POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Meccanica

Tesi di Laurea Magistrale

Politecnico di Torino

Ottimizzazione di un agente di *Q-learning* per il controllo di veicoli ibridi elettrici

Relatori:
Daniela Anna Misul
Claudio Maino

Candidato:
Matteo Acquarone

26 Luglio 2021

Abstract

Un'efficace strategia di gestione dell'energia (EMS) influisce in modo importante sulle prestazioni e sul controllo del consumo di carburante di veicoli ibridi elettrici. Mentre la maggior parte degli EMS utilizzati fino a questo momento seguono algoritmi basati su regole fisse (rule-based), recentemente sono stati sperimentati con successo algoritmi di ottimizzazione, tra i quali quelli di Reinforcement Learning. Nonostante gli ottimi risultati ottenuti con questi recenti metodi manca ancora uno studio specifico sulla scelta della funzione *reward* da utilizzare. L'obiettivo principale della tesi è quindi quello di trovare una funzione di *reward* per algoritmi di *Q-learning* che permetta di controllare efficacemente sia lo stato di carica della batteria (SOC) che il consumo di carburante (FC) di veicoli ibridi elettrici: un coefficiente di peso β è utilizzato per decidere su quale delle due grandezze focalizzarsi maggiormente. Inoltre, viene svolta un'analisi parametrica sul rapporto *exploration-exploitation* e sul *learning rate* in relazione al β utilizzato. Infine, viene effettuato un lavoro di miglioramento dell'agente utilizzato: mentre nella prima parte del lavoro l'agente impara solo dalle fasi di trazione del veicolo, nella seconda parte vengono sviluppate versioni dell'agente in grado di gestire sia la trazione che la frenata.

Indice

Indice delle figure

Indice delle tabelle

1 Introduzione

1.1 Motivazioni

Negli ultimi anni l'opinione pubblica sta sempre più rivolgendo la propria attenzione verso tematiche legate alla tutela dell'ambiente. Per quanto riguarda il vasto mondo dei trasporti, queste nuove idee hanno portato alla nascita del concetto di mobilità sostenibile. Per mobilità sostenibile si intende "che i sistemi di trasporto corrispondano ai bisogni economici, sociali e ambientali della società, minimizzandone contemporaneamente le ripercussioni negative" [1].

Attualmente oltre il 98% dei veicoli fa uso di motori a combustione interna. Tali sistemi di propulsione portano con sé diverse problematiche legate all'emissione di gas inquinanti nocivi per l'uomo o dannosi per l'ambiente, quali anidride carbonica (CO2), monossido di carbonio (CO) e ossidi di azoto (NOx) [2]. Siccome il settore dei trasporti consuma maggiori quantità di combustibili fossili rispetto a qualsiasi altro settore e produce, di conseguenza, grandi quantità di tali agenti inquinanti, risulta necessario imporre un cambio radicale di tendenza verso l'elettrificazione dei veicoli. Diversi governi nazionali hanno quindi iniziato a intervenire per lo sviluppo di veicoli elettrificati ed elettrici: l'utilizzo del motore elettrico (EM) per affiancare o sostituire il motore a combustione interna (ICE) permette infatti di ridurre in modo importante i consumi di carburante. Secondo quanto riportato nel Global EV Outlook 2020 [3], se venissero rispettate le condizioni dello "Stated Policies Scenario", nel 2030 la percentuale di veicoli elettrici raggiungerebbe circa il 7% del totale dei veicoli globali. Il numero di veicoli elettrici ed elettrificati salirebbe fino ad arrivare molto vicino ai 150 milioni: ciò permetterebbe di ridurre il numero di milioni di barili di petrolio utilizzati al giorno di 2,5mb/d.

1.2 Veicoli ibridi

1.2.1 Tipologie di veicoli elettrificati

Le principali tipologie di veicoli elettrificati sono i veicoli puramente elettrici (BEV) e i veicoli ibridi elettrici (HEV). Mentre i primi fanno uso della sola batteria come fonte di potenza per alimentare il motore elettrico e generare trazione, i secondi combinano l'utilizzo del motore elettrico alimentato da batteria e del motore a combustione interna. Di conseguenza, i veicoli BEV garantiscono l'annullamento delle emissioni inquinanti *Tank-to-Wheel* (TTW) grazie all'assenza del motore a combustione interna; allo stesso tempo però presentano differenti problematiche legate a peso, tempo di ricarica e durata del ciclo di vita della batteria. Al contrario, i veicoli ibridi HEV producono emissioni TTW a causa dell'utilizzo dell'ICE, ma non soffrono dei problemi riguardanti la batteria poiché quest'ultima viene utilizzata in misura minore rispetto ai BEV.

Entrambe le tipologie di veicolo descritte permettono di avere minori emissioni rispetto ai classici veicoli dotati solamente di motore a combustione interna: nei HEV, la presenza del motore elettrico permette di far lavorare l'ICE a potenze minori e in zone di lavoro a maggiore efficienza.

1.2.2 Tipologie di veicoli ibridi elettrici

A loro volta i veicoli HEV presentano diverse configurazioni possibili. Possono essere però individuate due tipologie principali di architetture utilizzate: *Serie* e *Parallelo*. Nei veicoli *Serie* il motore a combustione interna è utilizzato per fornire potenza al motore elettrico o per ricaricare la batteria: esso è infatti direttamente connesso al motore elettrico. È quindi proprio l'EM in ultimo luogo a fornire la potenza necessaria alla trazione del veicolo. Al contrario, nei veicoli *Parallelo* sia l'ICE che l'EM trasferiscono la potenza generata a un dispositivo intermedio che permette il collegamento tra i due motori e solo successivamente alle ruote. Vi sono infine architetture più complesse chiamate *Serie-Parallelo* o *Power-Split* che sono una via di mezzo tra le due categorie descritte.

Esiste un altro tipo di classificazione dei veicoli ibridi in base al grado di ibridizzazione: si distinguono veicoli *mild hybrid*, *full hybrid* (FHEV) e *plug-in* (PHEV). Nei veicoli *mild hybrid* il motore elettrico non può garantire da solo la trazione del veicolo, ma deve essere sempre affiancato dal motore a combustione interna. Nei FHEV, invece, la trazione può essere garantita sia dal solo motore a combustione interna che dal solo motore elettrico. Infine, i PHEV consentono di ricaricare la batteria attraverso una fonte esterna di energia elettrica.

1.2.3 Stato dell'arte delle strategie di gestione dell'energia di veicoli ibridi

Per quanto riguarda i veicoli ibridi diventa quindi di fondamentale importanza la scelta della distribuzione del flusso energetico tra i diversi componenti: in ogni istante di tempo di utilizzo del veicolo bisogna infatti determinare le frazioni di potenza fornite da motore a combustione interna e motore elettrico. Sono stati quindi sviluppati numerosi metodi di gestione dell'energia (EMS) con obiettivi molto vari tra cui, per esempio, la minimizzazione del consumo di carburante o il controllo dello stato di carica della batteria [4].

Le strategie di controllo di veicoli ibridi possono essere suddivise in due principali categorie: i metodi basati su regole fisse e i metodi di ottimizzazione. I primi determinano la distribuzione dell'energia tra motore elettrico e ICE nelle diverse condizioni di guida combinando diverse regole di controllo. Questi algoritmi hanno il grosso vantaggio di essere facilmente implementabili e applicabili online sul veicolo: per queste ragioni sono attualmente gli algoritmi più utilizzati nella pratica per il controllo di veicoli ibridi. D'altra parte, questi metodi presentano diversi svantaggi: per ottenere regole di controllo efficaci è necessaria esperienza tecnica e lo sviluppo di modelli elaborati per la rappresentazione dei diversi cicli di guida; di conseguenza questi metodi difettano di scarsa adattabilità a condizioni e cicli di guida sconosciuti.

A causa di questi evidenti svantaggi dei metodi basati su regole fisse, un grande sforzo nel mondo della ricerca è stato rivolto allo sviluppo di efficaci metodi di ottimizzazione. Al contrario dei metodi precedentemente descritti, i metodi di ottimizzazione cercano di ottimizzare il controllo degli HEV senza affidarsi a regole fisse. Tra di essi sono presenti metodi di ottimizzazione globale, come il *Dynamic Programming* (DP), e locale, come l'*Equivalent Consumption Minimization Strategy* (ECMS) e il *Model Predictive Control* (MPC). Utilizzando algoritmi come il DP si riescono ad ottenere risultati ottimali in termini di minimizzazione dei consumi di carburante e una buona adattabilità ai diversi cicli di guida. In compenso, questi metodi di ottimizzazione globale non sono adatti ad applicazioni in tempo reale sui veicoli perché richiedono la conoscenza pregressa del ciclo di guida e risultano molto pesanti dal punto di vista computazionale. D'altra parte, vi sono i metodi di ottimizzazione locale che possono teoricamente funzionare in tempo reale, poiché non richiedono la conoscenza a priori del ciclo di guida: l'ECMS appartiene a questa categoria. Esso si basa sulla minimizzazione istantanea di una funzione di costo dipendente soltanto dalle condizioni del sistema all'istante di tempo considerato. Per definire la funzione di costo è necessario determinare un fattore di equivalenza per comparare l'energia elettrica e l'energia del combustibile. Tuttavia, il valore ottimale del fattore di equivalenza può essere ottenuto solo con la conoscenza a priori del ciclo di guida. Per ovviare questo problema molti ricercatori hanno rivolto la loro attenzione verso una recente versione dell'ECMS: l'*Adaptive* ECMS. Tale algoritmo permette di aggiornare il parametro di controllo basandosi sulla richiesta di potenza all'istante di tempo considerato. Un altro esempio di metodo di ottimizzazione non globale è l'MPC. Quest'ultimo ottimizza le variabili di controllo basandosi su informazioni riferite a istanti di tempo futuri, ottenute grazie a modelli predittivi. Di conseguenza il suo grosso svantaggio è che le sue prestazioni dipendono pesantemente dall'accuratezza delle previsioni [5].

A causa degli evidenti limiti dei metodi sopra descritti, il mondo della ricerca ha recentemente iniziato a interessarsi ad algoritmi di *Reinforcement Learning* (RL). Gli algoritmi di RL presentano diversi vantaggi rispetto a quelli descritti precedentemente: essi combinano l'ottimizzazione globale della DP con l'implementazione in tempo reale tipica dei metodi basati su regole fisse [6]. Si riesce infatti ad ottenere un controllo adattativo: la scelta della distribuzione di potenza tra motore elettrico e ICE è in grado di adattarsi alle diverse condizioni di guida a cui è sottoposto il veicolo senza dover ricorrere a

previsioni di condizioni future come invece avviene per l'ECMS e il MPC. L'algoritmo di RL che è stato principalmente indagato fino a questo momento per applicazioni legate agli HEV è il *Q-learning* (QL) [7]. Le prime strategie di QL implementate si sono basate sull'utilizzo del *Q-learning* tabulare, in cui le variabili utilizzate sono discrete. Molto recentemente con lo sviluppo di nuovi algoritmi di RL, si sono provati ad applicare al problema anche metodi di *Deep Reinforcement Learning*. Questi ultimi combinano i concetti di RL con quelli del *Deep Learning* cioè con l'utilizzo di reti neurali. Come evoluzione del QL tabulare sono quindi stati utilizzati algoritmi quali *Deep Q-Networks* [8], *Double Deep Q-learning* [9] e *Deep Deterministic Policy Gradients* [10] e [11] grazie ai quali si è riuscito a garantire l'utilizzo di variabili continue [12] e [13], eliminando gli errori dovuti alla discretizzazione. I risultati finali ottenuti sono ottimi, comparabili a quelli ottenuti con l'ottimizzazione globale del DP.

1.3 Scopo di questo lavoro

In linea generale, una comune difficoltà nel trattare problemi complessi di RL è la definizione di un'efficace funzione di *reward*. Come verrà spiegato in modo più approfondito successivamente, il *reward* è una grandezza fondamentale per l'apprendimento dell'agente. L'obiettivo generale di qualsiasi algoritmo di RL è infatti quello di massimizzare la somma dei *reward* ottenuti lungo l'orizzonte temporale considerato. Di conseguenza il *reward* deve essere collegato all'obiettivo fisico che si vuole raggiungere: nel caso degli HEV il controllo dello stato di carica e la minimizzazione del consumo di carburante (FC). Il primo obiettivo del lavoro di tesi è quindi quello di trovare un'efficace funzione di *reward* adatta a soddisfare tali risultati.

L'altro obiettivo del lavoro riguarda il miglioramento dell'agente utilizzato per ottenere i controlli desiderati. Partendo da un agente già sviluppato su MATLAB, si è cercato di modificarlo in modo che potesse gestire non solo le fasi di trazione del veicolo, ma anche quelle di frenata. Sono state quindi sviluppate diverse versioni di agenti, evidenziando per ognuna i relativi vantaggi e svantaggi.

2 *Q-learning*

2.1 Concetti di base del Reinforcement Learning

L'algoritmo utilizzato in questo lavoro di tesi è definito *Q-learning* tabulare. Esso è uno tra gli algoritmi di *Reinforcement Learning* (RL) maggiormente utilizzati. Prima di entrare nel merito dello specifico algoritmo di *Q-learning*, sono introdotte in questa sezione nozioni di base del *Reinforcement Learning*.

2.1.1 Supervised, Unsupervised e Reinforcement Learning

Per *Reinforcement Learning* si intende una delle tre principali aree in cui si può suddividere il *Machine Learning* che a sua volta appartiene al vasto mondo dell'Intelligenza Artificiale. L'RL si distingue quindi dalle altre due aree del *Machine Learning*, definite *Supervised Learning* e *Unsupervised Learning*.

Nel *Supervised Learning* un esperto "supervisore" esterno fornisce all'algoritmo un set di dati iniziali con le relative soluzioni. Una volta che l'algoritmo ha correttamente terminato l'apprendimento, esso è in grado di trovare la soluzione per dati differenti da quelli utilizzati durante l'allenamento. Strumenti molto spesso utilizzati in quest'area del *Machine Learning* sono le reti neurali.

Per quanto riguarda l'*Unsupervised Learning,* viene fornito all'algoritmo un set di dati senza relative soluzioni. Lo scopo dell'algoritmo è in questo caso trovare la non nota organizzazione dei dati e raggrupparli in diverse categorie accumunate da determinate similitudini.

Infine, in algoritmi di RL, durante l'allenamento l'agente impara quali sono le migliori azioni da compiere all'interno dell'ambiente per raggiungere la massima somma possibile dei *reward*. Il *reward*, indicato genericamente con il simbolo *r*, definisce in generale l'obiettivo che si vuole raggiungere.

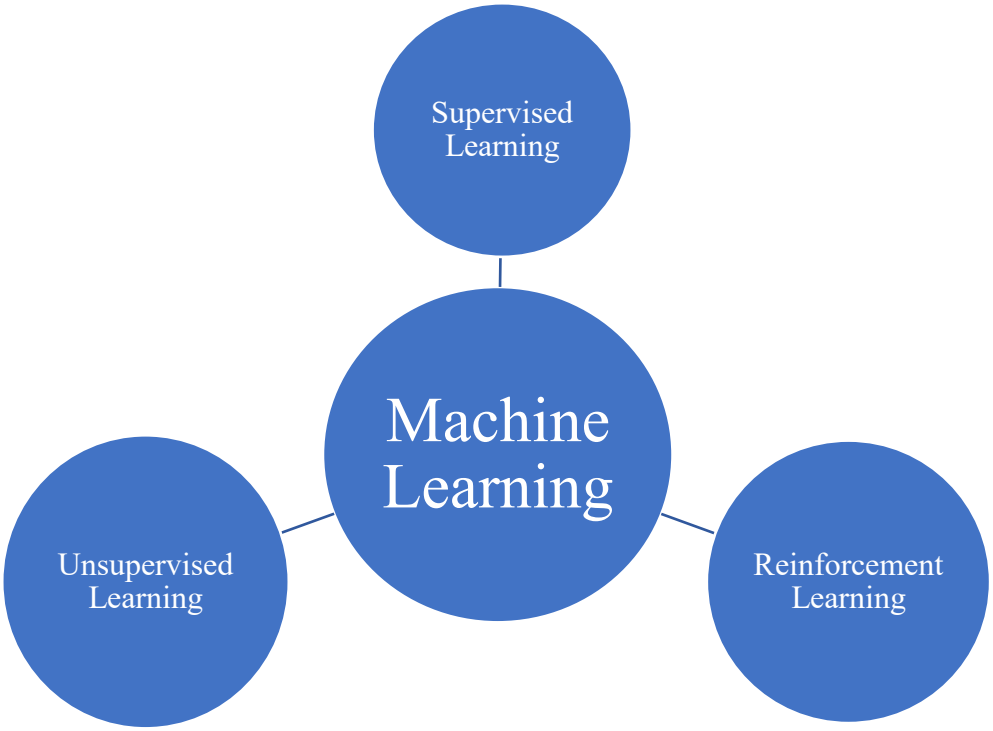

Figura 2.1 Suddivisione del Machine Learning

2.1.2 Elementi principali del Reinforcement Learning

Tutti gli algoritmi di RL hanno alcuni elementi in comune. Gli enti principali in un problema di RL sono sempre due: l'agente e l'ambiente. L'agente è l'entità "intelligente" che ha il compito di scegliere quale azione *a* compiere in ogni istante di tempo. L'ambiente invece è tutto ciò che si trova al di fuori dell'agente: nel caso del controllo di veicoli ibridi, ne fanno parte per esempio il veicolo, il guidatore, le condizioni della strada e le condizioni metereologiche.

L'obiettivo dell'agente è quello di scegliere in ogni istante temporale l'azione che permetta di ottenere la maggiore somma dei futuri *reward*. Per fare ciò l'agente ha chiaramente bisogno di informazioni provenienti dall'ambiente: a ogni istante di tempo esso dovrebbe ricevere quindi la rappresentazione dello stato *s* dell'ambiente, cioè l'insieme di tutte quelle grandezze necessarie alla scelta dell'azione. Tuttavia, ciò che l'agente riceve realmente sono le osservazioni e non gli stati. L'osservazione differisce dallo stato in quanto potrebbe non contenere tutte le informazioni necessarie all'agente per prendere l'azione corretta. Idealmente l'osservazione dovrebbe quindi coincidere con lo stato ma molto frequentemente ciò non avviene a causa di informazioni scorrette o di un numero minore di informazioni presenti nell'osservazione. Il sottile problema della differenza tra stato e azione verrà tralasciato nel resto di questa breve trattazione teorica del RL. Ad ogni modo prima di iniziare l'allenamento, l'agente non sa quali siano le azioni corrette da prendere poiché non possiede alcuna conoscenza di come risponderà l'ambiente. L'allenamento è quindi quel processo attraverso cui l'agente arriva a determinare quale azione prendere ogni volta che giunge a un determinato stato. Se l'algoritmo è impostato correttamente rispetto al problema da risolvere e l'allenamento risulta efficace si dovrebbe ottenere come risultato finale una politica ottimale. Per politica π si intende la mappatura che collega gli stati dell'ambiente alle azioni da prendere quando l'ambiente si trova in quegli stati [14].

Altro elemento sempre presente nel RL è il segnale di *reward*. Esso è una grandezza importante per l'apprendimento dell'agente. Si ricorda infatti che l'obiettivo dell'agente è quello di massimizzare la somma dei *reward* ottenuti nel tempo: si parla quindi di *value function* di uno stato *v(s)* come la somma totale dei *reward* che un agente può aspettarsi di accumulare nel futuro, partendo da quello stato. La scelta della funzione *reward* diventa quindi fondamentale in questo genere di problemi: essa deve essere collegata all'obiettivo fisico che si vuole raggiungere. È utile anche definire in questo caso il *return* R_t al tempo *t* che è definito come somma dei *reward* futuri ottenuti dopo l'istante di tempo *t*. Da notare la differenza sottile rispetto alla *value function* che è invece la somma dei *reward* attesi e non di quelli realmente ottenuti e che quindi risulta essere soltanto una stima del *return*.

2.1.3 Processo Markoviano e schema di base di funzionamento RL

Per applicare correttamente la maggior parte degli algoritmi di RL è necessario considerare come ambiente un sistema Markoviano. Si consideri l'istante di tempo presente *t*; per un sistema markoviano, lo stato del sistema al tempo presente *t* contiene tutte le informazioni riguardanti gli istanti di tempo passati, cioè possiede la proprietà di Markov. Inoltre, se lo stato del sistema al tempo *t* possiede la proprietà di Markov, lo stato che si avrà all'stante successivo a *t* dipenderà solo dallo stato al tempo *t* e dall'azione presa. Per questa ragione lavorare con un sistema Markoviano risulta essere una condizione ideale per i problemi di RL: la scelta della corretta azione al tempo *t* dipenderà soltanto dallo stato del sistema all'istante di tempo *t*. Anche quando lo stato del sistema non possiede la proprietà di Markov è ancora appropriato pensarlo come approssimazione di uno stato Markoviano. Chiaramente più lo stato del sistema si avvicina a possedere la proprietà di Markov, più si otterranno risultati migliori con algoritmi di RL [14].

Introdotti tutti questi elementi si può passare a spiegare lo schema di base di funzionamento di un algoritmo RL. Ci si pone nella condizione in cui l'agente deve ancora completare l'apprendimento delle azioni corrette. Partendo inizialmente da un istante generico di tempo *t* dove l'ambiente presenta uno stato *s,* l'agente sceglie quindi l'azione *a* per quel determinato stato *s*. Il metodo con cui l'agente sceglie tale azione verrà ampiamente discusso nei capitoli successivi. A questo punto, l'ambiente è influenzato da *a* e all'istante successivo a *t* arriva allo stato successivo *s'*. Viene quindi calcolato il *reward r(s,a)* riferito a quello stato *s* e quell'azione *a* utilizzando la *funzione di reward*. L'agente riceve tale valore di *reward* e lo utilizza per imparare la politica migliore secondo il principio di ottimalità di Bellman.

Figura 2.2 Schema generico di algoritmi di RL

2.1.4 Problemi episodici e continuativi

Per problemi episodici si intendono quei problemi dove l'interazione tra agente e ambiente si può suddividere in sequenze ben definite e separate tra di loro, chiamate episodi. La fine di ogni episodio, che avviene all'istante di tempo finale indicato con T, può essere data dal raggiungimento di particolari condizioni di termine del problema per cui risulta impossibile proseguire. Esempi di problemi episodici sono i giochi da tavolo come gli scacchi in cui una volta raggiunto l'obiettivo, il gioco termina. Se si considerano problemi episodici, il *return* al tempo t può quindi essere semplicemente ottenuto come somma dei *reward* ottenuti dopo l'istante di tempo t. Al contrario, se consideriamo problemi continuativi, cioè che si estendono quindi per un periodo di tempo potenzialmente infinito, non è opportuno considerare il *return* come semplice somma dei *reward*: il *return* risulterebbe infatti infinito e non potrebbe essere massimizzato. Per questa ragione viene utilizzato il *discounted return*, definito come:

$$R_t = \sum_{j=0}^{\infty} \gamma^j r_{t+j+1}$$

Dove:

- γ è il *discount factor*
- r_{t+j+1} è il *reward* ottenuto al tempo *t* +j+1

Il *discount factor* γ è un fattore compreso sempre tra $0 \leq \gamma \leq 1$. Tale parametro influisce sul valore al tempo *t* dei *reward* futuri. Essendo minore di uno, l'influenza di un *reward* sul *return* è tanto minore quanto più il *reward* è lontano nel tempo rispetto all'istante temporale *t*. In questo modo il *return* al tempo *t* di un problema continuativo è un numero finito ed è possibile massimizzarlo. La scelta del parametro γ nei problemi di RL risulta molto importante in quanto, tanto più esso si avvicina a uno, tanto più i *reward* futuri assumono maggiore importanza per l'agente; al contrario, tanto più γ si avvicina a zero tanto più l'agente considera meno importanti i *reward* ottenuti in istanti di tempo lontani da *t*.

Per sistemi markoviani a questo punto la funzione *value* v_π dello stato *s*, riferita alla politica π si può definire come il *discounted return* atteso partendo dallo stato *s* e seguendo la politica π da lì in poi:

$$v_\pi(s) = E_\pi\left[\sum_{j=0}^{\infty} \gamma^j r_{t+j+1} \,|s\right]$$

Con il simbolo E_π si fa riferimento al fatto che i *reward* sono quelli attesi seguendo la politica π e non quelli realmente ottenuti. Siccome l'algoritmo utilizzato in questo lavoro di tesi è il *Q-learning*, è necessario introdurre un'altra grandezza simile alla *value function*: *l'action-value function* q_π. Si definisce *action-value* dello stato *s* e azione *a* riferita alla politica π come il *discounted return* atteso partendo dallo stato *s*, scegliendo l'azione *a* e dall'istante di tempo successivo seguendo la politica π:

$$q_\pi(s,a) = E_\pi\left[\sum_{j=0}^{\infty} \gamma^j r_{t+j+1} \,|s,a\right]$$

A partire dall'espressione di $q_\pi(s,a)$ si può ricavare l'equazione di Bellman, fondamentale per molti algoritmi di RL:

$$q_\pi(s,a) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]$$

Dove:

- $\pi(s|a)$ è la probabilità di scegliere l'azione una volta arrivati allo stato *s* e seguendo la politica π
- $p(s',r|s,a)$ è la probabilità di arrivare allo stato *s'* con *reward r* partendo dallo stato *s* e scegliendo l'azione *a*

2.1.5 Principio di ottimalità di Bellman

Per riprendere quanto detto precedentemente con parole diverse, l'obiettivo degli algoritmi di RL è quello di ottenere la politica ottimale $\hat{\pi}$, cioè quella politica che massimizza il *discounted return* a ogni istante di tempo. Il principio di ottimalità di Bellman afferma quindi che "una politica ottimale ha la proprietà che qualsiasi siano lo stato e azione iniziale, le azioni successive devono costituire una politica ottimale rispetto allo stato risultante dalla prima azione" [15]. Una volta compreso il significato di politica ottimale, si può affermare che una politica π si può definire migliore di un'altra π' se il suo *discounted return* atteso è maggiore di quello di π'. Si possono quindi definire la *value function* ottimale e l'*action-value function* ottimale come:

$$v_{\hat{\pi}}(s) = \max_\pi \left(v_\pi(s)\right) \qquad q_{\hat{\pi}}(s,a) = \max_\pi \left(q_\pi(s,a)\right)$$

Da qua si possono ricavare le equazioni di ottimalità di Bellman:

$$v_{\hat{\pi}}(s) = \max_a \left(\sum_{s',r} p(s', r|s, a)[r + \gamma v_{\hat{\pi}}(s')] \right)$$

$$q_{\hat{\pi}}(s, a) = \sum_{s',r} p(s', r|s, a)[r + \gamma \max_{a'}(q_{\hat{\pi}}(s', a'))]$$

Queste formulazioni saranno di fondamentale importanza per l'algoritmo di RL che verrà usato in questo lavoro di tesi, il *Q-learning*. Per raggiungere la politica ottimale la politica scelta dall'agente deve essere migliorata durante l'allenamento.

2.2 Algoritmo di *Q-learning*

L'algoritmo *Q-learning* è uno tra i più utilizzati algoritmi di RL. Si tratta di un metodo di *Temporal-Difference Learning* per allenare l'agente, basato sull'iterazione del valore della *action-value function* per raggiungere la politica ottimale. Questo algoritmo garantisce la convergenza a tale politica ottimale. Una volta terminato l'allenamento, l'agente agisce in pura *exploitation*: per un generico stato *s* in cui si trova l'ambiente, prende l'azione *a* che considera migliore, cioè l'azione per cui il *Q(s,a)* riferito allo stato *s* è maggiore.

2.2.1 Iterazione del valore della *value function*

Al contrario di altri metodi di RL nel *Q-learning* si utilizza una variante dell'iterazione della *value function* per aggiornare e migliorare la politica durante l'allenamento. Questa iterazione combina al suo interno in un unico passaggio i due passaggi della iterazione della politica: la valutazione e il miglioramento della politica. Nell'iterazione del *value*, ogni volta che si incontra uno stato *s* si aggiorna il relativo *value* dello stato *s* in questo modo:

$$v_{j+1}(s) = \max_a \left(\sum_{s',r} p(s', r|s, a)[r + \gamma v_j(s')] \right)$$

Per $v_j(s)$ si intende il valore che la *v(s)* possedeva alla precedente iterazione, mentre con $v_{j+1}(s)$ si intende il valore della *v(s)* dopo l'aggiornamento. *j* indica quindi il numero di volte in cui la *v(s)* è stata iterata. Con l'iterazione del *value* è garantita la convergenza alla politica ottimale.

