio: il grafico mostra il conteggio del numero di azioni che il dispositivo smart IoT ha effettuta durante gli episodi sperimentati. Sull'asse x viene mostrato il numero di azioni per episodio e sull'asse y viene mostrato quante volte il dispositivo ha compiuto x azioni per episodio durante l'intera simulazione.
- Boxplot dei valori influenzati: i grafici mostrano informazioni relative alla statistica descrittiva dell'influenza sui valori dei parametri risultante dal lavoro di monitoraggio del dispositivo coinvolto nella simulazione. I grafici mettono in relazione i boxplot di tutti i dispositivi del progetto. Viene prodotto un boxplot per la temperatura e un box plot per l'umidità.
- CSV con la statistica descrittiva dei valori: il CSV di riferimento mette a disposizione le seguenti informazioni relative ai valori di umidità e temperatura influenzati dal lavoro dei dispositivi coinvolti nelle simulazioni:
  - device: nome del dispositivo a cui appartengono i dati
  - mean: media semplice.
  - median: mediana.
  - trim_mean_1: media trimmata al 10 %
  - trim_mean_2: media trimmata al 20 %
  - trim_mean_3: media trimmata al 30 %
  - trim_mean_4: media trimmata al 40 %
  - minimum_value: valore minimo rilevato.
  - maximum_value: valore massimo rilevato.
  - mode: valore più frequente.
  - first_quantile: quantile al 0.25.
  - third_quantile quantile al 0.75.
  - variance: varianza.
  - standard_deviation: deviaizone standard.
  - iqr: range interquartile.
  - mad: distanza assoluta dalla media.
  - kurtosis: indice di curtosi.
  - skewnessI indice di simmetria.

- Informazioni di convergenza: il file TXT mostra la rappresentazione del momento in cui l'agent si è specializzato nel lavoro assegnato. Oltre alla tabella di convergenza il file mostra il numero di episodi e di passi temporali che sono stati necessari per raggiungere la tabella di convergenza.
- Scatter plot delle ricompense: il grafico mostra lo storico delle ricompense che il dispositivo smart IoT ha ottenuto ad ogni passo temporale di attività.
- Scatter plot delle attività di bootstrap: il grafico mostra lo storico dei valori che il dispositivo smart IoT calcola utilizzando la Q-funtion al ricevimento della ricompensa che valuta le azioni intraprese ad ogni passo temporale di attività del dispositivo.

4 Prototipo

Il prototipo è stato pensato e progettato organizzandolo nelle porzioni che vengono illustrate nella figura 2.15. La porzione di sinistra, indicata dal riquadro giallo, è dedicata alle azioni. La porzione centrale, identificata dal riquadro viola, è dedicata al sensore DHT11. La porzione destra è dedicata all'ambiente e agli eventi, essa viene suddivisa in due sotto porzioni porzione. La prima sotto porzione è dedicata agli stati e viene identificata dal riquadro verde. La seconda sotto porzione è identificata dal riquadro blu ed è dedicata agli episodi.

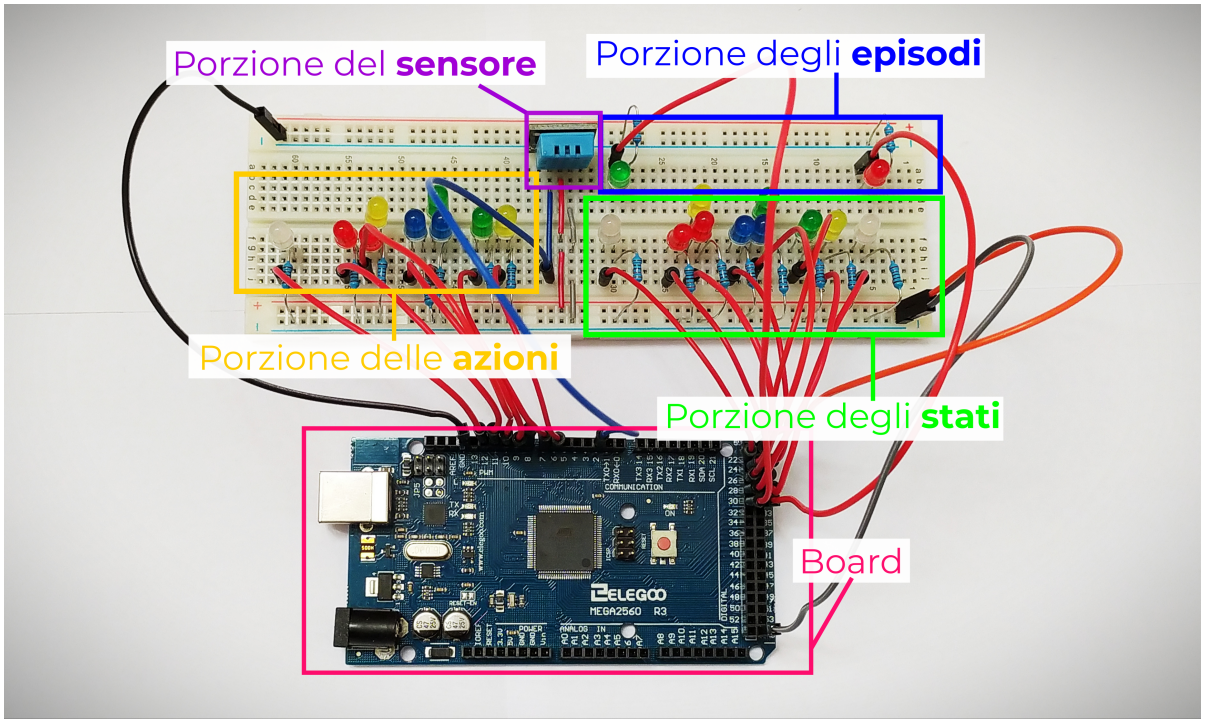


Figura 2.15: Partizionamento del prototipo.

Per condurre gli esperimenti utilizzando il prototipo è stato considerato uno scenario più semplice rispetto a quello delle specifiche del progetto. Nello scenario semplificato si considera il solo parametro ambientale della temperatura. Il task è sempre episodico e consiste sempre nel monitorare il parametro ambientale ed effettuare interventi qualora fosse necessario. L'inizio dell'episodio viene rappresentato da 3 accensioni intermittenti del LED verde posto nella porzione del prototipo dedicata agli episodi. Il termine dell'episodio viene rappresentato da 3 accensioni intermittenti del LED rosso collocato nella porzione degli episodi.

Gli stati che l'ambiente di questo scenario può assumere sono indicati nella lista seguente:

- Start: stato di partenza dell'episodio, esso viene indicato con l'intero 0. Quando l'ambiente è in questo stato si accende del primo LED bianco della porzione del prototipo dedicata agli stati.

- Heat: stato in cui il dispositivo rileva la necessità di accendere il sistema di riscaldamento. Questo stato viene identificato con l'intero 1. Lo stato viene rappresentato mediante l'accensione del LED rosso posto nella porzione del prototipo dedicata agli stati.

- Stop Heat: stato in cui il dispositivo rileva la necessità di arrestare il sistema di riscaldamento. Questo stato viene identificato con l'intero 2. Lo stato viene rappresentato dall'accensione dei LED rosso e giallo posti nella porzione del prototipo dedicata agli stati.

- Cool: stato in cui il dispositivo rileva la necessità di attivare il sistema di refrigerazione. Questo stato viene identificato con l'intero 3. Lo stato viene rappresentato mediante l'accensione del LED blu posto nella porzione del prototipo dedicata agli stati.

- Stop Cool: stato in cui il dispositivo rileva la necessità di arrestare il sistema di refrigerazione. Questo stato viene identificato con l'intero 4. Lo stato viene rappresentato mediante l'accensione del LED blu posto nella porzione del prototipo dedicata agli stati.

- Sleep: stato in cui il dispositivo rileva la convenienza di entrare in risparmio energetico. Questo stato viene idetificato con l'intero 5. Lo stato viene rappresentato mediante l'accensione dei LED blu e verde presenti nella porzione del prototipo dedicata agli stati.

- Report: stato in cui il dispositivo rivela la necessità di trasmettere i dati di intervento al centro di raccolta dati. Questo stato viene identificato dall'intero 6. Lo stato viene rappresentato mediante l'accensione del LED verde posto nella sezione del prototipo dedicata agli stati.

- Terminal: stato rilevato dal dispositivo che ha terminato l'episodio. Questo stato viene identificato dall'intero 7. Lo stato viene rappresentato mediante l'accensione del LED bianco collocato più a destra nella porzione del prototipo dedicata agli stati.

Le azioni a disposizione del prototipo sono:

- sense(): Azione che utilizza il dispositivo DHT11 per misurare realmente la temperatura ambientale. Tale azione ha ID uguale a 0. Questa azione accende il LED bianco situato nella porzione del prototipo dedicata alle azioni.

- heat(): questa azione simula la trasmissione, all'attuatore della temperatura, dei dati e del comando di attivazione del sistema di riscaldamento. Questa azione accende il LED rosso nella porzione del prototipo dedicata alle azioni. L'azione viene identificata dall'ID 1.

- stopHeat(): questa azione simula la trasmissione, all'attuatore della temperatura, dei dati e del comando utilizzato per arrestare il sistema di riscaldamento. Questa azione accende i LED rosso e giallo nella porzione del prototipo dedicata alle azioni. L'ID di questa azione è 2.

- cool(): questa azione simula la trasmissione, all'attuatore della temperatura, dei dati e del comando di attivazione del sistema di refrigerazione. L'azione ha ID 3.

- stopCool(): questa azione simula la trasmissione, all'attuatore della temperatura, dei dati e del comando utilizzato per arrestare il sistema di refrigerazione. Questa azione accende il LED blu posizionato nella porzione del prototipo dedicata alle azioni. L'ID di questa azione è 4.

- sleep(): questa azione simula l'entrata in sleep accendendo il LED verde nella porzione del prototipo dedicata alle azioni. L'azione ha ID 5.

- report(): questa azione simula la trasmissione, al centro di raccolta dati, dei dati relativi all'intervento sui parametri ambientali da parte del prototipo. L'azione accende il LED giallo presente nella porzione del prototipo dedicata alle azioni. Questa azione ha ID 6.

Nell'esperimento 4, che risulta essere un test del prototipo, non viene utilizzato alcun attuatore e si utilizza, invece, una lista di valori arbitrari per testare il comportamento del prototipo e per verificarne il corretto funzionamento. Nell'esperimento 5 invece si utilizza la misurazione reale dei valori ambientali condotta tramite sensore DHT11 . In questo esperimento gli effetti di influenza sui valori di temperatura da parte degli attuatori vengono simulati. Gli effetti del sistema di refrigerazione vengono ottenuti mediante del ghiaccio contenuto in un sacchetto di plastica. Gli effetti del sistema di riscaldamento vengono ottenuti mediante un asciuga capelli che soffia aria calda.

Il prototipo viene equipaggiato di una Q-table che coincide con la policy ottimale per il task che deve svolgere. Questo viene fatto perché le dimostrazioni e gli esperimenti di apprendimento vengono condotte attraverso il simulatore, il prototipo lo si utilizza per vedere come l'agente di Q-learning specializzato gestisce un prototipo del dispositivo smart IoT progettato e studiato. La Q-table, il codice utilizzato per governare il dispositivo e ulteriori informazioni vengono illustrati nel listing A.15. Nel listing vengono mostrati due metodi sense. Il primo in cui viene utilizzato il sensore è la funzione utilizzata nell'esperimento 5 condotto per mostrare il comportamento del prototipo in un caso reale. Il secondo che utilizza la lista di valori di test è la funzione utilizzata nel

test del prototipo condotto nell'esperimento 4. Come si può osservare nella sezione del listing A.15 gli episodi sperimentati dal prototipo durano di più rispetto a quelli previsti dal progetto descritto nelle specifiche. La durata maggiore dell'episodio è dovuta al fatto che nell'esperimento 5 le veci degli attuatori sono fatte da operatori umani che utilizzano ghiaccio e asciuga capelli per intervenire. Questo è stato fatto dunque per poter osservare con calma e ad occhio nudo i comportamenti del dispositivo mediante l'osservazione dell'accensione dei LED di controllo.

L'alimentazione del dispositivo avviene mediante collegamento USB con il calcolatore, in questo modo è possibile effettuare la trasmissione dei dati prodotti durante gli esperimenti condotti.

I dati raccolti durante la sperimentazione sono stati salvati su di un calcolatore all'interno di un file TXT, come quello mostrato in figura 2.16, avvalendosi della funzionalità di logging di PuTTY. Successivamente i dati raccolti sono stati estratti dal file per essere e rappresentati fraficamente attraverso l'utilizzo di un modulo Python appositamente programmato che viene mostrato nel listing A.16.

```
16  0,0,22.40,-1,3,3,-1,6,6,100,7
17  0,0,21.70,-1,5,5,-1,7
18  0,0,21.00,-1,5,5,-1,7
19  0,0,20.50,-1,5,5,-1,7
20  0,0,20.20,-1,5,5,-1,7
21  0,0,20.00,-1,4,4,-1,6,6,100,7
22  0,0,19.80,-1,5,5,-1,7
23  0,0,19.70,-1,5,5,-1,7
```

Figura 2.16: Esempio di output degli esperimenti con il prototipo

5 Descrizione degli esperimenti condotti

Il sistema di sperimentazione software descritto nella sezione precedente è stato utilizzato per condurre diversi esperimenti. Ai fini dello studio di tesi vengono riportati di seguito le informazioni relative a due esperimenti che hanno consentito di dimostrare gli obiettivi dello studio. In ogni esperimento i parametri che descrivono l'ambiente sono sempre gli stessi: temperature che descrive la temperatura in gradi C°, humidity che descrive la percentuale di umidità e wifi che descrive lo stato di connettività wifi. In ogni esperimento quello che cambia sono il secondo di partenza della simulazione, il secondo di termine della simulazione e le informazioni che caratterizzano i parametri utilizzati per descrivere l'ambiente della simulazione dell'esperimento e le relative liste dei valori associate ai parametri stessi.

Ogni esperimento viene condotto sottoponendo i tre dispositivi descritti nella sezione del sistema di sperimentazione software ai valori gestiti dalla simulazione. I dispositivi simulati coinvolti negli esperimenti sono dunque i seguenti 3:

- Diligent: si tratta di un dispositivo diligente precisamente istruito e che segue una semplice logica if than else per svolgere il suo lavoro. Questo dispositivo non entra mai in risparmio energetico ed effettua l'operazione di sensing ad ogni secondo del giorno simulato.

- Energy saving: si tratta anch'esso di un dispositivo semplice istruito mediante logica if then esle. Tale dispositivo se rileva una condizione ambientale in cui tutti i valori di tutti i parametri sono all'interno dei limiti stabiliti entra in modalità sleep diventando inattivo per 1 secondo.

- Smart: si tratta di un dispositivo dotato di agente di Reinforcement Learning. Il dispositivo, all'inizio della simulazione, non sa come comportarsi. Durante la simulazione apprende grazie alla tecnica del Q-learning il modo più efficiente su come svolgere il lavoro assegnato.

Nelle sottosezioni che seguono verranno descritte le particolarità che caratterizzano le simulazioni dei due esperimenti condotti per ottenere i risultati da discutere.

5.1 Esperimento 1 - Convergenza alla policy ottimale

Le informazioni relative ai parametri continui e discreti utilizzati per descrivere l'ambiente dell'esperimento 1 sono riportati nella tabella 2.13 e nella tabella 2.14:

Tabella 2.13: Informazioni dei parametri continui utilizzati nell'esperimento 1 della tesi.

name	*lower_limit_value*	*desired_value*	*upper_limit_value*
temperature	20.00	22.00	24.00
humidity	30.00	35.00	40.00

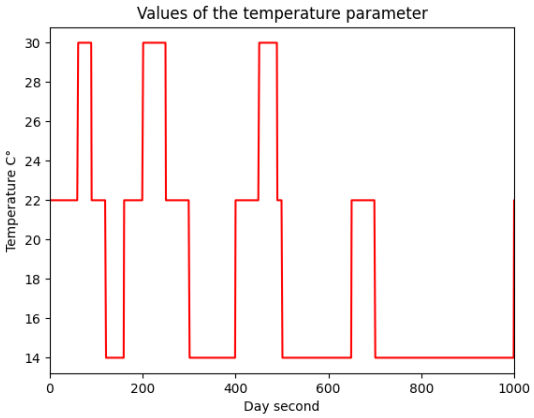
Tabella 2.14: Informazioni dei parametri discreti utilizzati nell'esperimento 1 della tesi.

name	*value_names*	*values*	*lower_limit_value*	*desired_value*	*upper_limit_value*
wifi	offline#online	0#1	0	1	float("inf")

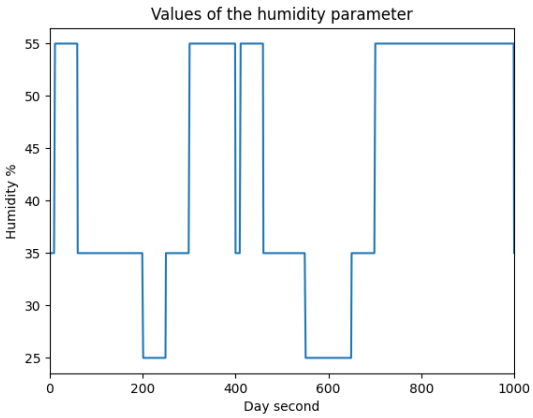
La lista dei valori dei parametri utilizzata nell'esperimento 1 è stata ottenuta utilizzando il sistema di modellazione dei valori dei parametri. Le risultanti liste generate relative ai parametri temperature, humidity e wifi vengono mostrate rispettivamente nella figura 2.17a, nella figura 2.17b e, per finire, nella figura 2.17c.

La simulazione inizia al secondo 0 e termina al secondo 1000 della giornata simulata. La modellazione delle liste dei valori dei parametri mostrate nelle figure di riferimento è servita a consentire all'agent del dispositivo smart di sperimentare tutti gli stati previsti dal sistema decisionale del dispositivo. In questo modo è stato possibile osservare il processo di apprendimento e di specializzazione in modo completo.

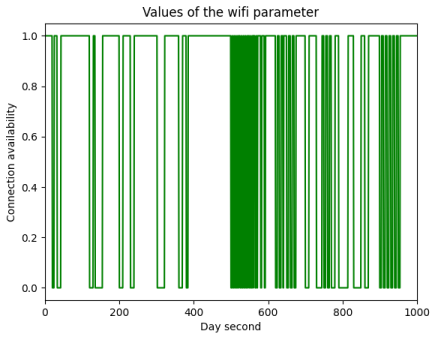
All'agent è stato attribuito un $\epsilon = 1$, un fattore di decremento $\epsilon\_decrease = 0.01$ e un valore minimo di casualità $minimum\_\epsilon = 0.3$ in modo da avere un agent che molto frequentemente compie decisioni casuali. Questo atteggiamento propenso all'esplorazione fornisce all'agent una maggior probabilità di apprendere una Q-table che tende alla policy ottimale.



(a) Lista dei valori del parametro temperature utilizzata nell'esperimento.



(b) Lista dei valori del parametro humidity utilizzata nell'esperimento.



(c) Lista dei valori del parametro wifi utilizzata nell'esperimento.

Figura 2.17: Valori dei parametri ambientali

5.2 Esperimento 2 - Valori di umidità e temperatura ricavati da misurazioni reali

Le informazioni relative ai parametri continui e discreti che descrivono l'ambiente dell'esperimento 2 sono riportate rispettivamente nella tabella 2.15 e nella tabella 2.16.

Tabella 2.15: Informazioni dei parametri continui utilizzati nell'esperimento 1 della tesi.

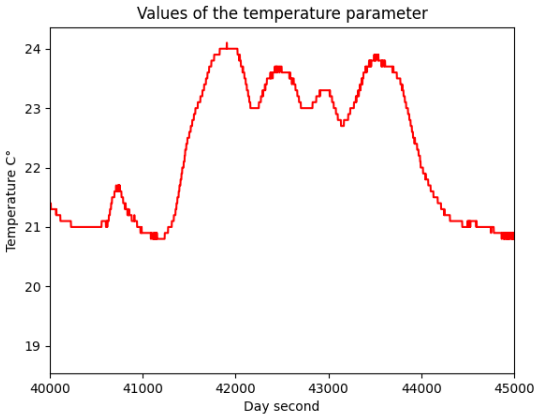
name	*lower_limit_value*	*desired_value*	*upper_limit_value*
temperature	21.50	22.00	22.50
humidity	42.00	44.00	46.00

Tabella 2.16: Informazioni dei parametri discreti utilizzati nell'esperimento 1 della tesi.

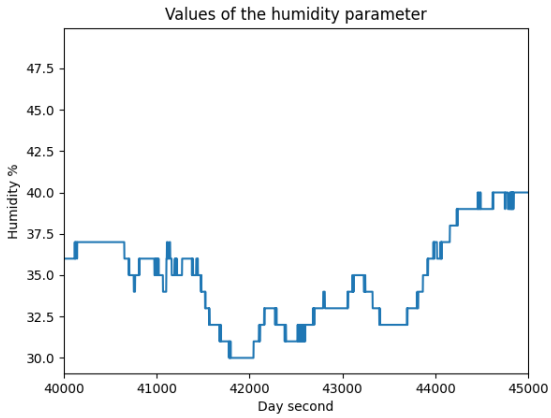
name	*value_names*	*values*	*lower_limit_value*	*desired_value*	*upper_limit_value*
wifi	offline#online	0#1	0	1	float("inf")

La lista dei valori dei parametri utilizzati per la simulazione dell'esperimento 2 viene generata utilizzando le informazioni estratte dal set di dati ottenuto mediante le misurazioni condotte in un ambiente real. Tali misurazioni sono state effettuate tramite l'utilizzo del dispositivo di misurazione realizzato con scheda di sviluppo per microcontrollori ELEGOO Uno R3. Le risultanti liste dei valori associati ai parametri temperature, humidity e wifi che sono state ottenute con questo metodo vengono mostrate rispettivamente nella figura 2.18a, nella figura 2.18b e nella figura 2.18c. La simulazione parte dal secondo 40000 e termina al secondo 45000, questi estremi sono stati scelti perché consentono di estrarre dal set di dati una delle zone con maggiore variazione ed escursione dei valori dei parametri ambientali. Nella figura 2.19a e nella figura 2.19b vengono riportate le liste dal secondo 0 al secondo 86400 senza limitazioni dettate dal secondo di inizio e termine della simulazione.

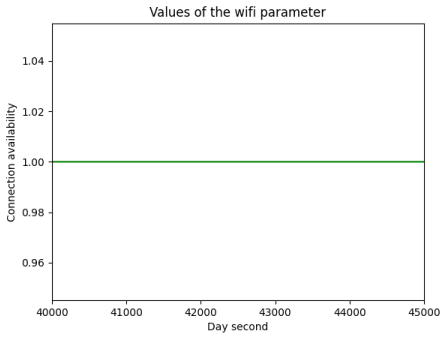
All'agent è stato attribuito un $\epsilon = 1$, un fattore di decremento $\epsilon\_decrease = 0.05$ e un valore minimo di casualità $minimum\_\epsilon = 0.0$. In questo esperimento l'agent ha uno spirito meno esplorativo e, ad un certo punto, abbandonerà le scelte casuali. L'agent continuerà a modificare la sua Q-table in base ai risultati delle operazioni di bootstrap condotte durante il processo di apprendimento.



(a) Esperimento 2 - Lista dei valori del parametro temperature limitata ai secondi di simulazione.

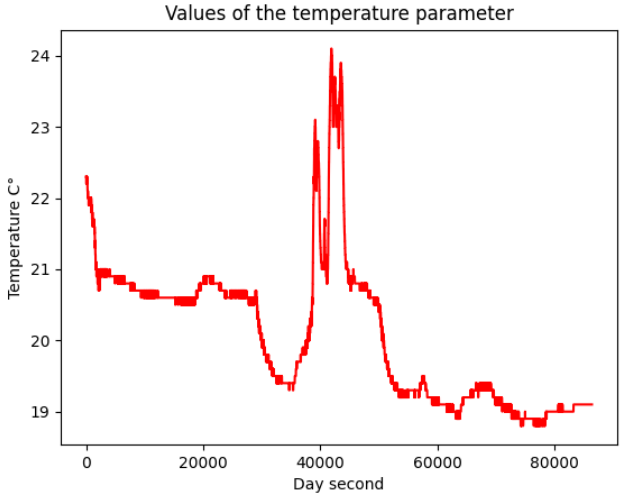


(b) Esperimento 2 - Lista dei valori del parametro humidity limitata ai secondi di simulazione.

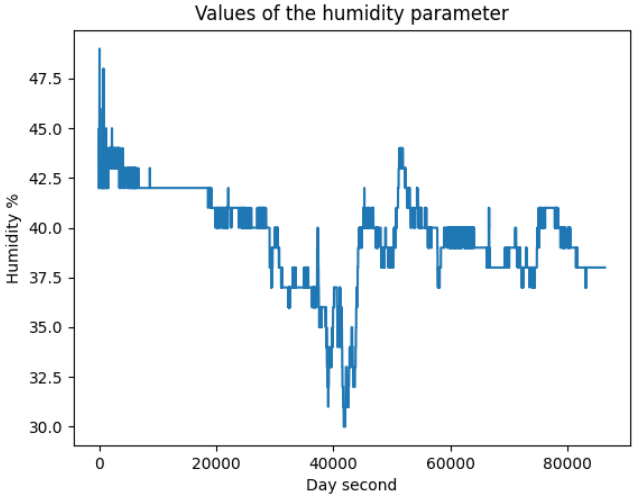


(c) Esperimento 2 - Lista dei valori del parametro wifi.

Figura 2.18: Valori estratti dal set di dati.



(a) Esperimento 2 - Lista dei valori del parametro temperature mostrata per intero.



(b) Esperimento 2 - Lista dei valori del parametro humidity mostrata per intero.

Figura 2.19: Esperimento 2 - Rappresentazione del set di dati dei valori reali di temperatura e umidità mostrato per intero.

5.3 Esperimento 3 - Dispositivo specializzato e valori di umidità e temperatura ricavati da misurazioni reali

Questo esperimento mette in relazione l'esperimento 1 e l'esperimento 2. I valori dei parametri continui e discreti sono quelli mostrati rispettivamente nella tabella 2.15 e nella tabella 2.16 riportate nella sottosezione precedente. Le liste dei valori dei parametri temperature, humidity e wifi sono mostrate rispettivamente nella figura 2.18a, nella figura 2.18b e nella figura 2.18c della sottosezione precedente.

All'agent è stato attribuito un $\epsilon = 0$, un fattore di decremento $\epsilon\_decrease = 0.00$ e un valore minimo di casualità $minimum\_\epsilon = 0.0$. Questo è stato fatto perché, in questo esperimento, l'agent dispone di una Q-table che converge a quella ottimale. Ci si aspetta che l'agent possa quindi affidarsi alle sole scelte di sfruttamento per poter condurre efficientemente il proprio lavoro in qualunque ambiente descritto dai parametri ambientali previsti delle specifiche del progetto. La Q-table fornita all'agent è quella che è stata ottenuta attraverso la conduzione dell'esperimento 1 e viene mostrata in figura 2.20.

```
Imported Q-table
[[  79.     79.     79.     50.12  100.   -100.  ]
 [  62.14   62.2    79.     62.19   62.19  -99.9 ]
 [  62.14   79.     62.2    55.82   61.56  -99.9 ]
 [  -1.64   -0.9    62.2    -1.     -1.    -90.  ]
 [  55.98   79.     61.58   34.04   62.2   -99.9 ]
 [  -1.71   -1.71   -0.99   -1.64   -1.64  -99.9 ]
 [  79.     62.14   61.58   62.2    62.14  -99.99]
 [  -0.9    43.79   62.2    -0.9    48.71  -99.9 ]
 [  62.13   62.2    48.17   43.75   -0.9   -99.9 ]
 [  -1.64   27.24   39.31   -1.64   -1.64  -99.  ]
 [  62.2    61.58   43.79   37.63    0.      0.  ]
 [  37.96   -0.99   48.76   33.68   37.62  -90.  ]
 [  79.     62.2    62.14   47.53   62.19 -100.  ]
 [   0.     43.78   62.2    -0.9    -0.9   -99.  ]
 [   0.     62.2    43.44   34.21    0.    -90.  ]
 [   0.     33.7    48.76   -0.99   33.7   -90.  ]
 [  61.52    0.     -0.9    -0.9     0.    -90.  ]
 [   0.     -0.9    38.99   -0.9    -0.9   -99.  ]
 [  -1.32   -1.4    -1.35   -0.95   -1.37 -100.  ]
 [  -1.8    -1.8    -1.8    -1.8    -1.8    -1.  ]
 [   0.      0.      0.      0.      0.      0.  ]]
```

Figura 2.20: Esperimento 3 - Q-table importata dal dispositivo smart che costituisce la linea di condotta da seguire.

5.4 Esperimento 4 - Test del prototipo

In questo esperimento è stato impiegato il prototipo hardware del dispositivo smart IoT mostrato in figura 2.5. Dal momento che gli esperimenti relativi all'apprendimento mediante Q-learning sono stati condotti negli esperimenti 1, 2 e 3, si è pensato di utilizzare il prototipo per osservarne il funzionamento sotto le direttive di un agent di Q-learning già specializzato. Il dispositivo si considera specializzato perché è stato equipaggiato con una Q-table che coincide con la policy ottimale associata al compito che il dispositivo deve svolgere. L Q-table in questione viene mostrata in figura 2.21. In questo esperimento

```
int qTable[8][7] = {
  {1,0,0,0,0,0,0}, // Start -> sense()
  {0,1,0,0,0,0,0}, // Need Heat -> heat()
  {0,0,1,0,0,0,0}, // Stop Heat -> stopHeat()
  {0,0,0,1,0,0,0}, // Need Cool -> cool()
  {0,0,0,0,1,0,0}, // Stop Cool -> stopCool()
  {0,0,0,0,0,1,0}, // Sleepable -> sleep()
  {0,0,0,0,0,0,1}, // Report -> report()
  {0,0,0,0,0,0,0}  // Terminal
};
```

Figura 2.21: Esperimento 4 - Q-table equipaggiata al prototipo.

si è condotto un test per verificare il corretto funzionamento dell'implementazione del prototipo. L'azione di sense() utilizzata è quella che, come spiegato nella sezione dedicata al prototipo, estrae i valori di temperatura da una lista arbitraria. La lista arbitraria è stata realizzata con lo scopo di osservare se il dispositivo fosse in grado di esplorare correttamente tutti gli stati previsti dallo scenario semplificato utilizzato per gli esperimenti con il prototipo. La lista di valori arbitrari della temperatura viene mostrata nella tabella 2.17.

21	16	19	20	20	23	21	20	20

Tabella 2.17: Esperimento 4 - Valori arbitrari di temperatura utilizzati per condurre il test.

La lista delle traiettorie che ci si aspetta che il dispositivo sperimenti viene mostrate di seguito:

- Valore arbitrario = 21: Start, sense(), Sleepable, sleep(), Terminal
- Valore arbitrario = 16: Start, sense(), Need Heat, heat(), Report, report(), Terminal
- Valore arbitrario = 19: Start, sense(), Sleepable, sleep(), Terminal
- Valore arbitrario = 20: Start, sense(), Stop Heat, stopHeat(), Report, report(), Terminal
- Valore arbitrario = 20: Start, sense(), Sleepable, sleep(), Terminal
- Valore arbitrario = 23: Start, sense(), Need Cool, cool(), Report, report(), Terminal
- Valore arbitrario = 21: Start, sense(), Sleepable, sleep(), Terminal
- Valore arbitrario = 20: Start, sense(), Stop Cool, stopColl(), Report, report(), Terminal
- Valore arbitrario = 20: Start, sense(), Sleepable, sleep(), Terminal

In questo esperimento ci si aspetta che il prototipo produca la sequenza di accensioni dei LED di controllo che corrispondono alle traiettorie indicate nella lista precedente. Inoltre ci si aspetta che, dopo l'esecuzione del test, i dati di output prodotti dal prototipo riportino la descrizione di traiettorie che coincidono con quelle attese dal test.

5.5 Esperimento 5 - Prototipo in uno scenario reale

In questo esperimento il prototipo oltre ad essere dotato della policy ottimale utilizza anche la funzione di sense che gli consente di percepire realmente i valori del parametro temperatura, relativo all'ambiente che sta monitorando, grazie al sensore DHT11.

In questo esperimento anche se si fa riferimento allo scenario semplificato si vuole comunque riprodurre gli effetti dell'attuatore dedicato all'unfluenza dei valori ambientali. Il sistema di refrigerazione è stato simulato mediante l'utilizzo di ghiaccio in un sacchetto di plastica posto sul sensore di temperatura. Il sistema di riscaldamento è stato simulato utilizzando un asciuga capelli che ha soffiato aria calda sul sensore di temperatura.

In questo esperimento dunque l'operatore umano ha fatto le veci dell'attuatore rispondendo ai segnali luminosi emessi dai LED posti nella porzione del prototipo dedicata alle azioni. Dopo che il dispositivo ha superato il test condotto nell'esperimento 4, quello che ci si aspetta di osservare in questo esperimento è il corretto svolgimento del lavoro di gestione dell'agente di Q-learning anche in scenari reali in cui gli effetti degli attuatori, simulati dall'operatore umano, non sono efficienti come quelli realmente implementabili.

Come detto nella sezione dedicata al prototipo gli episodi che esso sperimenta durano di più di quelli previsti dallo scenario descritto nelle specifiche del progetto. I grafici che verranno riprodotti nei risultati fanno dunque riferimento a episodi che hanno durata maggiore di 1 secondo. Come già spiegato questo è stato fatto per poter osservare ad occhio nudo, attraverso l'accensione dei LED, L'inizio degli episodi, le transizioni e le operazioni scelte dall'agent che governa il prototipo.

Capitolo 3

Analisi dei risultati

In questo capitolo verranno mostrati i risultati prodotti dalle simulazioni condotte durante l'esperimento 1 e durante l'esperimento 2. Per ogni esperimento verranno mostrati i grafici e le informazioni fondamentali, prodotte dal sistema di analisi di dati, che sono indispensabili per illustrare e comprendere l'esito dell'esperimento.

1 Esperimento 1

I grafici mostrati in figura 3.1 e in figura 3.2 fanno riferimento al risultato del lavoro di monitoraggio del dispositivo diligent. Quelli mostrati in figura 3.3 e in figura 3.4 fanno riferimento ai risultati del lavoro del dispositivo energy saver. I grafici che invece si possono osservare in figura 3.5 e in figura 3.6 sono quelli che mostrano il risultato del lavoro di monitoraggio del dispositivo smart. I grafici mostrano come i tre sistemi di monitoraggio hanno influenzato i valori dei parametri che hanno assunto valori oltre i limiti consentiti.

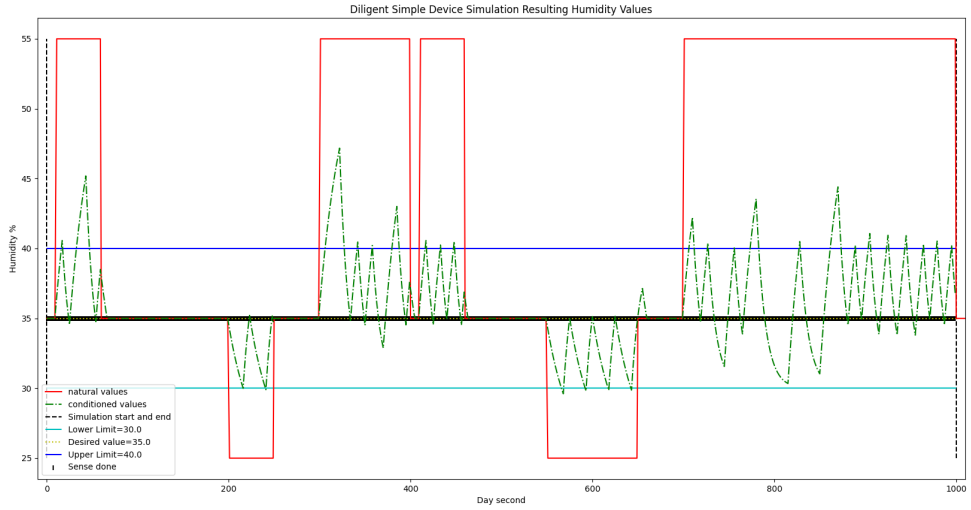


Figura 3.1: Esperimento 1 - Gestione dell'umidità da parte del dispositivo diligent.

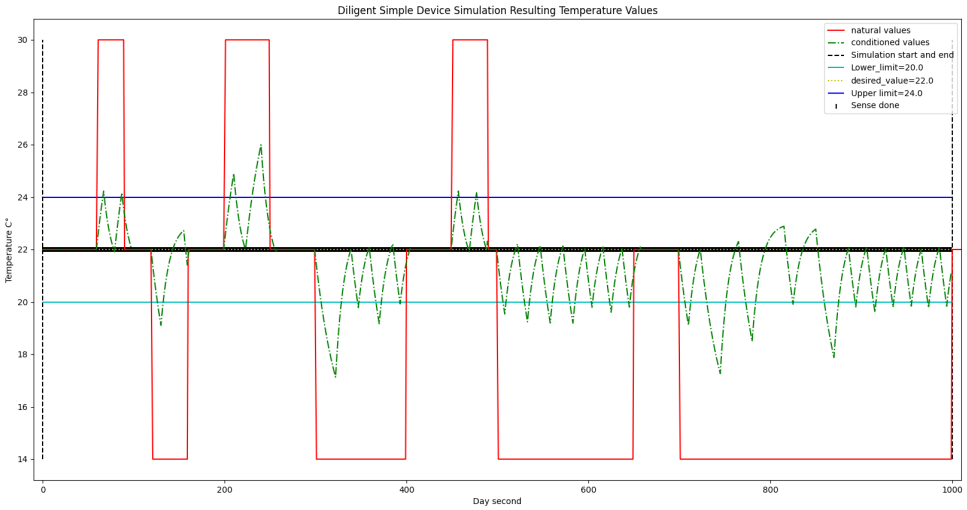


Figura 3.2: Esperimento 1 - Gestione della temperatura da parte del dispositivo diligent.

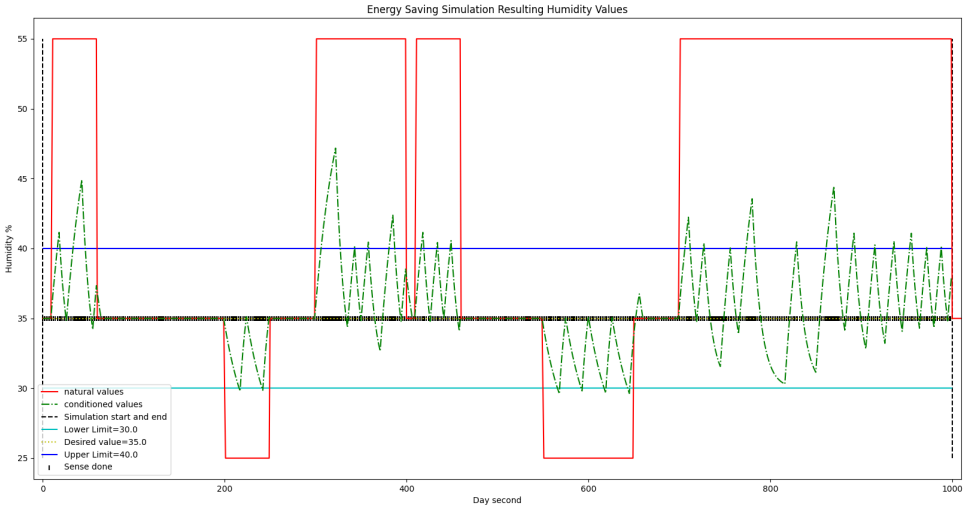


Figura 3.3: Esperimento 1 - Gestione dell'umidità da parte del dispositivo energy saver.

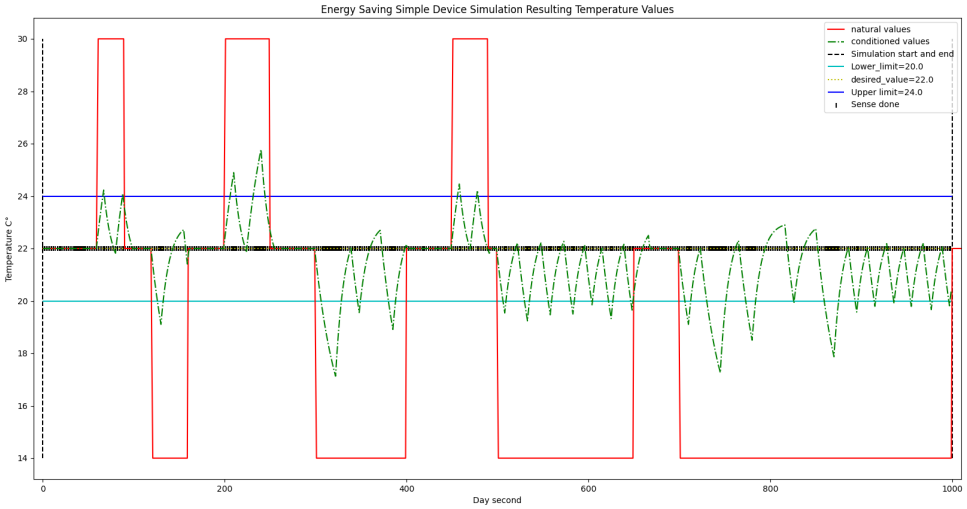


Figura 3.4: Esperimento 1 - Gestione della temperatura da parte del dispositivo energy saver.

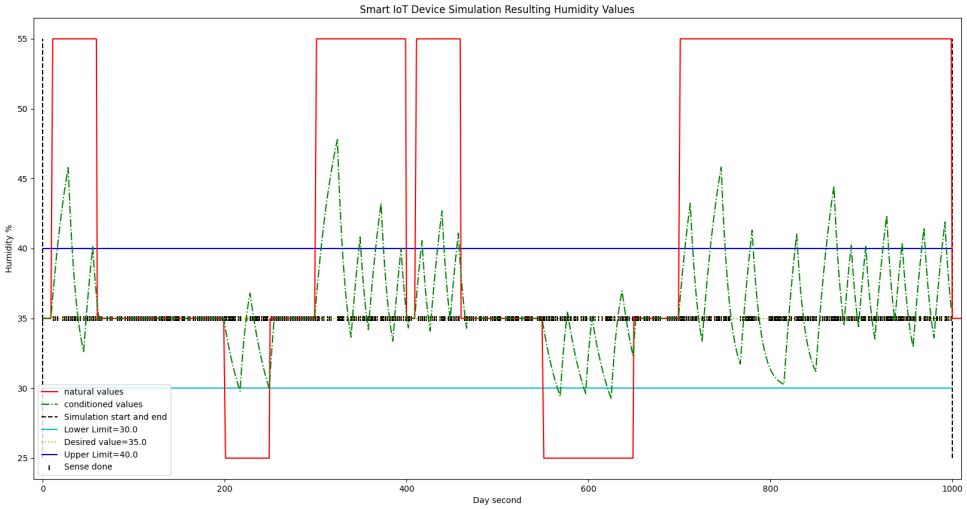


Figura 3.5: Esperimento 1 - Gestione dell'umidità da parte del dispositivo smart.