2.2.2 Metodo Temporal-Difference (TD)

Il *Q-learning* fa parte dei così detti metodi *Temporal-Difference*. In tali metodi l'agente impara direttamente dall'esperienza delle azioni all'interno dell'ambiente senza quindi la conoscenza della dinamica dello stesso, cioè della probabilità $p(s', r|s, a)$ di arrivare a uno specifico stato *s'* partendo dallo stato *s* e scegliendo l'azione *a*. I grossi vantaggi dei metodi TD rispetto ad altri metodi come *Monte-Carlo* o *Dynamic Programming* sono che per aggiornare la politica basta aspettare un singolo passo temporale e che non è necessaria la conoscenza della dinamica del modello. Ad ogni modo nei metodi TD non è possibile utilizzare direttamente l'iterazione del *value* sopra spiegata poiché non si è a conoscenza della dinamica del sistema e quindi del termine $p(s', r|s, a)$. Per questa ragione è necessario effettuare un aggiornamento leggermente differente della *value function* per lo stato *s*. Inoltre, nella maggior parte dei metodi TD si cerca di stimare l'*action-value function* ottimale invece della normale *value function*: ciò avviene anche nel *Q-learning*.

2.2.3 Aggiornamento della *Q(s,a)* nel *Q-learning*

Essendo un metodo TD, durante l'allenamento ad ogni passo temporale viene aggiornato il valore stimato dell'*action-value*. Finora sono state considerate le variabili *v* e *q* che si riferiscono ai reali *value* e *action-value function*. Tuttavia, tali valori non sono noti, di conseguenza dovremmo utilizzare una loro stima. Viene indicata quindi con *Q* la stima dell'action-*value Q*. L'obiettivo del *Q-learning* è quindi

quello di determinare una stima *Q(s,a)* che sia abbastanza vicina al valore ottimale $q_{\hat{\pi}}(s,a)$. Ad ogni passo temporale dell'allenamento con l'ambiente nello stato *s* e presa un'azione *a*, il valore di *Q(s,a)* viene aggiornato mediante una formula ricavata dall'equazione di Bellman precedentemente illustrata:

$$Q(s,a) = Q(s,a) + \alpha\left[r + \gamma \max_{a'}\big(Q(s',a')\big) - Q(s,a)\right]$$

Dove:

- γ è il *discount factor*
- α è il *learning rate*
- Q è la stima del valore di $q_{\hat{\pi}}(s,a)$

Cambiare parametri come *γ, α* o il rapporto tra *exploration* ed *exploitation* fa cambiare significativamente i risultati finali dell'apprendimento. *γ*, come già spiegato precedentemente, permette di regolare l'importanza attribuita ai *reward* futuri da parte dell'agente. Il *learning rate* è invece un parametro, compreso tra 0 e 1, che indica quanto velocemente impara l'agente. Più il *learning rate* è elevato, più infatti la *Q(s,a)* cambia se la differenza $r + \gamma \max_{a'}\big(Q(s',a')\big) - Q(s,a)$, detta errore TD, è diversa da zero. Di conseguenza, l'utilizzo di un *α* alto permette di arrivare vicino al valore di convergenza della *Q* più velocemente. D'altro canto, un *α* elevato può non essere sempre la scelta più efficace come verrà mostrato in seguito.

2.2.4 *Q-learning* tabulare e discretizzazione di stati e azioni

L'algoritmo che è stato utilizzato in questo lavoro di tesi è il *Q-learning* tabulare. In questo caso i valori di ogni *Q(s,a)* sono contenuti all'interno di una matrice chiamata *Q-table*. Ogni riga della matrice corrisponde a un particolare stato, mentre ogni colonna a una particolare azione; ogni cella della matrice contiene quindi il valore di *Q(s,a)*. Ad ogni istante temporale *t* avviene l'aggiornamento del valore della corrispondente *Q* dello stato *s* e azione *a* presenti all'istante *t*, con l'utilizzo della formula descritta nella sottosezione precedente. Prima di iniziare l'allenamento dell'agente, quest'ultimo non conosce ancora la politica ottimale. Il valore iniziale di ogni *Q(s,a)* presente all'interno della *Q-table* è teoricamente arbitrario. Chiaramente la scelta di tale valore iniziale può influire sull'allenamento e indirettamente sul rapporto tra *exploration* ed *exploitation* che verrà discusso nella sottosezione successiva. Per utilizzare questo algoritmo relativamente semplice di *Q-learning* si è costretti a utilizzare variabili discrete sia per quanto riguarda gli stati che per quanto riguarda le azioni. Lo stato può generalmente comprendere più grandezze fisiche, di numero generico *N*; ognuna di esse deve quindi essere discretizzata secondo una propria griglia di discretizzazione. Uno stato specifico corrisponde quindi agli *N* corrispondenti valori discretizzati delle grandezze fisiche. Anche le azioni devono essere discretizzate allo stesso modo degli stati e possono comprendere più variabili di controllo. La discretizzazione di queste variabili porta comunque ad errori che non sempre si possono ritenere trascurabili: la scelta del passo di discretizzazione è molto importante per conservare il valore fisico delle prove.

2.2.5 Allenamento dell'agente e rapporto tra *exploration-exploitation*

Vi sono due modalità di scelta dell'azione da parte dell'agente per ogni istante di tempo: *exploration* ed *exploitation*. In *exploration*, l'agente sceglie casualmente una delle azioni discretizzate. In *exploitation*, arrivati allo stato *s* l'agente sceglie l'azione che considera essere la migliore, cioè quella per cui vi è il massimo valore di *Q(s,a)*. Come detto precedentemente, prima di iniziare l'allenamento, l'agente non sa quali siano le azioni migliori da prendere e la *Q-table* è inizializzata in modo che tutti i valori di *Q(s,a)* siano uguali tra di loro. Durante l'allenamento vi è la necessità di bilanciare accuratamente il rapporto tra *exploration* ed *exploitation*. Infatti, agire sempre in pura *exploitation* durante tutto l'allenamento porterebbe a pessimi risultati in quanto l'agente tenderebbe a prendere sempre le stesse azioni: una volta arrivato al generico stato *s,* l'azione *a* cui corrisponde il massimo valore di *Q(s,a)* sarebbe un'azione che è già stata presa precedentemente per quello stato *s* (ammettendo che i *reward* siano positivi). Di conseguenza non verrebbero neanche considerate azioni differenti, che avrebbero

potuto portare a risultati migliori. Considerando invece l'altra condizione estrema di pura *exploration*, l'agente ogni volta che incontra uno stato prende sempre un'azione casuale. In questo modo è molto improbabile che vi sia qualche episodio di allenamento in cui l'agente ottenga buoni risultati. Riprendendo i risultati dello studio effettuato nell'articolo "Parametric study on reinforcement learning optimized energy management strategy for a hybrid electric vehicle" [16], l'effetto generale dell'aumento dell'*exploitation* è l'aumento del numero di esperienze positive, cioè il numero di episodi in cui si ottiene un *discounted return* finale maggiore rispetto all'episodio precedente. Tuttavia, è anche molto importante la qualità oltre che la quantità delle esperienze positive: si possono avere importanti variazioni della qualità di tali esperienze solo se l'esplorazione è sufficientemente alta. In linea generale risulta quindi necessario bilanciare con attenzione il rapporto tra *exploration* ed *exploitation*.

Uno tra i metodi più semplici e più utilizzati per il bilanciamento tra esplorazione ed *exploitation* è il metodo *ε-greedy*. Questo metodo consiste nel fissare un valore della costante ε minore di uno. In ogni istante di tempo l'agente prende un'azione casuale, quindi esplora, con una probabilità pari a ε, mentre prende un'azione di *exploitation* con probabilità pari a 1- ε. Il parametro ε è quindi direttamente collegato all'esplorazione: più è elevato, maggiore è l'esplorazione e viceversa. Lasciare il parametro ε costante durante tutto l'allenamento è una scelta molto comoda e spesso efficace, ma non sempre garantisce i risultati migliori. Può risultare molto utile, infatti, far decrescere il rapporto tra esplorazione ed *exploitation* con l'aumentare del numero dell'episodio di allenamento, cioè utilizzare un ε decrescente durante l'allenamento. Intuitivamente si comprende infatti che all'inizio dell'allenamento, quando l'agente non ha ancora interagito molto con l'ambiente, sia molto utile mantenere alto il livello di esplorazione in modo che l'agente possa provare e valutare azioni nuove; al contrario, quando ci si avvicina alla fine dell'allenamento, è più opportuno invece utilizzare prevalentemente azioni già esplorate in modo da generare episodi con elevato *discounted return*, seppur mantenendo almeno un minimo di esplorazione per garantire la possibilità di miglioramenti della politica. A puro titolo di esempio è riportato l'andamento del rapporto tra esplorazione ed *exploitation* ottenuto in una prova utilizzando un ε decrescente linearmente. Il rapporto tra esplorazione ed *exploitation* (Explore to exploit ratio nel grafico) per ogni episodio è in questo caso calcolato come il rapporto tra il numero di volte in cui l'azione è scelta in modo casuale e il numero di volte in cui come azione è presa quella che massimizza il *Q(s,a)*.

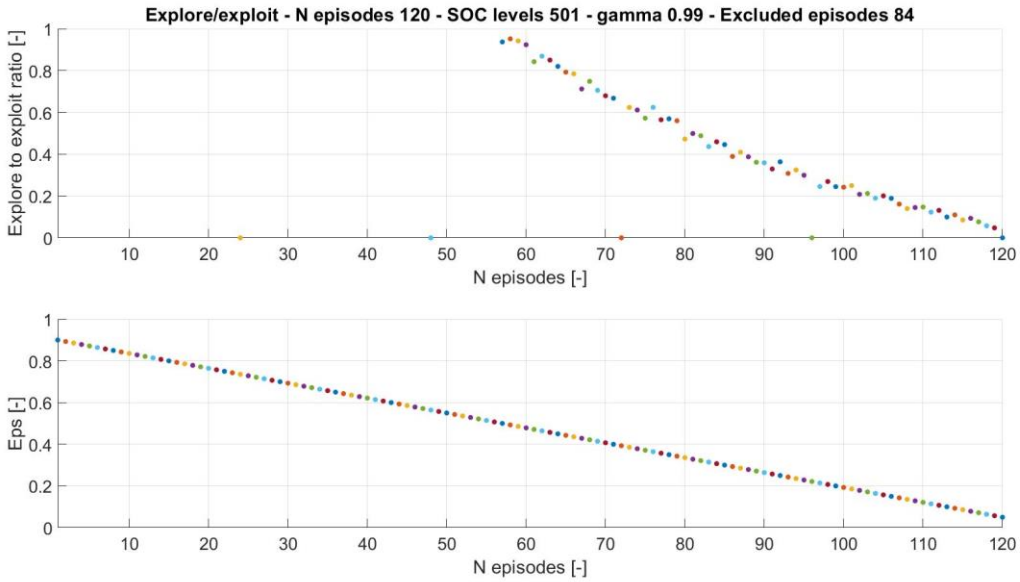

Figura 2.3 Exploration-exploitation con ε decrescente linearmente

2.3 Applicazione del *Q-learning* per il controllo di veicoli ibridi

Ora che è stato introdotto l'algoritmo utilizzato in questo lavoro di tesi, è opportuno spiegarne l'applicazione al particolare problema del controllo di veicoli ibridi. In primo luogo, come in ogni problema di RL, è necessario definire con precisione quale sia l'obiettivo fisico che si vuole raggiungere. In questa applicazione gli obiettivi possono essere diversi, quali la minimizzazione del consumo di carburante, il controllo dello stato di carica della batteria e il controllo delle emissioni inquinanti. In questo lavoro di tesi ci si concentrerà soltanto sui primi due obiettivi citati. Una volta definiti con precisione gli obiettivi fisici, si può iniziare a ragionare sulla scelta del *reward*, stati e azioni. Partendo dal *reward*, quest'ultimo dovrà essere collegato direttamente agli obiettivi fisici decisi precedentemente: le funzioni di *reward* possibili sono diverse e si discuterà della scelta tra una di esse successivamente. La scelta degli stati e delle azioni risulta essere di fondamentale importanza. Si ricorda che lo stato dell'ambiente deve contenere le informazioni necessarie all'agente per prendere l'azione corretta. Essendo un problema di gestione dell'energia tra motore elettrico e ICE, è sempre scelta come azione la distribuzione della potenza (PF) tra i due motori; può anche essere presente tra le azioni il rapporto di trasmissione del cambio, cioè la marcia, e lo stato di accensione dell'ICE. In questo lavoro di tesi verranno considerate tutte le grandezze come azioni. Anche lo stato dell'ambiente può comprendere diverse grandezze fisiche. Assolutamente necessario per la scelta del PF risulta essere lo stato di carica della batteria (SOC): il SOC non può infatti andare oltre determinati limiti imposti a priori per garantire sia il corretto funzionamento della batteria che il completamento del ciclo di guida del veicolo. Bisogna però ricordare che in un problema di RL sarebbe opportuno che lo stato si avvicinasse ad essere uno stato Markoviano. Per questa ragione sarebbe molto utile aggiungere altre grandezze allo stato dell'ambiente, quali per esempio la velocità del veicolo e la potenza richiesta allo stesso. Tuttavia, l'aggiunta di variabili agli stati o azioni, così come l'utilizzo di griglie di discretizzazioni più fitte per tali variabili, può comportare problemi computazionali: il numero delle coppie stato-azione e il conseguente numero di *Q(s,a)* cresce molto velocemente; la matrice *Q-table* assume quindi dimensioni maggiori e sono necessari sempre più episodi per allenare efficacemente l'agente. Per il *Q-learning* tabulare, questi problemi computazionali vengono spesso indicati con il nome *"curse of dimensionality"*. Per questa ragione nella maggior parte del lavoro di tesi viene utilizzato un unico stato, il SOC.

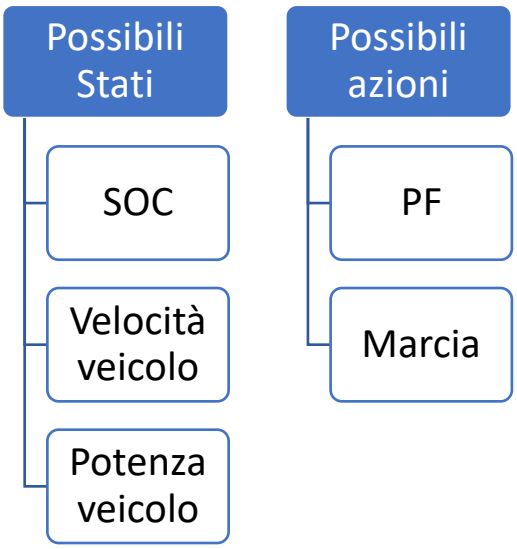

Figura 2.4 Possibili stati e azioni per applicazione del RL al controllo di veicoli ibridi

3 Modellazione del veicolo e ciclo di guida

Il codice precedentemente implementato su MATLAB permette di studiare diverse architetture di veicoli ibridi e diversi cicli di guida del veicolo. Nel corso del lavoro le prove sono state fatte su una particolare architettura di un veicolo ibrido parallelo, definita nel codice come p2. In questo capitolo, verrà quindi descritta brevemente l'architettura e il modello del veicolo utilizzati nel lavoro di tesi.

3.1 Architettura p2

3.1.1 Schema di base dell'architettura p2

In questo lavoro viene considerato un veicolo con architettura p2. Si tratta di un veicolo ibrido a configurazione parallelo, quindi avente un solo motore elettrico (EM1) e un motore a combustione interna (ICE) collegati attraverso un giunto di coppia (TC). Gli altri elementi fondamentali presenti nel veicolo sono: la batteria (Battery) che fornisce potenza elettrica al motore in trazione o la riceve in condizioni di frenata rigenerativa; l'inverter (Inv) interposto tra batteria e motore; le trasmissioni atte al cambio marcia (TR) e il differenziale sull'assale primario (FDp). In questa configurazione, sono anche presenti due frizioni: una atta a disaccoppiare l'EM1 e l'altra atta a disaccoppiare l'ICE. Ciò risulta possibile poiché il giunto di coppia posizionato tra i due motori si trova a monte della trasmissione TR. Il modello di veicolo considerato è a trazione anteriore. Nella Figura 5 è mostrato uno schema di questa architettura.

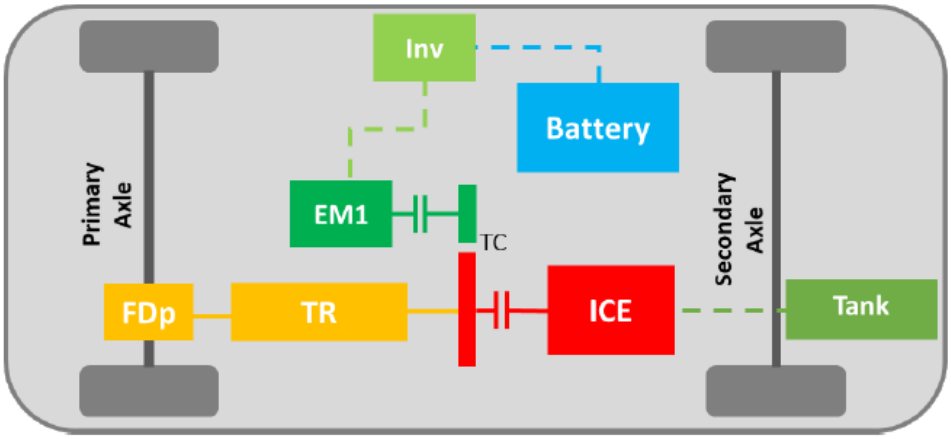

Figura 3.1 Schema dell'architettura p2

3.1.2 Possibili distribuzioni della potenza

La distribuzione di potenza tra EM1 e ICE può essere regolata in diversi modi. Viene quindi utilizzato un coefficiente α per indicare il rapporto tra potenza inviata al motore EM1 (P_{EM1}) e la potenza complessiva richiesta al veicolo (P_v):

$$\alpha_{PF} = \frac{P_{EM1}}{P_v}$$

Questo coefficiente α risulta molto utile per distinguere le diverse modalità di distribuzione della potenza all'interno del codice:

- $\alpha = 1$ indica un PF totalmente elettrico: comprende sia la trazione puramente elettrica che la frenata rigenerativa
- $\alpha = 0$ indica un PF totalmente termico: tutta la potenza è fornita dall'ICE
- $0 < \alpha < 1$ si riferisce a uno split della potenza tra motore elettrico e ICE
- $\alpha < 0$ indica che ci si trova in modalità di ricarica della batteria (*battery charging*)

3.2 Modello del veicolo

Il modello del veicolo utilizzato è molto semplificato: si tratta di un veicolo monotraccia che percorre traiettorie rettilinee. Tutti i fenomeni che caratterizzano un veicolo in curva come moto di rollio o angoli di deriva delle ruote non vengono quindi considerati. Inoltre, non vengono considerati i trasferimenti di carico e il conseguente beccheggio dovuto a fasi di accelerazione o frenata del veicolo. Non vengono infine considerati i limiti delle forze sviluppabili dagli pneumatici. Per calcolare la potenza necessaria per far muovere il veicolo sono quindi impostate equazioni di calcolo delle potenze resistenti al moto dovuti alle seguenti forze resistenti: resistenza aereodinamica, resistenza legata al rotolamento, resistenza dovuta alla pendenza della strada e resistenza legata all'inerzia. Tali potenze sono calcolate mediante le seguenti equazioni:

$$P_{rot} = k_{rot} m_v g \cos(S)\, V_v$$

$$P_p = m_v g \sin(S)\, V_v$$

$$P_{aria} = \frac{1}{2} \rho_{aria} C_x A_{front} V_v^3$$

$$P_{inerzia} = m_v \frac{\Delta V_v}{\Delta t} V_v$$

Dove:

- k_{rot} è il coefficiente di resistenza a rotolamento
- m_v è la massa totale del veicolo [kg]
- S è l'inclinazione della strada [rad]
- V_v è la velocità del veicolo [m/s]
- ρ_{aria} è la densità dell'aria
- C_x è il coefficiente di resistenza aereodinamica
- A_{front} è l'area della sezione frontale del veicolo [m^2]
- ΔV è la variazione di velocità [m/s] che avviene nel tempo Δt [s]

Da questa formula viene calcolata la potenza totale richiesta al veicolo come:

$$P_{tot} = P_{rot} + P_p + P_{aria} + P_{inerzia}$$

Successivamente la potenza complessiva richiesta al veicolo è calcolata aggiungendo alla P_{tot} la potenza dissipata a causa dell'inerzia dei diversi componenti del veicolo.

Sulla base di tali potenze viene definita la condizione di trazione del veicolo: considerando un istante di tempo t e il passo di discretizzazione temporale Δt, il veicolo viene considerato in trazione nell'intervallo di tempo compreso tra $t - \Delta t$ e t se la somma delle potenze all'istante di tempo $t - \Delta t$ e all'istante t è maggiore di zero.

3.3 Modellazione dei diversi componenti

Ogni componente del veicolo è stato modellizzato in modo da poter costruire un modello del veicolo del completo sufficientemente affidabile. Vengono di seguito analizzati diversi componenti, spiegando le assunzioni e formule che sono stati utilizzate. Tale modello era stato implementato in lavori precedenti sul codice di MATLAB che viene utilizzato in questo lavoro di tesi.

3.3.1 Dati dei diversi componenti

I dati utilizzati sono stati presi da componenti reali di un veicolo:

- Motore a combustione interna: i dati per questo componente sono stati presi sperimentalmente. In primo luogo, è stata considerata la caratteristica velocità angolare-coppia dell'ICE, ricavando dati fondamentali per il funzionamento del motore, come la velocità angolare massima e minima e i limiti legati alla potenza. Sono state inoltre fornite la massa, il momento di inerzia, l'efficienza e la mappatura del consumo di carburante considerato funzione della velocità angolare e potenza del motore.
- Trasmissione del cambio: sono stati considerate sei rapporti di trasmissione possibili. Anche in questo caso, i dati forniti sono sperimentali: momento di inerzia, massa e mappatura dell'efficienza della trasmissione in funzione del rapporto di trasmissione innestato.
- Motore elettrico: sono stati forniti i dati riguardanti la caratteristica velocità angolare-coppia dell'EM, con il conseguente limite massimo di velocità angolare, limite sulla potenza massima raggiungibile ed efficienza come funzione di velocità angolare e potenza.
- Differenziale e giunto di coppia: sono stati forniti rapporto di trasmissione ed efficienza dei due componenti.
- Inverter: è stata fornita solo l'efficienza come dato.
- Batteria: vengono forniti diversi dati quali il *C-rate*, la capacità e tensione nominali V_c di una singola cella, il rapporto nominale potenza-energia della batteria PE ,la capacità e tensione $V_{n,batt}$ nominali della batteria, il numero di celle in una singola unità N_c. I rimanenti dati importanti della batteria vengono calcolati con il dimensionamento della batteria stessa a partire da questi dati. Inoltre, vengono date la tensione di circuito aperto e la resistenza equivalente come funzioni dello stato di carica della batteria. Sono imposti dei limiti superiore (SOC_{max}) e inferiore (SOC_{min}) del SOC.

3.3.2 Modello della batteria

Il codice di MATLAB utilizzato permette di dimensionare la batteria utilizzata nel veicolo a partire dai dati descritti nella sottosezione precedente. Viene considerata una batteria al litio caratterizzata dalla seguente struttura. Le celle sono collegate in serie e vanno a costituire un'unità; a loro volta le unità sono connesse in parallelo andando a costituire un modulo; i moduli sono infine collegati in serie andando a costituire il pacco batteria. Essendo forniti come dati iniziali il numero di celle per unità, è necessario calcolare il numero di unità per modulo e il numero di moduli nel pacco batteria per completare il dimensionamento della stessa. In primo luogo, è necessario calcolare la capacità E_{batt} della batteria espressa in Wh utilizzando il rapporto PE e la potenza massima del motore elettrico:

$$E_{batt} = \frac{P_{EM,max}}{PE}$$

Si può da qua calcolare la capacità del pacco batteria C_{batt} e il numero di unità in parallelo N_{pu} per formare un modulo considerando che la capacità di una singola cella è uguale alla capacità di un'unità C_u:

$$C_{batt} = \frac{E_{batt}}{V_{n,batt}}$$

$$N_{pu} = \frac{C_{batt}}{C_u}$$

Per calcolare il numero di moduli in serie N_{sm} vengono utilizzate le seguenti relazioni in cui sono indicate con V_m la tensione di un modulo, che è uguale a quello di un'unità:

$$V_u = V_c \cdot N_c$$

$$N_{sm} = \frac{V_{n,batt}}{V_m}$$

Possono essere infine calcolate altre informazioni utili relative alla batteria, come la potenza massima della batteria in relazione al SOC, le massime correnti di scarica e carica della batteria.

3.4 Ciclo di guida

Per studiare il consumo del carburante di un veicolo risulta essere molto utile fare uso di cicli di guida normati. Un ciclo di guida associa a ogni istante temporale il valore di velocità che il veicolo deve avere in quell'istante di tempo. Utilizzare cicli di guida normati permette di ottenere risultati più facilmente confrontabili e leggibili. In questo lavoro di tesi si utilizza il ciclo di guida *Worldwide harmonized Light vehicles Test Procedure* (WLTP), introdotto come standard dal 2017 per il controllo delle emissioni e del consumo di carburante di un veicolo. Tale test, nonostante sia un test normato, è molto più vicino alle condizioni di guida reali di un veicolo rispetto ai cicli utilizzati in precedenza come il NEDC. Nella Figura 3.2 è mostrato il ciclo di guida WLTP3.

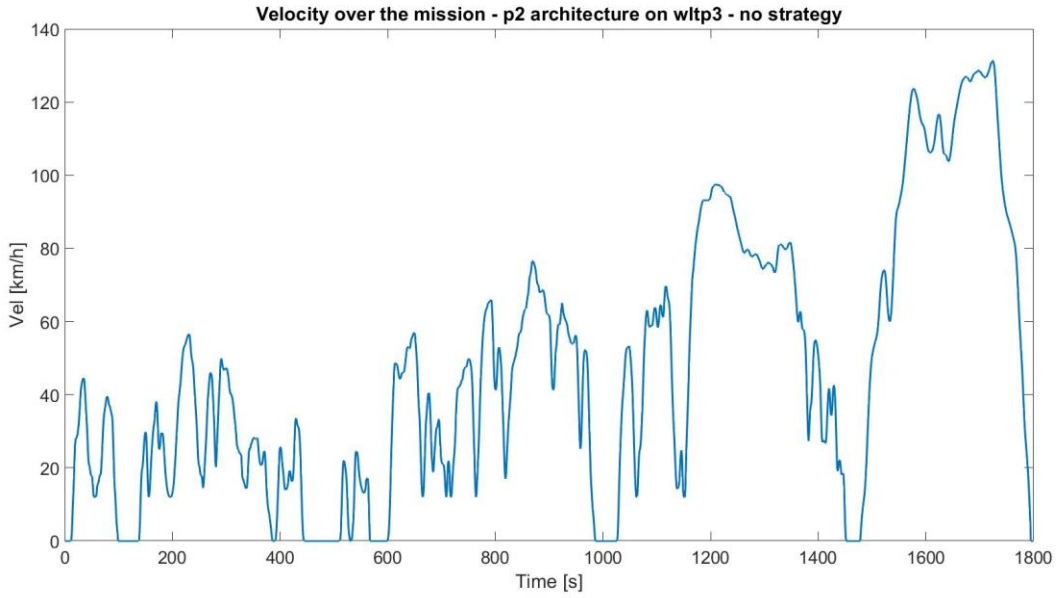

Figura 3.2 Ciclo di guida WLTP3

4 Descrizione del codice di *Q-learning*

Il lavoro di tesi parte dallo studio di un codice implementato su MATLAB durante precedenti lavori. Per comprendere a fondo il lavoro svolto e le modifiche effettuate è necessario quindi spiegare brevemente il funzionamento di base del codice. Come modello del veicolo è stato implementato il modello descritto nel capitolo precedente e come ciclo di guida il WLTP3. Nella sezione 4.1 vengono elencati i passaggi costituenti l'algoritmo, per poi chiarirli in modo più preciso nei paragrafi successivi.

4.1 Passaggi principali dell'algoritmo

In primo luogo, avviene il caricamento dei dati riferiti al veicolo e ai suoi componenti. Successivamente vi è il caricamento dei dati del ciclo di guida, comprensivi quindi della velocità, pendenza della strada per ogni istante di tempo del ciclo. È necessario, infatti, scegliere un passo temporale per applicare poi successivamente l'algoritmo di *Q-learning*: l'agente sceglierà l'azione per ogni passo temporale. Come successivo passaggio dell'algoritmo vi è la creazione delle matrici delle configurazioni contenenti tutte le possibili combinazioni delle variabili di controllo, cioè le variabili riferite alle azioni, per ogni istante temporale del ciclo. Vengono poi create le altre matrici delle configurazioni contenenti dati importanti per l'analisi del ciclo, quali la potenza e velocità richieste al veicolo e ai suoi componenti in ogni istante di tempo. In questa fase vengono anche riempite le matrici di *feasibility*, cioè quelle matrici contenenti informazioni sulla possibilità fisica di prendere una determinata azione in un determinato istante temporale. Le matrici generate fino a questo momento forniscono importanti informazioni riguardo al veicolo durante il ciclo di guida. Con questi dati di partenza, si può implementare il vero e proprio algoritmo di controllo tramite *Q-learning*. Dopo aver definito i parametri $\alpha, \gamma, \varepsilon$ e il numero di episodi di allenamento, è implementato un ciclo for su tutti gli episodi di allenamento. Per ogni episodio è necessario compiere un ciclo for su tutti gli istanti temporali di un episodio. Per ogni istante di tempo vengono prelevati i valori delle grandezze contenute nelle matrici delle configurazioni per quell'istante di tempo. A questo punto, una volta determinato lo stato dell'ambiente in quell'istante di tempo, viene scelta un'azione tra quelle considerate fisicamente valide con il metodo *ε-greedy*. Si calcola quindi il SOC dell'istante di tempo successivo e lo stato successivo. Infine, viene calcolato il *reward* ottenuto tramite la funzione di *reward*. Si aggiorna quindi il valore della *Q(s,a)* riferita allo stato dell'ambiente in quell'istante di tempo e all'azione effettivamente presa. Un singolo episodio può terminare per due ragioni: o perché si giunge all'ultimo istante temporale del ciclo, oppure perché si arriva a una situazione in cui non vi è neanche un'azione considerata fisicamente valida. Alla fine di ogni episodio viene calcolato il *discounted return* dell'episodio stesso e vengono salvati i valori finali SOC e consumo di combustibile. Infine, all'ultimo episodio di allenamento viene applicata esclusivamente *exploitation* e sono quindi salvati i risultati finali derivati dall'allenamento dell'agente.