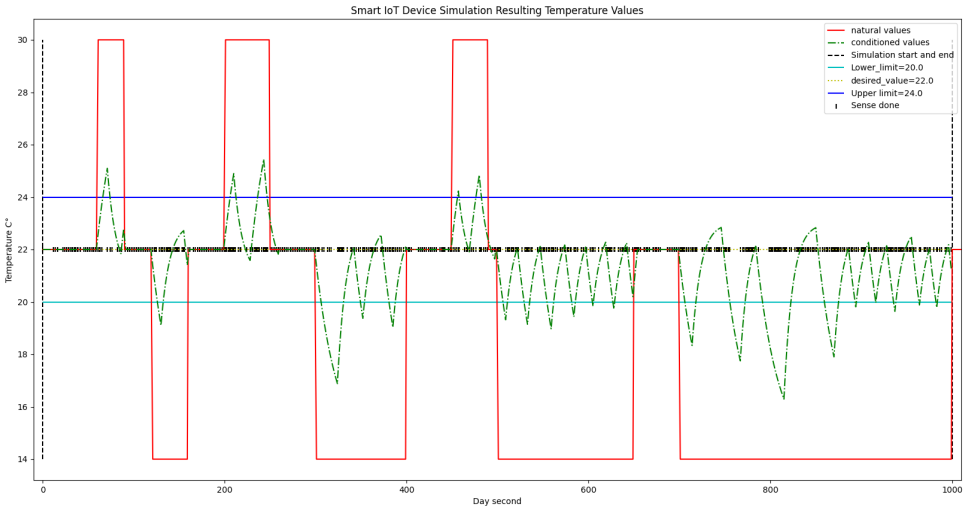


Figura 3.6: Esperimento 1 - Gestione della temperatura da parte del dispositivo smart.

I grafici a barre della figura 3.7 e della figura 3.8 mostrano per quanto tempo i tre dispositivi di monitoraggio hanno mantenuto i valori dei parametri entro e oltre i limiti. Come ci si aspettava il dispositivo diligent è quello che è stato in grado di far mantenere per più tempo ai parametri temperature e umidity valori entro i limiti.

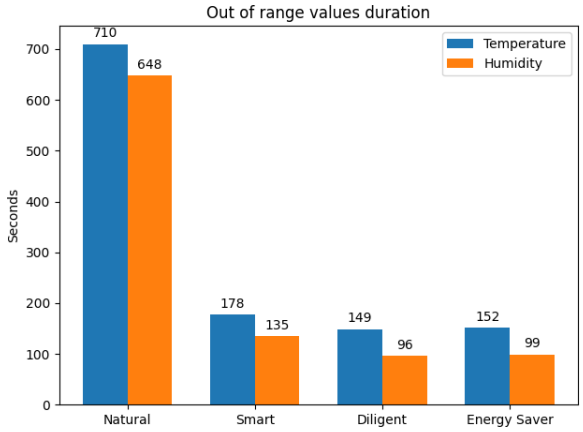


Figura 3.7: Esperimento 1 - Permanenza oltre i limiti dei valori dei parametri.

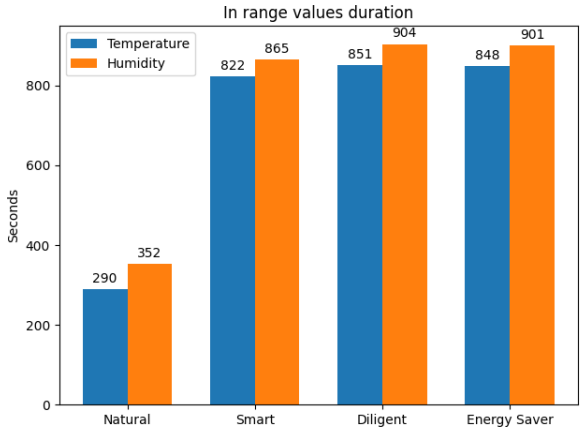


Figura 3.8: Esperimento 1 - Permanenza entro i limiti dei valori dei parametri.

Il grafico della figura 3.9 mostra la frequenza con la quale i vari dispositivi compiono le azioni a disposizione in un secondo durante la simulazione dell'esperimento 1, questa simulazione ha durata di 1000 secondi simulati che corrispondono a 1000 passi temporali di simulazione. L'elevato numero di azioni che il dispositivo smart ha compiuto sono dovute al processo di apprendimento in cui il dispositivo ha dovuto provare e sbagliare prima di capire la giusta sequenza di azioni utili al raggiungimento del goal degli episodi. Si consideri inoltre che, come indicato nella sezione della descrizione dell'esperimento, l'agent del dispositivo smart IoT ha un'elevata probabilità di compiere scelte esplorative durante i passi temporali delle traiettorie degli episodi perché ha un $minimum\_\epsilon = 0.3$. Come mostrato dal grafico, il dispositivo diligent non è mai entrato in sleep. Il dispositivo smart è entrato in sleep anche quando non avrebbe dovuto per via delle scelte esplorative. Il dispositivo energy saver è entrato in sleep ogni volta che i valori erano entro i limiti.

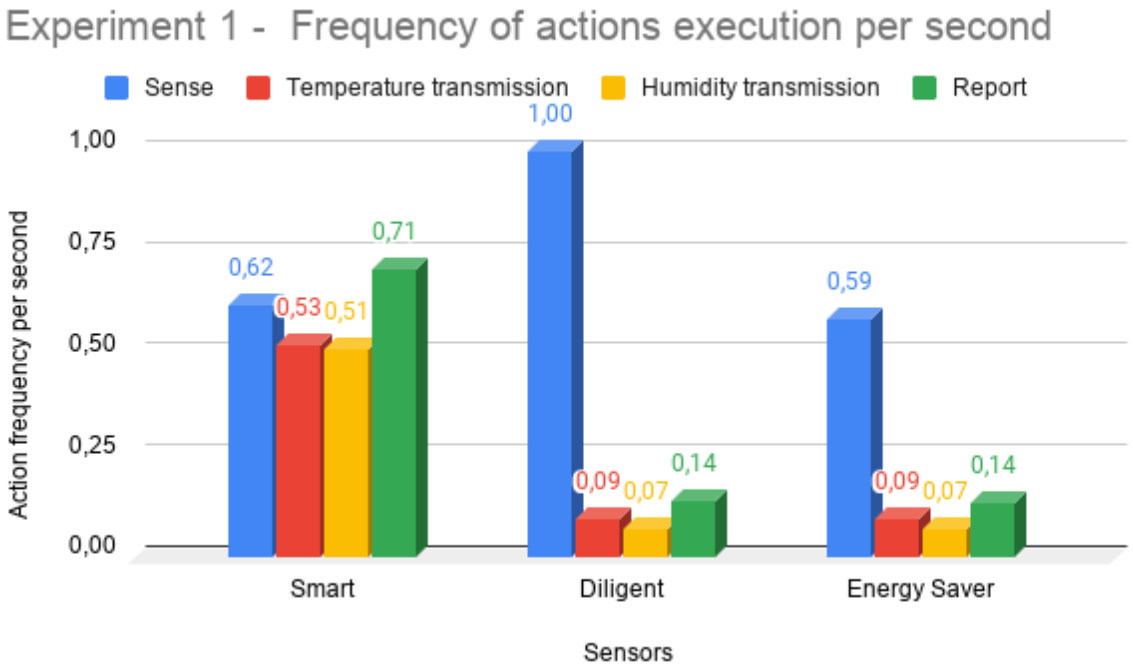


Figura 3.9: Esperimento 1 - Frequenza di esecuzione al secondo delle azioni da parte dei dispositivi.

Il grafico mostrato in figura 3.10 dimostra che il dispositivo smart esplora molto ottenendo, di conseguenza, numerose ricompense negative. Questi eventi hanno consentito all'agent di provare rapidamente a svolgere tante azioni durante uno stato fino a quando non ha compiuto l'azione di sleep o fino a quando non ha compiuto la giusta azione che ha consentito la transizione di stato. La prova di quanto appena detto viene fornita dall'istogramma mostrato in figura 3.11 in cui viene riportato il conteggio del numero di azioni compiute per ogni episodio della simulazione dell'esperimento 1. Si nota infatti che la maggior parte degli episodi è stato caratterizzato dal compimento di 3 o 4 azioni. Ciò significa che il dispositivo anche se ha sperimentato e sbagliato molto è riuscito a capire come comportarsi nonostante le elevate decisioni casuali compite. Il grafico a barre riportato nella figura 3.12 mostra il numero di passi temporali delle traiettorie di ogni episodio sperimentato dal dispositivo smart. Per via del comportamento molto esplorativo del dispositivo è possibile osservare che alcuni episodi sono stati composti da numerosi passi temporali. Questo perché il dispositivo ha inizialmente compiuto molte scelte casuali che spesso non hanno condotto al compimento di un'azione capace di terminare o di far procedere l'episodio. Il numero di passi temporali delle traiettorie coincide con il numero di azioni compiute dal dispositivo e di conseguenza il grafico è una conferma visiva del conteggio delle azioni riportato in figura 3.11.

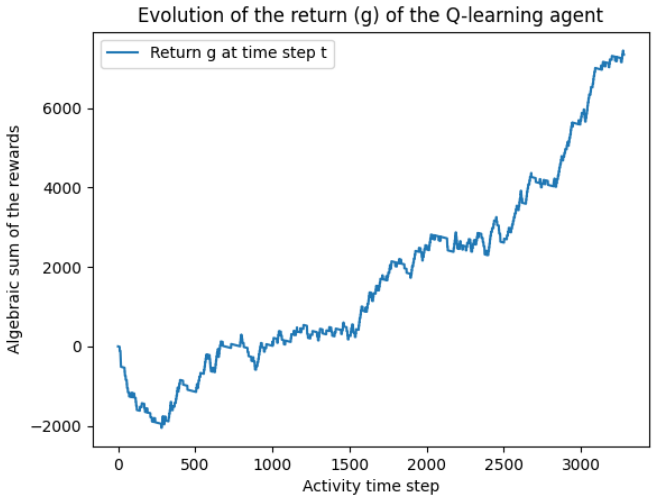


Figura 3.10: Esperimento 1 - Plot dell'evoluzione della somma delle ricompense.

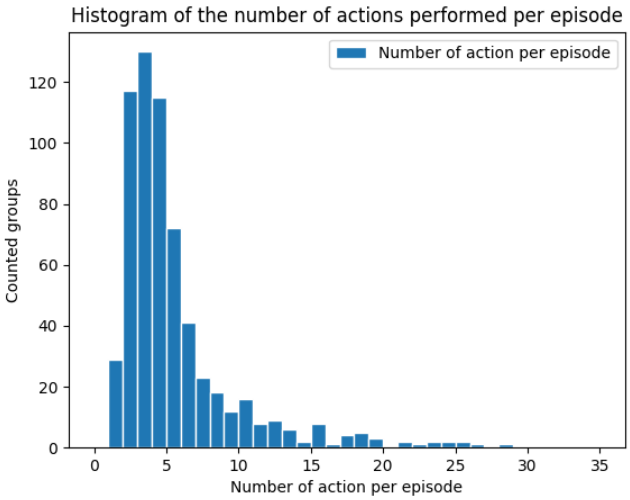


Figura 3.11: Esperimento 1 - Istogramma del numero di azioni per episodio.

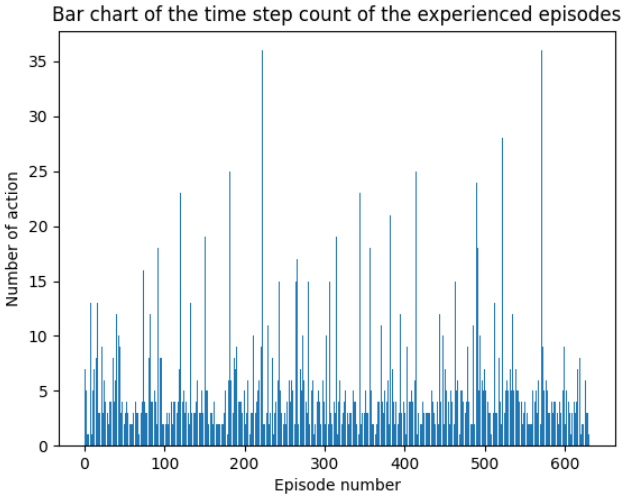


Figura 3.12: Esperimento 1 - Il grafico a barre mostra il numero di passi temporali che compongono gli episodi sperimentati dal dispositivo smart.

La figura 3.13 mostra che il dispositivo apprende una policy equivalente a quella ottimale, detta Q-table di convergenza, dopo aver sperimentato 3260 passi temporali delle traiettorie di 525 episodi. Le ricompense attribuite alle coppie stato azione ricevute dall'agent non cambiano in questo dominio del problema e, di conseguenza, il dispositivo ha terminato il processo di apprendimento. In tal caso il dispositivo potrà svolgere il suo lavoro affidandosi esclusivamente alle scelte di sfruttamento e senza necessità ulteriore di aggiornamento della Q-table. In questo contesto la Q-table di convergenza può essere esportata e utilizzata per fornire il comportamento a dispositivi che possono di conseguenza evitare di svolgere il processo di apprendimento perché capaci di comportarsi nel modo desiderato.

Number of episodes needed to converge: 525
Number of time steps of the trajectories of the episodes necessary to converge: 3260

s index	s name	temperature_intervention	humidity_intervention	reconnect	sense	report	sleep
state 0	INTERVENTION [0, 0, 0]	79	79	79	50.12	100	-100
state 1	INTERVENTION [1, 0, 0]	62.14	62.2	79	62.19	62.19	-99.9
state 2	INTERVENTION [0, 1, 0]	62.14	79	62.2	55.82	61.56	-99.9
state 3	INTERVENTION [1, 1, 0]	-1.64	-0.9	62.2	-1	-1	-90
state 4	INTERVENTION [0, 2, 0]	55.98	79	61.58	34.04	62.2	-99.9
state 5	INTERVENTION [1, 2, 0]	-1.71	-1.71	-0.99	-1.64	-1.64	-99.9
state 6	INTERVENTION [0, 0, 1]	79	62.14	61.58	62.2	62.14	-99.99
state 7	INTERVENTION [1, 0, 1]	-0.9	43.79	62.2	-0.9	48.71	-99.9
state 8	INTERVENTION [0, 1, 1]	62.13	62.2	48.17	43.75	-0.9	-99.9
state 9	INTERVENTION [1, 1, 1]	-1.64	27.24	39.31	-1.64	-1.64	-99
state 10	INTERVENTION [0, 2, 1]	62.2	61.58	43.79	37.63	0	0
state 11	INTERVENTION [1, 2, 1]	37.96	-0.99	48.76	33.68	37.62	-90
state 12	INTERVENTION [0, 0, 2]	79	62.2	62.14	47.53	62.19	-100
state 13	INTERVENTION [1, 0, 2]	0	43.78	62.2	-0.9	-0.9	-99
state 14	INTERVENTION [0, 1, 2]	0	62.2	43.44	34.21	0	-90
state 15	INTERVENTION [1, 1, 2]	0	33.7	48.76	-0.99	33.7	-90
state 16	INTERVENTION [0, 2, 2]	61.52	0	-0.9	-0.9	0	-90
state 17	INTERVENTION [1, 2, 2]	0	-0.9	38.99	-0.9	-0.9	-99
state 18	WAKE_UP 18	-1.32	-1.4	-1.35	-0.95	-1.37	-100
state 19	STANDARD 19	-1.8	-1.8	-1.8	-1.8	-1.8	-1
state 20	TERMINAL 20	0	0	0	0	0	0

Figura 3.13: Esperimento 1 - Informazioni relative al momento in cui il dispositivo smart si è specializzato.

La Q-table che l'agent ha appreso al termine della simulazione continuando a sperimentare il processo di apprendimento è quella che si può osservare in figura 3.14. La Q-table ottenuta a fine sperimentazione prende il nome di Q-table risultante.

In questo esperimento si dimostrato che il dispositivo è in grado di apprendere a svolgere il lavoro per la quale è stato progettato. Il dispositivo è riuscito ad apprendere una Q-table di convergenza prima del termine della simulazione e riuscendo a esplorare tutti gli stati. Lo scenario simulato è però poco probabile ed è servito appositamente a verificare che il dispositivo apprendesse qualora si trovasse in uno scenario favorevole alla sperimentazione di tutti gli stati possibili del problema esaminato.

s index	s name	temperature_intervention	humidity_intervention	reconnect	sense	report	sleep
state 0	INTERVENTION [0, 0, 0]	79	79	79	50.08	100	-100
state 1	INTERVENTION [1, 0, 0]	62.19	62.2	79	62.2	62.2	-99.9
state 2	INTERVENTION [0, 1, 0]	62.2	79	62.2	55.82	62.2	-100
state 3	INTERVENTION [1, 1, 0]	-1.64	-0.9	62.2	-1	-1	-90
state 4	INTERVENTION [0, 2, 0]	62.14	79	62.14	47.74	62.2	-99.99
state 5	INTERVENTION [1, 2, 0]	-1.71	-1.71	62.14	-1.64	-1.64	-99.9
state 6	INTERVENTION [0, 0, 1]	79	62.19	62.14	62.2	62.19	-100
state 7	INTERVENTION [1, 0, 1]	-0.9	43.79	62.2	-0.9	48.76	-99.9
state 8	INTERVENTION [0, 1, 1]	62.13	62.2	48.7	43.75	-0.9	-99.99
state 9	INTERVENTION [1, 1, 1]	-1.64	27.24	48.76	-1.64	-1.64	-99.9
state 10	INTERVENTION [0, 2, 1]	62.2	61.58	43.79	37.63	0	0
state 11	INTERVENTION [1, 2, 1]	37.96	-0.99	48.76	33.68	37.62	-90
state 12	INTERVENTION [0, 0, 2]	79	62.2	62.14	47.53	62.19	-100
state 13	INTERVENTION [1, 0, 2]	43.88	43.78	62.2	-0.9	-0.9	-99.9
state 14	INTERVENTION [0, 1, 2]	0	62.2	43.44	34.21	0	-90
state 15	INTERVENTION [1, 1, 2]	0	33.7	48.76	-0.99	33.7	-90
state 16	INTERVENTION [0, 2, 2]	62.2	0	-0.9	43.79	43.88	-99.9
state 17	INTERVENTION [1, 2, 2]	34.13	-0.9	48.76	-0.9	34.04	-99.9
state 18	WAKE_UP 18	-1.82	39.56	39.53	4.02	-2.31	-100
state 19	STANDARD 19	-1.8	-1.8	-1.8	-1.8	-1.8	-1
state 20	TERMINAL 20	0	0	0	0	0	0

Figura 3.14: Esperimento 1 - Q-table risultante della simulazione.

2 Esperimento 2

I grafici mostrati in figura 3.15 e in figura 3.16 mostrano come il dispositivo diligent ha condotto il suo lavoro durante la simulazione. Quelli mostrati in figura 3.17 e in figura 3.18 fanno riferimento al lavoro condotto dal dispositivo energy saver. Quelli mostrati invece in figura 3.19 e in figura 3.20 mostrano il risultato del dispositivo smart. Come per l'esperimento 1 precedentemente discusso i grafici riportati di seguito mostrano in che modo i tre sistemi abbiano influenzato i valori dei parametri che hanno assunto valori oltre ai limiti.

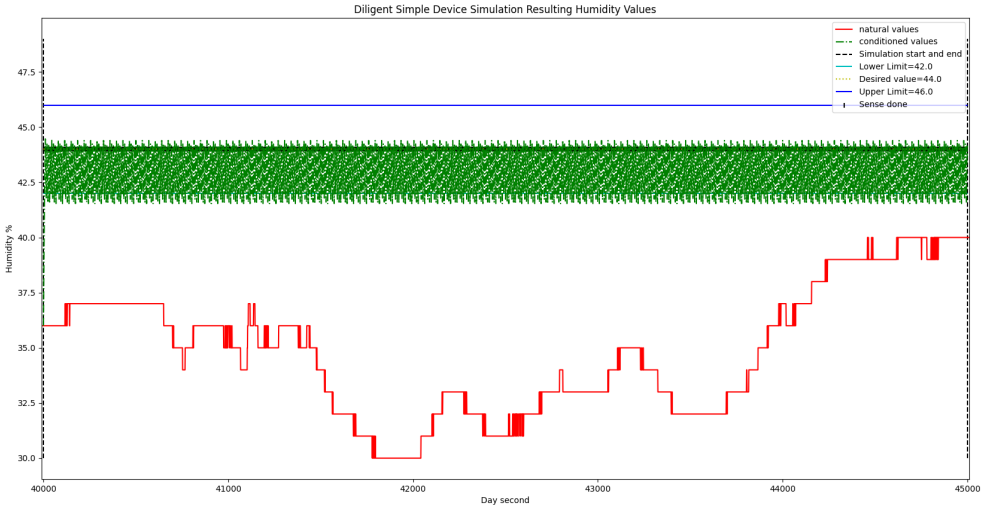


Figura 3.15: Esperimento 2 - Gestione dell'umidità da parte del dispositivo diligent.

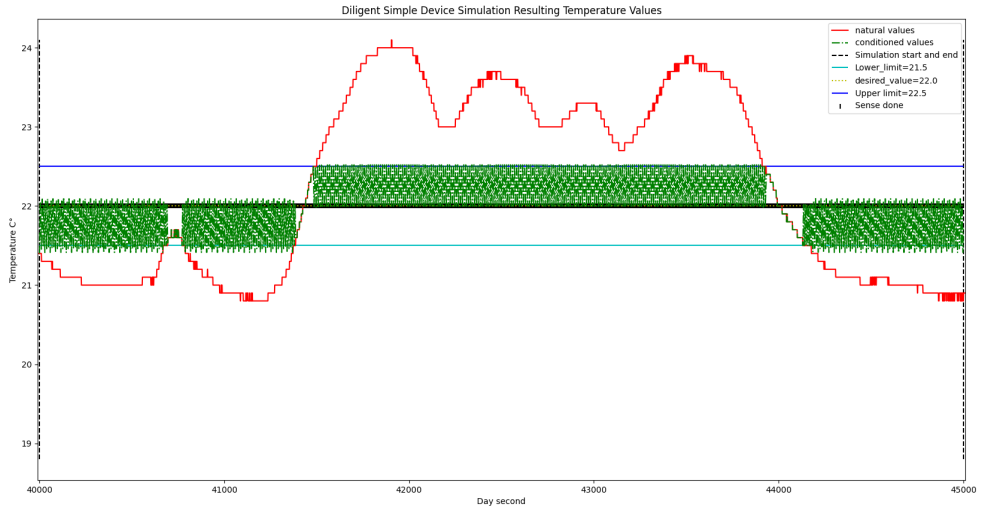


Figura 3.16: Esperimento 2 - Gestione della temperatura da parte del dispositivo diligent.

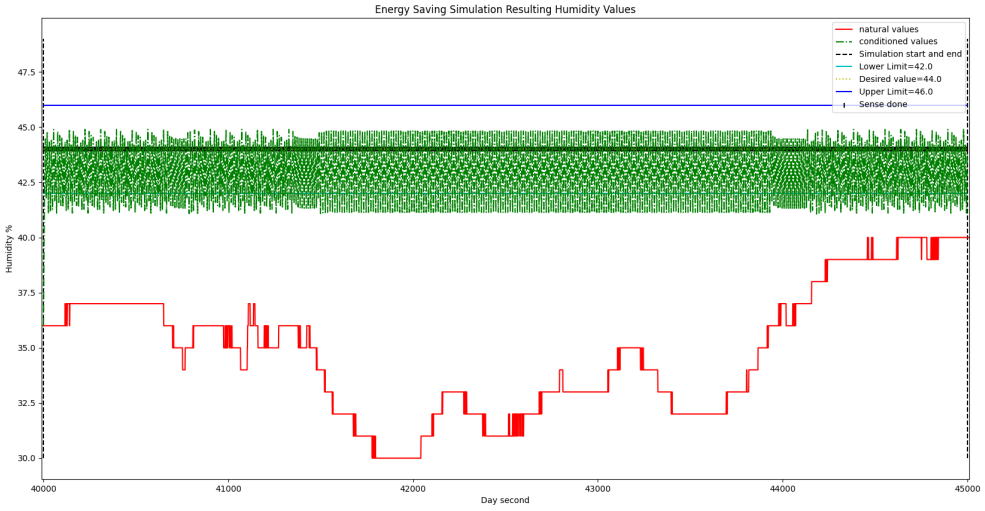


Figura 3.17: Esperimento 2 - Gestione dell'umidità da parte del dispositivo energy saver.

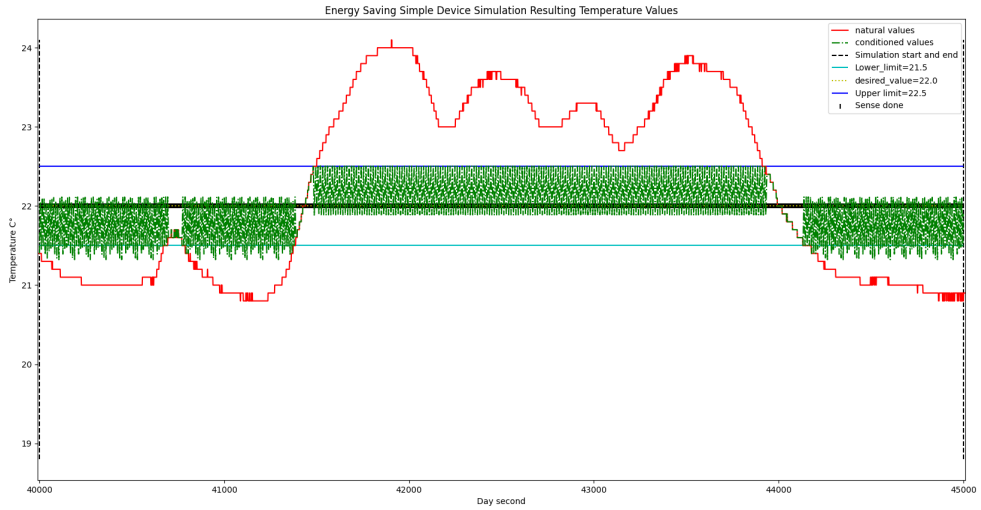


Figura 3.18: Esperimento 2 - Gestione della temperatura da parte del dispositivo energy saver.

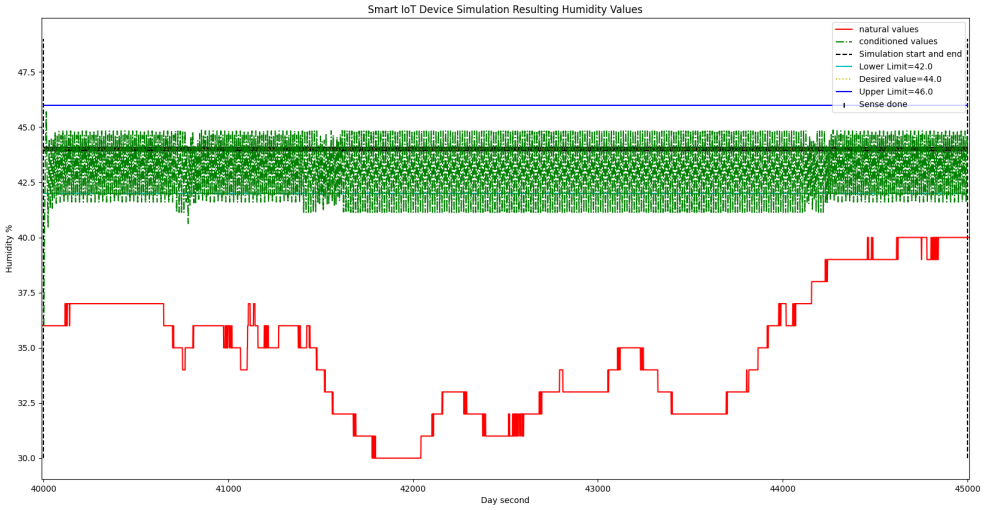


Figura 3.19: Esperimento 2 - Gestione dell'umidità da parte del dispositivo smart.

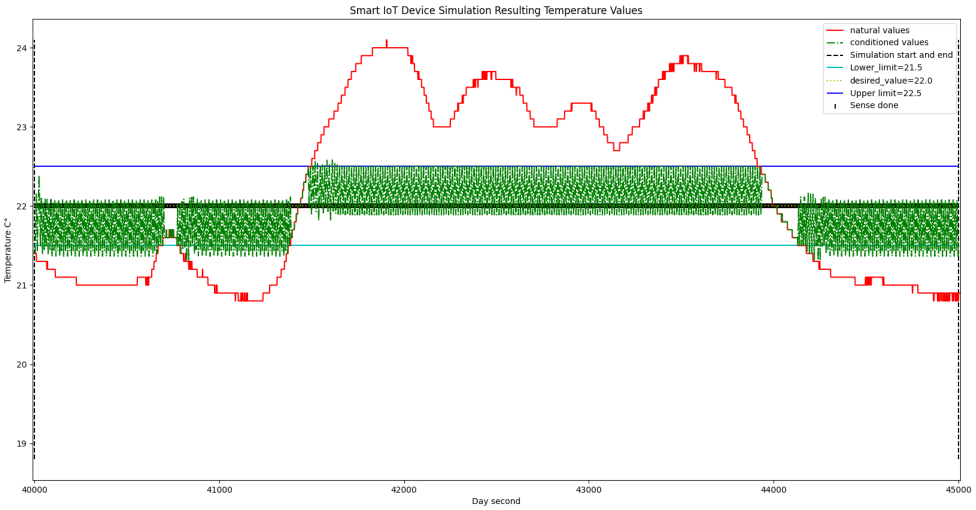


Figura 3.20: Esperimento 2 - Gestione della temperatura da parte del dispositivo smart.

I grafici a barre riportati in figura 3.21 e in figura 3.22 mostrano per quanto tempo i tre sistemi hanno mantenuto i valori dei parametri entro e oltre i limiti consentiti. Come per l'esperimento precedente, il dispositivo diligent è quello che riesce a far assumere per un tempo maggiore valori tollerabili ai parametri ambientali .

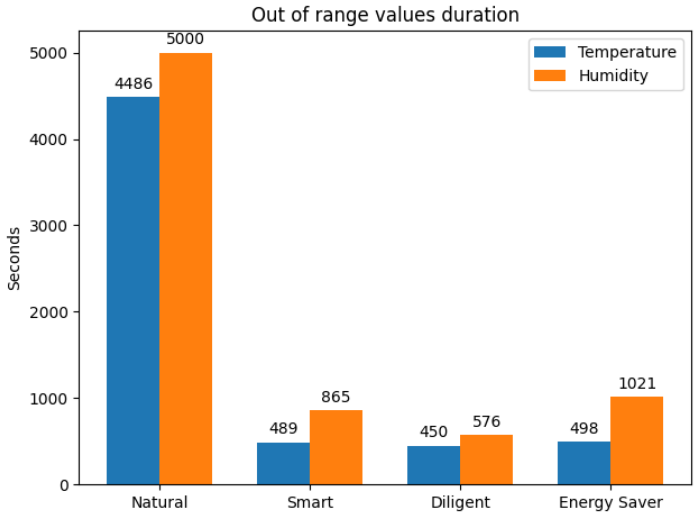


Figura 3.21: Esperimento 2 - Permanenza oltre i limiti del valore assunti dai parametri.

La figura 3.23 riporta la frequenza di esecuzione al secondo delle azioni da parte dei dispositivi durante l'esperimento 2, tale esperimento ha durata di 5000 secondi simulati che corrispondono a 5000 passi temporali di simulazione. In questo caso il dispositivo smart riesce a specializzarsi relativamente in fretta perché gli stati che l'esperimento consente di visitare sono minori di quelli dell'esperimento 1. Inoltre, gli stati sperimentati, vengono rivisitati frequentemente consentendo all'agent di capire in fretta la giusta sequenza di azioni per raggiungere il goal dell'episodio. Come ci si aspettava, e come dimostrato anche nell'esperimento 1, il dispositivo diligent è quello che compie più azioni di sensing mentre quello a risparmio energetico è quello che ne compie meno. Il conteggio delle operazioni di sleep associate al dispositivo diligent mostrano le occasioni in cui il dispositivo avrebbe potuto entrare in risparmio energetico, come però spiegato in precedenza il dispositivo diligent non entra mai in sleep. La scarsa frequenza di sense e il la minor efficienza del sistema di rappresentazione dello stato ambientale del dispositivo semplice energy saver comportano una maggior inefficienza sul controllo dei valori dei parametri. L'inefficienza di questo dispositivo comporta un maggior numero di trasmissioni del dispositivo energy saver rispetto a quello smart. La tabella 3.1 riporta i conteggi delle trasmissioni effettuate dai dispositivi. Il dispositivo diligent compie il maggior numero di trasmissioni perché riesce a identificare più situazioni problematiche.

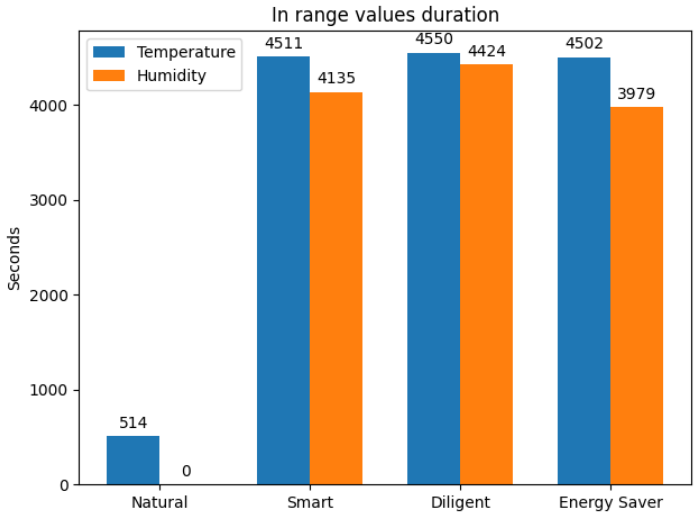


Figura 3.22: Esperimento 2 - Permanenza entro i limiti dei valori assunti dai parametri.

Dispositivo	Conteggio trasmissioni
smart	3091
diligent	3883
sleeper	3446

Tabella 3.1: Esperimento 2 - Conteggio delle trasmissioni

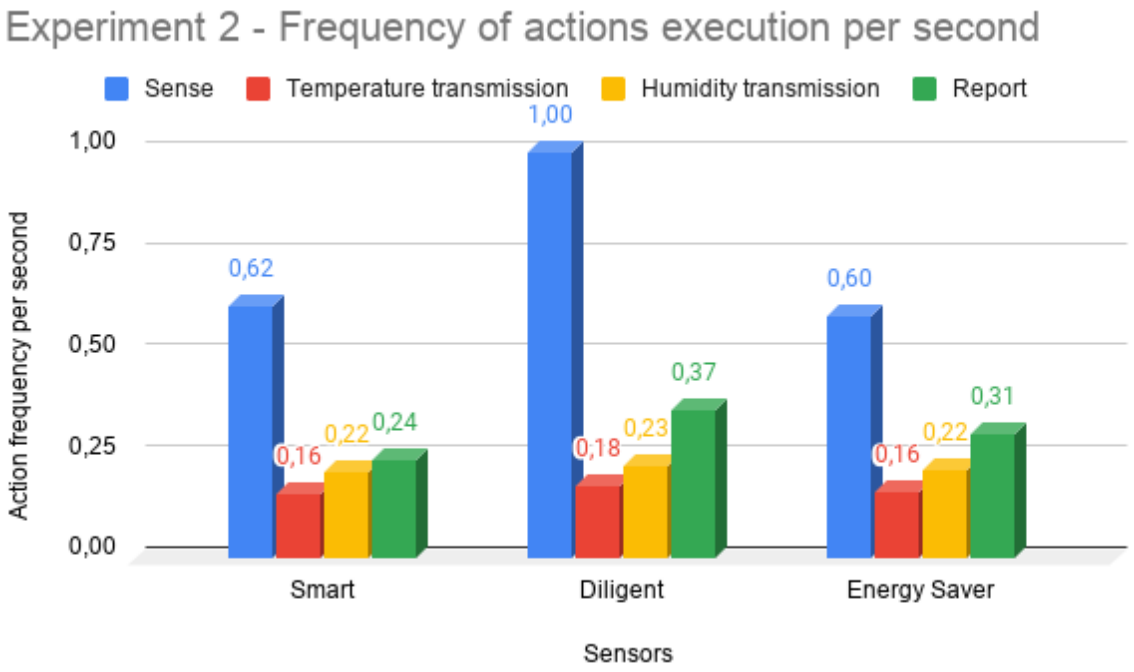


Figura 3.23: Esperimento 2 - Frequenza di esecuzione al secondo delle azioni da parte dei dispositivi.

L'evoluzione della return del dispositivo smart attraverso la simulazione è riportata in figura 3.24. In questo esperimento il dispositivo smart aveva una capacità di esplorazione limitata rispetto a quello dell'esperimento 1. Si può osservare che, inizialmente, il dispositivo ottiene ricompense e calcola valori di bootstrap peggiori. Succede però che, con il proseguire della simulazione, il dispositivo smart riesce a evitare le azioni con ricompensa negativa che corrispondono ai comportamenti non desiderati. Inoltre, il dispositivo smart apprende velocemente a effettuare le giuste sequenze di scelte per raggiungere prima il termine dell'episodio. La figura 3.25 è una prova del fatto che il dispositivo smart abbia sperimentato per un maggior numero di volte episodi in cui venivano compiute un numero di azioni comprese tra 2 e 3. La figura 3.26 supporta la figura 3.25 rafforzando la prova del fatto che il dispositivo abbia commesso più errori durante le fasi iniziali. I grafici mostrano inoltre che il dispositivo sia riuscito a specializzarsi estremamente più velocemente rispetto all'esperimento 1 descritto nella sezione precedente.

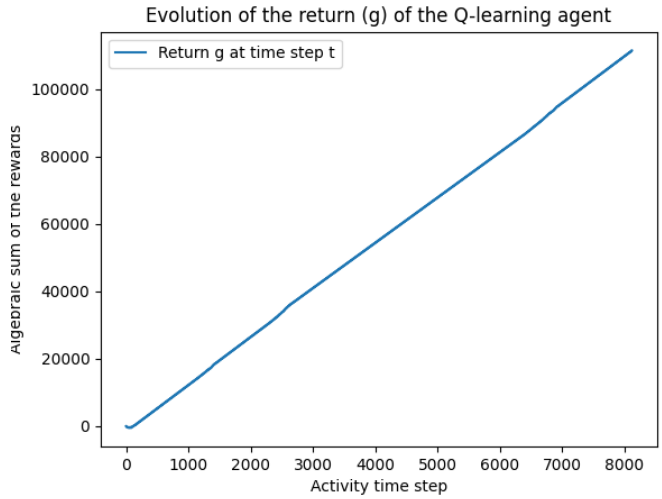


Figura 3.24: Esperimento 2 - Plot dell'evoluzione della somma delle ricompense ricevute.

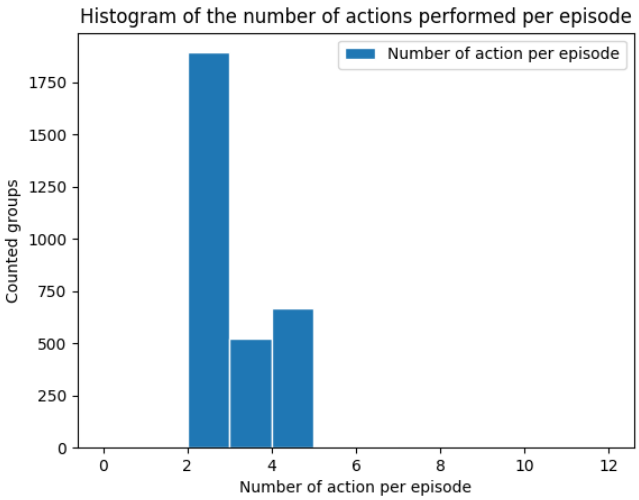


Figura 3.25: Esperimento 2 - Istogramma del conteggio delle azioni per episodio.

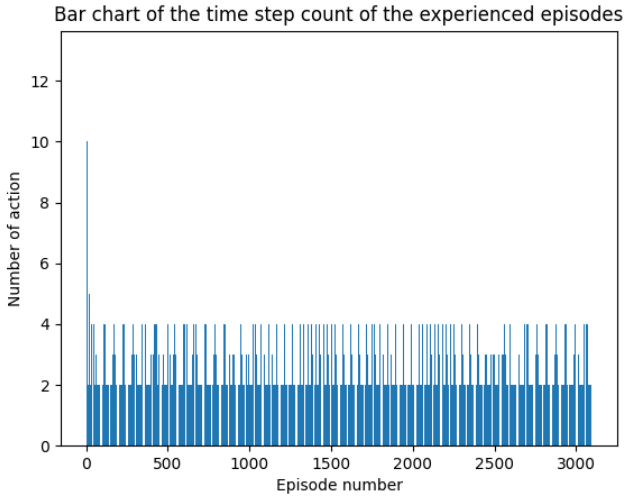


Figura 3.26: Esperimento 2 - Grafico a barre del numero di passi temporali per episodio.

Nell'esperimento 2 il dispositivo ha sperimentato un sottoinsieme della totalità degli stati possibili. Aver potuto sperimentare più frequentemente un limitato numero di stati ha consentito al dispositivo smart una specializzazione parziale che però è avvenuta in breve tempo. La specializzazione parziale è infatti avvenuta in 1836 passi temporali delle traiettorie di 495 episodi, come riportato in figura 3.27. Queste informazioni sono un'ulteriore prova che conferma quanto mostrato nei grafici precedenti che dimostravano un rapido apprendimento del dispositivo smart. Al termine della simulazione la Q-table risultante appresa dal dispositivo smart è quella mostrata in figura 3.28. La velocità di convergenza è comunque influenzata dalla probabilità ϵ delle scelte casuali. In alcuni casi potrebbe infatti capitare che l'agent identifichi al primo colpo l'azione giusta per lo stato incontrato, in questo modo sarebbe possibile fare transizioni più veloci con la conseguenza di una propagazione della ricompensa massima più rapida attraverso la Q-table.

Dunque, in questo esperimento è stato dimostrato che un agent può specializzarsi parzialmente e compiere in modo efficiente lavori che ripropongono abbastanza frequentemente gli stati di un sottoinsieme della totalità degli stati possibili. Il dispositivo ha imparato a comportarsi bene negli scenari che ha incontrato più di frequente e questo stato sufficiente per svolgere in modo abbastanza efficiente il lavoro assegnato.