4.2 Variabili di controllo e azioni

Come precedentemente accennato, le variabili di controllo che vanno a costituire un'azione sono la distribuzione della potenza, la marcia, caratterizzata dal proprio rapporto di trasmissione, e lo stato di accensione dell'ICE. Per stato di accensione dell'ICE si intende semplicemente la possibilità di avere l'ICE spento o acceso. La distribuzione del PF tra ICE e EM è indicata con il coefficiente α_{PF} definito precedentemente come $\alpha_{PF} = {P_{EM1}}/{P_{VEH}}$. Mentre la marcia e lo stato di accensione del motore sono variabili discrete, dal punto di vista fisico α potrebbe variare come una variabile continua tra 1 e -1. Per applicare il *Q-learning* tabulare si è però costretti a discretizzare le variabili riferite alle azioni: anche α_{PF} viene quindi discretizzato e può assumere solo un numero determinato di valori, determinato dall'utente. Il numero delle azioni N_{AZ} sarà quindi dato dal prodotto tra il numero di marce N_{GB}, numero di α_{PF} possibili N_{PF} e il numero di stati di accensione dell'ICE N_{ES} (sempre pari a 2):

$$N_{AZ} = N_{PF} \cdot N_{GB} \cdot N_{ES}$$

Ogni azione, infatti, è data dalla combinazione di un preciso PF, marcia e stato dell'ICE. Durante il lavoro di tesi il numero di marce N_{GB} è sempre tenuto costante e pari a 6; il numero di possibili distribuzioni di potenza è mantenuto sempre uguale a 7. Precisamente sono considerati i seguenti possibili valori di α_{PF}:

- $\alpha_{PF} = 1$: PF totalmente elettrico
- $\alpha_{PF} = 0$: PF totalmente termico
- $\alpha_{PF} = 0.25,\ \alpha = 0.5,\ \alpha = 0.75$: PF suddiviso tra ICE e EM
- $\alpha_{PF} = -0.5,\ \alpha = -1$: PF di ricarica della batteria attraverso l'ICE.

4.3 Matrici delle configurazioni e *feasibility*

Come precedentemente accennato nella sezione 4.1, durante la fase precedente all'implementazione dell'algoritmo di *Q-learning*, vengono create le matrici delle configurazioni. Ognuna di queste matrici ha un numero di righe pari al numero di intervalli di tempo N_{int} in cui è suddiviso il ciclo di guida e un numero di colonne pari al numero di azioni che l'agente può prendere ($N_{AZ} = N_{PF}N_{GB}N_{ES}$). Ogni matrice si riferisce a una determinata grandezza fisica calcolabile in ogni istante di tempo a partire dai dati del ciclo e dall'azione scelta in quell'istante. Ogni riga della matrice contiene quindi i valori che la grandezza fisica corrispondente può assumere per ogni possibile azione in un determinato intervallo di tempo.

Nel codice vengono calcolate in primo luogo le velocità che i singoli componenti del veicolo raggiungono nei diversi istanti di tempo e per le diverse azioni. Si ottengono quindi a partire dalla velocità del veicolo, le velocità angolari delle ruote, degli alberi del differenziale e del cambio, del motore a combustione interna, del motore elettrico. Per quanto riguarda alcuni di questi componenti, vi sono però dei limiti fisici legati alle loro velocità: per questa ragione vengono introdotte per la prima volta le matrici di *feasibility*. Con la parola *feasibility* ci si riferisce alla possibilità fisica di prendere una determinata azione in un determinato istante di tempo. La variabile di *feasibility* è di conseguenza booleana: pari a 1 se l'azione è in effetti *feasible,* cioè fisicamente prendibile in quell'istante di tempo, pari a zero se non è *feasible*. Le matrici di *feasibility* hanno quindi dimensioni pari a $N_{int} \cdot N_{AZ}$. Per quanto riguarda le velocità dei componenti vi sono limiti legati alla velocità minima e massima dell'ICE e alla velocità massima dell'EM. A titolo puramente esemplificativo, se in un istante di tempo e per una determinata azione la velocità dell'ICE assumerebbe un valore maggiore della velocità massima, il corrispondente valore della matrice di *feasibility*, riferito a quell'intervallo di tempo e a quell'azione, è nullo. Successivamente vengono calcolate le matrici delle configurazioni delle potenze relative ai singoli componenti del veicolo, quali ICE, EM, batteria. Anche in questo caso, è necessario aggiornare le matrici di *feasibility* in modo che vengano rispettati limiti fisici dei componenti sulla potenza. Per quanto riguarda gli intervalli di tempo del ciclo in cui vi è frenata, è stato deciso di imporre sempre la frenata rigenerativa attraverso le matrici di *feasibility*. In un intervallo di tempo di frenata viene imposta a 0 la variabile booleana di *feasibility* delle azioni con PF diverso dal puro elettrico. Se il valore assoluto della potenza di frenata richiesta in un istante di tempo supera la potenza massima del motore elettrico, il resto della potenza di frenata è comunque garantito dall'impianto frenante.

Le matrici di *feasibility* ottenute alla fine di questa parte di pre-processamento assumono un ruolo fondamentale anche nello sviluppo dell'algoritmo di *Q-learning*. Grazie ad esse, infatti, si riesce a garantire un allenamento più veloce ed efficace dell'agente: in ogni istante di tempo le azioni riconosciute come non *feasible* non vengono minimamente considerate dall'agente che invece può prendere una tra le sole azioni *feasible*. Questo garantisce all'agente di evitare la grossa e inutile perdita computazionale e di tempo durante l'allenamento di imparare a riconoscere le azioni *feasible* e non *feasible*. Nel caso si utilizzi come solo stato il SOC, l'utilizzo delle matrici di *feasibility* risulta essere necessario: non essendo presente nello stato alcuna informazione legata a velocità e potenza del veicolo, si genererebbero grossi problemi durante l'allenamento.

4.4 Allenamento dell'agente

4.4.1 Scelta dei parametri

Dopo aver terminato questa grossa parte di pre-processamento inizia il vero e proprio allenamento dell'agente di *Q-learning*. In primo luogo, devono essere impostati con attenzione i parametri tipici del *Q-learning* e da cui dipendono i risultati dell'allenamento. Le prime righe di codice relative a questa parte servono a inserire manualmente il *discount factor* γ, i valori iniziali del *learning rate* α e di ε. È necessario anche specificare il passo di discretizzazione dell'unica grandezza che costituisce lo stato: il SOC. Attraverso una funzione specifica, è poi determinato il numero di episodi di allenamento: esso nel codice dipende quindi dal numero di stati dell'ambiente e dal numero di istanti di tempo del ciclo di guida. Questa è una scelta logica in quanto, tanto maggiore è il numero di stati, tanto maggiore è il numero di coppie stato-azione di cui bisogna trovare il corretto *Q(s,a)*; inoltre, più è lungo il ciclo più l'agente probabilmente incontrerà un numero maggiore di stati dell'ambiente. Dopo la prima fase di settaggio dei parametri, si entra all'interno del singolo episodio di allenamento. Per ogni istante di tempo vengono quindi calcolati i valori che il SOC potrebbe assumere all'istante di tempo successivo facendo uso delle grandezze contenute nelle matrici delle configurazioni create precedentemente. Si ottiene un vettore di dimensioni pari a N_{AZ} in cui ogni cella contiene il valore che il SOC avrebbe nell'istante di tempo successivo se venisse presa l'azione corrispondente alla posizione della cella. Fino a questo punto non è stata ancora scelta l'azione da prendere.

4.4.2 Leggi di variazione della ε

Come accennato precedentemente per la scelta dell'azione viene applicata la tecnica *ε-greedy*, per bilanciare *exploration* ed *exploitation*. In generale è opportuno usare una ε decrescente con il numero di episodi. Le leggi utilizzate in questo lavoro per far diminuire la ε sono due. La prima fa decrescere ε in modo lineare secondo la seguente formula:

$$\varepsilon_{episodio} = \frac{N_{episodio}}{N_{episodio_{finale}}} \left(\varepsilon_{episodio_{finale}} - \varepsilon_{episodio_{iniziale}}\right) + \varepsilon_{episodio_{iniziale}}$$

Dove:

- $N_{episodio}$ è il numero dell'episodio di allenamento attuale
- $N_{episodio_{finale}}$ è il numero totale di episodi di allenamento
- $\varepsilon_{episodio_{finale}}$ è il valore che ε assume nell'ultimo episodio ed è imposto manualmente
- $\varepsilon_{episodio}$ è il valore che ε assume nell'episodio attuale
- $\varepsilon_{episodio_{iniziale}}$ è il valore che ε assume nel primo episodio ed è imposto manualmente

Nella Figura 4.1 si può vedere l'andamento grafico di ε con questa legge.

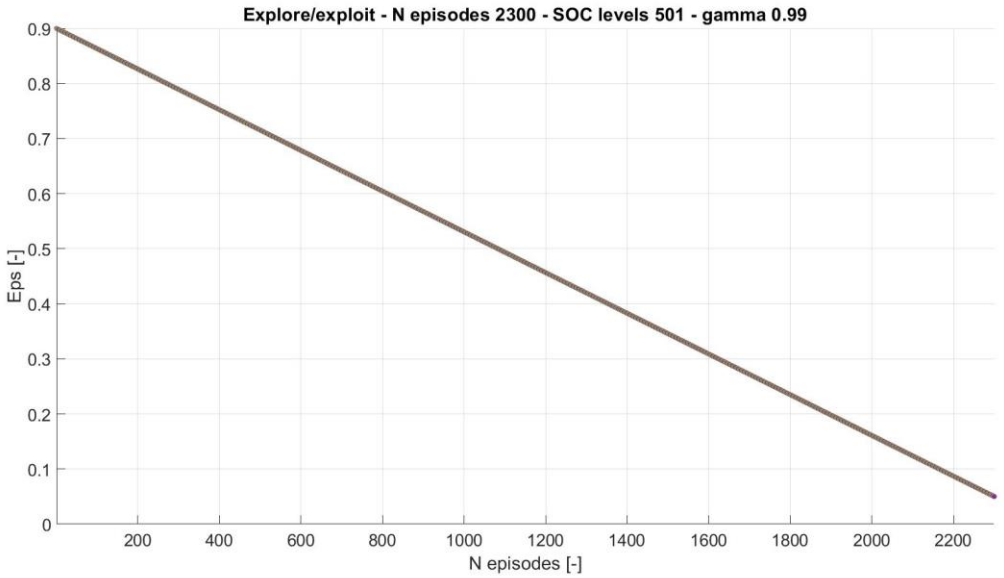

Figura 4.1 E rispetto al numero di episodi con legge lineare

La seconda legge per far decrescere ε è più complessa e viene indicata in questo lavoro con il nome "costante+parabola" per semplicità. Per un numero di episodi iniziali $N_{episodio_{eps}}$ fissati dall'utente, la ε è mantenuta costante e pari a 1. Dall'episodio numero $N_{episodio_{eps}}$ la ε viene fatta decrescere come una funzione quadratica rispetto al numero di episodi, mediante la seguente relazione:

$$\varepsilon_{episodio} = \begin{cases} -\dfrac{k}{N_{episodio_{finale}}}\,(N_{episodio} - N_{episodio_{eps}})^2 + \varepsilon_{episodio_{eps}} & se\ N_{episodio} \geq N_{episodio_{eps}} \\ 1 & se\ N_{episodio} < N_{episodio_{eps}} \end{cases}$$

Dove:

- $\varepsilon_{episodio_{eps}}$ è il valore della ε all'episodio numero $N_{episodio_{eps}}$
- k è una costante che dipende da $\varepsilon_{episodio_{eps}}$, $N_{episodio_{eps}}$ e $N_{episodio_{final}}$

Nella figura successiva è riportato il grafico rappresentante l'andamento di ε in funzione del numero di episodi. In questo particolare esempio è stato scelto un $N_{episodio_{eps}} = \frac{N_{episodio_{finale}}}{6}$.

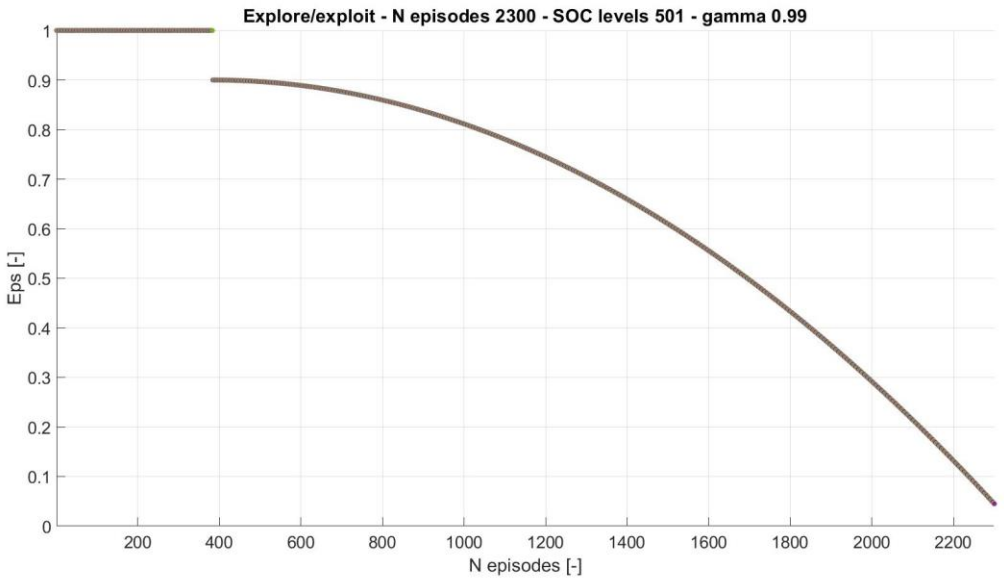

Figura 4.2 E rispetto al numero di episodi con la legge costante+parabola

4.4.3 Scelta dell'azione, nuovo stato e *reward*

Una volta calcolato l'ε corrispondente all'episodio preso in considerazione, si procede alla scelta dell'azione. Per implementare la strategia ε-*greedy*, si deve prima determinare se nell'istante di tempo considerato ci si trova in *exploration* o *exploitation*. Per fare ciò si usano due variabili ausiliarie x e y. Il valore della y è pari a un numero casuale compreso tra 0 e 1, ottenuto mediante la funzione rand di MATLAB; x è una variabile booleana pari a 1 se y è maggiore del valore di ε in quell'episodio. Per completezza si riporta il codice utilizzato su MATLAB:

$$y = rand;$$

$$x = y >= epsilon;$$

A questo punto, se la variabile booleana x è uguale a 0 si ha *exploration*, se è uguale a 1 viene presa l'azione considerata migliore dall'agente per quello stato (*exploitation*). Con questo metodo si riesce a garantire la corretta implementazione del metodo ε-*greedy*: la probabilità di essere in fase di esplorazione sarà pari a ε, mentre la probabilità d'essere in *exploitation* sarà pari a $1 - \varepsilon$. Si ricorda che in generale le azioni che l'agente può prendere in un determinato istante di tempo sono solamente le azioni *feasible*. Di conseguenza in *exploration*, viene scelta casualmente un'azione tra quelle fisicamente valide; in *exploitation*, viene scelta l'azione *a* per cui il valore della *Q(s,a),* riferita allo stato presente nell'istante di tempo considerato, è massimo. Inoltre, in un determinato numero di episodi durante l'allenamento è imposta per ogni istante temporale, pura *exploitation*: tali episodi sono definiti episodi di validazione e non sono utili al fine di allenare l'agente ma solo per analizzarne l'efficacia dell'apprendimento a prova conclusa, come vedremo successivamente.

Dopo aver quindi determinato l'azione da prendere, si può individuare il SOC all'istante di tempo successivo che corrisponde all'elemento nella posizione, relativa all'azione presa, del vettore precedentemente calcolato dei SOC possibili all'istante successivo. Il valore del SOC così calcolato non corrisponde ancora allo stato successivo poiché non è ancora stato discretizzato. Il valore appartenente alla griglia di discretizzazione del SOC più vicino al SOC non discretizzato viene quindi considerato come stato successivo.

Infine, viene calcolato il *reward* derivante dalla scelta dell'azione in quell'istante di tempo, attraverso l'apposita funzione. Si discuterà ampliamente nel capitolo 5 riguardo la scelta della funzione di *reward*, che dovrà dipendere sia dal SOC che dal consumo di carburante, cioè le due grandezze fisiche che si vogliono tenere sotto controllo.

4.4.4 Aggiornamento della *Q*-table

Dopo aver calcolato il *reward* derivato dalla scelta dell'azione *a* nell'istante di tempo *t* in cui l'ambiente è caratterizzato dallo stato *s*, si può aggiornare il valore della *Q(s,a)* con la seguente formula, dove s' e a' sono lo stato successivo e l'azione possibile nell'istante di tempo successivo:

$$Q(s,a) = Q(s,a) + \alpha\left[r + \gamma \max_{a'}\left(Q(s',a')\right) - Q(s,a)\right]$$

Attraverso l'aggiornamento del valore della *Q(s,a)* avviene l'apprendimento dell'agente, che impara a riconoscere le azioni che portano a un *discounted return* maggiore. Si ricorda che i valori della *Q* sono contenuti nella matrice *Q-table* in cui ogni cella corrisponde a una determinata coppia stato-azione. Una piccola imprecisione nell'algoritmo riguarda il termine $\max_{a'}\left(Q(s',a')\right)$: il massimo viene infatti considerato rispetto a tutte le azioni a' e non solo a quelle realmente *feasible* nell'istante di tempo successivo. Inoltre, nel codice è stato inizialmente deciso di aggiornare la *Q(s,a)* solo se nell'intervallo di tempo considerato è presente la trazione. Con questa scelta l'algoritmo ottiene risultati buoni come vedremo in seguito, ma non considera minimamente gli istanti di tempo di frenata. È importante sottolineare come la scelta dell'azione in frenata sia influente sui risultati finali ottenuti e sia collegata solo alla scelta della marcia: quindi teoricamente è scorretto tralasciare in toto dal problema le fasi di frenata che saranno analizzate più approfonditamente in capitoli successivi.

4.4.5 Salvataggio dati finali e analisi dell'allenamento

Alla fine di ogni episodio di allenamento vengono salvati i risultati più importanti ottenuti per poi analizzare l'efficacia dell'allenamento e i valori finali di SOC e FC. Vengono di conseguenza salvate le azioni che sono state prese (marce e PF), il SOC della batteria, il consumo di carburante, i *reward* ottenuti in ogni istante di tempo per ogni episodio. È stato possibile analizzare a fondo i risultati salvati da ogni prova grazie a delle porzioni di codice di analisi separate dal codice descritto fino a questo punto: grazie ad essi si possono riprendere i dati salvati da una singola prova ed analizzarli. Ciò è risultato estremamente utile per il confronto tra le differenti prove. Vengono quindi ora illustrati alcuni grafici fondamentali per analizzare l'andamento dell'allenamento. In primo luogo, si può ottenere una rappresentazione grafica in 3-D della *Q-table*, dove sui due assi orizzontali (SOC e Actions) sono plottati gli stati e le azioni, mentre sull'asse verticale (*Q value*) i valori assunti dalle *Q(s,a)* della *Q-table* a fine allenamento. Nella figura sottostante è riportato il grafico ottenuto da una prova a puro titolo di esempio.

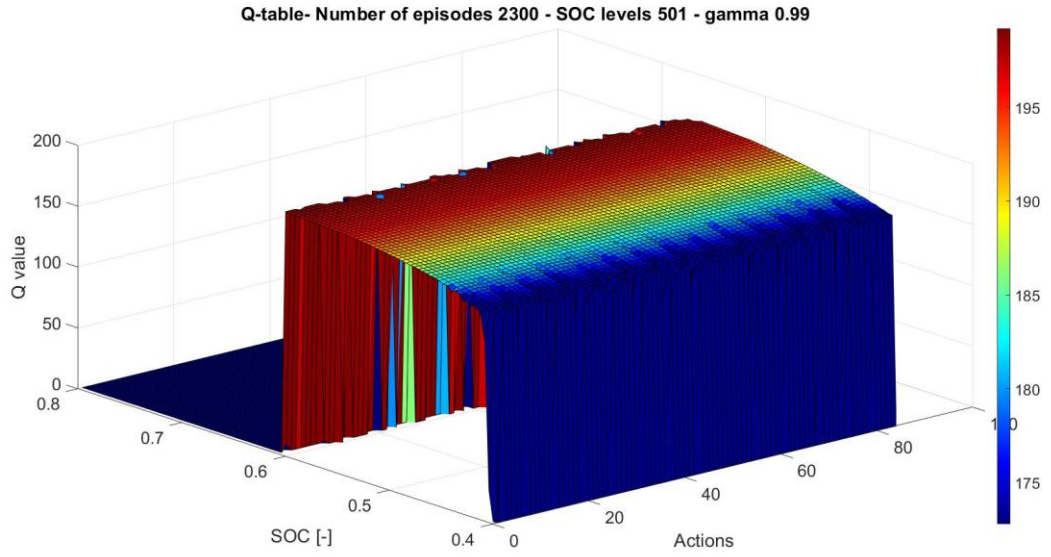

Figura 4.3 Rappresentazione in 3-D della Q-table

Può anche essere molto utile la rappresentazione tramite scatter dei *reward* ottenuti rispetto alle variabili SOC e consumo di combustibile in un intervallo di tempo (mfc) che si vogliono controllare. In questo modo si può osservare l'effettivo valore dei *reward* ottenuti durante una prova: ciò risulterà molto utile durante lo studio della funzione di *reward*. Nella figura sottostante è riportata la rappresentazione grafica descritta utilizzando la funzione scatter.

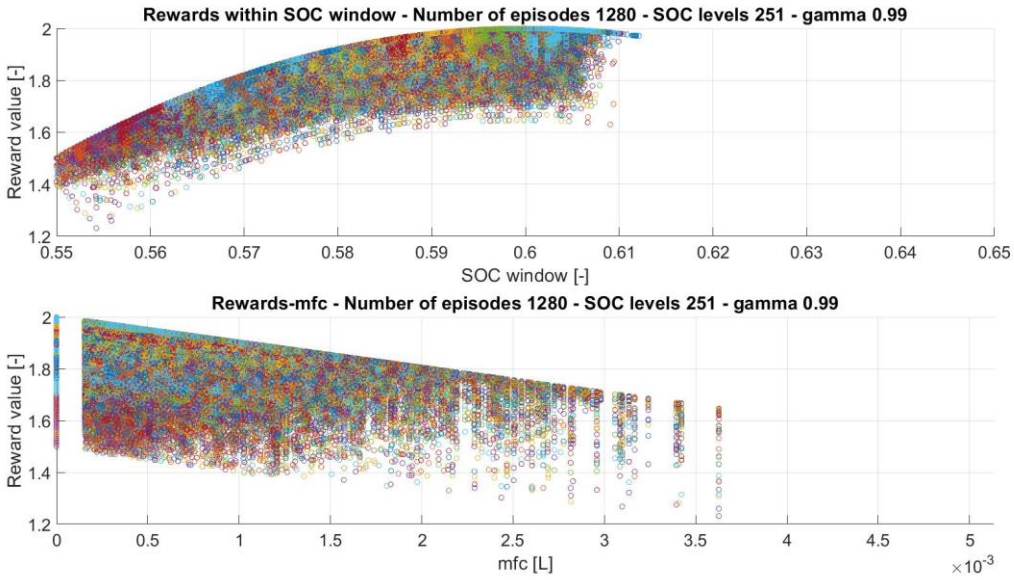

Figura 4.4 Reward ottenuti rispetto al SOC e consumo di carburante

Un altro grafico utile per analizzare il rapporto tra *exploration* ed *exploitation* è quello della sottosezione 2.2.5. Per capire se l'allenamento è avvenuto correttamente o meno è fondamentale guardare la grandezza che si vuole massimizzare per ogni problema di RL: la somma cumulata dei *discounted return* ottenuti durante la prova. Per semplicità nei grafici successivi ci si riferirà al valore della somma cumulata dei *discounted return*, attraverso il nome *discounted return*. Osservandone l'andamento grafico rispetto al numero di episodi, si possono trarre diverse considerazioni. Se l'allenamento è avvenuto idealmente, il *discounted return* dovrebbe crescere all'aumentare degli episodi, man mano che il livello di esplorazione diminuisce. Chiaramente le oscillazioni del *discounted return* intorno al suo valore di media mobile sono più ampie ed evidenti negli episodi iniziali dove prevale l'esplorazione. Negli episodi finali si dovrebbe invece idealmente notare una convergenza del *discounted return* al valore finale. Nel grafico successivo è riportato quindi l'andamento della somma cumulata del *discounted return* per ogni episodio. Inoltre, è stata plottata anche la media mobile di tale grandezza in modo da intuirne più facilmente l'andamento.

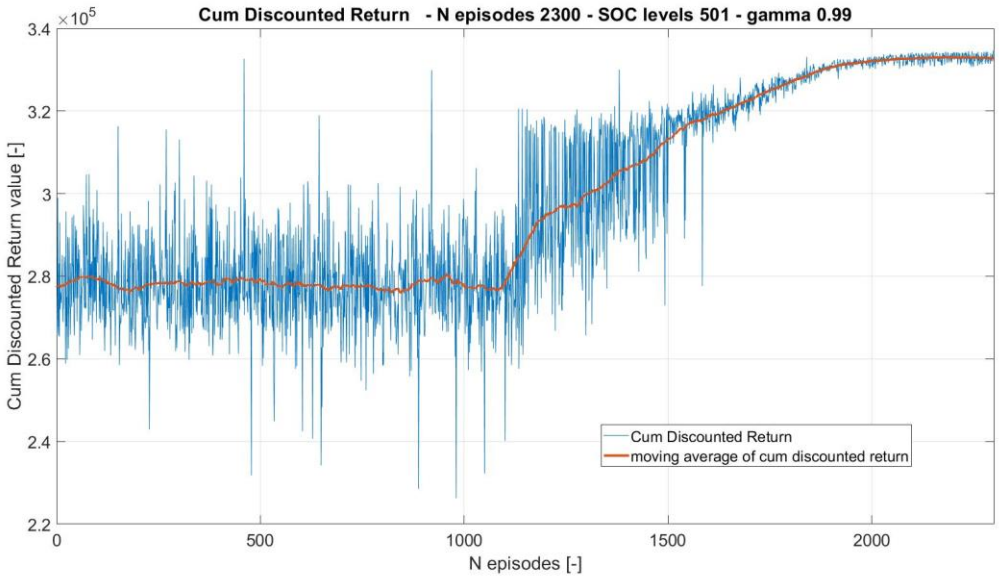

Figura 4.5 Somma cumulata dei discounted return

Altra grandezza interessante da osservare è la somma dei *reward* ottenuti durante gli episodi di validazione. In linea teorica tale grandezza dovrebbe crescere all'aumentare del numero dell'episodio di validazione. Infine, si può vedere l'andamento fisico di SOC e FC ottenuti durante tali episodi di validazione al fine di controllare che l'allenamento funzioni correttamente per il raggiungimento dell'obiettivo fisico desiderato, cioè il controllo del SOC e riduzione del consumo di carburante in questo lavoro. Nella figura successiva è riportato quest'ultimo grafico descritto.

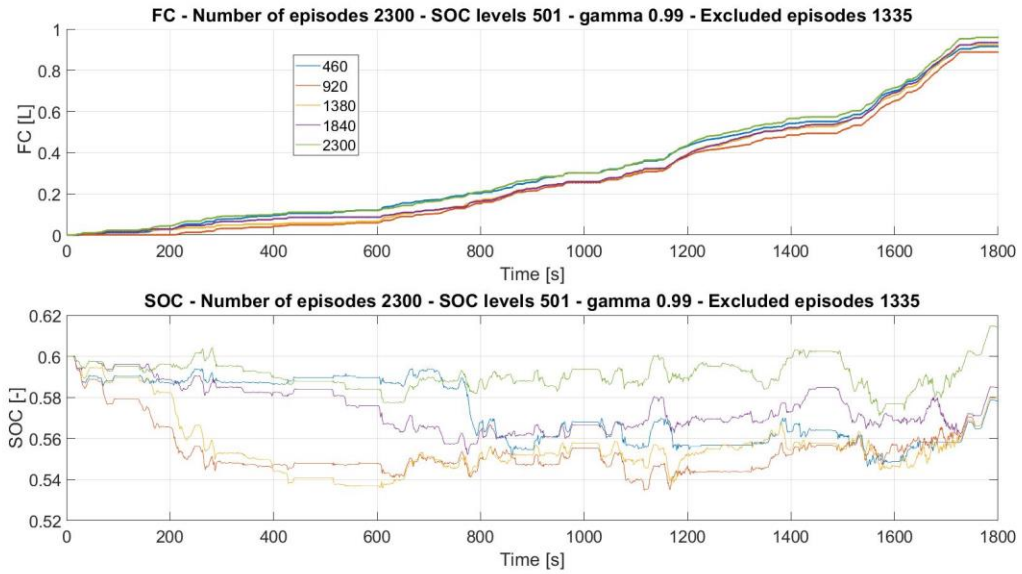

Figura 4.6 FC e SOC negli episodi di validazione

5 Scelta della funzione di *reward*

La scelta della funzione di *reward* è di fondamentale importanza per qualsiasi problema di RL. Si ricorda infatti che il *reward* è l'elemento essenziale attraverso cui avviene l'apprendimento dell'agente. Tale funzione deve essere fortemente collegata con le grandezze fisiche che si desidera controllare e impostata in modo tale da permettere il raggiungimento degli obiettivi fisici desiderati. Il range entro cui può variare il SOC in questa prima fase di lavoro è tra $[0.4,0.8]$.

5.1 Obiettivi fisici desiderati

In primo luogo, è quindi necessario individuare con precisione gli obiettivi fisici desiderati. Come citato precedentemente, si vuole ottenere il controllo sul SOC della batteria e sul consumo di carburante (FC).

Si parte quindi dall'analisi più specifica del controllo del SOC che si vuole ottenere. È necessario specificare che l'algoritmo di *Q-learning* sviluppato si adatta teoricamente sia ai veicoli di tipo *full hybrid* (FHEV) e che ai *plug-in hybrid* (PHEV). Per quanto riguarda un FHEV è opportuno garantire che il SOC finale a fine del ciclo guida analizzato sia maggiore o uguale del SOC della batteria a inizio ciclo. Si ricorda infatti che la batteria dei veicoli FHEV non può essere ricaricata tramite sorgenti di energia elettriche esterne al veicolo e di conseguenza, pensando a un'applicazione reale, il SOC a fine ciclo coincide con il SOC alla ripartenza del veicolo per un nuovo ciclo: nel caso in cui il nuovo ciclo di guida fosse particolarmente gravoso in termini di potenza richiesta e di consumo di energia elettrica della batteria, si potrebbe insorgere nel problema di raggiungere livelli troppo bassi di stato di carica. Per quanto riguarda invece i PHEV, essendo la batteria ricaricabile grazie a fonti esterne di energia elettrica, questo problema risulta essere meno evidente. Si può infatti concedere fino a un certo limite che il SOC finale sia minore del SOC iniziale. Sarebbe in questo caso da tenere in considerazione la durata temporale del ciclo di guida e dopo quanti cicli sarebbe necessario ricaricare la batteria con fonti esterne per capire la reale convenienza del metodo. Inoltre, va ricordato che per i veicoli FHEV i limiti legati al range entro il quale il SOC può variare possono essere molto stretti. Quindi altro obiettivo importante da raggiungere è il mantenimento del SOC all'interno del range desiderato per ogni ciclo preso in considerazione. Ad ogni modo è chiaro che i risultati siano molto dipendenti dal ciclo di guida utilizzato: anche per questa ragione è stato utilizzato il ciclo normato WLTP.

L'altro obiettivo fisico che si vuole raggiungere è la minimizzazione dei consumi di carburante. Per ottenere questo risultato è necessario richiedere la minore potenza possibile al motore a combustione interna e cercare di farlo lavorare in punti di lavoro ad alta efficienza. Chiaramente ciò comporta una richiesta di potenza maggiore alla batteria. I due obiettivi legati al SOC e all'FC sono, di conseguenza, in contrapposizione tra di loro. Controllare entrambi contemporaneamente è un difficile obiettivo da raggiungere, per questa ragione la scelta di una *funzione di reward* efficace non è un problema banale.

5.2 Analisi di possibili funzioni di *reward* per il solo controllo del SOC

Prima di dedicarsi al problema più complesso del controllo simultaneo di SOC e FC, si vuole determinare quali siano le migliori funzioni per il solo controllo del SOC. Questo problema risulta più semplice rispetto a quello completo, ma permette di avere un'idea sulla funzione completa da utilizzare per quanto riguarda il SOC. Si mostrano ora le diverse ipotesi di funzione *reward* provate. In questo capitolo le prove svolte sono prove "di debug", cioè hanno un numero di episodi ridotto: ciò permette di ottenere i risultati finali velocemente e farsi un'idea della potenzialità delle diverse funzioni di *reward* tentate.

5.2.1 Funzione a strati

Come primo tentativo si è analizzata una funzione di *reward* "a strati". La finestra del SOC viene suddivisa in N bande; a ognuna di esse è associato un valore costante di *reward*. Si consideri di essere all'istante di tempo *t* con uno stato *s* dell'ambiente, prendere l'azione *a* e arrivare quindi allo stato successivo *s'*. Il *reward* ottenuto in questo caso dipende solo dallo stato *s'*, quindi dal valore discretizzato del SOC nell'istante di tempo successivo a *t*. A seconda della banda in cui ricade lo stato successivo, si ottiene un diverso valore di *reward*. Può essere variato il numero N di bande così come la loro larghezza e la costante di *reward* associata a ognuna di esse. In generale le costanti k_i di *reward* sono poste tanto maggiori quanto più la banda a cui sono associata è vicina al valore del SOC a inizio ciclo. In questo modo ricevendo un *reward* più alto ogni volta che ci si trova in una banda di SOC più interna e quindi più vicina al valore iniziale, è più probabile che i valori del SOC incontrati durante l'episodio di validazione a fine allenamento appartengano a tale banda centrale. Indirettamente si dovrebbe ottenere quindi che anche il valore finale di SOC appartenga alla banda con *reward* maggiore e che non siano incontrate situazioni di non *feasibility* dovute al superamento del range massimo di SOC.

Nella rappresentazione successiva, viene considerato un esempio dell'applicazione di tale *reward*. Il numero di bande in cui è suddivisa la finestra di SOC [0.4,0.8] è pari a N=3 e la larghezza di ognuna delle bande è sotto riportata nella formula di $r(s,a)$. A ognuna di esse è associato un particolare valore di *reward* rappresentato dalle costanti k_i con i da 1 a 3, decrescenti da k_1 a k_3.

$$r(s,a) = \begin{cases} k_1 \quad se\ \mathrm{SOC}_{\mathrm{band}_{\mathrm{low}_1}} < SOC < \mathrm{SOC}_{\mathrm{band}_{\mathrm{high}_1}} \\ k_2 \quad se\ \mathrm{SOC}_{\mathrm{band}_{\mathrm{low}_2}} < SOC < \mathrm{SOC}_{\mathrm{band}_{\mathrm{low}_1}}\ oppure\ \mathrm{SOC}_{\mathrm{band}_{\mathrm{high}_1}} < SOC < \mathrm{SOC}_{\mathrm{band}_{\mathrm{high}_2}} \\ k_3 \quad se\ \mathrm{SOC}_{\mathrm{min}} < SOC < \mathrm{SOC}_{\mathrm{band}_{\mathrm{low}_2}}\ oppure\ \mathrm{SOC}_{\mathrm{band}_{\mathrm{high}_2}} < SOC < \mathrm{SOC}_{\mathrm{max}} \end{cases}$$

Dove:

- $\mathrm{SOC}_{\mathrm{band}_{\mathrm{low}_i}}\ e\ \mathrm{SOC}_{\mathrm{band}_{\mathrm{high}_i}}$ si riferiscono ai limiti inferiore e superiore della banda i
- $\mathrm{SOC}_{\mathrm{min}}$ e $\mathrm{SOC}_{\mathrm{max}}$ sono i limiti della finestra di SOC considerata

Come dimostrato dal grafico del *discounted return* in *Figura 5.1*, l'allenamento funziona correttamente, poiché si vede che la media mobile del *discounted return* ha un andamento crescente con il numero degli episodi.

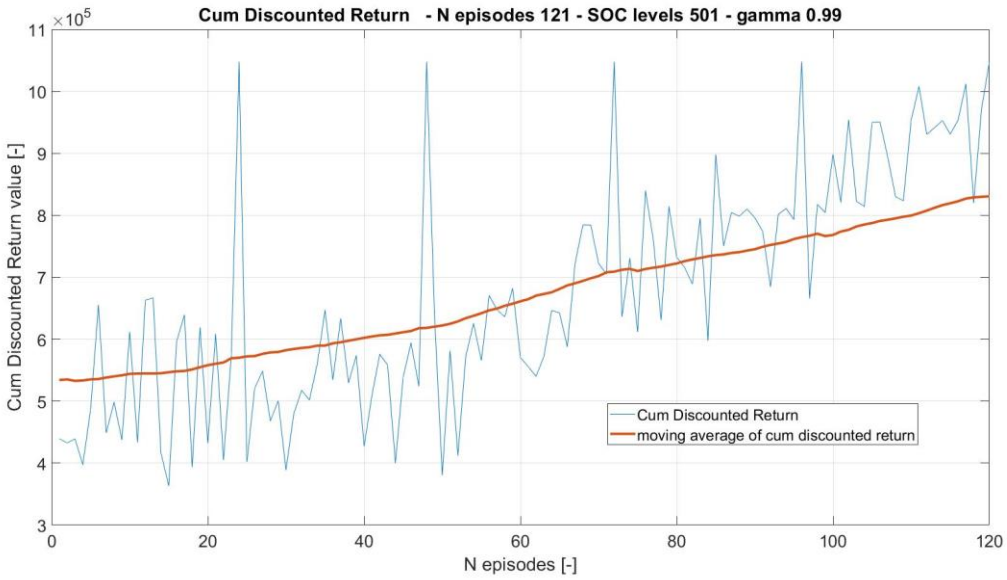

Figura 5.1 Discounted return per funzione di reward a strati

Tuttavia, i risultati finali ottenuti dipendono molto dal numero e dall'estensione delle bande in cui è suddivisa la finestra del SOC. Per un corretto utilizzo di questa funzione bisognerebbe studiare quali siano i parametri ottimali delle bande. Inoltre, l'applicazione di questa funzione del SOC nel problema completo risulta più complessa rispetto all'utilizzo della funzione a parabola spiegata successivamente.

5.2.2 Funzione a parabola

Un altro tentativo di funzione è stato quello della funzione di *reward* "a parabola". Essa presenta diversi vantaggi rispetto alla funzione vista precedentemente: permette di eliminare il problema della scelta del numero di bande e risulta più facilmente implementabile per il problema complesso che verrà studiato successivamente. Nonostante vengano introdotte le scelte arbitrarie della costante e coefficiente della parabola, queste scelte sono comunque meno complesse della scelta delle costanti k_i per ogni banda della funzione precedente. La funzione di *reward* utilizzata in questo caso è la seguente:

$$r(s,a) = cost - k(SOC(s') - SOC_{iniziale})^2$$

Dove:

- $cost$ è una costante positiva
- k è un coefficiente positivo
- $SOC_{iniziale}$ è il valore del SOC a inizio ciclo
- $SOC(s')$ è il valore del SOC discretizzato all'istante di tempo successivo quindi nello stato *s'*

Usando questa definizione, si nota come più il SOC discretizzato dell'istante di tempo successivo è vicino al SOC iniziale, più il termine negativo è ridotto e di conseguenza il *reward* è maggiore. Il *reward* massimo si ottiene quindi per un unico valore dello stato *s'* e non per un insieme di più valori come avveniva con la funzione precedente. Anche in questo caso l'allenamento funziona correttamente, come dimostrato dall'andamento del *discounted return* nella figura sottostante.

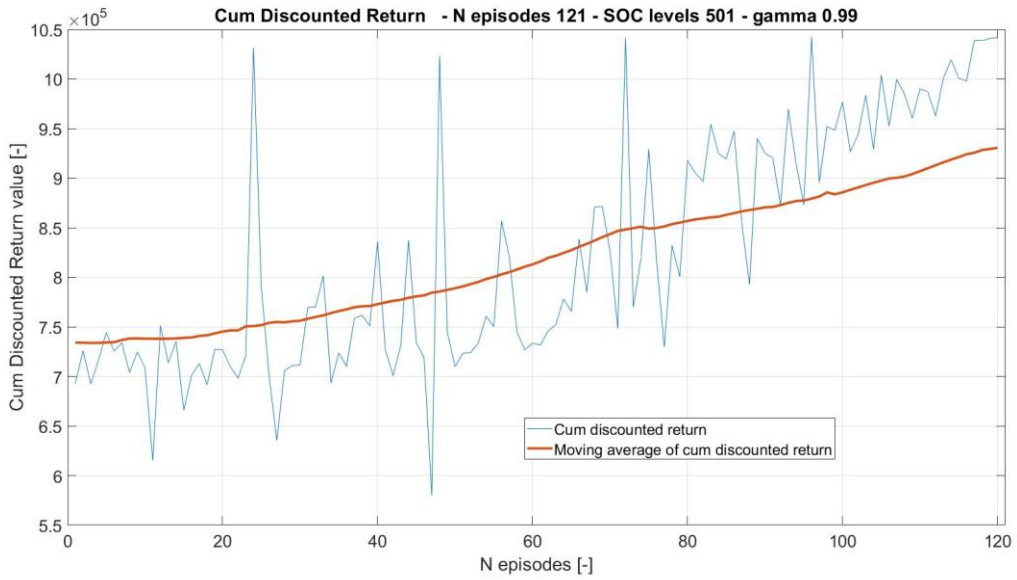

Figura 5.2 Discounted return per funzione di reward a parabola

Per i vantaggi sopra elencati si è preferito usare questa funzione per il passaggio al problema complesso.

5.3 Analisi di possibili funzioni di *reward* per il controllo del SOC e FC

In questo capitolo vengono analizzate le funzioni di *reward* di tentativo per la risoluzione del problema complesso di simultaneo controllo del FC e del SOC. In primo luogo, si è iniziato con l'analisi di una funzione di *reward* già presente nel codice, per poi passare all'implementazioni di nuove funzioni.

5.3.1 Funzione Seoul

La funzione chiamata "Seoul" è una funzione già implementata nel codice MATLAB iniziale. Con l'utilizzo di tale funzione è possibile ottenere anche *reward* negativi e non solo positivi. Finora sono state analizzate solo funzioni con *reward* sempre positivi. L'utilizzo di *reward* negativi fa cambiare molto l'allenamento. Si consideri che all'istante di tempo *t* del primo episodio di allenamento l'ambiente sia caratterizzato da uno stato *s*. L'agente sceglie un'azione *a* e riceve un *reward* negativo. Quando l'ambiente si ripresenterà nello stato *s*, l'agente se in *exploitation* non sceglierà mai nuovamente l'azione *a*, bensì prenderà un'azione nuova diversa. In generale con l'utilizzo di *reward* negativi l'agente in *exploitation* tenderà ad evitare sempre le azioni che portano *Q(s,a)* negativi per un determinato stato e di conseguenza si ottiene un aumento dell'esplorazione a parità di ε. A parte queste considerazioni l'utilizzo di *reward* negativi insieme a *reward* positivi tende a complicare maggiormente l'allenamento, perciò è una tecnica da utilizzare con cautela.

Riprendendo la funzione Seoul, essa è caratterizzata dalla seguente formula generale:

$$r = \begin{cases} -icemfc(t,a) + (SOC(s') - SOC_{iniziale})^2 & se\ SOC(s') > SOC_{lim} \\ -icemfc(t,a) - cost & se\ SOC(s') < SOC_{lim} \end{cases}$$

Dove:

- $icemfc(t,a)$ è il consumo di combustibile avvenuto in un intervallo di tempo che parte dall'istante *t*, una volta scelta l'azione *a*
- $cost$ è una costante positiva
- SOC_{lim} è un valore di SOC imposto come soglia, comunque maggiore del limite inferiore imposto al SOC

Questa funzione effettivamente prende in considerazione sia l'FC che il SOC. Se analizzata, si può però notare come porti a risultati pessimi in termini di SOC. Infatti, quando il $SOC(s') > SOC_{lim}$, il termine $(SOC(s') - SOC_{iniziale})^2$ tende ad essere tanto maggiore quanto il valore di $SOC(s')$ è più distante dal valore di $SOC_{iniziale}$. L'agente tende quindi in *exploitation* a prendere azioni che portino lontano dal valore di SOC iniziale che in realtà è il target. Inoltre, per $SOC(s') > SOC_{lim}$ è possibile ottenere *reward* negativi se il termine $icemfc(t,a)$ è maggiore di $(SOC(s') - SOC_{iniziale})^2$. D'altra parte, quando $SOC(s') < SOC_{lim}$ il *reward* ottenuto è sempre minore di 0. Ciò comporta delle difficoltà nell'allenamento dell'agente poiché per $SOC(s')$ molto vicino a SOC_{lim} si può ottenere il massimo del *reward* oppure *reward* negativi. Tale difficoltà nell'allenamento dell'agente è dimostrata dal pessimo andamento decrescente della media mobile del *discounted return*.

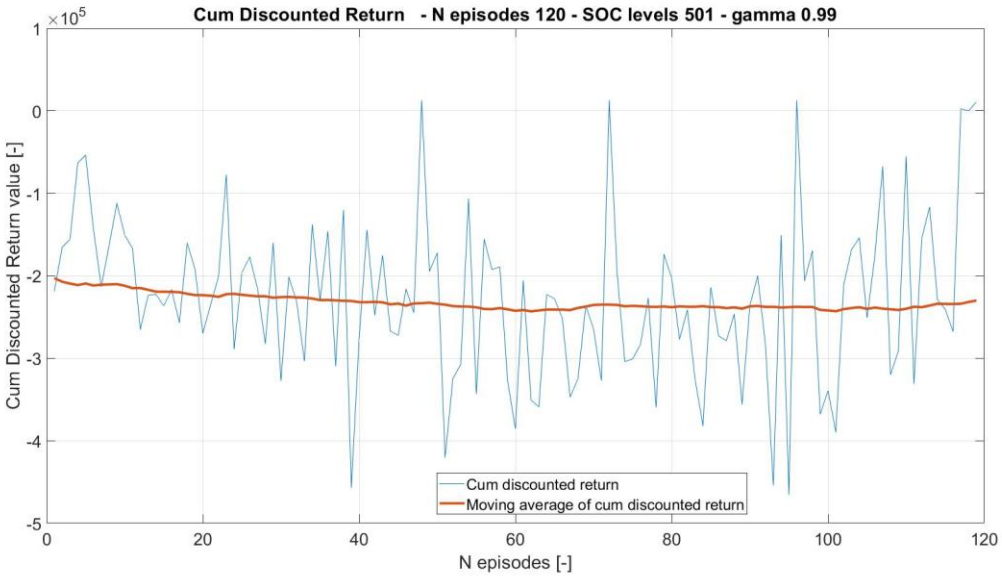

Figura 5.3 Discounted return per funzione di reward Seoul

Per quanto riguarda invece il consumo di carburante, la funzione di *reward* premia giustamente la condizione di minor consumo di carburante: tanto maggiore è il termine $icemfc(t,a)$ tanto più il *reward* diventa minore. A titolo esemplificativo si consideri una prova in cui $cost = -10$ e $SOC_{lim} = 0.45$. Può essere molto interessante osservare la *Q-table* che si ottiene al termine dell'allenamento. Come previsto, i valori di *Q(s,a)* più elevati si hanno per SOC di poco maggiori di SOC_{lim}, mentre per i valori di SOC di poco minori di SOC_{lim} i valori di *Q(s,a)* sono negativi.

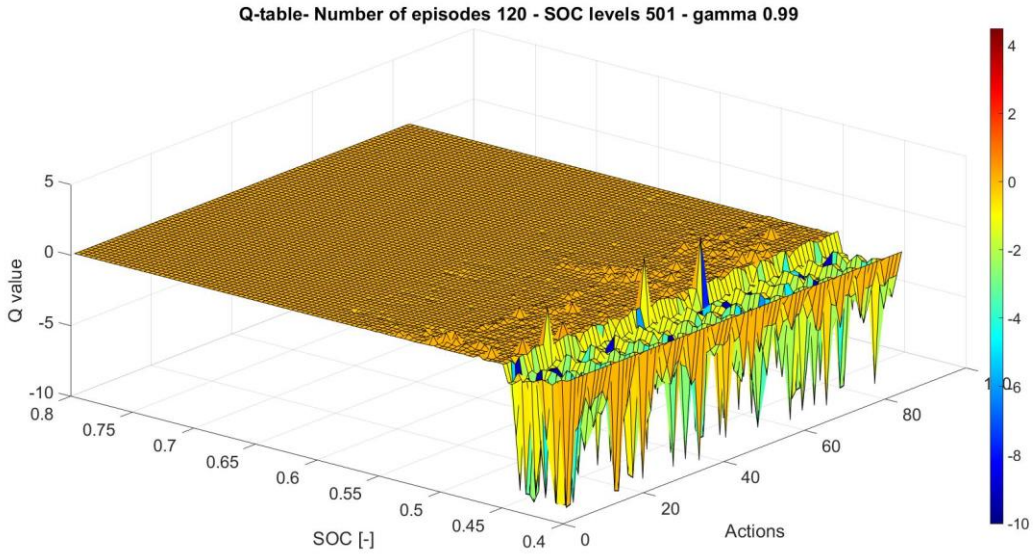

Figura 5.4 Q-table ottenuta per funzione di reward Seoul

Visti gli scarsi risultati ottenuti con questa funzione si sono provate diverse modifiche ad essa tra le quali l'aggiunta di una costante positiva per ottenere tutti *reward* positivi e l'aggiunta di un coefficiente per pesare l'influenza del consumo di carburante rispetto al SOC. Tuttavia, le modifiche provate non hanno portato a buoni risultati e per questo si è deciso di provare altre funzioni di *reward*.

5.3.2 Funzione Prezzo

Molto interessante è stato lo studio di una funzione di *reward* collegata al costo monetario dell'energia utilizzata, chiamata funzione "Prezzo". Tale funzione ha quindi come obiettivo quello di ridurre le spese di carburante ed energia elettrica durante il ciclo di guida. Attraverso il *reward* non si può esprimere questo obiettivo esteso a tutto il ciclo, bensì solo a un intervallo di tempo. La formulazione utilizzata per questa funzione è stata di conseguenza la seguente:

$$r(s,a) = \begin{cases} cost_1 - \mathrm{k} \cdot \left(price_{fc_{dt}} - price_{el_{dt}}\right) & se\ SOC(s') > SOC_{lim} \\ -cost_2 - icemfc(t,a) & se\ SOC(s') < SOC_{lim} \end{cases}$$

$$price_{fc_{dt}} = icemfc(t,a) \cdot price_{fc_L}$$

$$price_{el_{dt}} = E_{batt} \cdot \left(SOC(s) - SOC(s')\right) * price_{el_{Wh}}$$

Dove:

- $cost_1$, $cost_2$ e k sono costanti positive
- $price_{fc_{dt}}$ e $price_{el_{dt}}$ sono rispettivamente i costi del carburante e dell'energia elettrica utilizzati nell'intervallo di tempo *dt* considerato
- $price_{fc_L}$ è il costo al litro del carburante
- E_{batt} è la capacità della batteria espressa in Wh
- $price_{el_{Wh}}$ è il costo al Wh dell'energia elettrica

Per evitare i problemi di apprendimento dell'agente spiegati prima a causa dell'utilizzo di *reward* negativi con $SOC(s') > SOC_{lim}$, è opportuno porre il valore della $cost$ a un valore tale per cui $cost - k \cdot \left(price_{fc_{dt}} - price_{el_{dt}}\right) > 0$ per ogni valore di $price_{fc_{dt}}$ e $price_{el_{dt}}$ presenti nel ciclo di guida: in questo modo tutti i *reward* ottenuti per $SOC(s') > SOC_{lim}$ risultano maggiori di zero. Utilizzando però questa funzione è stato necessario introdurre *reward* negativi per $SOC(s') < SOC_{lim}$ con SOC_{lim} maggiore del SOC minimo della finestra del SOC considerata. Questo perché il $price_{fc_{dt}}$ è spesso maggiore del $price_{el_{dt}}$ scegliendo dei $price_{fc_L}$ e $price_{el_{Wh}}$ realistici e di conseguenza l'agente tende a privilegiare la riduzione del SOC piuttosto che il consumo di carburante: se non fossero introdotti *reward* negativi la maggior parte degli episodi risulterebbe escluso a causa di una riduzione del SOC oltre il limite inferiore. A titolo esemplificativo sono riportati nella figura successiva i *reward* ottenuti durante i diversi episodi di allenamento scegliendo una $cost_1 = 2$, $cost_2 = 10$ e k =100:

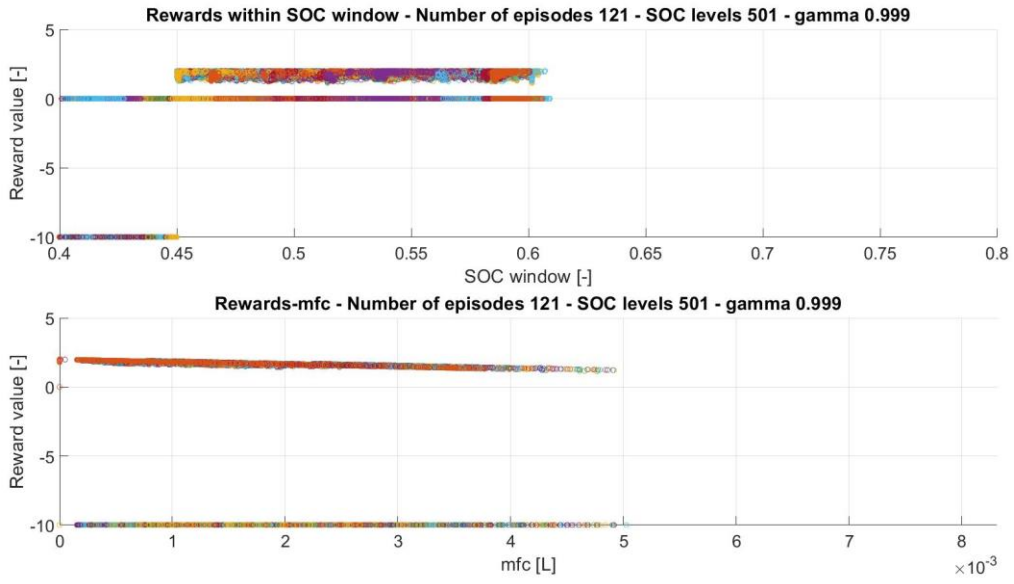

Figura 5.5 Reward ottenuti per funzione di reward Prezzo

Comunque, l'allenamento con questa funzione di *reward* risulta efficace come dimostra l'andamento crescente dei *discounted return* ottenuti nella Figura 5.6:

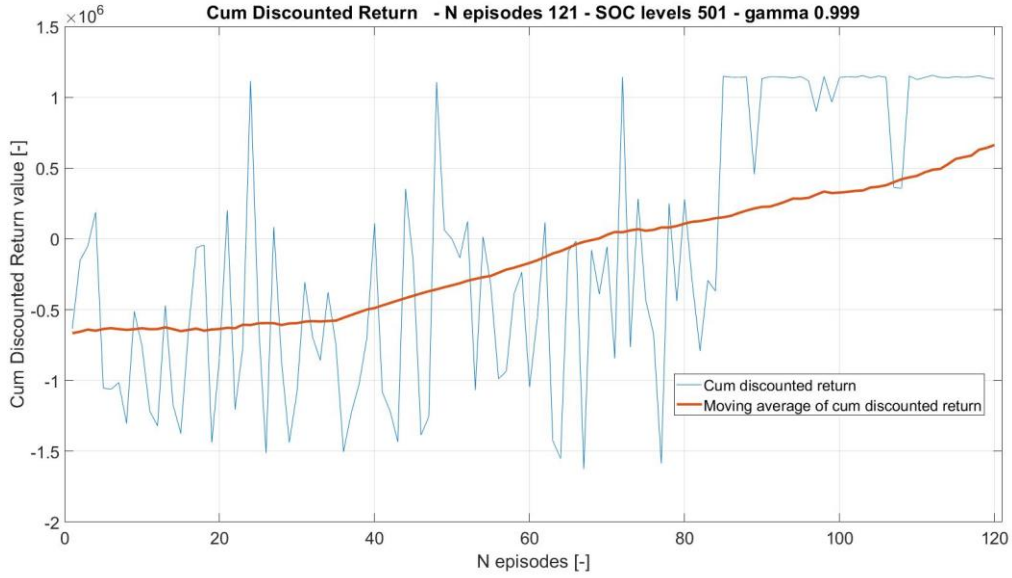

Figura 5.6 Discounted return per funzione di reward Prezzo

Il grosso problema di questa funzione è che non rispetta uno dei due obiettivi fisici definito a inizio capitolo. Mentre infatti si riesce ad ottenere una grossa riduzione del consumo di carburante durante l'allenamento, non si ottiene un SOC finale maggiore o uguale rispetto al SOC iniziale. Questo avviene poiché con questa funzione di *reward* si tende a privilegiare maggiormente la riduzione del SOC piuttosto che il consumo di carburante. Riprendendo la prova di esempio citata prima, si può notare come il SOC a fine ciclo dell'episodio di validazione a fine allenamento sia di molto inferiore a quello iniziale e si avvini al SOC_{lim} imposto di 0.525:

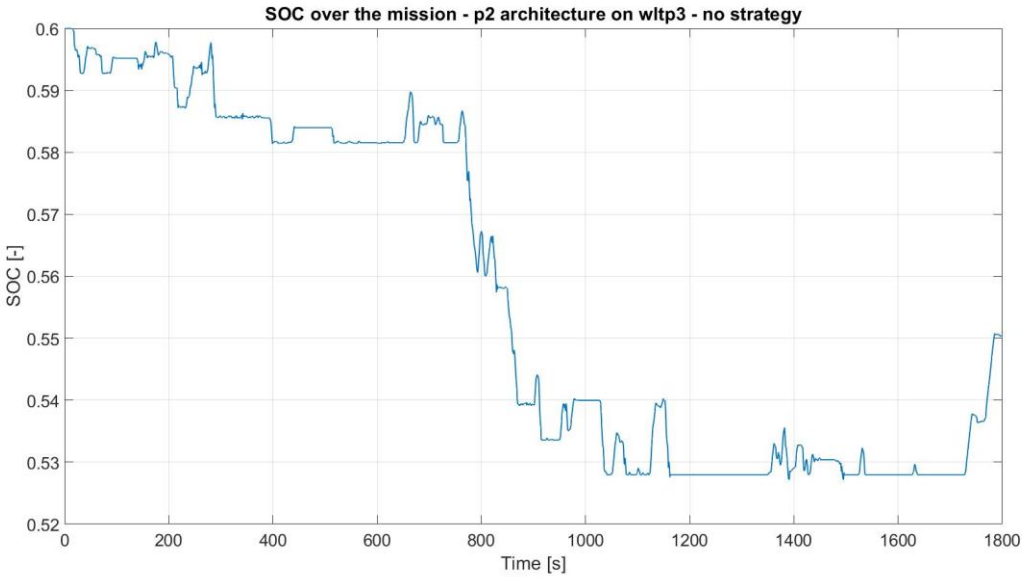

Figura 5.7 SOC per funzione di reward Prezzo

Infine, questa funzione presenta il problema relativo alla difficile definizione dei prezzi di carburante ed energia elettrica che sono in continuo cambiamento. Per tutte queste ragioni questa funzione non è poi stata utilizzata nel resto del lavoro, nonostante presenti caratteristiche interessanti.