Number of episodes needed to converge: 495
Number of time steps of the trajectories of the episodes necessary to converge: 1836

s index	s name	temperature_intervention	humidity_intervention	reconnect	sense	report	sleep
state 0	INTERVENTION [0, 0, 0]	-0.9	0	-0.9	-0.9	100	-90
state 1	INTERVENTION [1, 0, 0]	0	0	0	0	0	0
state 2	INTERVENTION [0, 1, 0]	-1	79	-0.9	-0.9	-0.9	-90
state 3	INTERVENTION [1, 1, 0]	0	0	0	0	0	0
state 4	INTERVENTION [0, 2, 0]	-0.99	79	-0.9	-0.9	-0.9	-90
state 5	INTERVENTION [1, 2, 0]	0	0	0	0	0	0
state 6	INTERVENTION [0, 0, 1]	79	0	0	-0.9	0	-90
state 7	INTERVENTION [1, 0, 1]	0	0	0	0	0	0
state 8	INTERVENTION [0, 1, 1]	61.57	-0.99	-1.64	-1.64	-1.64	-90
state 9	INTERVENTION [1, 1, 1]	0	0	0	0	0	0
state 10	INTERVENTION [0, 2, 1]	0	62.2	0	-0.9	-0.9	0
state 11	INTERVENTION [1, 2, 1]	0	0	0	0	0	0
state 12	INTERVENTION [0, 0, 2]	79	-1.64	-1.64	-1.55	-1.64	-90
state 13	INTERVENTION [1, 0, 2]	0	0	0	0	0	0
state 14	INTERVENTION [0, 1, 2]	-1.55	62.2	-1.64	-1.64	-1.64	-90
state 15	INTERVENTION [1, 1, 2]	0	0	0	0	0	0
state 16	INTERVENTION [0, 2, 2]	55.98	0	0	0	0	0
state 17	INTERVENTION [1, 2, 2]	0	0	0	0	0	0
state 18	WAKE_UP 18	-2.24	-2.24	-2.3	-1.04	-2.3	-90
state 19	STANDARD 19	-1.64	-1.64	-1.64	-1.64	-1.64	-1
state 20	TERMINAL 20	0	0	0	0	0	0

Figura 3.27: Esperimento 2 - Risultati dell'analisi della convergenza alla policy ottimale.

s index	s name	temperature_intervention	humidity_intervention	reconnect	sense	report	sleep
state 0	INTERVENTION [0, 0, 0]	-0.9	0	-0.9	-0.9	100	-90
state 1	INTERVENTION [1, 0, 0]	0	0	0	0	0	0
state 2	INTERVENTION [0, 1, 0]	-1	79	-0.9	-0.9	-0.9	-90
state 3	INTERVENTION [1, 1, 0]	0	0	0	0	0	0
state 4	INTERVENTION [0, 2, 0]	-0.99	79	-0.9	-0.9	-0.9	-90
state 5	INTERVENTION [1, 2, 0]	0	0	0	0	0	0
state 6	INTERVENTION [0, 0, 1]	79	0	0	-0.9	0	-90
state 7	INTERVENTION [1, 0, 1]	0	0	0	0	0	0
state 8	INTERVENTION [0, 1, 1]	62.2	-0.99	-1.64	-1.64	-1.64	-90
state 9	INTERVENTION [1, 1, 1]	0	0	0	0	0	0
state 10	INTERVENTION [0, 2, 1]	0	62.2	0	-0.9	-0.9	0
state 11	INTERVENTION [1, 2, 1]	0	0	0	0	0	0
state 12	INTERVENTION [0, 0, 2]	79	-1.64	-1.64	-1.55	-1.64	-90
state 13	INTERVENTION [1, 0, 2]	0	0	0	0	0	0
state 14	INTERVENTION [0, 1, 2]	-1.55	62.2	-1.64	-1.64	-1.64	-90
state 15	INTERVENTION [1, 1, 2]	0	0	0	0	0	0
state 16	INTERVENTION [0, 2, 2]	62.2	0	0	0	0	0
state 17	INTERVENTION [1, 2, 2]	0	0	0	0	0	0
state 18	WAKE_UP 18	-2.24	-2.24	-2.3	2.76	-2.3	-90
state 19	STANDARD 19	-1.64	-1.64	-1.64	-1.64	-1.64	-1
state 20	TERMINAL 20	0	0	0	0	0	0

Figura 3.28: Esperimento 2 - Q-table risultante.

3 Esperimento 3

In questo esperimento è stato dimostrato che il dispositivo dotato di Q-table che converge alla policy ottimale è il più efficiente e riesce a svolgere in modo specializzato il lavoro assegnato.

La figura 3.29 e 3.30 dimostra che il dispositivo diligent è quello che in ogni caso riesce a mantenere per più tempo i valori dei parametri entro i limiti stabiliti. Il dispositivo smart si è comportato in modo ancora più efficiente rispetto all'esperimento 2, questo è dovuto all'utilizzo di una Q-table di convergenza completa. In figura 3.31 si nota che il dispositivo smart è quello che ha prestazioni migliori dal punto di vista del risparmio energetico. Questo per via dei seguenti motivi:

- Il dispositivo smart esegue solo la giusta sequenza di azioni. Questo rende il dispositivo smart dell'esperimento 3 più efficiente di tutti gli altri dispositivi smart analizzati nelle altre simulazioni.

- Il dispositivo smart è dotato di sistema di rappresentazione dello stato ambientale. Questo sistema consente al dispositivo di identificare in modo più efficiente lo stato in cui l'ambiente si trova. I dispositivi semplici, non essendo dotati di alcun sistema, rappresentano lo stato ambientale e prendono le loro decisioni esclusivamente mediante una semplice sequenza di istruzioni if then else.

- la Q-table di convergenza importata dal dispositivo smart, la decisione sequenziale del Q-learning e il sistema di rappresentazione ambientale, lavorando in modo sinergico, consentono al dispositivo smart di essere il più efficiente dal punto di vista del risparmio energetico. Il dispositivo smart riesce infatti a identificare meglio i momenti in cui dormire e i momenti in cui intervenire effettuando trasmissioni.

Dunque, il risparmio della batteria è dovuto al fatto che il dispositivo smart riesce a compiere il minor numero di trasmissioni. Le trasmissioni sono infatti l'azione più dispendiosa a disposizione dei dispositivi di monitoraggio. Anche se il dispositivo energy saver compie meno operazioni di sensing effettua comunque troppe trasmissioni ed ha prestazioni peggiori del dispositivo smart. I conteggi delle trasmissioni compiute dai dispositivi vengono mostrati in nella tabella 3.2

Dispositivo	Conteggio trasmissioni
smart	3056
diligent	3883
sleeper	3446

Tabella 3.2: Esperimento 3 - Conteggio delle trasmissioni

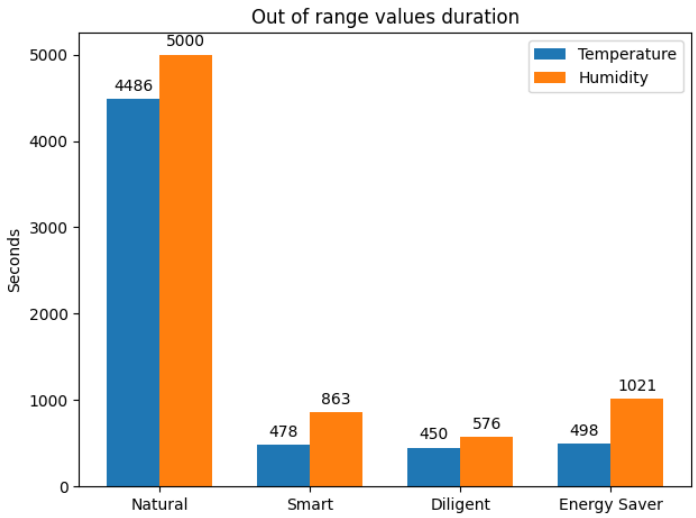


Figura 3.29: Esperimento 3 - Permanenza oltre i limiti del valore assunti dai parametri..

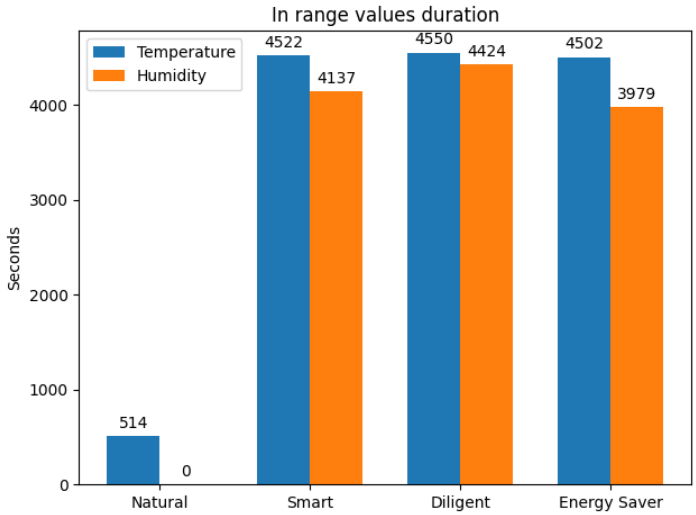


Figura 3.30: Esperimento 3 - Permanenza entro i limiti dei valori assunti dai parametri.

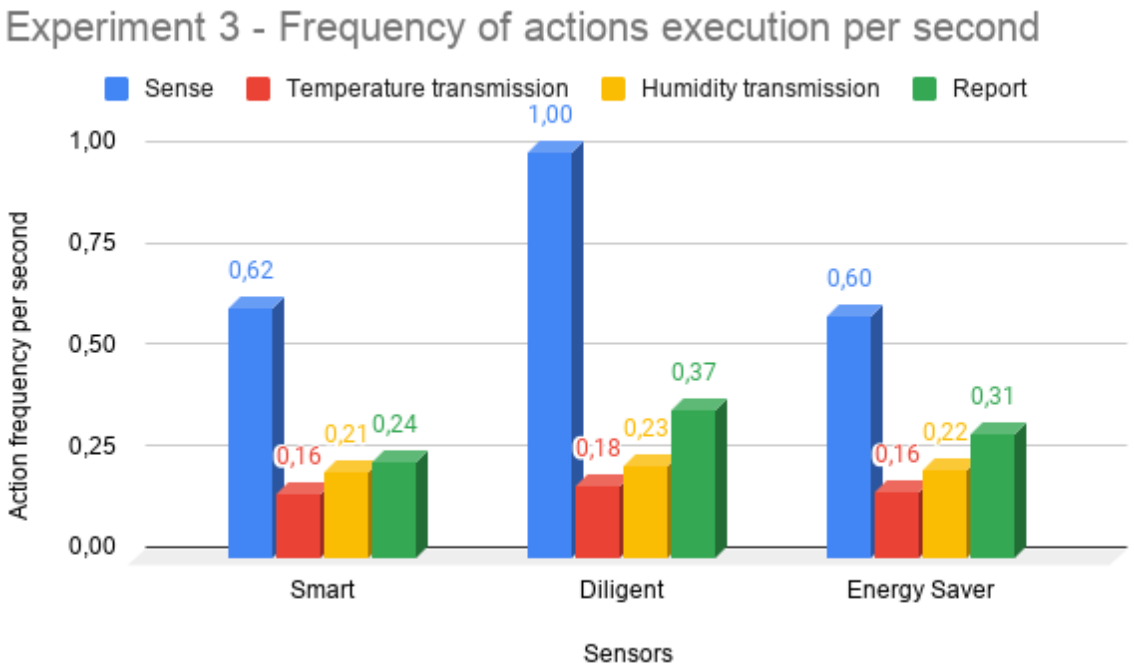


Figura 3.31: Esperimento 3 - Frequenza di esecuzione al secondo delle azioni da parte dei dispositivi.

Osservando la figura 3.32 si può constatare che il dispositivo compie solo azioni corrette. Per ogni azione diversa dal report ottiene una ricompensa di -1 e per ogni azione report compiuta ottiene 100 punti. Non vengono mai compiute azioni di sleep durante gli stati di intervento, questa è una prova della specializzazione del dispositivo smart che decide ed esegue le azioni in modo ottimale. Il dispositivo smart colleziona ricompense che garantiscono una crescita della return che ha andamento lineare. Una prova ulteriore di quanto detto fino'ora è data dall'istogramma mostrato in figura 3.33 dove si può osservare che il dispositivo ha sempre e solo compiuto le azioni strettamente indispensabili. Ulteriore prova del fatto che il dispositivo ha compiuto esclusivamente le azioni necessarie a raggiungere nel modo più rapido possibile il goal dell'episodio viene mostrata nel grafico a barre della figura 3.34, si nota infatti che gli episodi hanno al massimo una durata di 4 passi temporali. Gli episodi composti da 4 passi temporali prevedono un'azione sense, un'azione di intervento e un'azione di trasmissione. L'ultimo passo temporale dell'episodio è dedicato al raggiungimento dello stato terminale.

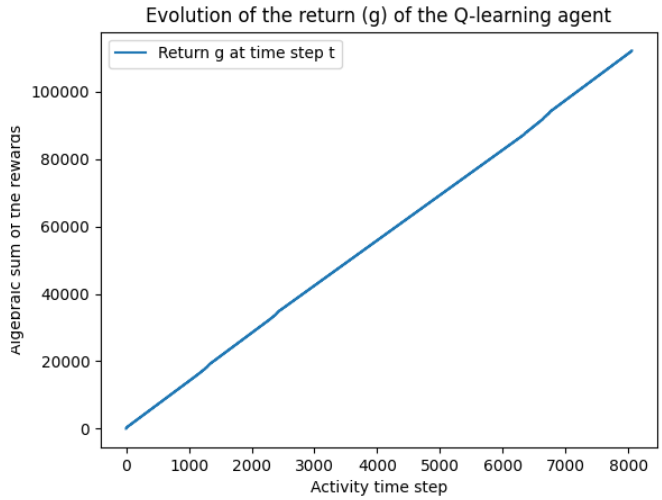


Figura 3.32: Esperimento 3 - Plot dell'evoluzione della somma delle ricompense ricevute.

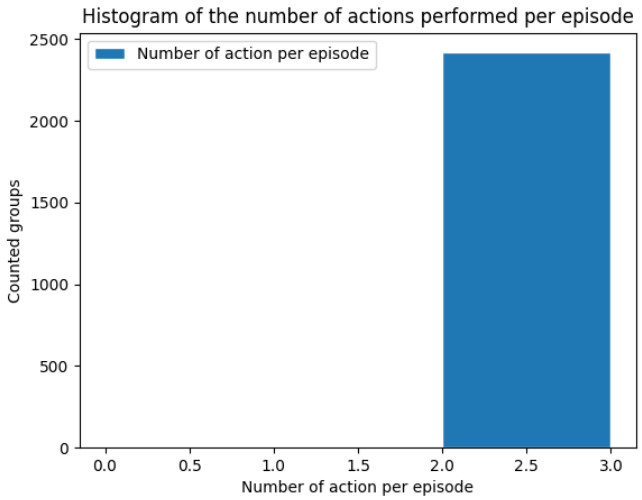


Figura 3.33: Esperimento 3 - Istogramma del conteggio delle azioni per episodio.

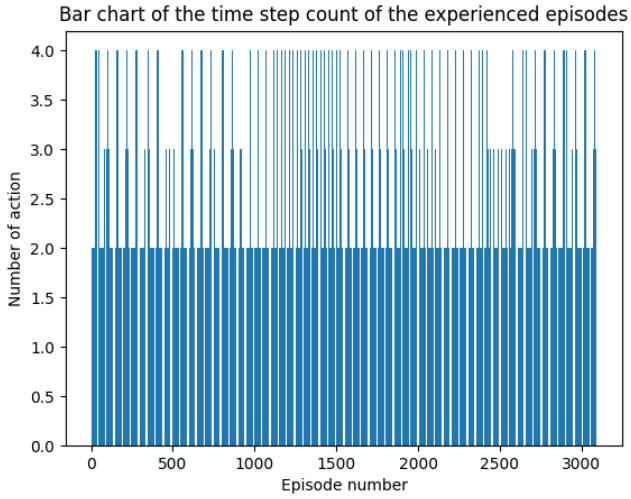


Figura 3.34: Esperimento 3 - Grafico a barre del numero di passi temporali per episodio.

Dal momento che il dispositivo smart ha importato una Q-table di convergenza, il sistema di analisi della convergenza ha rilevato che la specializzazione è avvenuta in 0 passi temporali come mostrato in figura 3.35. In questo esperimento il bootstrap è stato inibito e di conseguenza la Q-table risultante è la stessa della Q-table di convergenza importata ad inizio sperimentazione come mostrato in figura 3.36. L'inibizione è stata adottata per evitare che la qualità dell'azione sense nello stato 18 potesse divenire minore di quella delle altre azioni. Questo succede nel processo di apprendimento perché il sense varia il suo valore di bootstrap perché conduce verso stati ad alta ricompense oppure verso lo stato standard dove la ricompensa massima è -1. In contesti di apprendimento questa condizione è lecita e non da alcun problema perché il dispositivo riesce a raggiungere comunque la fine dell'episodio anche se potrebbe dover provare più azioni. In questo esperimento si è voluto ottenere risultati correlati al solo utilizzo di scelte greedy associate a una Q-table di convergenza.

Number of episodes needed to converge: 1
Number of time steps of the trajectories of the episodes necessary to converge: 0

s index	s name	temperature_intervention	humidity_intervention	reconnect	sense	report	sleep
state 0	INTERVENTION [0, 0, 0]	79	79	79	50.12	100	-100
state 1	INTERVENTION [1, 0, 0]	62.14	62.2	79	62.19	62.19	-99.9
state 2	INTERVENTION [0, 1, 0]	62.14	79	62.2	55.82	61.56	-99.9
state 3	INTERVENTION [1, 1, 0]	-1.64	-0.9	62.2	-1	-1	-90
state 4	INTERVENTION [0, 2, 0]	55.98	79	61.58	34.04	62.2	-99.9
state 5	INTERVENTION [1, 2, 0]	-1.71	-1.71	-0.99	-1.64	-1.64	-99.9
state 6	INTERVENTION [0, 0, 1]	79	62.14	61.58	62.2	62.14	-99.99
state 7	INTERVENTION [1, 0, 1]	-0.9	43.79	62.2	-0.9	48.71	-99.9
state 8	INTERVENTION [0, 1, 1]	62.13	62.2	48.17	43.75	-0.9	-99.9
state 9	INTERVENTION [1, 1, 1]	-1.64	27.24	39.31	-1.64	-1.64	-99
state 10	INTERVENTION [0, 2, 1]	62.2	61.58	43.79	37.63	0	0
state 11	INTERVENTION [1, 2, 1]	37.96	-0.99	48.76	33.68	37.62	-90
state 12	INTERVENTION [0, 0, 2]	79	62.2	62.14	47.53	62.19	-100
state 13	INTERVENTION [1, 0, 2]	0	43.78	62.2	-0.9	-0.9	-99
state 14	INTERVENTION [0, 1, 2]	0	62.2	43.44	34.21	0	-90
state 15	INTERVENTION [1, 1, 2]	0	33.7	48.76	-0.99	33.7	-90
state 16	INTERVENTION [0, 2, 2]	61.52	0	-0.9	-0.9	0	-90
state 17	INTERVENTION [1, 2, 2]	0	-0.9	38.99	-0.9	-0.9	-99
state 18	WAKE_UP 18	-1.32	-1.4	-1.35	-0.95	-1.37	-100
state 19	STANDARD 19	-1.8	-1.8	-1.8	-1.8	-1.8	-1
state 20	TERMINAL 20	0	0	0	0	0	0

Figura 3.35: Esperimento 3 - Risultato dell'analisi della convergenza alla policy ottimale.

s index	s name	temperature_intervention	humidity_intervention	reconnect	sense	report	sleep
state 0	INTERVENTION [0, 0, 0]	79	79	79	50.12	100	-100
state 1	INTERVENTION [1, 0, 0]	62.14	62.2	79	62.19	62.19	-99.9
state 2	INTERVENTION [0, 1, 0]	62.14	79	62.2	55.82	61.56	-99.9
state 3	INTERVENTION [1, 1, 0]	-1.64	-0.9	62.2	-1	-1	-90
state 4	INTERVENTION [0, 2, 0]	55.98	79	61.58	34.04	62.2	-99.9
state 5	INTERVENTION [1, 2, 0]	-1.71	-1.71	-0.99	-1.64	-1.64	-99.9
state 6	INTERVENTION [0, 0, 1]	79	62.14	61.58	62.2	62.14	-99.99
state 7	INTERVENTION [1, 0, 1]	-0.9	43.79	62.2	-0.9	48.71	-99.9
state 8	INTERVENTION [0, 1, 1]	62.13	62.2	48.17	43.75	-0.9	-99.9
state 9	INTERVENTION [1, 1, 1]	-1.64	27.24	39.31	-1.64	-1.64	-99
state 10	INTERVENTION [0, 2, 1]	62.2	61.58	43.79	37.63	0	0
state 11	INTERVENTION [1, 2, 1]	37.96	-0.99	48.76	33.68	37.62	-90
state 12	INTERVENTION [0, 0, 2]	79	62.2	62.14	47.53	62.19	-100
state 13	INTERVENTION [1, 0, 2]	0	43.78	62.2	-0.9	-0.9	-99
state 14	INTERVENTION [0, 1, 2]	0	62.2	43.44	34.21	0	-90
state 15	INTERVENTION [1, 1, 2]	0	33.7	48.76	-0.99	33.7	-90
state 16	INTERVENTION [0, 2, 2]	61.52	0	-0.9	-0.9	0	-90
state 17	INTERVENTION [1, 2, 2]	0	-0.9	38.99	-0.9	-0.9	-99
state 18	WAKE_UP 18	-1.32	-1.4	-1.35	-0.95	-1.37	-100
state 19	STANDARD 19	-1.8	-1.8	-1.8	-1.8	-1.8	-1
state 20	TERMINAL 20	0	0	0	0	0	0

Figura 3.36: Esperimento 3 - Q-table risultante.