5.3.3 Funzione Frazione FC

Un approccio completamente differente è stato utilizzato con un'altra funzione di tentativo, chiamata "Frazione FC". Questa funzione ha lo scopo principale di minimizzare il consumo di carburante, mantenendo però il SOC al di sopra di un certo valore definito dall'utente SOC_{lim}. Tale risultato è ottenuto utilizzando la seguente funzione di *reward*:

$$r(s,a) = \begin{cases} \dfrac{cost}{icemfc(t,a)} & se\ icemfc(t,a) \neq 0\ e\ SOC(s') > SOC_{lim} \\ \dfrac{cost}{icemfc(t,a) + penalità} & se\ icemfc(t,a) \neq 0\ e\ SOC(s') < SOC_{lim} \\ \dfrac{cost}{min(icemfc)} & se\ icemfc = 0 \end{cases}$$

Dove:

- $penalità$ e $cost$ sono costanti positive
- $\min(icemfc)$ indica il minimo valore di consumo di combustibile in un intervallo di tempo calcolato durante il ciclo studiato e contenuto quindi nella matrice delle configurazioni corrispondente alla grandezza $icemfc$

In questo caso i *reward* ottenuti sono quindi tutti positivi. Il termine $penalty$ serve a far diminuire il *reward* nel caso in cui il $SOC(s') < SOC_{lim}$. In questo modo, se $penalty$ viene posto a un valore sufficientemente elevato l'agente tenderà a scartare quelle azioni che portano a $SOC(s') < SOC_{lim}$. Regolando il valore SOC_{lim} e la penalty in modo appropriato è possibile ottenere risultati interessanti con questa funzione. Anche in questo caso, il *discounted return* ha un andamento crescente, sintomo di un allenamento avvenuto in modo efficace. Questa funzione presenta un problema che è però molto simile alla funzione Prezzo vista precedentemente: tendendo a privilegiare la riduzione di consumo di carburante, il valore del SOC tende ad abbassarsi e il SOC finale è molto vicino al SOC_{lim}. In Figura 5.8 viene riportato l'andamento del SOC avendo imposto il $SOC_{lim} = 0.45$ e $penality = 100$: come si può vedere il valore finale del SOC è vicino al SOC_{lim}.

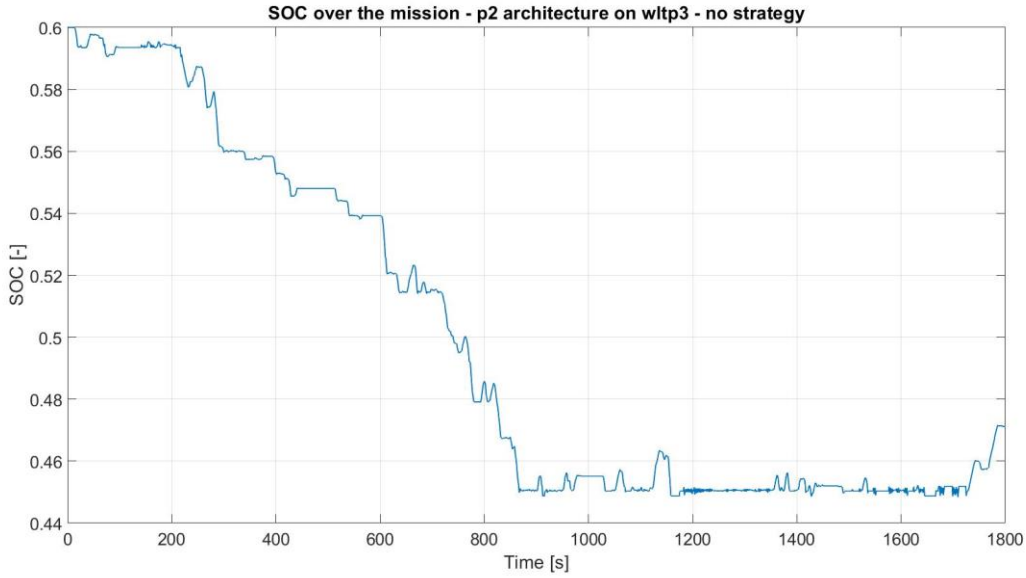

Figura 5.8 SOC per funzione di reward Frazione FC

Il grosso problema è che fissando invece un SOC_{lim} vicino al SOC a inizio ciclo non si riesce ad ottenere la minimizzazione del consumo di carburante voluta poiché l'agente tende a non portare il SOC a valori inferiori al SOC_{lim} imposto e dove aumentare il consumo di carburante. Nonostante tale funzione di *reward* abbia portato a risultati interessanti, il difficile settaggio dei parametri SOC_{lim} e *penalità*, ha portato a escludere questa soluzione.

5.3.4 Funzione Iperbole

Una funzione di *reward* che ha portato a ottimi risultati è la funzione "Iperbole". Questa funzione di *reward* riprende alcuni concetti visti per il controllo del solo SOC con la funzione parabola e li unisce alla minimizzazione del consumo di carburante. L'espressione della funzione di *reward* è la seguente:

$$r(s,a) = -k_1 \cdot icemfc(t,a) + \sqrt{cost - k_2 \cdot (SOC(s') - SOC_{iniziale})^2}$$

Dove:

- k_1, k_2 e $cost$ sono costanti positive

Questa funzione è composta da due addendi separati: mentre il termine $-cost_1 \cdot icemfc(t,a)$ è introdotto per garantire la minimizzazione del consumo di carburante, il secondo membro $\sqrt{cost_2 - k(SOC(s') - SOC_{iniziale})^2}$ permette di mantenere il SOC vicino al valore del SOC a inizio ciclo. Come precedentemente citato i due obiettivi sono implicitamente in contrasto tra di loro e la scelta dei termini costanti k_1, k_2 e $cost$ può influenzare molto i risultati finali. Se per esempio si aumenta il valore di k_1, si tende a favorire maggiormente la minimizzazione del FC, mentre se viene aumentato k_2 l'agente favorisce il controllo del SOC. Nella seguente figura sono riportati i valori dei *reward* ottenuti in una prova in cui sono stati posti $k_1 = -1000$, $cost = 400, k_2 = -5000$. I *reward* ottenuti seguono proprio l'andamento sottolineato precedentemente rispetto a SOC e $icemfc$.

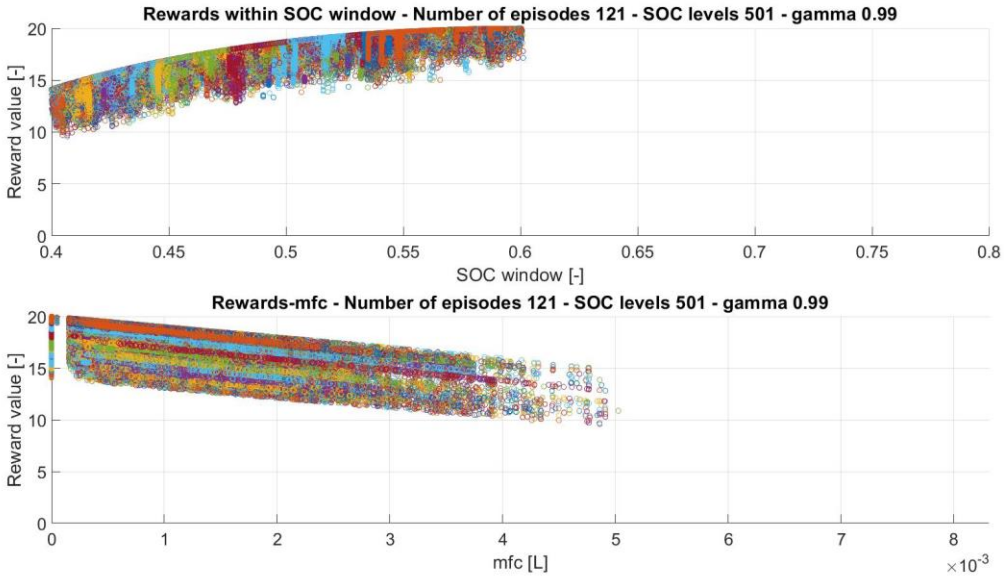

Figura 5.9 Reward ottenuti per funzione di reward a iperbole

Analizzando i risultati ottenuti con questa funzione, si può notare come l'andamento del *discounted return* dimostri un corretto apprendimento dell'agente:

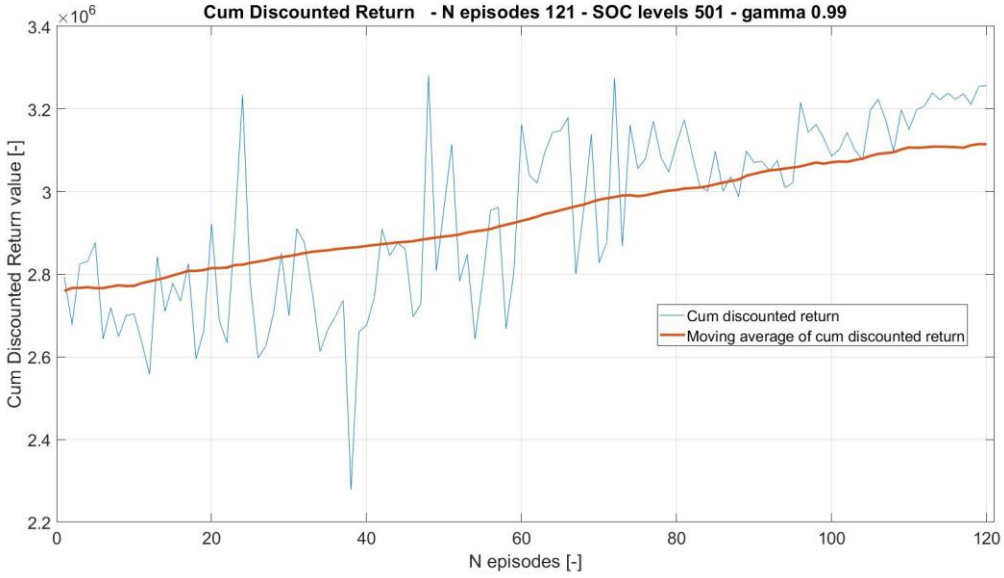

Figura 5.10 Discounted return per funzione di reward a iperbole

Guardando quindi il SOC finale si può vedere come è molto più vicino al SOC a inizio ciclo rispetto alle prove svolte con alcune delle funzioni di *reward* viste in precedenza:

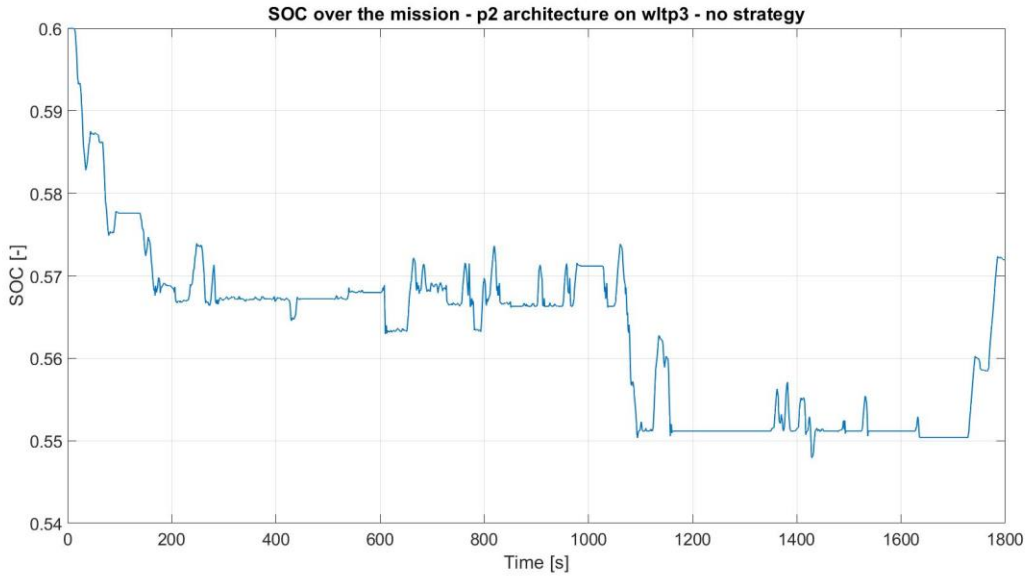

Figura 5.11 SOC per funzione di reward a iperbole

Questa funzione di *reward* si presenta quindi molto interessante: l'idea di avere due termini separati per il controllo di SOC e FC risulta efficiente. Tuttavia, con questa funzione risulta complessa la scelta delle costanti per definire quale obiettivo fisico si voglia privilegiare.

5.3.5 Funzione Parabola normalizzata

La funzione che ha permesso di ottenere i risultati migliori è certamente la funzione "Parabola normalizzata". Quest'ultima presenta diverse caratteristiche in comune con la funzione iperbole, ma garantisce una maggiore semplicità ed efficacia nel gestire l'importanza data al controllo del SOC rispetto alla minimizzazione dell'FC. Ecco in seguito la formula utilizzata:

$$r(s,a) = cost - amplifier \cdot (\beta \cdot icemfc_{norm}(t,a) + (1-\beta) \cdot \Delta SOC^2_{norm})$$

$$icemfc_{norm}(t,a) = \frac{icemfc(t,a)}{\max(icemfc)}$$

$$\Delta SOC^2_{norm} = \frac{(SOC(s') - SOC_{iniziale})^2}{(SOC_{max} - SOC_{iniziale})^2}$$

Dove:

- $\max(icemfc)$ è il massimo valore di consumo di carburante in un intervallo di tempo calcolato durante il ciclo studiato
- $cost$ e $amplifier$ sono due costanti positive
- β è un coefficiente di peso atto a determinare l'importanza attribuita al controllo del FC rispetto al controllo del SOC, $0 < \beta < 1$

Esattamente come la funzione iperbole, la funzione parabola normalizzata contiene due termini separati: il primo, $\beta \cdot icemfc_{norm}(t,a)$, permette di ottenere la minimizzazione dell'consumo di carburante, il secondo, $(1-\beta) \cdot \Delta SOC^2_{norm}$, garantisce il controllo del SOC. Si possono però notare diverse novità in questa funzione rispetto a quelle viste precedentemente. In primo luogo, è stata introdotta una normalizzazione dei termini legati al consumo di carburante $icemfc(t,a)$ e al SOC. Tale espediente risulta molto utile poiché le grandezze $icemfc(t,a)$ e $(SOC(s') - SOC_{initial})^2$ risultano molto differenti non solo dal punto di vista fisico, ma anche per quanto riguarda il loro ordine di grandezza. Per questa ragione viene introdotta la normalizzazione di tali quantità; le grandezze utilizzate $icemfc_{norm}(t,a)$ e ΔSOC^2_{norm} possono variare solo tra zero e uno:

$$0 < icemfc_{norm}(t, a) < 1$$

$$0 < \Delta SOC^2_{norm} < 1$$

In questo modo risulta più facile confrontarle all'interno della funzione di *reward*. L'altra novità di questa funzione è l'introduzione del coefficiente di peso $beta$ che permette di regolare l'importanza che si vuole dare alla minimizzazione dell'FC rispetto al controllo del SOC. Tale coefficiente può essere fatto variare tra zero e uno: se β aumenta, viene privilegiata la minimizzazione del FC, mentre se β diminuisce, viene privilegiato il controllo del SOC. Questa funzione ha portato a risultati molto positivi ed è stata per questo utilizzata nel resto del lavoro come funzione di *reward*. Nel prossimo capitolo verrà analizzata questa funzione mostrando i risultati ottenuti dal suo utilizzo.

6 Analisi della funzione Parabola normalizzata

Dopo le analisi svolte sulle diverse funzioni di *reward*, è stata scelta la funzione parabola normalizzata come funzione più adatta per ottenere gli obiettivi fisici desiderati. In questo capitolo verranno brevemente analizzati i risultati ottenuti nelle prove con un basso numero di episodi per poi passare alle prove con un numero di episodi maggiore. Verrà svolta un'analisi parametrica sul rapporto *exploration-exploitation* e sul *learning rate* α, individuando i migliori valori da utilizzare in funzione del coefficiente di peso β. In tutte le prove che seguiranno la finestra del SOC sarà mantenuta tra [0.4;0.8] con 501 livelli di discretizzazione. Inoltre, il *discount factor* γ è lasciato sempre pari a 0.99.

6.1 Prove con basso numero di episodi

Come prima fase dell'analisi sono state considerate prove di "debug" con un numero di episodi molto basso, pari a 120, atte a comprendere la potenziale efficacia della funzione di *reward* scelta. Una scelta opportuna in prove a così basso numero di episodi è sicuramente definire un fattore di apprendimento α elevato, quindi vicino all'unità. In questo modo i valori di Q contenuti nella *Q-table* cambiano in modo importante a ogni aggiornamento e si riescono a raggiungere a fine allenamento valori di Q più vicini a quelli di convergenza. Inoltre, si è dimostrato efficace l'utilizzo di un *discount factor* γ vicino a uno: è infatti utile in questo tipo di problemi avere un agente che tenga in grossa considerazione i *reward* degli istanti di tempo lontani da quello presente. Per queste ragioni per queste prove è stato scelto $\alpha = 0.9$ e $\gamma = 0.99$. Una volta determinati α e γ, si è proseguito a testare la reale efficacia del coefficiente di peso β. Chiaramente bisogna tenere presente che i risultati ottenuti non sono quelli ottimali a causa del numero basso di episodi, ma permettono comunque di comprendere l'influenza della variazione di β sui risultati finali. Sono state riportati nella Figura 6.1 i risultati di SOC e FC finali ottenuti da prove a β differenti. Le prove in considerazione hanno tutte $\alpha = 0.99$, $\gamma = 0.9$, $cost = 2$ e il rapporto tra *exploration* ed *exploitation* è determinato dalla strategia di variazione di ε chiamata nella sottosezione 4.4.2 come costante+parabola. L'unico parametro che varia tra una prova e l'altra è proprio β che assume i seguenti valori: $\beta = 0.1, \beta = 0.3, \beta = 0.5, \beta = 0.7$ *e* $\beta = 0.9$. Si può osservare come l'effetto del β sia quello sperato: più β aumenta più la riduzione di FC è maggiore e il SOC finale minore. Tuttavia, essendo prove con un numero basso di episodi, vi sono eccezioni a quanto detto. Per $\beta = 0.3$, per esempio, l'FC e il SOC finale sono maggiori rispetto alla prova con $\beta = 0.1$ e il SOC finale della prova con $\beta = 0.9$ è maggiore di quello della prova a $\beta = 0.7$. Aumentando il numero di episodi come verrà fatto nei capitoli successivi, si otterranno risultati migliori da questo punto di vista. Ad ogni modo, dalle prove di debug si conclude che la funzione di *reward* scelta soddisfa i requisiti definiti precedentemente.

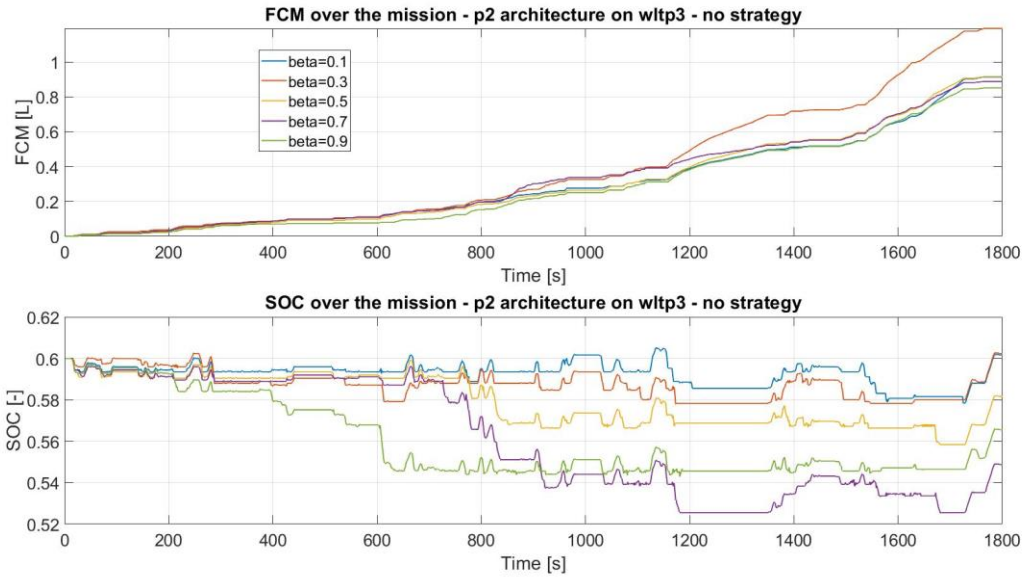

Figura 6.1 Confronto tra prove di "debug" con β diversi

Da questo momento in poi, in questo capitolo 6 verranno analizzate solamente prove complete cioè con un numero di episodi pari a 2300. Il numero di episodi è stato calcolato a partire dal numero di stati N_{stati} e dal numero di intervalli di tempo N_{int} in cui è suddiviso il ciclo di guida: più queste quantità sono elevate, più il tempo necessario per allenare l'agente è elevato. La formula utilizzata per trovare il numero di episodi completi è:

$$N_{episodi} = a \cdot N_{stati} + b \cdot N_{int}$$

Dove a e b sono posti pari a 1 per ottenere il numero di episodi per una prova completa, mentre sono stati posti uguali a 0.05 per le prove di debug.

6.2 Analisi parametriche al variare di β

Le analisi parametriche svolte in questa sezione hanno lo scopo di fornire delle indicazioni riguardo il rapporto ottimale tra *exploration- exploitation* e il *learning rate* ottimale per ogni β. Variando infatti il β, il problema di *Q-learning* cambia radicalmente e i risultati ottenuti sono completamente differenti. Sarebbe pertanto un errore lasciare a priori uguali i parametri ε e α per prove a β diverso. Inoltre, sono anche riportate considerazioni sul parametro $cost$. Di conseguenza, vengono analizzati tre valori di β molto differenti su cui svolgere queste analisi parametriche: ricordando che β può variare tra zero e uno, sono stati scelti $\beta = 0.1, \beta = 0.5, \beta = 0.9$.

6.2.1 β =0.1

Considerando $\beta = 0.1$, si predilige in modo marcato il controllo del SOC rispetto alla minimizzazione del FC. Riprendendo infatti la funzione di *reward* $rcost - amplifier \cdot (\beta \cdot icemfc_{norm}(t,a) + (1-\beta) \cdot \Delta SOC^2_{norm})$, si può notare come ponendo $\beta = 0.1$ il termine legato al FC venga notevolmente ridotto, mentre il termine legato al SOC cresca. Ciò è dimostrato dai seguenti grafici rappresentativi dei *reward* ottenuti durante una prova completa con $\beta = 0.1$: osservando il subplot superiore della Figura 6.2, si nota come allontanandosi dal valore di SOC iniziale (SOC=0.6), il *reward* diminuisca drasticamente fino ad arrivare al valore minimo per un SOC pari al limite inferiore della finestra (SOC=0.4). Considerando un singolo valore del SOC si vede che il range entro cui variano i *reward* ottenuti per quel SOC è piccolissimo se confrontato con l'ampiezza di tali range per β maggiori: l'influenza del consumo di carburante è in questo caso minima sui *reward* ottenuti. Le stesse conclusioni

si possono trarre osservando il secondo subplot che mette in relazione i *reward* ottenuti durante la prova, con il consumo di combustibile in un intervallo di tempo.

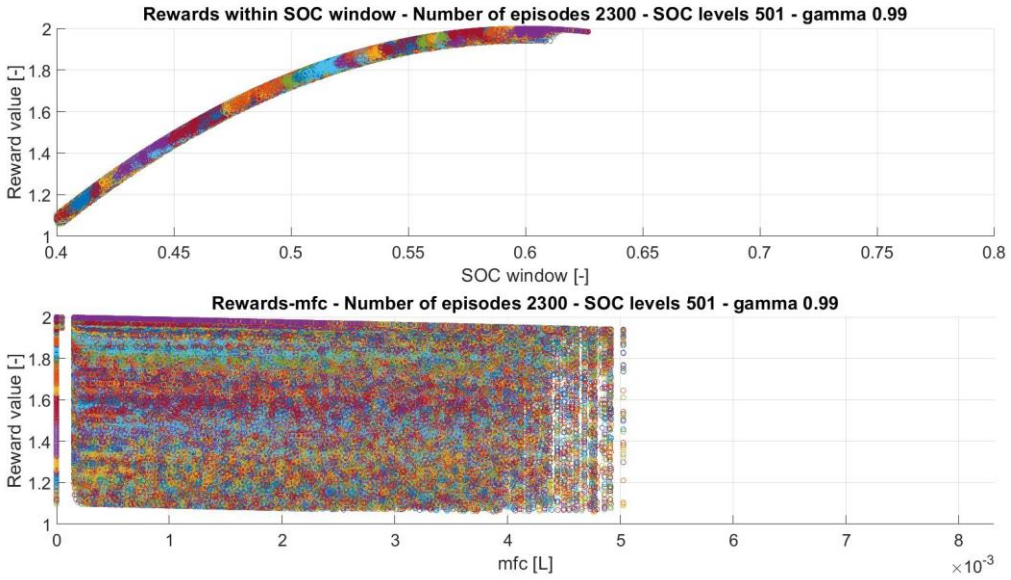

Figura 6.2 Reward ottenuti con β=0.1

Un altro grafico molto utile per studiare la dipendenza del *reward* dal valore di SOC e mfc è il seguente contour. In questa rappresentazione non sono riportati i *reward* effettivamente ottenuti durante la prova, bensì la funzione di *reward* è stata plottata rispetto al SOC e all'mfc per tutto il range di valori di SOC e mfc possibili. La scala dei colori si riferisce proprio al valore assunto dal *reward*.

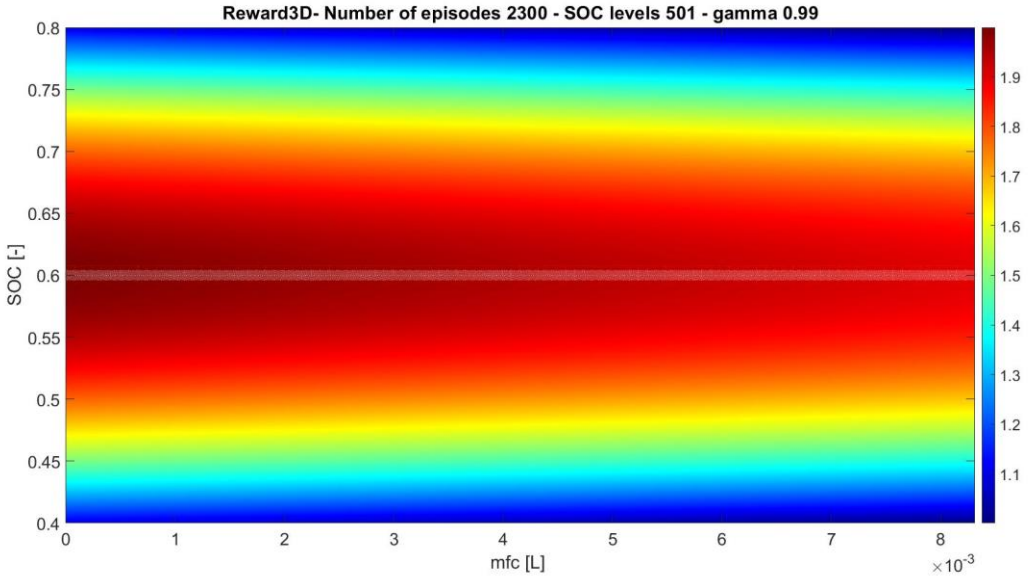

Figura 6.3 Contour della funzione di reward per β=0.1

Passando invece allo studio dell'allenamento dell'agente con l'utilizzo di questo valore di β, si può affermare che in generale vi sono meno problemi con l'utilizzo di β bassi piuttosto che alti. Infatti, privilegiando il controllo del SOC il numero di episodi esclusi si mantiene basso. Per episodi esclusi si

intendono quegli episodi che non arrivano a fine ciclo a causa del superamento dei limiti imposti al SOC. Avere un numero molto elevato di episodi esclusi, influisce negativamente sull'allenamento, poiché l'agente ha raramente sperimentato episodi effettivamente buoni in cui si riesce ad arrivare a fine ciclo. Con un β=0.1 questo problema è completamente scongiurato. Osservando ora il grafico del *discounted return* riportato in Figura 6.4, si nota un andamento crescente e la convergenza raggiunta negli episodi finali, indicatori tipici di un ottimo allenamento dell'agente. In questo particolare esempio, gli unici episodi esclusi presenti sono quelli in cui il livello di esplorazione è massimo, con $\varepsilon = 1$.

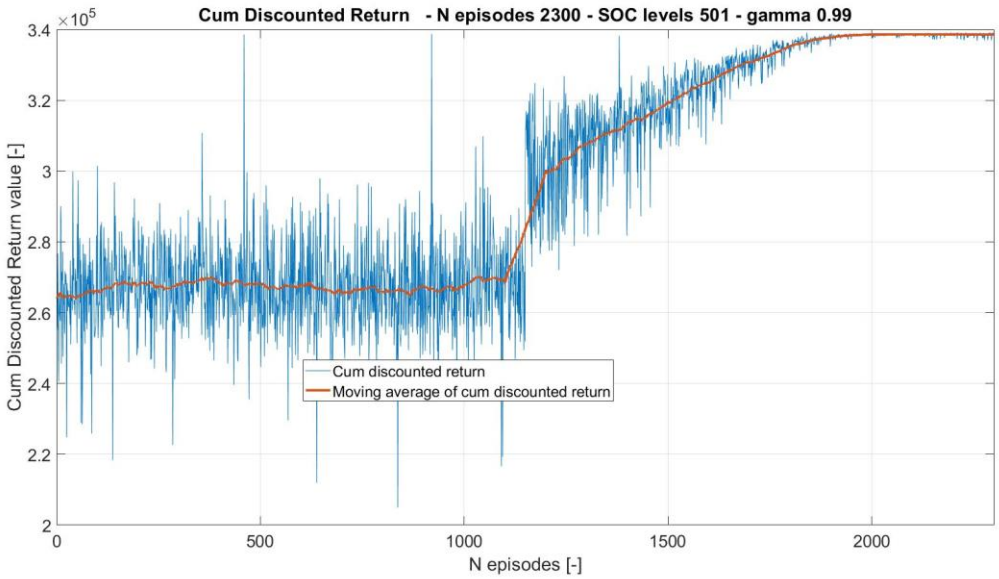

Figura 6.4 Discounted return per β=0.1

Inoltre, osservando la *Q-table* a fine allenamento si può notare una forma particolare: essa è molto simile alla forma parabolica assunta dal *reward* rispetto al SOC. Ciò è ragionevole in quanto più il SOC è vicino al limite inferiore della finestra di SOC, più il *reward* ottenuto risulta basso e di conseguenza il valore della *Q(s,a)* corrispondente è minore. Inoltre, si nota che per valori di SOC vicini al limite superiore della finestra di SOC i valori delle *Q* sono nulli: questo perché non vengono mai raggiunti valori così elevati di SOC in nessun episodio di allenamento.

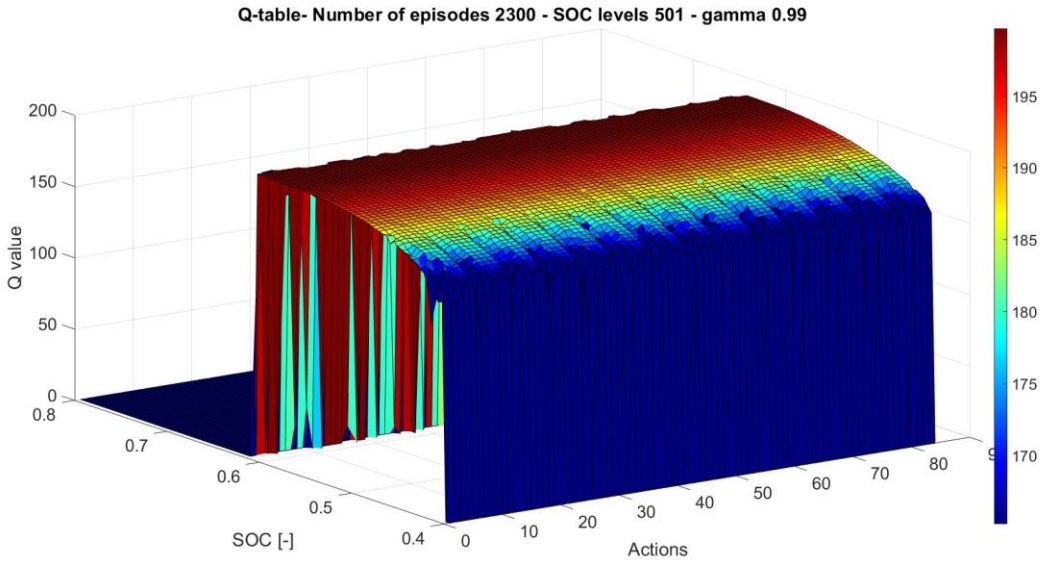

Figura 6.5 Q-table per β=0.1

Si passa ora invece all'analisi parametrica legata al rapporto *exploration- exploitation* per β=0.1. Si è provato quindi ad applicare le due strategie di decremento della ε viste alla sottosezione 4.4.2, osservando i differenti risultati ottenuti. I parametri comuni per le prove sono $\alpha = 0.9$ e $\gamma = 0.99$. Nelle tre prove si è cambiata solo la legge di variazione di ε: nella prima si è imposta la legge lineare con $\varepsilon_{episodio_{finale}} = 0.1$ e $\varepsilon_{iniziale} = 0.9$, nella seconda e nella terza la legge costante+parabola con $N_{episode_{eps}} = {N_{episodio_{finale}}}/{6}$ nella seconda e $N_{episode_{eps}} = {N_{episodio_{finale}}}/{2}$ nella terza.

In queste prove siccome il β è molto basso, la scelta di tale parametro influisce relativamente poco sui risultati finali dell'allenamento. Confrontando infatti i valori di SOC e FC in Figura 6.6 ottenuti durante gli episodi finali delle prove, non si riesce a distinguere quale tra esse fornisca i risultati migliori.

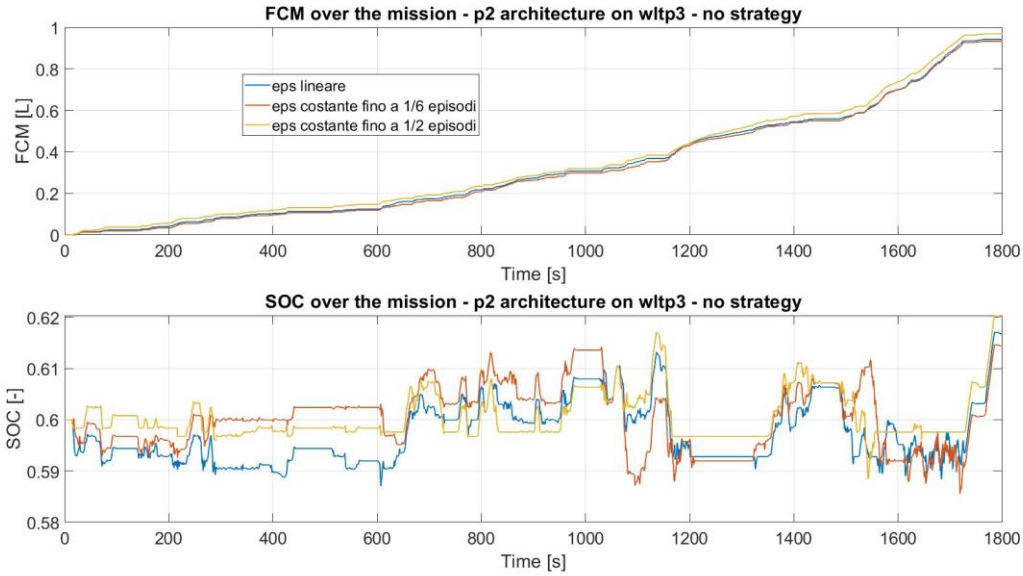

Figura 6.6 FC e SOC per β=0.1 al variare del rapporto exploration-exploitation

Di conseguenza, l'unico modo per valutare quale sia la prova migliore dal punto di vista dei risultati finali dell'allenamento è prendere in considerazione la somma dei *reward* o il *discounted return* ottenuti nell'episodio finale. Per semplicità viene presa in considerazione la somma dei *reward*, riportata nella tabella seguente. Nonostante la differenza sia minima, la migliore risulta essere l'ultima prova considerata.

Tabella 6.1 Somma dei reward per prove con β=0.9

Prova considerata	Somma dei *reward*
Eps lineare	3584.9
Eps=1 fino a 1/6	3585.2
Eps=1 fino a 1/2	3585.45

Tuttavia, la differenza più evidente che si può vedere è che la convergenza, per quanto riguarda il valore del *discounted return*, avviene a un numero di episodi minore quando l'esplorazione è minore come si può notare nella Figura 6.8, dove sono confrontate le prove con minore e maggiore esplorazione. Nei problemi di RL se si riesce a interrompere l'allenamento non appena si presenta la convergenza, migliorano i risultati sia in termine del tempo necessario per completare l'allenamento, sia per quanto riguarda la somma dei *reward* ottenuti. Se infatti continua l'allenamento dell'agente dopo che è stata raggiunta la convergenza del *discounted return*, i risultati possono peggiorare notevolmente. Nel capitolo 7 verrà mostrata l'implementazione di una funzione per garantire l'interruzione dell'allenamento prima di arrivare all'episodio stabilito come ultimo episodio prima di iniziare l'allenamento.

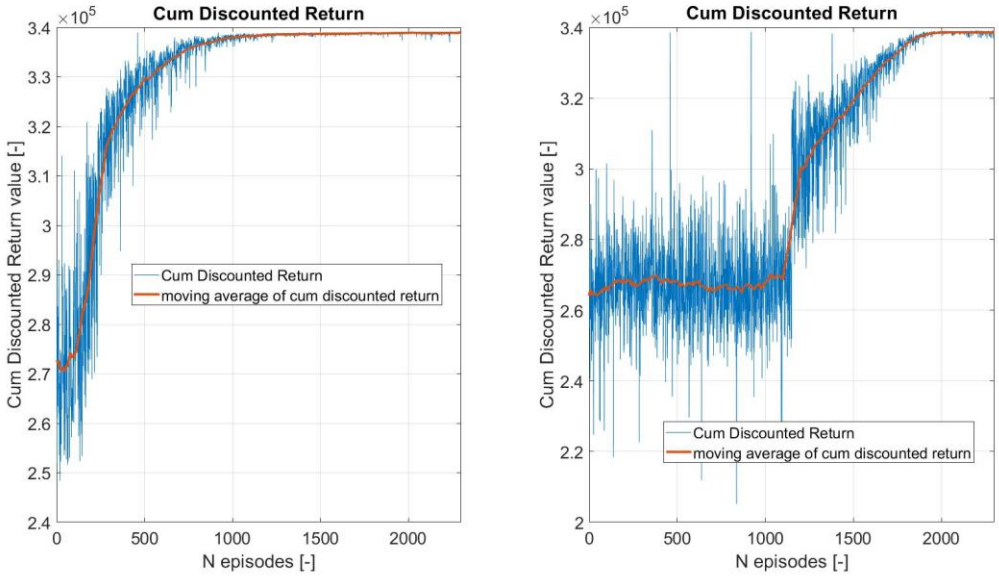

Figura 6.7 Confronto dei discounted return per due prove a differenti livelli di esplorazione

6.2.2 β =0.5

Ora si passa ad analizzare le prove con un β=0.5. Utilizzando questo coefficiente di peso si ottengono risultati intermedi rispetto alle prove con gli altri due β studiati. I termini legati al SOC e all'FC hanno lo stesso peso poiché $\beta = (1 - \beta)$. Analizzando quindi i *reward* effettivamente ottenuti durante una prova con questo coefficiente di peso, si possono notare due grosse differenze rispetto ai *reward* ottenuti con $\beta = 0.1$: a parità di consumo di carburante in un intervallo di tempo (mfc), i *reward* ottenuti hanno un range di variazione pari a 0.5, molto inferiore rispetto al *reward* analizzato prima; al contrario, a pari valore di SOC si può vedere che la variazione di *reward* è molto più elevata rispetto al $\beta = 0.1$. Ciò

conferma che il peso attribuito ai due obiettivi fisici desiderati è effettivamente cambiato. Osservando con attenzione il grafico in Figura 6.8 si può tuttavia notare un piccolo dettaglio: mentre la massima variazione del SOC comporta a una diminuzione del *reward* di 0.5 a pari mfc, la massima variazione del mfc a pari SOC porta a una variazione del *reward* di 0.3. Ci si sarebbe potuto aspettare che, avendo posto $beta = 0.5$, le due variazioni di *reward* dovessero essere uguali: in realtà ciò non avviene poiché in nessuno degli episodi viene mai raggiunto il $\max(icemfc)$ con cui viene normalizzato il valore di $icemfc(t,a)$. Viene comunque utilizzato il valore $\max(icemfc)$ per normalizzare l'$icemfc(t,a)$ al fine di mantenere il carattere generale della funzione di *reward* scelta.

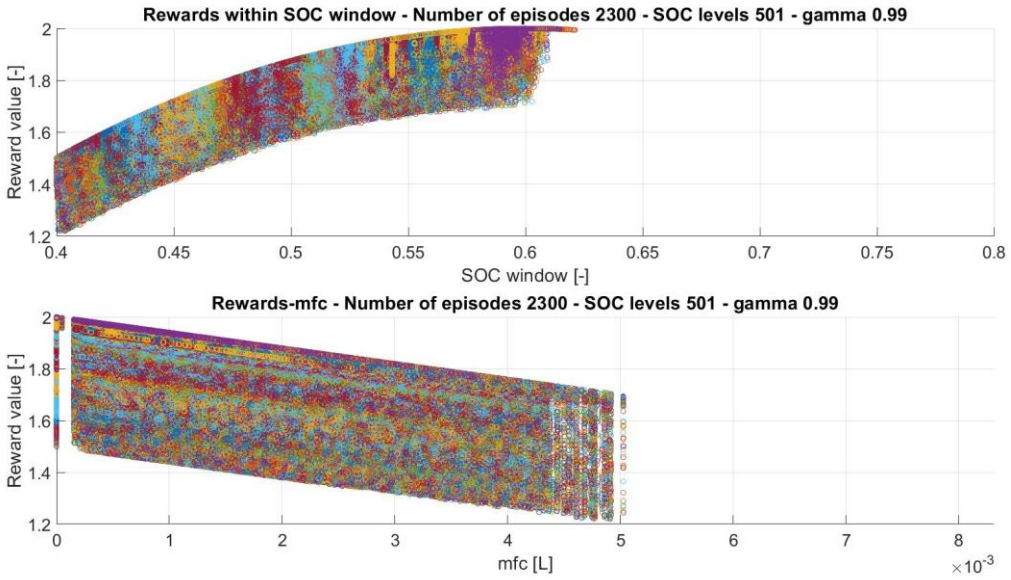

Figura 6.8 Reward ottenuti con β=0.5

Altro grafico interessante, attraverso cui si possono trarre le stesse conclusioni, è il contour della funzione di reward rispetto al SOC e mfc, rappresentato in Figura 6.9.

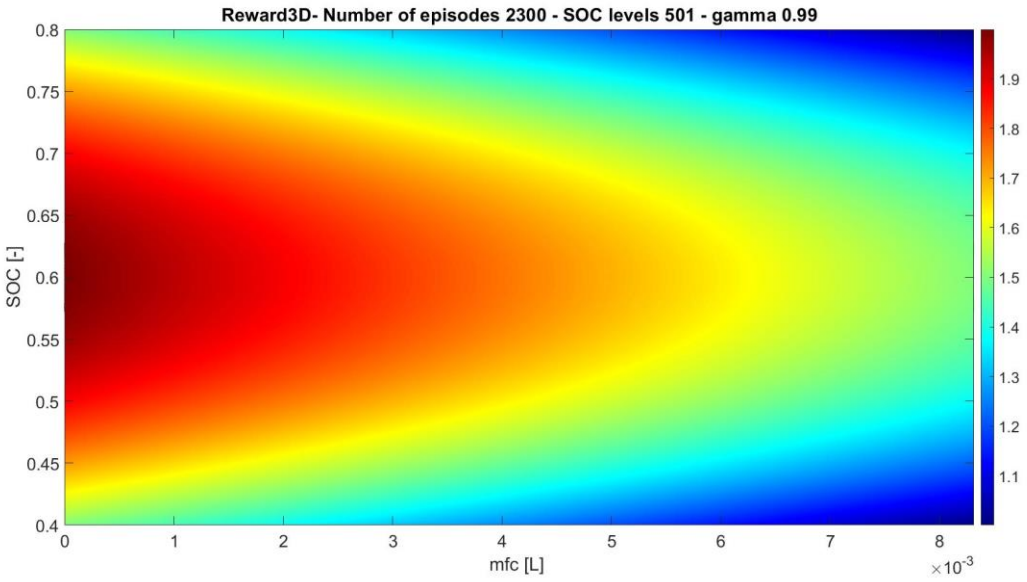

Figura 6.9 Contour della funzione di reward per β=0.5

Anche con l'utilizzo di questo coefficiente di peso, l'allenamento dell'agente non presenta problemi. Ciò è dimostrato in Figura 6.10 dall'andamento crescente del *discounted return*, dopo gli episodi di pura esplorazione. Inoltre, intorno all'episodio 2000 è raggiunta la convergenza di tale grandezza.

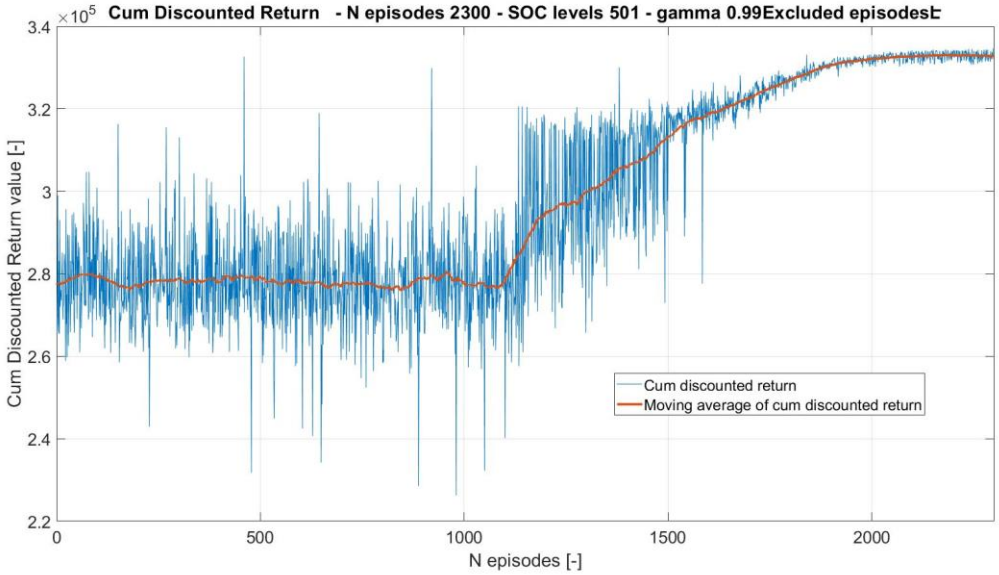

Figura 6.10 Discounted return per β=0.5

Osservando la *Q-table* si possono vedere le differenze previste rispetto a quella precedente: essa risulta più piatta, cioè la variazione dei valori *Q* riferiti a una stessa azione è minore rispetto a quella vista precedentemente. Mentre per *β* differenti il valore massimo delle *Q(s,a)* coincide, il valore minimo diverso da zero delle *Q(s,a)* risulta essere più elevato per *β*=0.5. La forma della *Q-table* è nuovamente simile alla forma dei *reward* ottenuti rispetto al SOC.

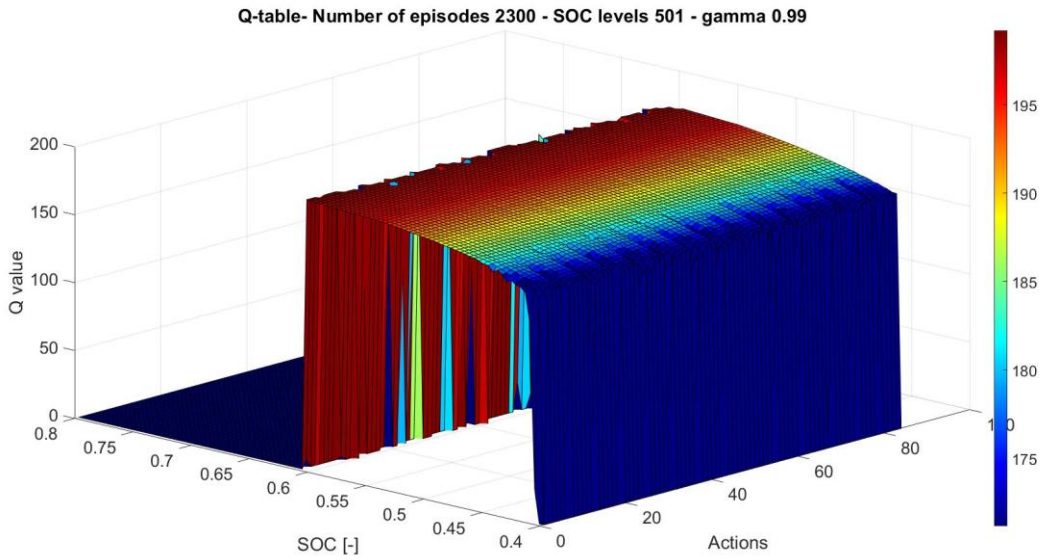

Figura 6.11 Q-table per β=0.5

Si passa ora all'analisi sull'influenza del rapporto *exploration-exploitation* sui risultati finali ottenuti. Esattamente come prima sono state effettuate tre prove con diversi livelli di esplorazione. I parametri comuni sono $\alpha = 0.9$ e $\gamma = 0.99$ e si è solo cambiata la legge di variazione di ε: nella prima prova si è imposta la legge lineare con $\varepsilon_{episodio_{finale}} = 0.1$ e $\varepsilon_{iniziale} = 0.9$, nella seconda e nella terza la legge costante+parabola con $N_{episodio_{eps}} = {}^{N_{episodio_{finale}}}/_{6}$ nella seconda, $N_{episodio_{eps}} = {}^{N_{episodio_{finale}}}/_{2}$ nella terza. In questo caso, i risultati ottenuti sono stati molto diversi dai precedenti: la variazione di ε ha influito maggiormente sui risultati finali che risultano palesemente diversi. Senza confrontare la somma dei *reward* ottenuti è comunque complesso determinare ad occhio quale sia la prova migliore. Se è infatti vero che la prova a esplorazione maggiore presenta un SOC finale di molto maggiore rispetto alle altre due prove, essa allo stesso tempo porta a un consumo di carburante più elevato.

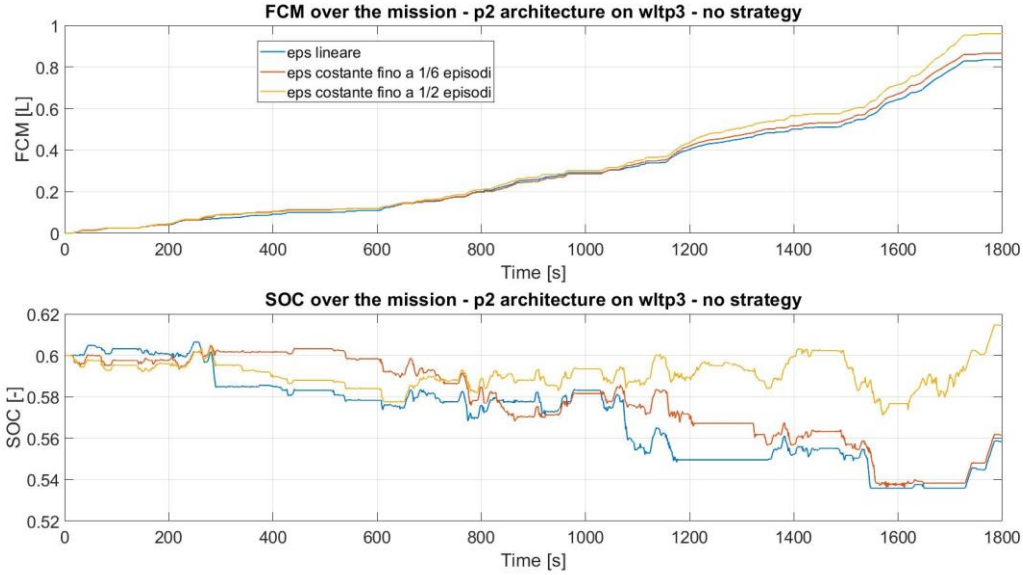

Figura 6.12 FC e SOC per β=0.5 al variare del rapporto exploration-exploitation

L'unico modo per confrontare le prove risulta quindi paragonare le somme dei *reward* ottenuti durante l'episodio finale di ciascuna prova, riportate nella tabella seguente. In questo modo si nota come l'allenamento dell'agente nell'ultima prova abbia garantito un allenamento più efficiente. Ciò è dovuto al fenomeno precedentemente citato per il β=0.5: il raggiungimento troppo precoce della convergenza del *discounted return* per prove a esplorazione minore porta a risultati finali peggiori. In questo caso risulta infatti più evidente questo fenomeno negativo. Come si può notare, il segnale di *discounted return* per la prova con ε che varia linearmente diventa sempre più rumoroso, cioè aumenta la varianza relativa ad esso, dopo che è stata raggiunta la convergenza.

Tabella 6.2 Somma dei reward per prove con beta=0.5

Prova considerata	Somma dei *reward*
Eps lineare	3520
Eps=1 fino a 1/6	3526
Eps=1 fino a 1/2	3537

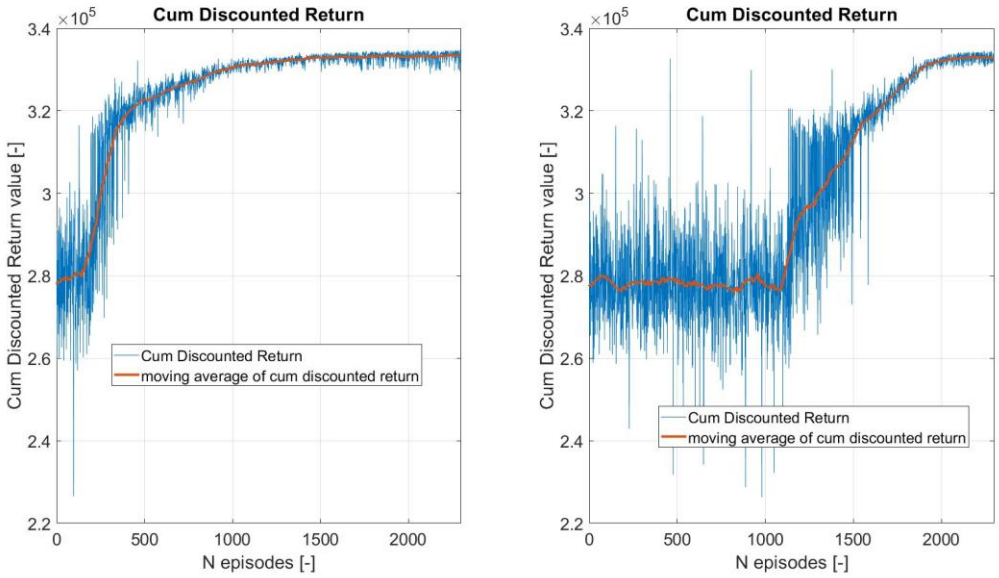

Figura 6.13 Confronto tra discounted return per prove con differenti livelli di esplorazione

Per β=0.5 si è inoltre provato a utilizzare anche un α decrescente con il numero di episodi. Tale soluzione non è risultata particolarmente efficace per questo β, poiché porta alla convergenza per un numero di episodi inferiore, con conseguente lieve peggioramento dei risultati. L'utilizzo di α decrescente verrà di conseguenza discusso più approfonditamente nella sottosezione successiva, dove ha portato a ottimi risultati per β=0.9.