4 Esperimento 4

Il prototipo ha superato il test condotto durante l'esperimento 4. La sequenza di illuminazione relativa alle conduzione degli episodi e alle sequenze di azioni eseguite che il prototipo ha mostrato erano quelle che ci si aspettava. In figura 3.37 si può osservare che il logging su monitor seriale dei dati prodotti dal prototipo in merito alle traiettorie coincide con quelli previsti dal test.

```
[Episode 1 Trajectory]
s_1 = 0, a_1 = 0 21.00 c°, r_2 = -1, s_2 = 5, a_2 = 5, r_3 = -1, s_3 = 7.
[Episode 2 Trajectory]
s_1 = 0, a_1 = 0 16.00 c°, r_2 = -1, s_2 = 1, a_2 = 1, r_3 = -1, s_3 = 6, a_3 = 6, r_4 = 100, s_4 = 7.
[Episode 3 Trajectory]
s_1 = 0, a_1 = 0 19.00 c°, r_2 = -1, s_2 = 5, a_2 = 5, r_3 = -1, s_3 = 7.
[Episode 4 Trajectory]
s_1 = 0, a_1 = 0 20.00 c°, r_2 = -1, s_2 = 2, a_2 = 2, r_3 = -1, s_3 = 6, a_3 = 6, r_4 = 100, s_4 = 7.
[Episode 5 Trajectory]
s_1 = 0, a_1 = 0 20.00 c°, r_2 = -1, s_2 = 5, a_2 = 5, r_3 = -1, s_3 = 7.
[Episode 6 Trajectory]
s_1 = 0, a_1 = 0 23.00 c°, r_2 = -1, s_2 = 3, a_2 = 3, r_3 = -1, s_3 = 6, a_3 = 6, r_4 = 100, s_4 = 7.
[Episode 7 Trajectory]
s_1 = 0, a_1 = 0 21.00 c°, r_2 = -1, s_2 = 5, a_2 = 5, r_3 = -1, s_3 = 7.
[Episode 8 Trajectory]
s_1 = 0, a_1 = 0 20.00 c°, r_2 = -1, s_2 = 4, a_2 = 4, r_3 = -1, s_3 = 6, a_3 = 6, r_4 = 100, s_4 = 7.
[Episode 9 Trajectory]
s_1 = 0, a_1 = 0 20.00 c°, r_2 = -1, s_2 = 5, a_2 = 5, r_3 = -1, s_3 = 7.
```

Figura 3.37: Esperimento 4 - Output del test che mostra le traiettorie sperimentate.

5 Esperimento 5

In questo esperimento si è potuto osservare il comportamento del prototipo smart IoT in uno scenario reale. Come mostrato nel grafico della figura 3.38, il sensore ha funzionato correttamente riuscendo a misurare i valori ambientali. I picchi di temperatura sono dovuti agli strumenti adottati al fine di simulare gli effetti degli attuatori. L'asciuga capelli riscalda molto velocemente il sensore rendendolo difficile da raffreddare. Il dispositivo ha comunque rappresentato correttamente lo stato ambientale in ogni momento ed ha scelto la giusta sequenza di azioni per completare il task. Questo appena detto è infatti osservabile in figura 3.43 dove è possibile osservare che il grafico a barre riporta traiettorie più lunghe in relazione agli episodi in cui sono state registrate temperature oltre i limiti stabiliti.

Il dispositivo è stato inoltre in grado di identificare correttamente tutti i momenti in cui entrare in sleep. Oltre alla prova fornita dalle corrette sequenze di accensioni dei LED osservate durante l'esperimento, è possibile constatare quanto detto fino ad ora guardando un estratto del file di output prodotto dal prototipo durante l'esperimento e che viene mostrato in figura 3.40. Consultando il file mostrato in figura 3.40 si possono identificare le azioni sense() effettuate durante lo stato Start osservando le righe in cui si può trovare la sequenza di numeri "0,0", si può invece identificare l'esecuzione dello sleep() durante lo stato Sleep osservando le righe in cui si può trovare la sequenza di numeri "5,5". Un'ulteriore prova di quanto appena detto la si ottiene osservando i grafici riportati in figura 3.42 e in figura 3.43. In tali grafici infatti si osserva che sono stati sperimentate molte traiettorie in cui sono state compiute solo due azioni, sense e sleep, in concomitanza di quegli episodi in cui i valori di temperatura erano entro i limiti stabiliti.

Come mostrato in figura 3.39, il prototipo smart IoT ha permesso di mantenere i valori di temperatura entro i limiti nell'82,9% degli episodi sperimentati. Questo è dovuto all'inefficienza degli effetti degli attuatori simulati. Con altri sistemi si sarebbe ottenuto un punteggio migliore. Nella figura 3.41 viene mostrata la frequenza, per passo temporale di sperimentazione, con cui il prototipo ha eseguito le azioni a disposizione durante l'esperimento. La figura mostra che è stato scelto dall'agent del prototipo di compiere esclusivamente le trasmissioni strettamente necessarie alla conduzione, in modo efficiente, del compito assegnato. Dunque, le azioni compiute atte a simulare le trasmissioni all'attuatore e al data center sono state effettuate esclusivamente nei casi in cui il valore della temperatura era effettivamente oltre i limiti stabiliti.

Il Prototipo ha sperimentato 129 episodi nei quali è stato scelto di entrare in sleep 147 volte. Negli altri esperimenti, condotti utilizzando il simulatore del sistema software di sperimentazione, ogni passo temporale di sperimentazione è consistito nello scorrimento di un secondo simulato in cui è stato sperimentato, dai dispositivi simulati, un intero episodio. Come spiegato, nelle sezioni dedicate al prototipo e agli esperimenti con il prototipo, gli episodi sperimentati dal prototipo durano più di un secondo in modo da poter

consentire all'operatore umano che conduce l'esperimento di poter intervenire, nei panni di attuatore, in modo reattivo in risposta ai segnali del dispositivo. Per questo motivo il grafico in figura 3.41, relativo alla frequenza di esecuzione delle azioni, utilizza come unità di tempo il passo temporale dell'esperimento. Per passo temporale dell'esperimento si intende quella partizione di tempo caratterizzata o dall'episodio che il prototipo sperimenta, oppure, dall'episodio che il dispositivo evita di sperimentare avendo scelto di entrare in sleep. Questo è stato fatto per creare un termine di paragone tra i risultati del simulatore e i risultati degli esperimenti con il prototipo. Nel simulatore l'entrata in sleep corrispondeva a mandare in risparmio energetico il dispositivo per 1 secondo, e quindi facendo andare avanti per un secondo il simulatore considerando le conseguenze sull'ambiente dell'inattività. Nello scenario reale l'azione di sleep viene conteggiata come passo temporale passato in risparmio energetico e quindi si calcola uno step in cui il dispositivo non ha potuto comunicare necessità di interventi, in questo caso è stata la fisica dell'ambiente reale a intervenire sull'evoluzione della temperatura sfruttando il delay impostato dall'azione di sleep. Dunque, il prototipo che ha scelto di entrare 147 volte in sleep ha evitato 147 episodi perché in quei passi temporali era in modalità di risparmio energetico e quindi inattivo.

Per concludere, se si osserva la figura 3.44 si ha un'ulteriore prova del corretto svolgimento, da parte del prototipo, del compito assegnato. Il prototipo ha ottenuto elevate ricompense quando ha segnalato le situazioni sfavorevoli. Il prototipo non ha mai ricevuto ricompense di -100 perché essendo specializzato non ha mai effettuato lo sleep() durante gli stati di intervento. Inoltre si osserva una diminuzione delle ricompense a partire circa dall'episodio 100, questo è dovuto al fatto che il dispositivo non ha più incontrato situazioni critiche in cui intervenire ottenendo elevate ricompense.

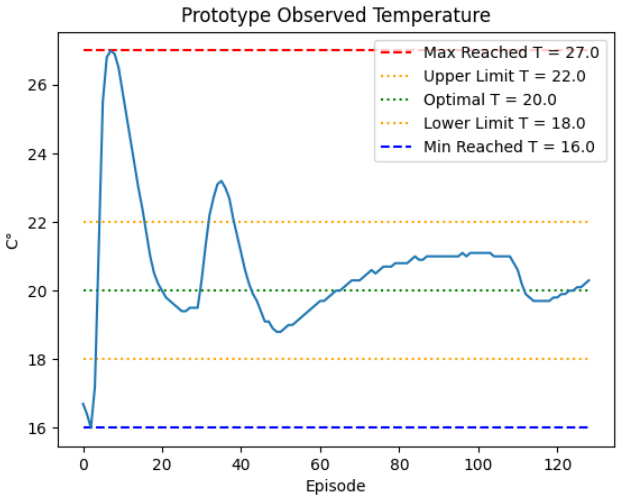


Figura 3.38: Esperimento 5 - Plot del lavoro di monitoraggio del prototipo.

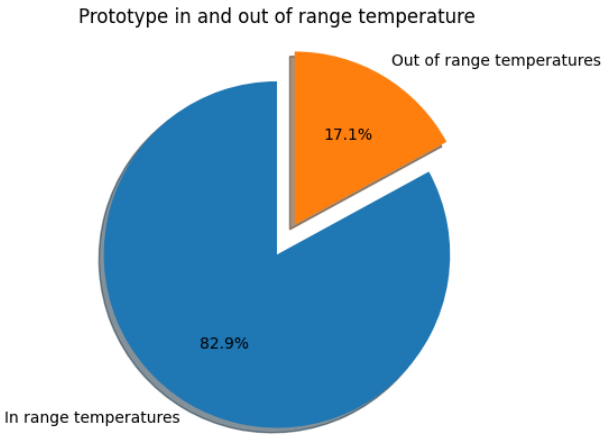


Figura 3.39: Esperimento 5 - Percentuale di temperature entro e oltre i limiti registrate dal prototipo.

```
0,0,19.90,-1,4,4,-1,6,6,100,7
0,0,19.70,-1,5,5,-1,7
0,0,19.40,-1,5,5,-1,7
0,0,19.10,-1,5,5,-1,7
0,0,19.10,-1,5,5,-1,7
0,0,18.90,-1,5,5,-1,7
0,0,18.80,-1,5,5,-1,7
0,0,18.80,-1,5,5,-1,7
0,0,18.90,-1,5,5,-1,7
0,0,19.00,-1,5,5,-1,7
0,0,19.00,-1,5,5,-1,7
0,0,19.10,-1,5,5,-1,7
0,0,19.20,-1,5,5,-1,7
0,0,19.30,-1,5,5,-1,7
0,0,19.40,-1,5,5,-1,7
0,0,19.50,-1,5,5,-1,7
0,0,19.60,-1,5,5,-1,7
0,0,19.70,-1,5,5,-1,7
0,0,19.70,-1,5,5,-1,7
0,0,19.80,-1,5,5,-1,7
```

Figura 3.40: Esperimento 5 - Estratto del file dati prodotto dal prototipo durante l'esperimento.

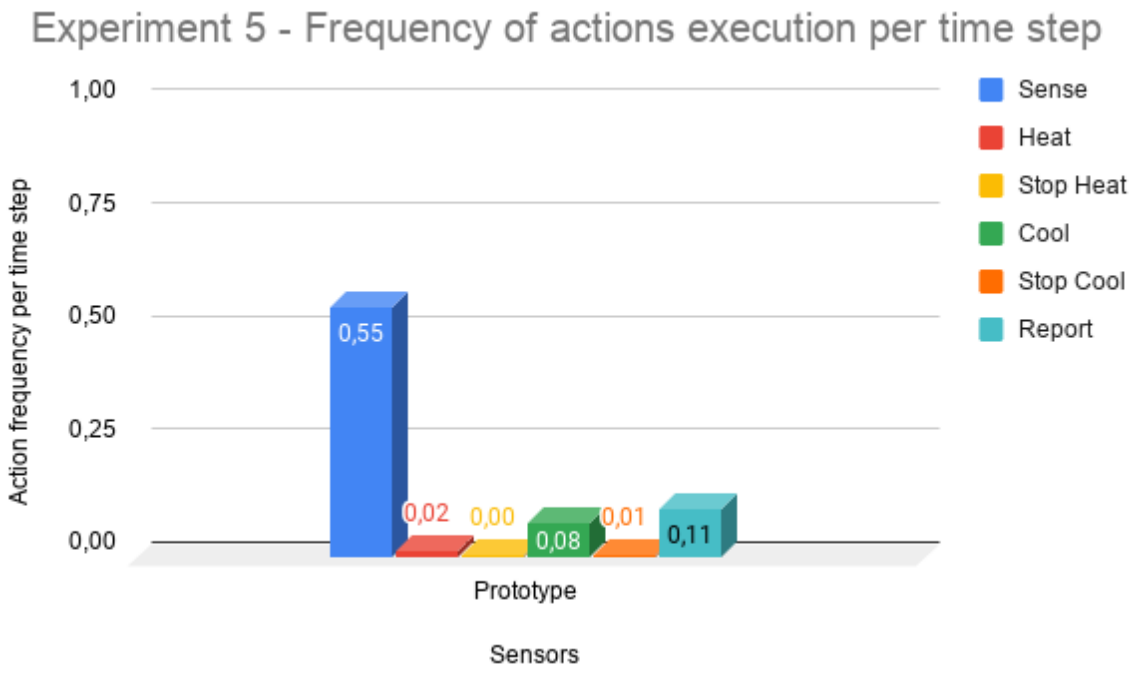


Figura 3.41: Esperimento 5 - Frequenza di esecuzione per passo temporale dell'esperimento delle azioni da parte del prototipo.

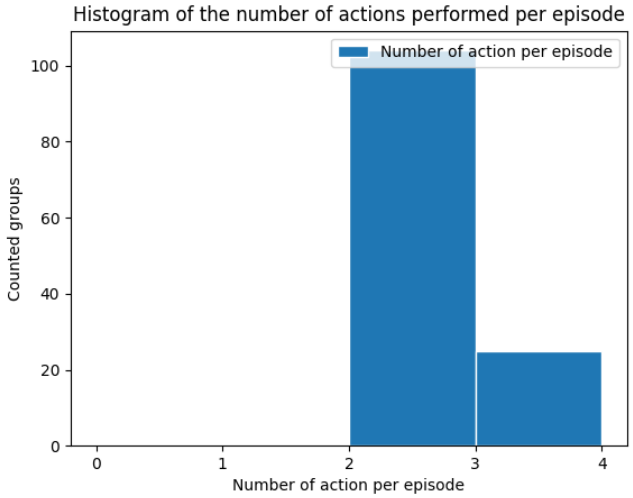


Figura 3.42: Esperimento 5 - Istogramma del conteggio delle azioni per episodio.

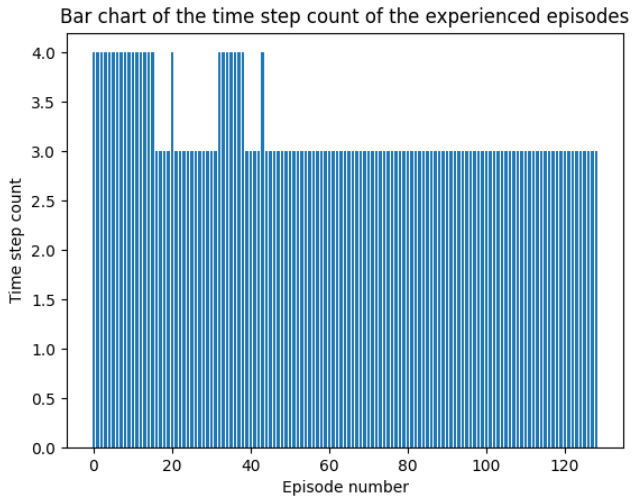


Figura 3.43: Esperimento 5 - Grafico a barra della durata degli episodi sperimentati.

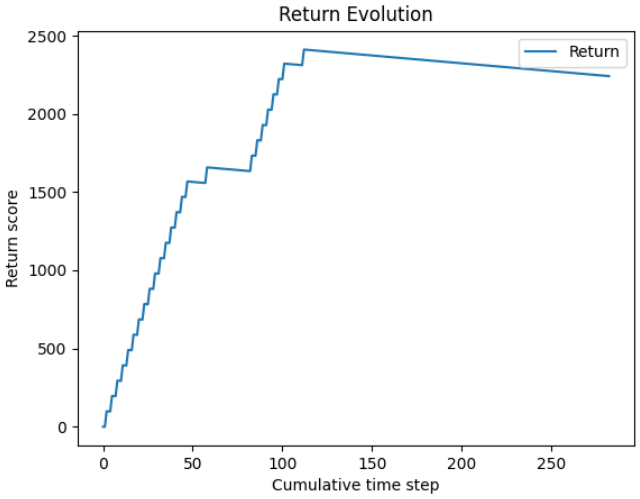


Figura 3.44: Esperimento 5 - Plot dell'evoluzione della return del prototipo.

Capitolo 4

Conclusioni

I risultati degli esperimenti hanno dimostrato che un problema descrivibile sotto forma di catena di decisioni è risolvibile mediante la tecnica di Reinforcement Learning che prende il nome di Q-learning. Per ottenere i risultati analizzati in questo studio sono stati simulati e testati tre diversi sistemi IoT di controllo ambientale. Durante gli esperimenti due sistemi di controllo ambientale sono stati governati da semplici dispositivi simulati che sono stati programmati senza tecniche di Machine Learning per monitorare l'ambiente. Il dispositivo semplice, chiamato diligent, non è mai entrato in modalità di risparmio energetico durante le simulazioni. Il suo scopo è stato quello di rappresentare il sistema più efficiente dal punto di visto di prontezza di intervento sui parametri ambientali che assumevano valori al di fuori dei limiti consentiti. Il dispositivo semplice, chiamato energy saver, è entrato in modalità di risparmio energetico durante le simulazioni in quei momenti in cui i parametri ambientali hanno assunto valori entro i limiti consentiti. Il suo scopo è stato quello di rappresentare il sistema in grado di ridurre il numero di azioni sense compiute durante la simulazione al fine di tentare di risparmiare energia. Il terzo dispositivo simulato di monitoraggio che è stato impiegato nella simulazione è stato dotato di agente di Q-learning. Questo dispositivo, a cui sono state rivolte le maggiori attenzioni di questo studio, ha dimostrato di poter apprendere passo dopo passo a svolgere il lavoro assegnato in modo corretto ed efficiente unendo i punti di forza dei due dispositivi semplici di controllo. Gli esperimenti condotti hanno dimostrato che, dopo la specializzazione, il dispositivo smart riesce a mantenere i valori dei parametri entro i limiti meglio del dispositivo energy saver e riesce a consumare meno del dispositivo diligent.

L'efficienza del dispositivo smart IoT, e del sistema in generale, dipende fortemente dal modo in cui viene modellato sotto forma di catena decisionale il problema affrontato. Più la modellazione risulta essere esaustiva, di conseguenza, maggiore il dispositivo riesce ad essere efficiente e a fornire funzionalità utili alla risoluzione del problema. Identificare un numero maggiore di stati del problema utili al completamento del task fornisce vantaggi al dispositivo che riesce ad apprendere il comportamento migliore in un insieme

più ampio di situazioni. Infatti, una suddivisione più articolata degli stati ambientali del problema esaminato in questo studio potrebbe consentire di identificare più situazioni in cui il dispositivo smart potrebbe trarre vantaggio dall'entrare in modalità di risparmio energetico senza compromettere la reattività di intervento sui parametri.

Uno degli aspetti che rallentano l'apprendimento del dispositivo smart è la frequenza con cui esso incontra uno stato. Anche in questo studio è stato confermato quanto detto in letteratura, il dispositivo impara a comportarsi meglio negli stati che incontra più di frequente [23]. La specializzazione in compiti svolti più frequentemente è una delle caratteristiche che il Reinforcement Learning ha in comune con gli esseri umani.

Gli esperimenti condotti utilizzando il prototipo hardware del dispositivo smart IoT hanno dimostrato la reale possibilità di risolvere i problemi descritti mediante il Q-learning utilizzando dispositivi che possono essere anche dotati di risorse limitate come, per esempio, le schede di sviluppo per microcontrollori e semplici componenti elettronici.

Lo studio ha dimostrato che, di fronte a un problema complesso e opportunamente modellato, è possibile realizzare un ambiente di allenamento simulato in cui l'agent, svolgendo il suo processo di apprendimento, riesce a identificare una Q-table che converge alla policy ottimale. Il risultato della fase di apprendimento può essere utilizzato come comportamento da fornire ai dispositivi utilizzati in contesti reali per poter svolgere compiti complessi. In questo modo i dispositivi riserverebbero le conseguenze delle decisioni sbagliate ad un contesto in cui gli errori non causano alcun tipo di danno o problema.

Uno sviluppo futuro del lavoro svolto potrebbe essere quello di predisporre un sistema decisionale condiviso tra diversi dispositivi smart collocati in ambienti con caratteristiche che possono differire anche di molto tra loro. Ogni dispositivo potrebbe inviare al centro decisionale le informazioni percepite relative all'ambiente che lo ospita. Il centro decisionale si occuperebbe della rappresentazione dello stato ambientale mediante le informazioni ricevute dai dispositivi e produrrebbe come output la decisione che verrà poi eseguita dal dispositivo che ha interrogato il centro decisionale. Il numero di dispositivi di questo scenario e la diversità tra gli ambienti che ospitano i dispositivi proporrebbero più probabilmente e più frequentemente informazioni su stati diversi al centro decisionale che riuscirebbe ad apprendere velocemente e sperimentando in fretta tutti gli stati. Inoltre, ogni dispositivo sarebbe composto da minor codice perché delegherebbe al centro decisionale le computazioni più complesse concentrandosi ad eseguire le azioni importanti a raggiungere l'obiettivo del lavoro. Disponendo di un centro decisionale condiviso risulterebbe più facile modificare e rendere più complessa la catena decisionale senza dover intervenire sui dispositivi della rete il cui compito sarebbe quello di trasmettere informazioni ed eseguire i risultati calcolati dal centro stesso. Se l'approccio centralizzato dovesse essere un problema si potrebbe pensare ad un approccio decentralizzato dove l'obiettivo resterebbe sempre quello di separare il centro decisionale dalla logica operativa del dispositivo.