6.2.3 β=0.9

Utilizzando un coefficiente di peso pari a 0.9, viene privilegiata principalmente la minimizzazione del consumo di carburante. Analizzando infatti i *reward* ottenuti in una prova con β=0.9, si può notare, infatti, come a parità di SOC il range di variazione del *reward*, pari a 0.55, sia molto più elevato rispetto alle prove con β inferiori; al contrario il range di variazione del *reward* a pari mfc è basso, pari a 0.1. Ciò conferma che il peso attribuito al FC è aumentato. Le stesse considerazioni si possono fare guardando il contour della funzione di *reward* rispetto a SOC e mfc.

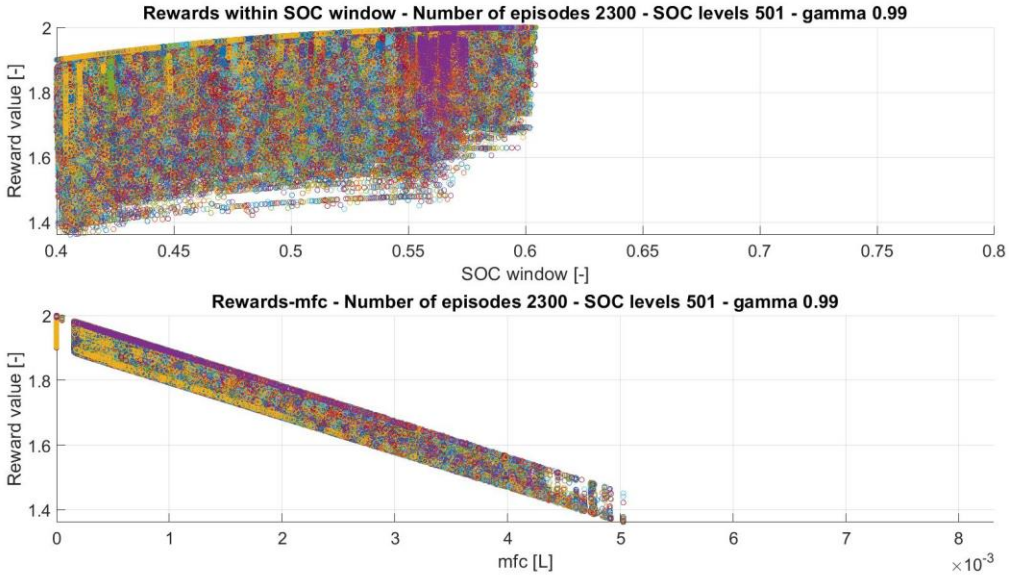

Figura 6.14 Reward ottenuti con β=0.9

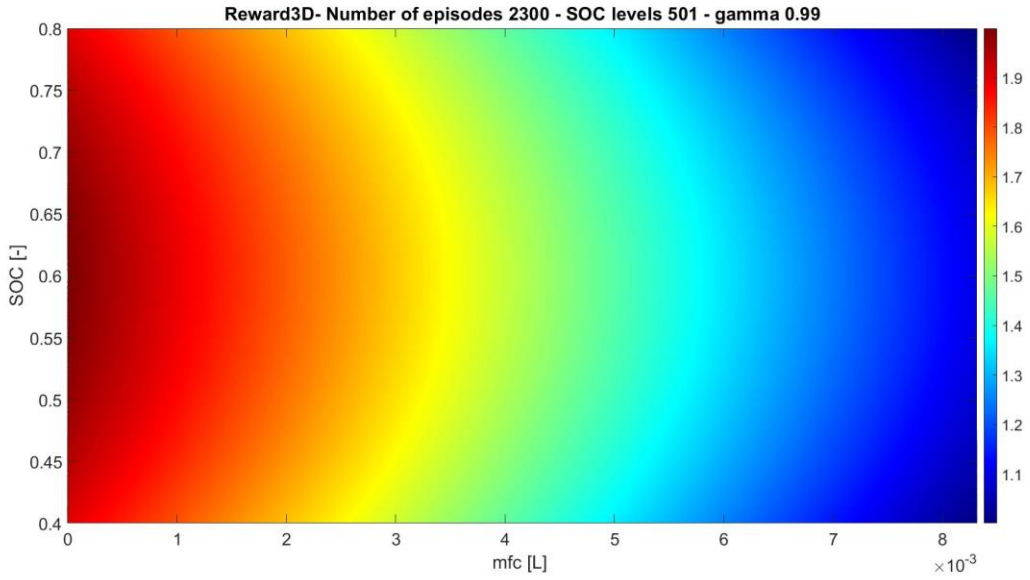

Figura 6.15 Contour della funzione di reward per β=0.5

In questo caso però l'allenamento è risultato molto più complesso da gestire, a causa dell'elevato numero di episodi esclusi. Privilegiando la minimizzazione del FC, infatti, l'agente ricorre più frequentemente all'utilizzo del motore elettrico rispetto a quello a combustione interna in fase di trazione: siccome ogni volta che è utilizzato l'EM per la trazione il valore del SOC scende, è più probabile che venga raggiunto il limite inferiore della finestra di SOC considerata. Un numero molto elevato di episodi esclusi influisce negativamente sull'allenamento dell'agente in quanto esso ha pochi episodi andati a buon fine su cui basare il proprio apprendimento. Nella Figura 6.16 viene mostrato l'andamento del *discounted return* per la migliore di queste prove. Si nota come l'andamento del *discounted return* sia molto differente rispetto a quello delle prove con β minore: durante gli episodi iniziali si evidenzia addirittura una diminuzione del *discounted return* e durante tutto l'allenamento esso mantiene una varianza molto elevata. La diminuzione del *discounted return* nei primi episodi è dovuta al fatto che l'agente tende a privilegiare le azioni a maggior consumo di SOC, portando alla conclusione dell'episodio prima dell'istante di tempo finale: a inizio allenamento il *reward* che si ottiene nell'istante di tempo presente ha molta più importanza dei *reward* futuri poiché i valori delle *Q(s,a)* sono ancora bassi. Solo andando avanti nell'allenamento l'agente comprende che può riuscire a ottenere un *discounted return* maggiore se riesce a concludere con successo l'episodio. L'alta varianza del *discounted return* durante tutto l'allenamento è dovuta proprio all'elevato numero di episodi esclusi che sono presenti anche per valori bassi di esplorazione.

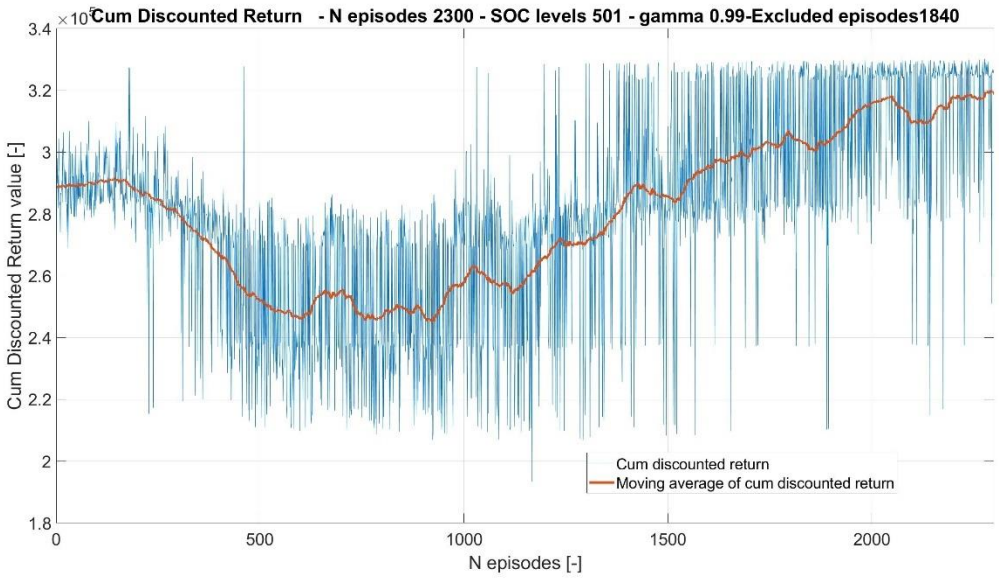

Figura 6.16 Discounted return per β=0.9

La *Q-table* ottenuta presenta una forma estremamente più piatta rispetto alle due viste in precedenza. Ciò è dovuto alla minore dipendenza della *Q(s,a)* dallo stato SOC dell'ambiente.

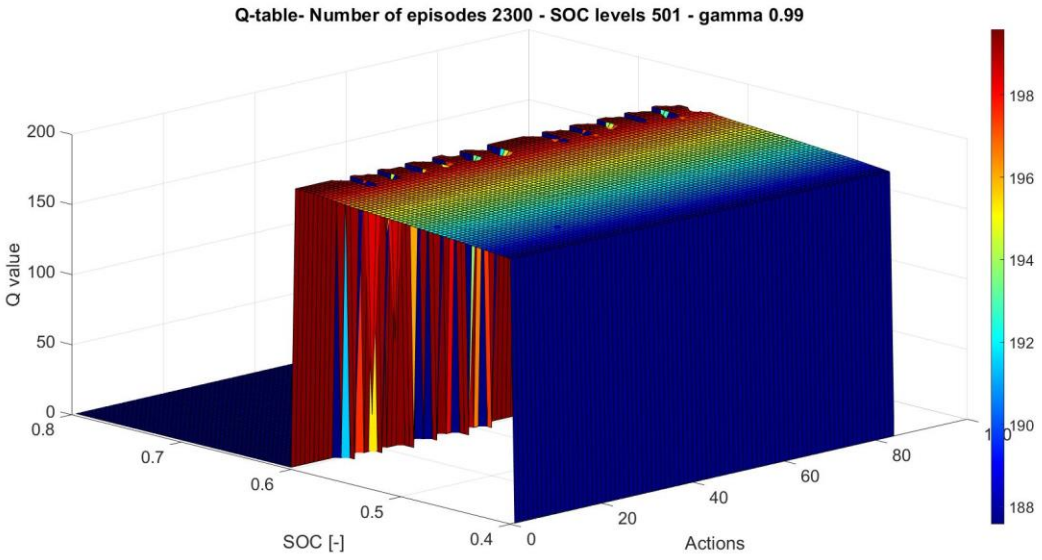

Figura 6.17 Q-table per β=0.9

Essendo l'allenamento così complicato, è importante scegliere con attenzione il rapporto *exploration-exploitation* che si vuole utilizzare. Infatti, scegliendo livelli alti di esplorazione, il numero di episodi esclusi tende ad aumentare ancora, facendo peggiorare ulteriormente l'allenamento. Per l'analisi sono state considerate prove con leggi di variazioni di ε uguali a quelle utilizzate con gli altri β. In questo caso i risultati migliori sono stati però ottenuti utilizzando un livello di esplorazione più basso, come prevedibile. Vengono quindi riportati in Figura 6.18 gli andamenti di FC e SOC ottenuti dalle prove considerate. Anche in questo caso è molto difficile riconoscere quale sia la prova migliore guardando solo queste grandezze fisiche. Sembrerebbe anzi che le prove a esplorazione maggiore portino a risultati migliori in quanto il consumo di carburante risulta minore.

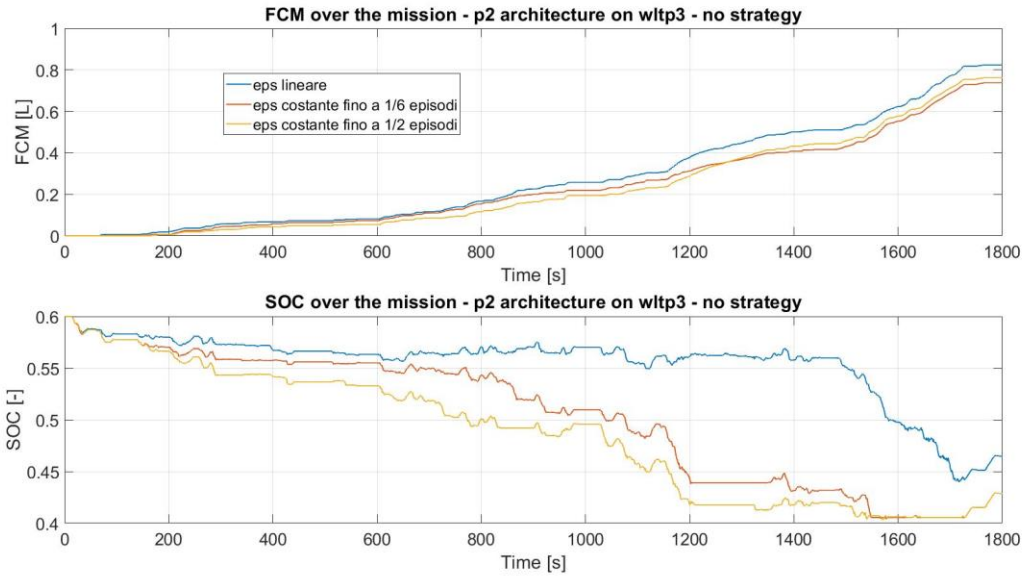

Figura 6.18 FC e SOC per β=0.9 al variare del rapporto exploration-exploitation

Tuttavia, osservando la somma dei *reward* ottenuti per ogni prova, si nota che il miglioramento nel SOC compensa i consumi di carburante maggiori: con la prova a esplorazione minore si ottiene un allenamento migliore e un numero minore di episodi esclusi, come si osserva nella Tabella 6.3.

Tabella 6.3 Somma dei reward e numero di episodi esclusi per prove con β =0.9

Prova considerata	Somma dei *reward*	Numero episodi esclusi
Eps lineare	3493	1909
Eps=1 fino a 1/6	3460	2076
Eps=1 fino a 1/2	3440	2126

6.2.3.1 Learning rate decrescente con il numero di episodi

Una soluzione molto interessante è stata quella di utilizzare *learning rate* α decrescente con il numero di episodi. Mentre per le prove con β inferiori né l'utilizzo di α costanti e minori di 0.9 né di α decrescenti con il numero di episodi sono stati vantaggiosi, per le prove con β=0.9 ciò ha portato a un miglioramento dei risultati. In generale, nei problemi di RL può essere molto utile utilizzare un *learning rate* decrescente per diverse ragioni. Ciò permette, per esempio, di raggiungere più velocemente la convergenza del valore dei *Q(s,a)* contenuti nella *Q-table*. Sicuramente è molto utile mantenere un valore di α elevato per i primi episodi di allenamento poiché i valori *Q(s,a)* sono molto bassi e ancora molto lontani dal valore di convergenza: imponendo un α alto la variazione dei *Q(s,a)* in ogni istante temporale sarà maggiore e *Q(s,a)* si avvicineranno più velocemente alla convergenza. Al contrario negli episodi finali, dove i valori di *Q* sono già vicini al valore di convergenza, mantenere un α elevato può portare a variazioni troppo grosse dei valori di *Q*, che quindi oscillano intorno al valore ottimale senza raggiungerlo; utilizzando invece α basso la variazione delle *Q* sarà piccola e potranno arrivare almeno teoricamente più vicino al valore ottimale con meno oscillazioni. Questa soluzione permette di far arrivare a convergenza l'algoritmo a un numero di episodi inferiore rispetto a un α costante. Per questa ragione, l'utilizzo di α decrescente risulta ottimale per le prove a β=0.9 dove si fatica arrivare a convergenza. È stata applicata anche per le prove a β inferiori, ma non sono stati ottenuti risultati migliori: come detto in precedenza in precedenza, il raggiungimento troppo precoce della convergenza fa solitamente peggiorare i risultati. Di conseguenza, vengono qua analizzate solo le prove con β=0.9. Sono state provate due leggi di variazione di α in funzione del numero dell'episodio. Per entrambe le leggi si cerca di mantenere α elevato per gli episodi iniziali per poi diminuirlo negli episodi finali. Come prima legge è stata utilizzata una funzione prima costante e poi lineare con la seguente formula:

$$\alpha = \begin{cases} \alpha_{iniziale} \ \ se \ N_{episodio} < N_{episodio_{retta}} \\ \dfrac{N_{episodio} - N_{episodio_{finale}}}{N_{episodio_{retta}} - N_{episodio_{finale}}} \cdot \left(\alpha_{iniziale} - \alpha_{finale}\right) + \alpha_{finale} \ se \ N_{episodio} > N_{episodio_{retta}} \end{cases}$$

Dove:

- $\alpha_{iniziale}$ e α_{finale} sono i valori di α a inizio e fine allenamento
- $N_{episodio}$ è il numero dell'episodio per cui si calcola α
- $N_{episodio_{retta}}$ è il numero dell'episodio in cui inizia la diminuzione di α

Con l'utilizzo di questa legge di α si sono ottenuti buoni risultati, ma vi sono diversi parametri da settare: $\alpha_{iniziale}$, α_{finale} e $N_{episodio_{retta}}$. Per questa ragione è stata sviluppata una seconda legge parabolica di α che permette l'utilizzo di un parametro in meno ($N_{episodio_{retta}}$) :

$$\alpha = \frac{\alpha_{finale} - \alpha_{iniziale}}{N_{episodio}^{\ 2}_{finale}} \cdot N_{episodio}^2 + \alpha_{iniziale}$$

Per la sua maggiore semplicità, è stata quindi usata questa seconda legge. In Figura 6.19 è riportato l'andamento di α rispetto al numero dell'episodio corrispondente, con $\alpha_{iniziale} = 0{,}9, \alpha_{finale} = 0.1$.

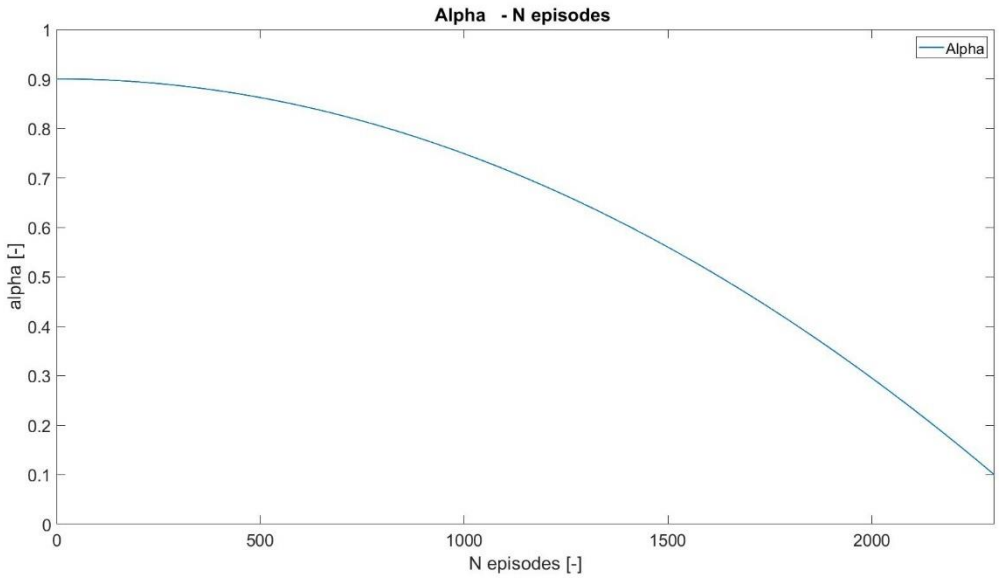

Figura 6.19 Learning rate decrescente rispetto al numero dell'episodio di allenamento

Ora viene quindi applicata questa legge di variazione del *learning rate* per una prova a β=0.9. In Figura 6.20 viene quindi riportato il confronto tra gli andamenti di SOC e FC per una prova a $\alpha = 0.9$ costante e un'altra con α decrescente utilizzando i parametri $\alpha_{iniziale} = 0.9$, $\alpha_{finale} = 0.1$; gli altri parametri risultano uguali per le due prove. Si nota che la prova con α decrescente fornisce i risultati migliori: il SOC finale risulta maggiore e l'FC minore rispetto a quelli ottenuti con la prova ad α costante. Ciò è confermato anche dalla somma più elevata dei *reward* ottenuti e da un numero di episodi esclusi minore come riportato nella tabella seguente.

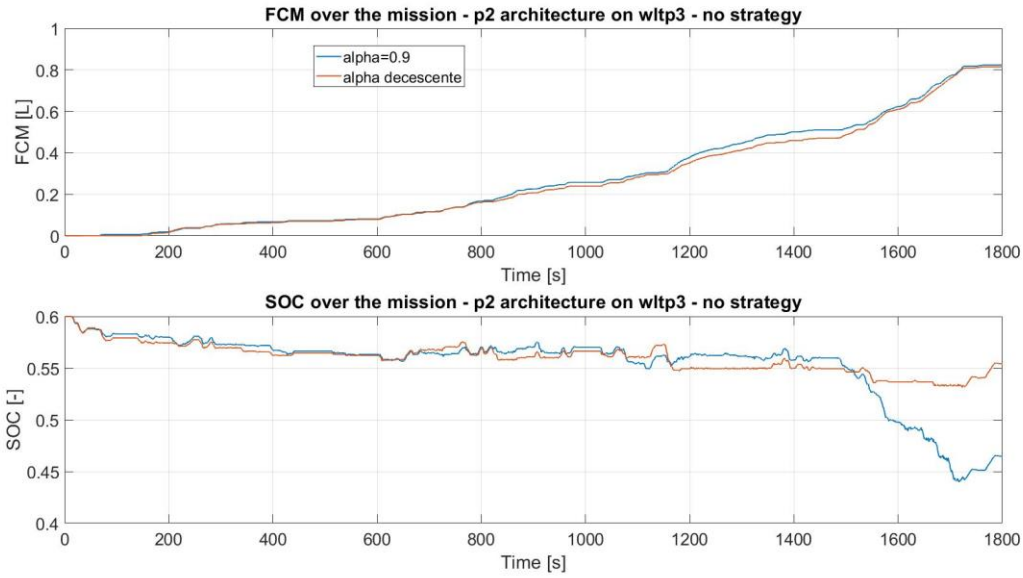

Figura 6.20 Confronto tra prove con α costante e α decrescente

Tabella 6.4 Somma dei reward e numero di episodi esclusi per prove con beta=0.9 per α diversi

Prova considerata	Somma dei *reward*	Numero episodi esclusi
Alpha=0.9	3493	1849
Alpha decrescente	3501	1656

In conclusione è opportuno notare come un *learning rate* decrescente con il numero dell'episodio possa essere applicato anche a prove con β minore di 0.9: per ottenere buoni risultati bisognerebbe fermare l'allenamento prima dell'episodio considerato all'inizio come finale.

6.2.3.2 Parametro *cost*

La scelta del parametro $cost$ presente all'interno della funzione di *reward* influisce sui risultati finali. La sua analisi viene riportata nella sottosezione 6.2.3 poiché proprio per un β=0.9 la variazione di questo parametro ha portato i più grandi cambiamenti dei risultati. Ammettendo di mantenere l' $amplifier$ pari a 1, la variazione di $cost$ porta a *reward* ottenuti differenti. Aumentando tale parametro è possibile ottenere *reward* maggiori e influenzati in misura minore dalla riduzione di *reward* pari a $amplifier \cdot (beta \cdot icemfc_{norm}(t,a) + (1 - beta) \cdot \Delta SOC^2_{norm})$. Ciò porta a risvolti diretti sull'allenamento: l'agente, infatti, tende a preferire in modo ancora più evidente gli episodi che si concludono all'istante di tempo finale del ciclo e a sfavorire gli episodi esclusi: se infatti l'episodio si conclude prima della fine del ciclo di guida, l'agente ottiene molto meno *discounted return*. Per cercare di alleviare il problema del numero elevato di episodi esclusi nelle prove a β=0.9 si è provato quindi ad alzare il parametro $cost$. Effettivamente il numero di episodi esclusi è diminuito e i risultati sono cambiati. Di seguito sono quindi riportati i risultati di SOC e FC finali ottenuti per tre prove svolte con $cost = 2, cost = 8, cost = 20$ e tutti gli altri parametri uguali tra le diverse prove. Si può notare come all'aumento del $cost$ si abbia un aumento del SOC finale e del consumo di carburante. Ciò va in contraddizione con il significato del β=0.9 che tende a preferire la riduzione di FC. Di conseguenza, è stata mantenuta la $cost = 2$. Ad ogni modo risulta complicato confrontare quantitativamente le prove svolte con differenti parametri di *reward*. Per questa ragione è stata implementata una funzione per il confronto dei *reward* ottenuti con prove a parametri differenti e sono stati confermati i migliori risultati ottenuti con la prova con $cost = 2$. Tale funzione di confronto tra *reward* differenti verrà poi spiegata nel capitolo 6.4.

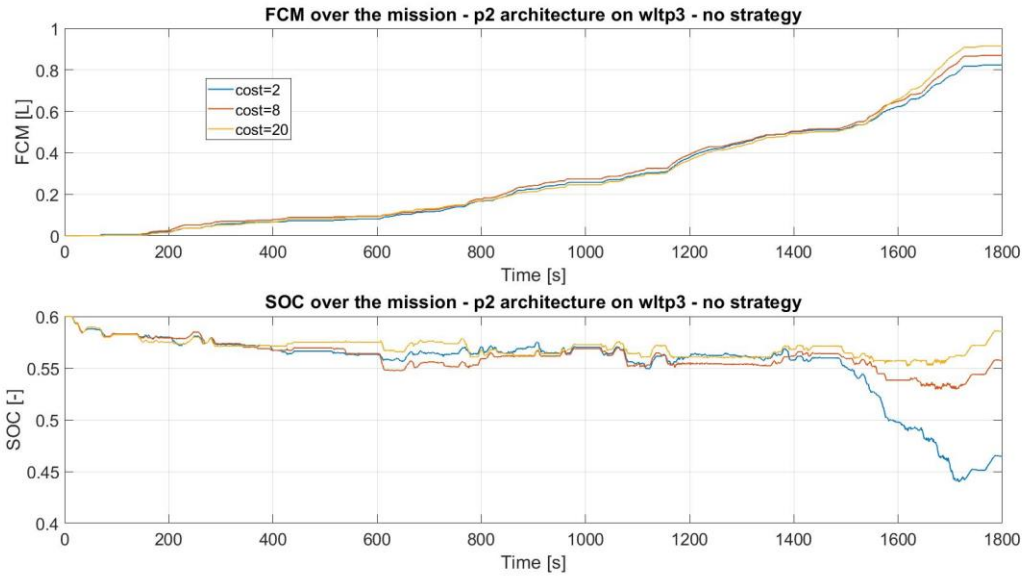

Figura 6.21 Confronto di FC e SOC per β=0.9 al variare del parametro cost

6.3 Effetto complessivo del coefficiente di peso *β*

Con le informazioni ottenute dall'analisi parametrica, sono state svolte anche prove con *β* differenti dai tre visti precedentemente per valutare l'effettiva efficacia della variazione del coefficiente di peso. I risultati finali ottenuti sono ottimi, in quanto *β* rispetta il proprio scopo: più *β* è elevato più l'FC è minore, più il *β* è basso più il SOC finale è maggiore. Sono state svolte prove con sei valori differenti di *β*: 0.1, 0.3 ,0.5, 0.7, 0.8, 0.9. Gli andamenti del SOC e FC ottenuti a fine allenamento di ciascuna prova sono stati riportati in Figura 6.22. Per garantire una più facile lettura dei risultati finali sono riportati dei grafici di Pareto: il primo in Figura 6.23 rappresenta semplicemente i SOC e FC finali ottenuti dalle diverse prove considerate; il secondo invece rappresenta il SOC finale e il risparmio di carburante ottenuto in percentuale rispetto al consumo di carburante per una prova con *β*=0, cioè dove viene imposto il solo controllo del SOC e non dell'FC.