Per concludere, si può affermare che lo studio condotto ha prodotto i risultati di apprendimento attesi. Questo studio potrebbe essere uno spunto su come implementare

il sistema decisionale remoto che i dispositivi IoT potrebbero utilizzare per comportarsi in modo smart.

Appendice A

Codice

Listing A.1: Codice della classe SmartIoTDevice

```
class SmartIoTDevice:
    def __init__(self, env_representation_sys,
                 action_system, agent, reward_system):
        self.env_representation_sys = \
            env_representation_sys
        self.action_system = action_system
        self.agent = agent
        self.reward_system = reward_system
```

Listing A.2: Codice della classe SimpleDevice

```
def __init__(self, parameters, sleep_duration):
    self.action_system = SimpleActionSystem(sleep_duration)
    self.parameters = {}
    for parameter in parameters:
        self.parameters.update({parameter.name: parameter})

    actuators = {}
    temperature_actuator = SimpleTemperatureActuator(
        self.parameters.get("temperature").lower_limit_value,
        self.parameters.get("temperature").desired_value,
        self.parameters.get("temperature").upper_limit_value)
    actuators.update(
        {temperature_actuator.name: temperature_actuator}
    )

    humidity_actuator = SimpleHumidityActuator(
        self.parameters.get("humidity").lower_limit_value,
        self.parameters.get("humidity").desired_value,
        self.parameters.get("humidity").upper_limit_value)

    actuators.update(
        {temperature_actuator.name: temperature_actuator}
    )

    actuators.update(
        {humidity_actuator.name: humidity_actuator}
    )

    self.actuator_system = SimpleActuatorSystem(actuators)
    self.connected = None

    # Give actuator to action
    self.action_system.actions.get("t_intervention").actuator =\
        self.actuator_system.actuators.get("temperature_actuator")
    self.action_system.actions.get("h_intervention").actuator = \
        self.actuator_system.actuators.get("humidity_actuator")
```

Listing A.3: Funzione di inizializzazione dei parametri continui.

```
def initialize_continuous_parameters(continuous_parameters_info):
    continuous_parameters = []

    for continuous_parameter_information in continuous_parameters_info:
        # Information extraction
        name = continuous_parameter_information[0]
        lower_limit_value = continuous_parameter_information[1]
        desired_value = continuous_parameter_information[2]
        upper_limit_value = continuous_parameter_information[3]

        continuous_parameter = ContinuousEnvironmentParameter(
            name, lower_limit_value, desired_value, upper_limit_value)
        continuous_parameters.append(continuous_parameter)

    return continuous_parameters
```

Listing A.4: Funzione di inizializzazione dei parametri discreti.

```
def initialize_discrete_parameters(discrete_parameters_info):
    discrete_parameters = []
    for discrete_parameter_information in discrete_parameters_info:
        name = discrete_parameter_information[0]
        value_names = discrete_parameter_information[1].split("#")
        values = discrete_parameter_information[2].split('#')
        name_value_pairs = []
        i = 0
        while i < len(values):
            pair = [value_names[i], int(values[i])]
            name_value_pairs.append(pair)
            i += 1
        lower_limit_value = discrete_parameter_information[3]
        if lower_limit_value == "float(\"inf\")":
            lower_limit_value = float("inf")
        else:
            if "." in lower_limit_value:
                lower_limit_value = float(lower_limit_value)
            else:
                lower_limit_value = int(lower_limit_value)
        if "." in discrete_parameter_information[4]:
            desired_value = float(discrete_parameter_information[4])
        else:
            desired_value = int(discrete_parameter_information[4])
        upper_limit_value = discrete_parameter_information[5]
        if upper_limit_value == "float(\"inf\")":
            upper_limit_value = float("inf")
        else:
            if "." in lower_limit_value:
                upper_limit_value = float(upper_limit_value)
            else:
                upper_limit_value = int(upper_limit_value)
        discrete_parameter = DiscreteEnvironmentParameter(
            name, name_value_pairs, lower_limit_value, desired_value,
            upper_limit_value)
        discrete_parameters.append(discrete_parameter)
    return discrete_parameters
```

Listing A.5: Codice Arduino IDE per misurare temperatura ed umidità.

```
// Get current date time.
DateTime now = ds3231.now();

// Get temperature event and print its value.
sensors_event_t event;
dht.temperature().getEvent(&event);
if (isnan(event.temperature)) {
  Serial.print("NaN");
}
else {
  temperature = event.temperature;
}
// Get humidity event and print its value.
dht.humidity().getEvent(&event);
if (isnan(event.relative_humidity)) {
  Serial.print("NaN");
}
else {
  humidity = event.relative_humidity;
}
```

Listing A.6: Codice Arduino IDE per stampare su monitor seriale.

```
// Send data to serial monitor
  Serial.print(now.year());
  Serial.print("/");
  Serial.print(pad(now.month()));
  Serial.print("/");
  Serial.print(pad(now.day()));
  Serial.print(formattedDate);
  Serial.print(",");
  Serial.print(pad(now.hour()));
  Serial.print(":");
  Serial.print(pad(now.minute()));
  Serial.print(":");
  Serial.print(pad(now.second()));
  Serial.print(formattedDate);
  Serial.print(",");
  Serial.print(temperature);
  Serial.print(",");
  Serial.println(humidity);
```

Listing A.7: Funzione di inizializzazione dei valori dei parametri ambientali mediante valori di default.

```
def initialize_default_parameters_values(parameters_values_info):
    parameters_values = []
    for info in parameters_values_info:
        # Information extraction
        name = info[0]
        if "." in info[1]:
            default_value = float(info[1])
        else:
            default_value = int(info[1])
        parameter_values = ParameterValue(name, default_value)
        parameters_values.append(parameter_values)
    return parameters_values
```

Listing A.8: Codice della classe degli interruttori

```
class Switch:
    def __init__(self, name, possible_states, trigger_type):
        """ A switch gives information about the need of
            intervention.

        :param name: the name of the switch.
        :param possible_states: number of possible states.
        :param trigger_type: Determines how to set the state.
        """
        self.name = name
        self.state = None  # instructions on the need to act.
        self.possible_states = possible_states
        self.trigger_type = trigger_type
```

Listing A.9: Funzione di conversione da base mista a base decimale.

```
    def convert_mixed_radix_value_to_decimal_number(
        self, mixed_radix_number_representation):
        decimal_number = 0
        weight_selector = 0
        for mixed_radix_value in mixed_radix_number_representation:
            calculated_value = mixed_radix_value
            digit_position_weight = \
                self.digit_position_weights[weight_selector]
            weight_selector += 1
            for power in digit_position_weight:
                calculated_value *= power

            decimal_number += calculated_value
        return decimal_number
```

Listing A.10: Codice utilizzato per realizzare un agente di Q-learning.

```
class Agent:
    def __init__(self):
        self.g = 0  # The agent return

    def decide_action(self, s_t):
        pass


class QLearningAgent(Agent):
    def __init__(self, q_table, epsilon,
                 epsilon_decreasing_amount,
                 epsilon_minimum_value, alpha, gamma):
        """ Learns the work.

        :param q_table:
        :param epsilon: Usually 1.0.
        :param epsilon_decreasing_amount: Usually 0.2.
        :param epsilon_minimum_value: Usually 0.01.
        :param alpha: Usually 0.9.
        :param gamma: Usually 0.8
        """
        self.epsilon = epsilon  # random choice prob.
        self.epsilon_decreasing_amount = \
            epsilon_decreasing_amount
        self.epsilon_minimum_value = epsilon_minimum_value
        self.alpha = alpha  # The learning rate of the agent.
        self.gamma = gamma  # Future reward's discount factor.
        self.q_table = q_table  # state-action quality map.
        super(QLearningAgent, self).__init__()
```

```
def decide_action(self, s_t):
    # Random number to compare with the epsilon value.
    randomly_extracted_number = random.random()

    if randomly_extracted_number <= self.epsilon:
        decided_action_index = \
            self.decide_action_randomly()
    else:
        decided_action_index = \
            self.decide_action_using_q_table(s_t)

    # Decrease in the randomness of decisions.
    if self.epsilon >= self.epsilon_minimum_value:
        self.epsilon -= \
            self.epsilon * self.epsilon_decreasing_amount
    result = decided_action_index

    return result

def decide_action_using_q_table(self, s_t):
    """ Return best quality score from the s_t Q-table row.
    :s_t: actual environment state.
    """
    env_state_q_table_row = self.q_table[s_t, :]
    best_q_table_action_value = \
        np.max(env_state_q_table_row)
    best_q_table_action_value_indexes = \
        np.where(env_state_q_table_row
            == best_q_table_action_value)[0]
    best_q_table_action_randomly_chosen_index = \
        random.choice(best_q_table_action_value_indexes)

    return best_q_table_action_randomly_chosen_index

def decide_action_randomly(self):
    """ Choose and return a random action id number.
    """
    action_number = np.shape(self.q_table)[1] - 1
    randomly_decided_action_index = \
        random.randint(0, action_number)
    return randomly_decided_action_index
```

```
def bootstrap(self, r_t_t, s_t, a_t, s_t_t):
    """ Use the Q-function to update the Q-table.
    Assigns, using the Q-function, a quality score to
    a given action chosen by the agent in a given state.
    The calculated value is used to update the the Q-table
    for the given state-action pair.
    The formula used is the exact implementation of the
    Q-function shown at page 131 of the book Reinforcement
    Learning An Introduction Second Edition by Sutton and
    Barto.

    :param r_t_t: The reward for the decided action.
    :param s_t: The decision state.
    :param a_t: The decided action
    :param s_t_t: The reached state
    :return:
    """
    # The old state-action quality value.
    old_saq = self.q_table[s_t, a_t]

    # State-action pair with Higher value.
    max_saq = max(self.q_table[s_t_t, :])

    # The learning rate.
    # The higher the faster the agent learns.
    alpha = self.alpha

    # The discount factor.
    # The higher the more relevant are new rewards.
    gamma = self.gamma

    # Q function used to updated s-a q.
    updated_saq = \
      old_saq + (alpha * (r_t_t + (gamma * max_saq) - old_saq))

    # Q-table state-action quality update.
    self.q_table[s_t, a_t] = updated_saq

    return updated_saq
```

Listing A.11: Codice del modulo utilizzato per implementare e gestire la R-table.

```
import numpy


def build_r_table(possible_env_state_number,
                  number_of_action,
                  best_rewarded_state_action_information):
    """ Table used to assign rewards.

    :param best_rewarded_state_action_information: job info.
    :param possible_env_state_number: R-table's rows.
    :param number_of_action: R-table's columns.
    :return:
    """
    rows_number = possible_env_state_number + 3
    cols_number = number_of_action
    r_table = numpy.zeros((rows_number, cols_number))

    # Fill all non final row columns with -1
    row_index = 0
    while row_index < rows_number - 1:
        col_index = 0
        while col_index < cols_number:
            r_table[row_index, col_index] = -1
            col_index += 1
        row_index += 1

    # Set up the task goals
    for info in best_rewarded_state_action_information:
        row = info[0]
        col = info[1]
        reward = info[2]
        r_table[row, col] = reward
    return r_table[:]
```

Listing A.12: Classe parent di tutte le azioni

```
class Action:
    def __init__(self, name, env_representation_sys,
                 energy_cost, commitment_time):
        """ This is a generalization of the system's actions.

        :param name: action name.
        :param env_representation_sys: represents states.
        :param energy_cost: battery drain.
        :param commitment_time: device inactivity duration.
        """
        self.name = name
        self.cost = energy_cost
        self.commitment_time = commitment_time
        self.env_representation_sys = \
            env_representation_sys
```

Listing A.13: Classe specializzata dell'azione che consente al dispositivo di entrare in sleep mode.

```
class Sleep(Action):
    def __init__(self,
                 env_representation_sys,
                 energy_cost,
                 commitment_time):
        super(Sleep,
              self).__init__("sleep",
                             env_representation_sys,
                             energy_cost, commitment_time)

    def execute(self):
        if (
          self.env_representation_sys.env_state.s_t
          ==
          self.env_representation_sys.standard_state_index):
            return [True, self.commitment_time]
        else:
            return [False, 0]
```

Listing A.14: Codice del modulo episode.py.

```
class Episode:
    def __init__(self, number, day_second, total_t,
    t_sensed_parameter_values, trajectory,
    episode_commitment_time,
    separate_action_count_dictionary):
        self.number = number
        self.day_second = day_second
        self.total_t = total_t
        self.sensed_parameter_information = \
            t_sensed_parameter_values
        self.trajectory = trajectory
        self.commitment_time = episode_commitment_time
        self.separate_action_count_dictionary = \
            separate_action_count_dictionary


class Trajectory:
    def __init__(self):
        self.step_sequence = []


class TrajectoryStep:
    def __init__(self):
        self.connected = None
        self.s_t = None
        self.a_t = None
        self.a_t_name = None
        self.a_t_legit = None
        self.s_t_t = None
        self.r_t_t = None
        self.g_t_t = None
        self.bootstrapped_value = None
        self.q_table_clone_at_this_step = None
        self.q_table_string_representation_at_this_step = None
```

Listing A.15: Prototype Code.

```
// Constant Delays
const uint16_t DHT11_SENSING_DELAY = 1000;
const uint16_t ACTION_LIGHT_DELAY = 500;
const uint16_t STATE_LIGHT_ON_DELAY = 1000;
const uint16_t TIME_STEP_DELAY = 300;
const uint16_t TERMINAL_STATE_DELAY = 1000;
const uint16_t BEFORE_ACTION_DELAY = 500;
const uint16_t AFTER_ACTION_DELAY = 500;
const uint16_t EPISODE_START_DELAY = 100;
const uint16_t EPISODE_TERMINATION_DELAY = 100;


// DHT11 config
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <DHT_U.h>
const uint8_t DHT_TYPE = DHT11;
const uint8_t DHT11_PIN = 2;
DHT_Unified dht(DHT11_PIN, DHT_TYPE);

// Environment parameter config
const float LOWER_LIMIT_T = 18.00;
const float UPPER_LIMIT_T = 22.00;
const float DESIRED_T = 20.00;
float sensedT = -100;
int actuatorMode = -100;

// State transition temperature value test list:
float testT[9] = {21,16,19,20,20,23,21,20,20};
int testTIndex = 0;
```

```
// Q-table explanation and config
// Columns action order:
//   - 0: sense
//   - 1: heat (red on)
//   - 2: stop heat (red off)
//   - 3: cool (blue on)
//   - 4: stop cool (blue off)
//   - 5: sleep (green on)
//   - 6: report (yellow)
// Rows state order:
//   - 0: Start
//   - 1: Need heat
//   - 2: Stop heat
//   - 3: Need cool
//   - 4: Stop cool
//   - 5: Sleepable
//   - 6: Report
//   - 7: Terminal
int qTable[8][7] = {
  {1,0,0,0,0,0,0}, // Start -> sense
  {0,1,0,0,0,0,0}, // Need Heat -> heat
  {0,0,1,0,0,0,0}, // Stop Heat -> stopHeat()
  {0,0,0,1,0,0,0}, // Need Cool -> cool()
  {0,0,0,0,1,0,0}, // Stop Cool -> stopCool()
  {0,0,0,0,0,1,0}, // Sleepable -> sleep()
  {0,0,0,0,0,0,1}, // Report -> report()
  {0,0,0,0,0,0,0}  // Terminal
};
```

```
// This function prints the Q-table
void representQTable(){
  Serial.println("------------------");
  Serial.println("[Q-table]");
  for (int i = 0; i < 8; i++){
    for (int j = 0; j < 7; j++){
      if(j < 6){
        Serial.print(qTable[i][j]);
        Serial.print(",");
      } else {
        Serial.print(qTable[i][j]);
      }
    }
    Serial.println("");
  }
  Serial.println("------------------");
}

// R-table explanation and definition
// Good reward for report in report state
// Bad reward for sleepeng on work
int rTable[8][7] = {
  {-1,-1,-1,-1,-1,-100,-1},
  {-1,-1,-1,-1,-1,-100,-1},
  {-1,-1,-1,-1,-1,-100,-1},
  {-1,-1,-1,-1,-1,-100,-1},
  {-1,-1,-1,-1,-1,-100,-1},
  {-1,-1,-1,-1,-1,-1,-1},
  {-1,-1,-1,-1,-1,-100,100},
  {0,0,0,0,0,0,0}
};

// reward system
int getReward(int s, int a){
  return rTable[s][a];
}
```

```
// This funciton prints the R-table
void representRTable(){
  Serial.println("--------------------");
  Serial.println("[R-table]");
  for (int i = 0; i < 8; i++){
    for (int j = 0; j < 7; j++){
      if(j < 6){
        Serial.print(rTable[i][j]);
        Serial.print(",");
      } else {
        Serial.print(rTable[i][j]);
      }
    }
    Serial.println("");
  }
  Serial.println("--------------------");
}

// State representation variables and constatnts
int state;
int actualOnSPin = -1; // Status LED currently on
int TERMINAL = 7;

// State visual representation LED pins configuration
const int S_0_LED_PIN = 22; // start
const int S_1_LED_PIN = 23; // Need Heat
const int S_2_LED_PIN = 24; // Stop heat
const int S_3_LED_PIN = 25; // Need cool
const int S_4_LED_PIN = 26;  // Stop cool
const int S_5_LED_PIN = 27;  // Sleep
const int S_6_LED_PIN = 28;  // Report
const int S_7_LED_PIN = 29;  // Termonal

// Episode visual representation LED pins
const int E_START_LED_PIN = 30;
const int E_TERMINATION_LED_PIN = 31;
```

```
// Function used to determine the environment state
void determineState(int decidedA){
  if (decidedA == 0 && state == 0){
    if (sensedT == -100) {
      state = 5;
    }
    else if(sensedT <= LOWER_LIMIT_T){
      state = 1;
      actuatorMode = 1;
    }
    else if (sensedT >= DESIRED_T && actuatorMode == 1){
      state = 2;
      actuatorMode = 0;
    }
    else if (sensedT >= UPPER_LIMIT_T){
      state = 3;
      actuatorMode = -1;
    }
    else if (sensedT <= DESIRED_T && actuatorMode == -1){
      state = 4;
      actuatorMode = 0;
    }
    else {
      state = 5;
    }
  }
  else if (decidedA == 1 && state == 1) {
    state = 6;
  }
  else if (decidedA == 2 && state == 2) {
    state = 6;
  }
  else if (decidedA == 3 && state == 3) {
    state = 6;
  }
  else if (decidedA == 4 && state == 4) {
    state = 6;
  }
  else if (decidedA == 5) {
    state = 7;
  }
```

```
  else if (decidedA == 6 && state == 6) {
    state = 7;
  }
  else{
   // Wrong action decided and executed
   // The state remains the same
  }
}

// Function used to turn on current environment state LED
void turnOnS(int sPin){
  // Check if the led of a state is already on.
  if (actualOnSPin != -1){
    digitalWrite(actualOnSPin, LOW);
  }
  digitalWrite(sPin, HIGH);
  actualOnSPin = sPin;
}
```

```
// function that associates the current environmental
// state to the pin of the LED to be turned on
int determineSPin(int determinedS){

  // State led turn on barrier
  int sPin;
  switch (state) {
    case 0:
    sPin = S_0_LED_PIN;
    break;

    case 1:
    sPin = S_1_LED_PIN;
    break;

    case 2:
    sPin = S_2_LED_PIN;
    break;

    case 3:
    sPin = S_3_LED_PIN;
    break;

    case 4:
    sPin = S_4_LED_PIN;
    break;

    case 5:
    sPin = S_5_LED_PIN;
    break;

    case 6:
    sPin = S_6_LED_PIN;
    break;

    case 7:
    sPin = S_7_LED_PIN;
    break;
  }
  return sPin;
}
```

```
// Function that by means of other functions determines
// the LED associated with the current state which must
// be lit and which turns on the LED in question
void representsCurrentS(int determinedS){
  int sPin = determineSPin(determinedS);
    turnOnS(sPin);
}

// Action visual representation LED pin config
const int A_0_LED_PIN = 12; // sense()
const int A_1_LED_PIN = 11; // heat()
const int A_2_LED_PIN = 10; // stopHeat()
const int A_3_LED_PIN = 9;  // cool()
const int A_4_LED_PIN = 8;  // stopCool()
const int A_5_LED_PIN = 7;  // sleep()
const int A_6_LED_PIN = 6;  // report()

int actualAPin = -1; // LED of the action currently running
```

```
// Actions function definition

// Action 0
// Sense action that uses the DHT11 sensor to
// obtain the temperature value of the real
// monitored environment
float sense() {
  digitalWrite(A_0_LED_PIN, HIGH);
  sensedT;
  // Get sensedT event and print its value.
  sensors_event_t event;
  dht.temperature().getEvent(&event);
  if (isnan(event.temperature)) {
    //Serial.print("NaN");
    sensedT = -100.00;
  }
  else {
    sensedT = event.temperature;
  }
  digitalWrite(A_0_LED_PIN, LOW);
  return sensedT;
}
```

```
// Action 0
// Test sense action that extracts arbitrary
// temperature valuesfrom the test array
// used to test the correct functioning of
// the prototype in experiment 4.
//void sense() {
//  digitalWrite(A_0_LED_PIN, HIGH);
//
//  // Debug sense print
//  //Serial.println("[Test sense]");
//  //Serial.print("Initial sensedT: ");
////  Serial.print(sensedT);
////  Serial.print(", ");
//
//  sensedT = testT[testTIndex];
//
//
//  // Debug sense print
////  Serial.print("testTIndex: ");
////  Serial.print(testTIndex);
////  Serial.print(", ");
////  Serial.print("Resulting sensedT: ");
////  Serial.print(sensedT);
////  Serial.println(".");
//
//  if (testTIndex < 8){
//    testTIndex++;
//  }else{
//    testTIndex = 0;
//  }
//
//  delay(DHT11_SENSING_DELAY);
//  digitalWrite(A_0_LED_PIN, LOW);
//}
```

```
// Action 1
void heat(){
  digitalWrite(A_1_LED_PIN, HIGH);
  delay(ACTION_LIGHT_DELAY);
  digitalWrite(A_1_LED_PIN, LOW);
}

// Action 2
void stopHeat(){
  digitalWrite(A_2_LED_PIN, HIGH);
  delay(ACTION_LIGHT_DELAY);
  digitalWrite(A_2_LED_PIN, LOW);
}

// Action 3
void cool(){
  digitalWrite(A_3_LED_PIN, HIGH);
  delay(ACTION_LIGHT_DELAY);
  digitalWrite(A_3_LED_PIN, LOW);
}

// Action 4
void stopCool(){
  digitalWrite(A_4_LED_PIN, HIGH);
  delay(ACTION_LIGHT_DELAY);
  digitalWrite(A_4_LED_PIN, LOW);
}

// Action 5
void sleep(){
  digitalWrite(A_5_LED_PIN, HIGH);
  delay(ACTION_LIGHT_DELAY);
  digitalWrite(A_5_LED_PIN, LOW);
}

// Action 6
void report(){
  digitalWrite(A_6_LED_PIN, HIGH);
  delay(ACTION_LIGHT_DELAY);
  digitalWrite(A_6_LED_PIN, LOW);
}
```

```
// This function uses the id of the action decided
// by the Q-learning agent to perform the
// actual execution.
void execute(int decidedA){
  switch (decidedA) {

    case 0:
    sense();
    break;

    case 1:
    heat();
    break;

    case 2:
    stopHeat();
    break;

    case 3:
    cool();
    break;

    case 4:
    stopCool();
    break;

    case 5:
    sleep();
    break;

    case 6:
    report();
    break;
  }
}
```

```
// Function that makes the greedy choice by extracting
// the higher quality action from the Q-table.
int decide(int s){
  int maxQ = 0;
  int maxQIndex;
  for (int i = 0; i < 7; i++){
    int q = qTable[s][i];
    if (q > maxQ){
      maxQ = q;
      maxQIndex = i;
    }
  }
  return maxQIndex;
}


// Variables of episodes and trajectories
int episodeN = 1;
int trajectoryTimeStep = 1;

// Function used to visually represent the beginning
// of the episode.
void startEpisode(){
  int index = 0;
  while (index < 3){
    digitalWrite(E_START_LED_PIN, HIGH);
    delay(EPISODE_START_DELAY);
    digitalWrite(E_START_LED_PIN, LOW);
    delay(EPISODE_START_DELAY);
    index++;
  }
}
```

```
// Function used to visually represent the end
// of the episode. The function also keeps track
// of the episode progerssion and resets the step
// count of the trajectories of the experienced episodes.
void endEpisode(){
  int index = 0;
  while (index < 3){
    digitalWrite(E_TERMINATION_LED_PIN, HIGH);
    delay(EPISODE_TERMINATION_DELAY);
    digitalWrite(E_TERMINATION_LED_PIN, LOW);
    delay(EPISODE_TERMINATION_DELAY);
    index++;
  }
  // Reset the trajectory step number
  trajectoryTimeStep = 1;
  // Increase the episode number
  episodeN++;
}
```

```
// Produces the experiment data
// mode:
// - 0 for heading
// - 1 for intermediate step
// - 2 for terminal step
// The trajectory structure is the following:
//   - for the first time step:
//       state, action, sensedT, reward
//   - for intermediate time step:
//       state, action, reward
//   - for terminal time step:
//       state
void sendExperimentData(int decidedA, int r, int mode){
  switch (mode){
    case 1:
    // Test trajectory print s_t
    Serial.print(state);
    Serial.print(",");

    if (decidedA == 0) {
      Serial.print(decidedA);
      Serial.print(",");
      Serial.print(sensedT);
      Serial.print(",");
    } else {
      // Test trajectory print a_t
      Serial.print(decidedA);
      Serial.print(",");
    }

    // Test trajectory print a_t
    Serial.print(r);
    Serial.print(",");
    break;

    case 2:
    // Test trajectory print s_t
    Serial.println(state);
    break;
  }
}
```

```
// Function used to produce the printout of
// human-understandable information relating
// to the trajectories tested.
void logTrajectoryStep(int decidedA, int r, int mode){
  switch (mode){

    case 0:
    // Test trajectory print episode
    Serial.print("[Episode ");
    Serial.print(episodeN);
    Serial.println(" Trajectory]");
    break;

    case 1:
    // Test trajectory print s_t
    Serial.print("s_");
    Serial.print(trajectoryTimeStep);
    Serial.print("=");
    Serial.print(state);
    Serial.print(",");

    if (decidedA == 0) {
      // Test trajectory print a_t
      Serial.print("a_");
      Serial.print(trajectoryTimeStep);
      Serial.print("=");
      Serial.print(decidedA);
      Serial.print(",");
      Serial.print(sensedT);
      Serial.print(",");
    } else {
      // Test trajectory print a_t
      Serial.print("a_");
      Serial.print(trajectoryTimeStep);
      Serial.print("=");
      Serial.print(decidedA);
      Serial.print(",");
```

```
    }

    // Test trajectory print a_t
    Serial.print("r_");
    Serial.print(trajectoryTimeStep + 1);
    Serial.print("=");
    Serial.print(r);
    Serial.print(",");
    break;

    case 2:
    // Test trajectory print s_t
    Serial.print("s_");
    Serial.print(trajectoryTimeStep);
    Serial.print("=");
    Serial.println(state);
    break;

  }
}
```

```
// Prototype initialization function
void setup() {

  // Initialize DHT11.
  dht.begin();

  // Opens serial port
  Serial.begin(9600);

  // s pins config
  pinMode(S_0_LED_PIN, OUTPUT);
  pinMode(S_1_LED_PIN, OUTPUT);
  pinMode(S_2_LED_PIN, OUTPUT);
  pinMode(S_3_LED_PIN, OUTPUT);
  pinMode(S_4_LED_PIN, OUTPUT);
  pinMode(S_5_LED_PIN, OUTPUT);
  pinMode(S_6_LED_PIN, OUTPUT);
  pinMode(S_7_LED_PIN, OUTPUT);

  // e pin config
  pinMode(E_START_LED_PIN, OUTPUT);
  pinMode(E_TERMINATION_LED_PIN, OUTPUT);

  // a pins config
  pinMode(A_0_LED_PIN, OUTPUT);
  pinMode(A_1_LED_PIN, OUTPUT);
  pinMode(A_2_LED_PIN, OUTPUT);
  pinMode(A_3_LED_PIN, OUTPUT);
  pinMode(A_4_LED_PIN, OUTPUT);
  pinMode(A_5_LED_PIN, OUTPUT);
  pinMode(A_6_LED_PIN, OUTPUT);

}
```

```
// Prototype work execution function
void loop() {
   // Log Episode header
   //logTrajectoryStep(-100, -100, 0);

  // Episode start state
  state = 0;

  // Episode start
  startEpisode();

  while (state != TERMINAL){

    // Represent s_t
    representsCurrentS(state);



    // Action decision
    int decidedA = decide(state);

    // Delay before action execution
    delay(BEFORE_ACTION_DELAY);

    // Action execution
    execute(decidedA);

    // Get Reward
    int r = getReward(state, decidedA);

    // Log Intermediate step
    //logTrajectoryStep(decidedA, r, 1);

    // Send intermediate time step data
    sendExperimentData(decidedA, r, 1);

    // Delay after action execution
    delay(AFTER_ACTION_DELAY);

    // S_t+1 determination
    determineState(decidedA);
```

```
    // Increase time step
    trajectoryTimeStep++;
  }

  // Represent the terminal state
  representsCurrentS(state);

  // Log lest trajectory step
  //logTrajectoryStep(-100, -100, 2);

  // Send terminal time step data
    sendExperimentData(-100, -100, 2);

  // Turn off terminal state light
  delay(TERMINAL_STATE_DELAY);
  digitalWrite(S_7_LED_PIN, LOW);


  // Episode termination
  endEpisode();

  delay(TIME_STEP_DELAY);
}
```

Listing A.16: Modulo Python di analisi dei risultati degli esperimento condotti con prototipo.

```
import matplotlib.pyplot as plt
from numpy import arange

LOWER_LIMIT_T = 18.00
DESIRED_T = 20.00
UPPER_LIMIT_T = 22.00

temperature = []
states = []
actions = []
action_per_episode = []
steps = []
transmissions = []
rewards = []
g = 0
g_history = []

out_of_range_t_seconds = 0
in_range_t_seconds = 0

with open("data/file.txt") as file:
    rows = file.readlines()
    episodes = len(rows)
    senses = [0] * episodes
    heats = [0] * episodes
    stop_heats = [0] * episodes
    cools = [0] * episodes
    stop_cools = [0] * episodes
    sleeps = [0] * episodes
    reports = [0] * episodes

    # Extracting data
    episode_index = 0
    for row in rows:
        split_row = row.split(",")

        states_row = [0] * 8
        actions_row = [0] * 7
```

```
transmission_count = 0
executions_count = 0
steps_count = 0
index = 0
for info in split_row:
    if index == 0 or index == 4 or \
        index == 7 or index == 10:
        # States
        if info == "0":
            states_row[0] = 1
            steps_count += 1
        elif info == "1":
            states_row[1] = 1
            steps_count += 1
        elif info == "2":
            states_row[2] = 1
            steps_count += 1
        elif info == "3":
            states_row[3] = 1
            steps_count += 1
        elif info == "4":
            states_row[4] = 1
            steps_count += 1
        elif info == "5":
            states_row[5] = 1
            steps_count += 1
        elif info == "6":
            states_row[6] = 1
            steps_count += 1
        elif info == "7" or info == "7\n":
            states_row[7] = 1
            steps_count += 1
    # Actions
    elif index == 1 or index == 5 or index == 8:
        if info == "0":
            actions_row[0] = 1
            senses[episode_index] = 1
            executions_count += 1
        elif info == "1":
            actions_row[1] = 1
            transmission_count += 1
```

```
            heats[episode_index] = 1
            executions_count += 1
        elif info == "2":
            actions_row[2] = 1
            transmission_count += 1
            stop_heats[episode_index] = 1
            executions_count += 1
        elif info == "3":
            actions_row[3] = 1
            transmission_count += 1
            cools[episode_index] += 1
            executions_count += 1
        elif info == "4":
            actions_row[4] = 1
            transmission_count += 1
            stop_cools[episode_index] = 1
            executions_count += 1
        elif info == "5":
            actions_row[5] = 1
            sleeps[episode_index] = 1
            executions_count += 1
        elif info == "6":
            actions_row[6] = 1
            transmission_count += 1
            reports[episode_index] = 1
            executions_count += 1
    # Sensed temperature
    elif index == 2:
        extracted_t = float(info)
        temperature.append(extracted_t)
        if extracted_t >= UPPER_LIMIT_T or \
            extracted_t <= LOWER_LIMIT_T:
            out_of_range_t_seconds += 1
        else:
            in_range_t_seconds += 1
    # Returns
    elif index == 3 or index == 6 or index == 9:
        r = int(info)
        rewards.append(r)
        g += r
        g_history.append(g)
```

```
            index += 1
        episode_index += 1
        states.append(states_row)
        actions.append(actions_row)
        transmissions.append(transmission_count)
        steps.append(steps_count)
        action_per_episode.append(executions_count)

# Temperature plot
x_lim = episodes - 1
min_temperature = min(temperature)
max_temperature = max(temperature)
plt.plot(temperature)
plt.xlabel("Episode")
plt.ylabel("C")
plt.title("Prototype Observed Temperature")
plt.hlines(max_temperature, 0,
           x_lim, colors="red",
           linestyles="dashed",
           label="Max Reached T = " + str(max_temperature))
plt.hlines(UPPER_LIMIT_T, 0,
           x_lim, colors="orange",
           linestyles="dotted",
           label="Upper Limit T = " + str(UPPER_LIMIT_T))
plt.hlines(DESIRED_T, 0,
           x_lim, colors="green",
           linestyles="dotted",
           label="Optimal T = " + str(DESIRED_T))
plt.hlines(LOWER_LIMIT_T, 0,
           x_lim, colors="orange",
           linestyles="dotted",
           label="Lower Limit T = " + str(LOWER_LIMIT_T))
plt.hlines(min_temperature, 0,
           x_lim, colors="blue",
           linestyles="dashed",
           label="Min Reached T = " + str(min_temperature))
plt.legend(loc="best")
plt.savefig("output/temperature.png")
plt.close()
```

```
# Plot rewards
reward_scatter_x = arange(len(rewards))
plt.title("Reward sequence")
plt.xlabel("Cumulative time step")
plt.ylabel("Reward score")
plt.scatter(reward_scatter_x, rewards, label="Reward score")
plt.legend(loc="best")
plt.savefig("output/rewards.png")
plt.close()

# Plot g
reward_scatter_x = arange(len(g_history))
plt.title("Return Evolution")
plt.xlabel("Cumulative time step")
plt.ylabel("Return score")
plt.plot(g_history, label="Return")
plt.legend(loc="best")
plt.savefig("output/return.png")
plt.close()

# Out of range t bar chart
labels = 'In range temperatures', 'Out of range temperatures'
sizes = [in_range_t_seconds, out_of_range_t_seconds]
max_value = max(sizes)
explode_indexes = []
finder_index = 0
for size in sizes:
    if size == max_value:
        explode_indexes.append(finder_index)
    finder_index += 1
explode = [0.0] * len(sizes)
for explode_index in explode_indexes:
    explode[explode_index] = 0.2

fig1, ax1 = plt.subplots()
ax1.pie(sizes,
        explode=explode,
        labels=labels,
        autopct='%1.1f%%',
        shadow=True, startangle=90)
# Equal aspect ratio ensures that pie is drawn as a circle.
```

```
ax1.axis('equal')
plt.title("Prototype in and out of range temperature")
plt.savefig("output/in_out_range_percentage.png")
plt.close()

# Action count
labels = ['Prototype action executions']
width = 0.16  # the width of the bars
bar1 = arange(len(labels))  # the label locations
bar2 = [i + width for i in bar1]
bar3 = [i + width for i in bar2]
bar4 = [i + width for i in bar3]
bar5 = [i + width for i in bar4]
bar6 = [i + width for i in bar5]
bar7 = [i + width for i in bar6]

fig, ax = plt.subplots()
rects0 = ax.bar(bar1, senses.count(1),
                width, label='Sense count')
rects1 = ax.bar(bar2, heats.count(1),
                width, label='Heat count')
rects2 = ax.bar(bar3, stop_heats.count(1),
                width, label='Stop heat count')
rects3 = ax.bar(bar4, cools.count(1),
                width, label='Cool count')
rects4 = ax.bar(bar5, stop_cools.count(1),
                width, label='Stop cool count')
rects5 = ax.bar(bar6, sleeps.count(1),
                width, label='Sleep count')
rects6 = ax.bar(bar7, reports.count(1),
                width, label='Report count')

# Add some text for labels,
# title and custom x-axis tick labels, etc.
ax.set_ylabel('Number of action execution')
ax.set_title('Action count')
ax.set_xticks(bar4)
ax.set_xticklabels(labels)
ax.legend()
```

```
def autolabel(rects):
    """Attach a text label above each bar
       in *rects*, displaying its height."""
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(height),
                    xy=(rect.get_x() + rect.get_width() / 2,
                        height),
                    # xy=(1, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')


autolabel(rects0)
autolabel(rects1)
autolabel(rects2)
autolabel(rects3)
autolabel(rects4)
autolabel(rects5)
autolabel(rects6)

fig.tight_layout()

plt.savefig("output/action_count.png")
plt.close()

# Episode duration Histogram chart
bins = range(0, 5, 1)
plt.title("Histogram of the number of actions "
          "performed per episode")
plt.hist(action_per_episode, bins=bins,
         edgecolor='white',
```

```
            label='Number of action per episode')
plt.ylabel("Counted groups")
plt.xlabel("Number of action per episode")
plt.xticks(bins)
plt.legend(loc='best')
plt.savefig("output/action_per_episode.png")
plt.close()

# Episode duration
plt.title("Bar chart of the time step "
          "count of the experienced episodes")
x = arange(len(steps))
plt.bar(x, steps, label="Actions performed")
plt.ylabel("Time step count")
plt.xlabel("Episode number")
plt.savefig("output/time_step_count_bar_chart.png")
plt.close()
```

Bibliografia

[1] https://azure.microsoft.com/it-it/overview/internet-of-things-iot/iot-technology-protocols.

[2] https://gym.openai.com.

[3] https://www.bostondynamics.com/.

[4] https://www.statista.com.

[5] https://www.treccani.it/vocabolario.

[6] Mark A Adams. Reinforcement theory and behavior analysis. *Behavioral Development Bulletin*, 9(1):3, 2000.

[7] Cristian Aflori and Mitica Craus. Grid implementation of the apriori algorithm. *Advances in engineering software*, 38(5):295–300, 2007.

[8] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6. Ieee, 2017.

[9] Richard Bellman. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences of the United States of America*, 38(8):716, 1952.

[10] Richard Brathwait. *The Yong Mans Gleanings*. 1613.

[11] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 132–149, 2018.

[12] Issam El Naqa, Ruijiang Li, and Martin J Murphy. *Machine learning in radiation oncology: theory and applications*. Springer, 2015.

[13] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. Overview of supervised learning. In *The elements of statistical learning*, pages 9–41. Springer, 2009.

[14] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. Unsupervised learning. In *The elements of statistical learning*, pages 485–585. Springer, 2009.

[15] Anthony Hyman. *Charles Babbage: Pioneer of the computer*. Princeton University Press, 1985.

[16] Rob Kitchin and Gavin McArdle. What makes big data, big data? exploring the ontological characteristics of 26 datasets. *Big Data & Society*, 3(1):2053951716631130, 2016.

[17] Jian-hua Li. Cyber security meets artificial intelligence: a survey. *Frontiers of Information Technology & Electronic Engineering*, 19(12):1462–1474, 2018.

[18] SA McLeod. Bf skinner: Operant conditioning. *Retrieved September*, 9(2009):77, 2007.

[19] Richard Meyes, Hasan Tercan, Simon Roggendorf, Thomas Thiele, Christian Büscher, Markus Obdenbusch, Christian Brecher, Sabina Jeschke, and Tobias Meisen. Motion planning for industrial robots using reinforcement learning. *Procedia CIRP*, 63:107–112, 2017.

[20] P Ivan Pavlov. Conditioned reflexes: an investigation of the physiological activity of the cerebral cortex. *Annals of neurosciences*, 17(3):136, 2010.

[21] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[22] Hassan Soubra and Alain Abran. Functional size measurement for the internet of things (iot) an example using cosmic and the arduino open-source platform. In *Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement*, pages 122–128, 2017.

[23] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[24] Edward L Thorndike. Animal intelligence: an experimental study of the associative processes in animals. *The Psychological Review: Monograph Supplements*, 2(4):i, 1898.

[25] Alan M Turing. Computing machinery and intelligence. *Creative Computing*, 6(1):44–53, 1980.

[26] M Waltz and K Fu. A heuristic approach to reinforcement learning control systems. *IEEE Transactions on Automatic Control*, 10(4):390–398, 1965.

[27] Sun-Chong Wang. Artificial neural network. In *Interdisciplinary computing in java programming*, pages 81–100. Springer, 2003.

[28] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[29] Felix Wortmann and Kristina Flüchter. Internet of things. *Business & Information Systems Engineering*, 57(3):221–224, 2015.

Ringraziamenti

Ringrazio in primo luogo il mio relatore, il Chiarissimo Professor Marco Di Felice, per avermi guidato verso la scoperta del Reinforcement Learning e per aver reso possibile la realizzazione di questo elaborato.
Ringrazio inoltre il mio correlatore, il Dottor Angelo Trotta, per avermi seguito, consigliato e sostenuto durante tutte le fasi del lavoro di tesi condotto.
Ringrazio la mia famiglia per il sostegno che mi ha fornito durante il periodo di studio, per il supporto dato nei momenti di difficoltà e per tutto l'aiuto che ho ricevuto e che mi ha consentito di raggiungere gli obiettivi che mi ero fissato.
Ringrazio Laura per essere stata al mio fianco in ogni momento, per aver condiviso con me la passione e la curiosità verso la scienza e per aver rinvigorito in ogni istante il desiderio di raggiungere questo traguardo in modo da poter iniziare una carriera ricca di soddisfazioni.
Ringrazio anche i miei colleghi e amici, Cristian e Giacomo, per aver trasformato le sfide proposte dal percorso di studi in piacevoli e memorabili opportunità di crescita formativa e personale.
Un ringraziamento va anche ad Andrea che, nonostante la distanza fisica, è riuscito a starmi vicino in diversi momenti importanti sapendosi rivelare un ottimo amico.