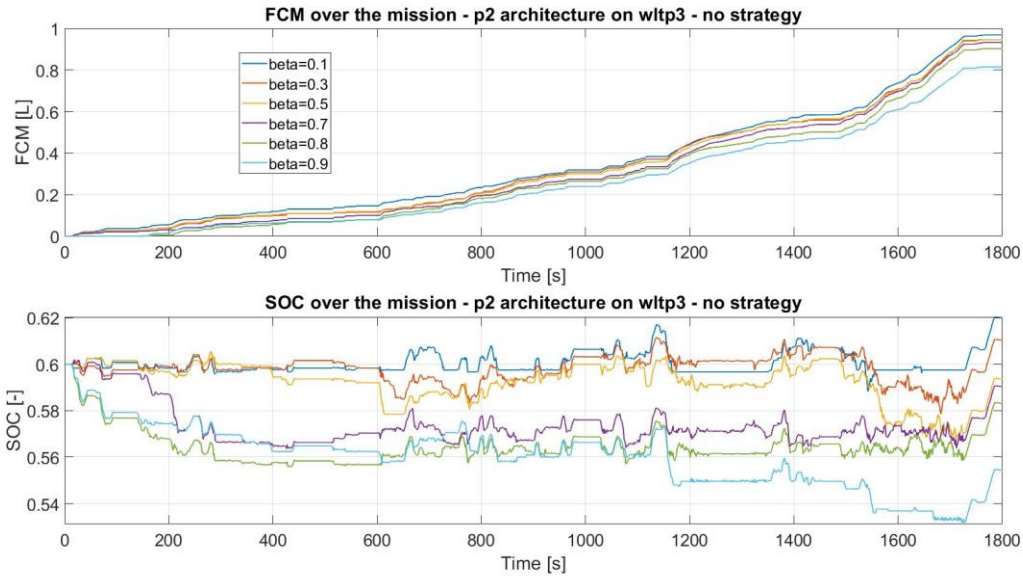

Figura 6.22 Confronto SOC e FC per prove a β differenti

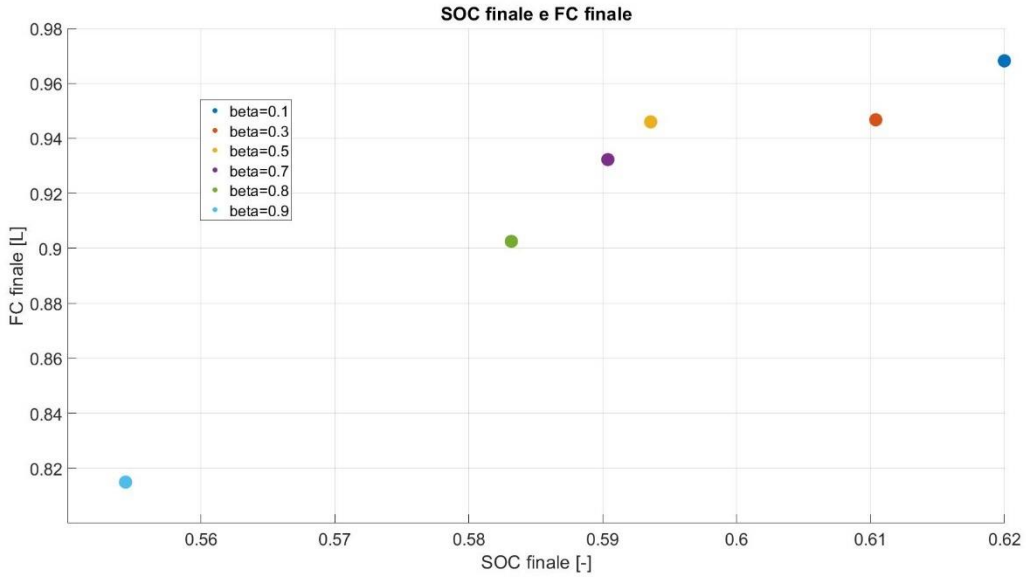

Figura 6.23 Pareto di SOC finale e FC per prove a β differenti

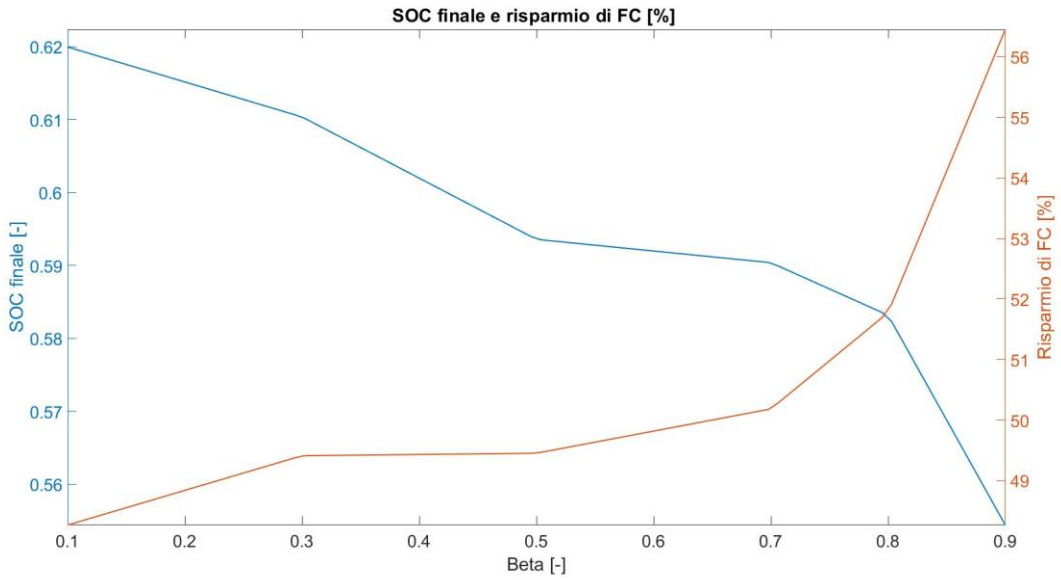

Figura 6.24 Pareto di SOC e risparmio percentuale di FC per prove a β differenti

6.4 Aggiunte al codice

Come precedentemente citato è stato utilizzato come base per questo lavoro un codice precedentemente implementato su MATLAB. Per risolvere alcuni problemi incontrati nello studio svolto sulla funzione di *reward*, sono state implementate due nuove funzioni.

6.4.1 Interruzione dell'allenamento

In primo luogo, è stata implementata una funzione per fermare l'allenamento dell'agente qualora si giunga a convergenza del *discounted return*. Come precedentemente citato, in molte occasioni ciò può essere estremamente utile per evitare che la continuazione dell'allenamento oltre la convergenza porti

al peggioramento dei risultati. L'altro grosso vantaggio di questa funzione è che permette di diminuire il tempo necessario per completare l'allenamento. Tuttavia, non è stato trovato in letteratura un metodo sempre valido per riconoscere la convergenza di un algoritmo di RL, né uno adatto per il problema specifico trattato in questo lavoro. Di conseguenza la funzione sviluppata è basata sui dati ottenuti empiricamente dalle prove svolte ed i valori dei parametri scelti sono adatti solo per i dati e i parametri considerati in queste prove. Tuttavia, si può estendere il valore della funzione a prove più generiche, utilizzando soglie differenti.

L'algoritmo della funzione utilizzata si basa sull'utilizzo delle medie mobili. In primo luogo, si stabilisce un numero minimo di episodi dopo il quale l'allenamento potrebbe essere fermato. A questo punto, per ogni episodio dell'allenamento si guarda se gli episodi compresi tra quello attuale e quello precedente di $N_{step_{conclusi}}$ episodi, siano tutti episodi conclusi correttamente, cioè non vi siano episodi esclusi: se vi fossero infatti episodi esclusi non sarebbe corretto fermare l'allenamento. Dopo aver verificato questa condizione è necessario capire se il *discounted return* è giunto a convergenza o meno. Per far ciò, viene calcolata la differenza tra la media dei *discounted return* degli episodi compresi tra l'episodio attuale e l'episodio numero $N_{episodio_{attuale}} - N_{step_1}$ e la media dei *discounted return* degli episodi compresi tra l'episodio numero $N_{episodio_{attuale}} - N_{step_1}$ e l'episodio numero $N_{episodio_{attuale}} - 2 \cdot N_{step_1}$. N_{step_1} è un numero sempre positivo, determinato in funzione del numero di episodi di allenamento. Chiaramente utilizzare solo questo controllo sarebbe troppo superficiale e l'allenamento si potrebbe fermare in modo scorretto a seconda della scelta di N_{step_1}. Per questo gli stessi identici passaggi son applicati per calcolare le differenze $diff_i$ tra le medie dei *discounted return* utilizzando N_{step_i} con i che va da 1 a n. Per ognuna delle n $diff_i$ calcolate bisogna controllare che sia inferiore a una costante $cost_i$. Se tutte le n differenze sono minori delle rispettive costanti di confronto, allora si conclude che l'algoritmo è arrivato a convergenza e si ferma l'allenamento all'episodio successivo. Il numero n di controlli è arbitrario, ma è consigliabile utilizzare $n > 2$. Per semplicità i numeri N_{step_i} sono tali da soddisfare la seguente relazione: $N_{step_i} = N_{step_{i+1}} \cdot 2$. Per generalizzare questo algoritmo basterebbe porre tutte le costanti $cost_i$ pari a zero.

Viene di seguito riportato il confronto tra due prove con β=0.5: una in cui l'allenamento viene fermato prima dell'episodio indicato a inizio allenamento come finale, l'altra in cui l'allenamento prosegue fino alla fine. I parametri utilizzati nelle due prove sono i medesimi. Osservando gli andamenti del SOC e FC ottenuti, si vede come la prova con stop allenamento porti a un SOC finale maggiore, ma allo stesso tempo un FC maggiore: non si riesce quindi a capire ad occhio quale tra le due prove dia risultati migliori. Tuttavia, si può notare che per la prima, la somma dei *reward* ottenuti durante l'episodio finale è maggiore e l'allenamento risulta più corto, cioè si ferma a un numero di episodi inferiore rispetto alla seconda.

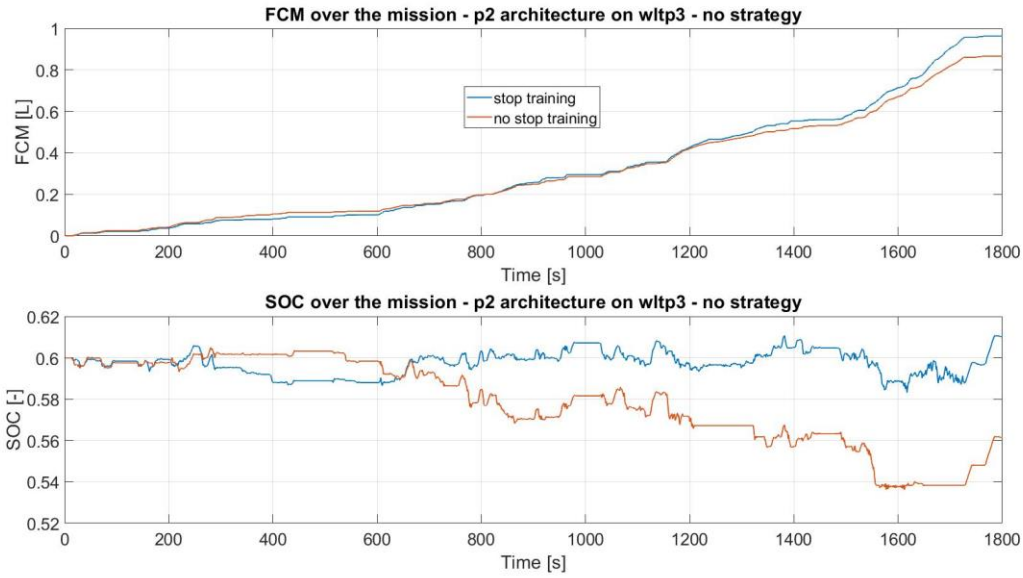

Figura 6.25 Confronto di SOC e FC di prove con e senza stop allenamento

Tabella 6.5 Somma dei reward e numero di episodi di allenamento con e senza interruzione

Prova considerata	Somma dei *reward*	Numero di episodi di allenamento
Interruzione allenamento	3536	1852
No interruzione allenamento	3524	2300

6.4.2 Confronto tra prove con *reward* differenti

Un'altra funzione molto utile è stata creata per confrontare i risultati di prove con *reward* differenti. Come visto più volte nel corso di questo lavoro non è immediato capire quale prova tra due confrontate abbia effettivamente fornito i migliori risultati. Se le funzioni di *reward* presentano parametri con valori diversi non si può neanche confrontare la somma dei *reward* o il *discounted return* dell'episodio finale, poiché sono stati calcolati in modo diverso nelle due prove. L'unico modo per effettuare il confronto tra i *reward* è ricalcolarli rispetto alla medesima funzione di *reward*. Consideriamo quindi due prove differenti in cui è stata usata la funzione di *reward* Parabola, ma con parametri differenti. Una volta quindi caricati i valori di SOC e FC ottenuti con l'episodio finale della seconda prova, ricalcoliamo i *reward* ottenuti a ogni istante di tempo del ciclo con la funzione di *reward* con i parametri utilizzati nella prima prova. La formula utilizzata per il calcolo dei *reward* r_2 della seconda prova utilizzando la funzione di *reward* della prima è quindi la seguente:

$$r_2 = cost_1 - amplifier_1\left(beta_1 \cdot icemfc_{norm_2}(t,a) + (1-beta_1)\cdot \Delta SOC^2_{norm_2}\right)$$

$$icemfc_{norm_2}(t,a) = \frac{icemfc_2(t,a)}{\max(icemfc_2)}$$

$$\Delta SOC^2_{norm_2} = \frac{(SOC_2(s') - SOC_{initial_2})^2}{(SOC_{high_2} - SOC_{initial_2})^2}$$

Dove le grandezze con pedice 1 sono riferite alla prima prova, mentre le grandezze con pedice 2 sono riferite alla seconda. Dopo aver calcolato i *reward* della seconda prova in questo modo, la loro somma può essere confrontata direttamente con la somma dei *reward* della prima prova: la prova per cui la somma dei *reward* è maggiore, è la prova migliore. È da specificare che questo confronto ha significato solo con prove simili. È stato molto utile, per esempio, per quelle prove svolte a β=0.9, ma a $cost$ differente. I *reward* delle due prove hanno infatti il medesimo significato e ha senso il confronto: se si

calcolano i *reward* della seconda prova rispetto alla funzione della prima o viceversa, la prova migliore è sempre la medesima. Confrontare invece prove a β molto differenti non ha particolare significato: è normale che calcolando i *reward* della seconda prova rispetto alla funzione della prima, venga individuata come prova migliore la prima, e nel caso opposto in cui si calcolino i *reward* della prima prova rispetto alla funzione della seconda, la prova migliore sia la seconda.

7 Problema della frenata

Come citato nei capitoli precedenti, l'agente utilizzato fino a questo momento è in grado di gestire correttamente solo la scelta dell'azione in intervalli di tempo di trazione. Tuttavia, per un veicolo ibrido elettrico, le fasi di frenata risultano molto importanti, poiché permettono di ricaricare la batteria: per un veicolo ibrido Parallelo, la scelta scorretta della marcia in frenata può portare a un recupero di potenza minore. Inoltre, l'agente considerato finora impara solo in intervalli di tempo di trazione. Questo comporta una sottostima dei risultati ottenibili poiché dal punto di vista dell'agente non sono previste fasi di frenata nel ciclo di guida. La frenata porterebbe infatti a ottenere valori più elevati di *reward* rispetto alla trazione: utilizzando la frenata rigenerativa, si ottiene un aumento del SOC, senza alcun consumo di carburante.

In questo capitolo, si analizzano diverse soluzioni al problema della frenata: l'aggiornamento della stessa *Q-table* sia in frenata che in trazione, l'aggiunta di una *Q-table* per gestire la sola frenata, l'aggiunta di PF di frenata e l'aggiunta di uno stato di trazione-frenata che si affianca al SOC.

7.1 Aggiornamento della stessa *Q-table* per frenata e trazione

In questa soluzione proposta, si mantiene la medesima *Q-table* utilizzata fino a questo momento. L'unica differenza riguarda l'aggiornamento dei valori di Q contenuti nella *Q-table*. Fino ad ora, la *Q-table* è stata aggiornata solo in intervalli di tempo di trazione: in questa prima modifica la *Q-table* è aggiornata anche in intervalli di tempo di frenata. Questa prima soluzione si è in realtà rivelata completamente fallimentare: nelle prove svolte con numero di episodi basso, tutti gli episodi si rivelano episodi esclusi. Analizzando la *Q-table* finale che si ottiene se ne capisce la ragione: per quasi tutti gli stati, anche quelli riferiti a SOC vicini al limite inferiore della finestra di SOC, l'azione a cui corrisponde il massimo *Q(s,a)* è un'azione con PF puramente elettrico. Di conseguenza, quando l'intervallo di tempo considerato è di trazione e l'agente è in *exploitation*, quest'ultimo tende quasi sempre a scegliere un'azione di PF puro elettrico facendo diminuire di molto il SOC. A causa delle azioni scelte, durante l'episodio il SOC diminuisce così tanto da raggiungere il limite inferiore della finestra di SOC considerata prima dell'ultimo istante di tempo del ciclo di guida. I valori così alti di *Q(s,a)* per azioni di PF puro elettrico sono dovute proprio alla modifica effettuata. Quando l'ambiente si trova in un intervallo di tempo di frenata e nello stato s, le azioni considerate *feasible* sono solamente le azioni con PF puramente elettrico per ottenere la frenata rigenerativa. Il *reward* che si ottiene è elevato, grazie all'aumento del SOC ottenuto senza consumo di carburante. A questo punto, si aggiorna quindi il valore di *Q(s,a)* con la seguente formula:

$$Q(s,a) = Q(s,a) + \alpha\left[r + \gamma \max_{a'}\big(Q(s',a')\big) - Q(s,a)\right]$$

Quando l'ambiente si troverà al medesimo valore di SOC, quindi nello stesso stato s, in un istante di tempo di trazione, l'agente in *exploitation* tenderà a scegliere l'azione con PF puramente elettrico che in frenata ha portato a un *reward* così elevato. A causa di questo grosso problema insito nella modifica stessa, l'agente è considerato completamente inadeguato.

7.2 *Q-table* per la frenata

La seconda soluzione proposta è l'utilizzo di una *Q-table* aggiuntiva per la sola frenata. In questo caso sono quindi presenti due *Q-table* separate, una per la trazione, l'altra per la frenata. Le dimensioni delle due matrici sono uguali e le coppie stato-azione sono le medesime per le due matrici. Per ogni coppia stato-azione sono presenti quindi due valori distinti della Q: uno riferito alla trazione e uno riferito alla frenata. Ora che sono state introdotte le due matrici, si può spiegare come vengono utilizzate in questo

algoritmo. Ponendosi in un istante di tempo *t* di trazione con corrispondente stato *s* dell'ambiente e scelta dall'agente l'azione *a*, viene aggiornato con la solita formula il corrispondente valore di *Q(s,a)*, esattamente come avveniva con l'agente precedente. Se però l'istante di tempo *t* corrisponde a un istante di frenata, viene aggiornato il valore $Q_{frenata}(s,a)$ appartenente alla matrice *Q-table* per la frenata, secondo la seguente formula:

$$Q_{frenata}(s,a) = Q_{frenata}(s,a) + \alpha_{frenata}[r + \gamma_{frenata}\max_{a'}(Q(s',a')) - Q_{frenata}(s,a)]$$

Dove i parametri $\alpha_{frenata}$ e $\gamma_{frenata}$ sono il *learning rate* e *discount factor* usati per la frenata che possono essere imposti diversi dalla trazione. È interessante osservare come $\alpha_{frenata}$ sia stato imposto, nella maggior parte delle prove, decrescente con il numero dell'episodio. Ciò è ragionevole in quanto le coppie stato-azione effettivamente aggiornate nella *Q-table* per la sola frenata siano molte meno rispetto a quelle della *Q-table* per la trazione: l'utilizzo di $\alpha_{frenata}$ decrescente permette di arrivare alla convergenza del valore di $Q_{frenata}(s,a)$ in modo più preciso. In questo caso, il $Q_{frenata}(s,a)$ riferito quindi allo stato e azione successivi appartiene alla *Q-table* della frenata. In questo modo è come se l'agente prevedesse che dopo un'istante di trazione anche l'istante successivo sia di trazione e che a un istante di frenata segua sempre un altro istante di frenata. Ciò non è sempre vero e dal punto di vista teorico è un'assunzione scorretta. Un altro importante dettaglio da determinare è quello di decidere sotto quali condizioni aggiornare la *Q-table* di frenata. È stato ritenuto opportuno aggiornarla non in tutti gli istanti di tempo in cui non è presente la trazione, bensì sono esclusi gli istanti di tempo in cui la potenza richiesta al veicolo è nulla, quindi a veicolo fermo. L'aggiornamento della matrice in tali condizioni fornirebbe informazioni inutili all'agente poiché la scelta della marcia a veicolo fermo risulta completamente inutile. Viene riportata in Figura 7.1, la rappresentazione 3-D di una *Q-table* per la sola frenata ottenuta a fine allenamento di una prova con β=0.5. Si può notare, come atteso, che buona parte dei valori delle $Q_{frenata}(s,a)$ sono nulli: gli unici valori diversi da zero sono quelli riferiti ad azioni di PF puramente elettrico, le uniche azioni *feasible* in frenata. Si può inoltre notare che, a parità di stato e azione con PF puramente elettrico considerati, il valore $Q_{frenata}(s,a)$ è maggiore rispetto al rispettivo valore della *Q(s,a)* di trazione.

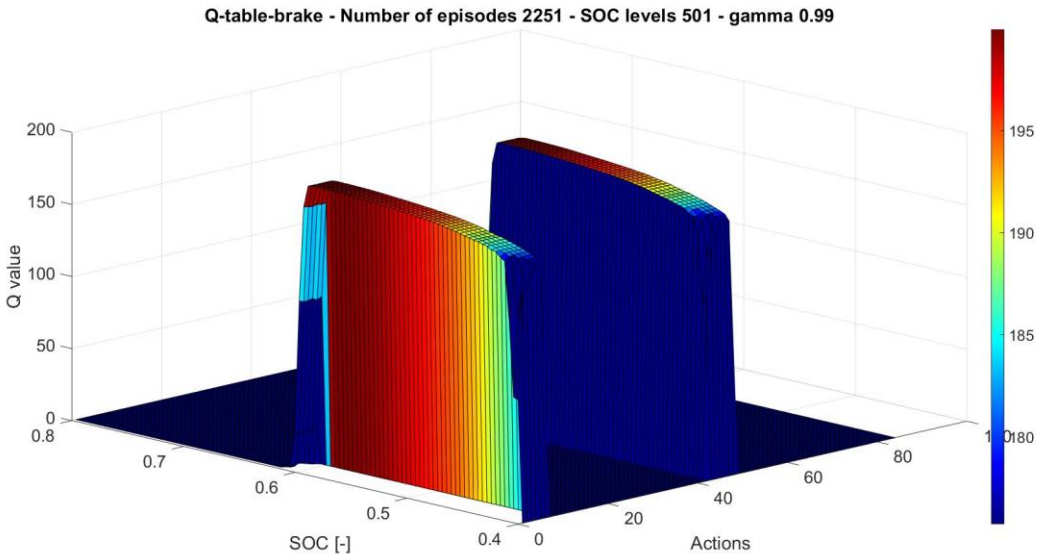

Figura 7.1 Q-table per la sola frenata

Sono state quindi svolte delle prove con diversi β per confrontare i risultati ottenuti con l'utilizzo di due *Q-table* invece che una sola. Si inizia quindi prendendo in considerazione tre prove svolte a parametri comuni $\beta = 0.5$, $\varepsilon = 1$ fino a metà episodi e poi decrescente con la legge costante+parabola, $\alpha = 0.9$ e $\gamma = 0.99$: la prima con una sola *Q-table* per la sola trazione, la seconda presenta due *Q-table* aggiornate in ogni istante temporale e la terza con due *Q-table* aggiornate in ogni istante temporale in cui il veicolo non risulta fermo. Osservando gli andamenti di SOC e FC, la prova peggiore è la seconda: il SOC si

abbassa troppo, fino a raggiungere un SOC finale di 0.563. Come prevedibile, l'aggiornamento inutile della *Q*-table in frenata in quegli istanti in cui il veicolo è fermo porta a un peggioramento dei risultati.

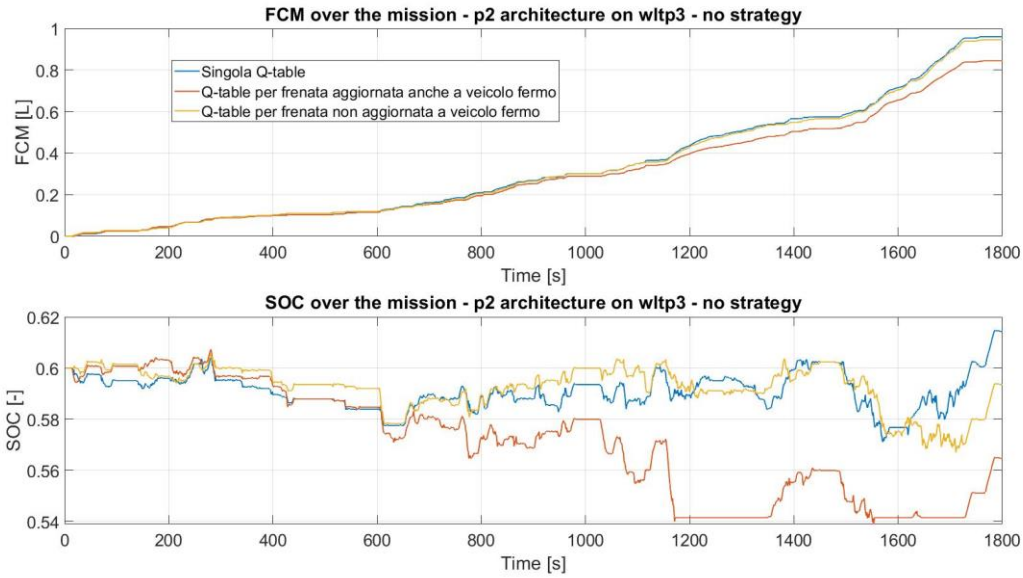

Figura 7.2 Confronto di SOC e FC per prove a singola e doppia Q-table

Guardando invece la somma dei *reward*, si nota che la prova migliore, sebbene di poco, è la prova con *Q-table* per frenata che non viene aggiornata a veicolo fermo, ma soltanto quando la velocità del veicolo è maggiore di zero.

Tabella 7.1 Somma dei reward per prove a singola e doppia Q-table

Prova considerata	**Somma dei *reward***
Singola *Q-table*	3537.4
Q-table per frenata aggiornata anche a veicolo fermo	3520
Q-table per frenata non aggiornata a veicolo fermo	3538.2

Per gli altri β è stata di conseguenza abbandonata completamente l'opzione di aggiornare la *Q-table* anche a veicolo fermo e sono state confrontate soltanto le altre due configurazioni. Per β=0.9 il confronto risulta essere ancora più interessante: anche solo confrontando gli andamenti di SOC e FC si vede come la prova con l'utilizzo delle due *Q-table* dia risultati migliori. Si nota, infatti, come con quest'ultima prova si riesca a ottenere un FC minore e un SOC finale maggiore.

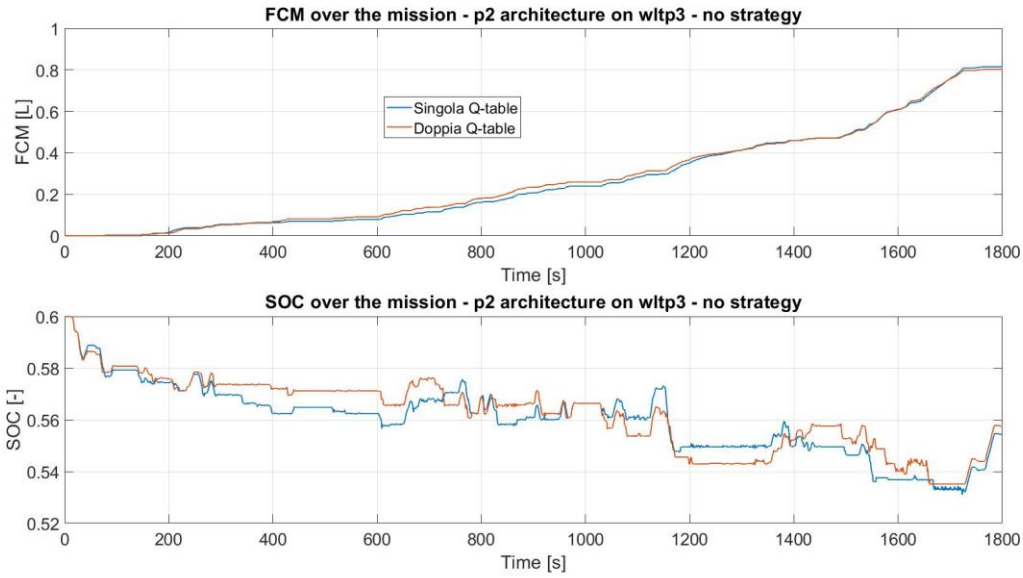

Figura 7.3 Confronto di FC e SOC per prove con β=0.9 a singola e doppia Q-table

Al contrario, per quanto riguarda le prove con β=0.1, i risultati ottenuti con le prove a due *Q-table* sono leggermente peggiori. Ciò può essere dovuto al fatto che con β=0.1 si raggiungono molto più frequentemente valori del SOC maggiori del valore iniziale: ciò potrebbe peggiorare l'apprendimento dell'agente.

In conclusione, analizzando i risultati ottenuti, questa soluzione risulta interessante, ma non risolve tutti i problemi legati alla frenata, in quanto con l'utilizzo di due *Q-table* separate, frenata e trazione sono allenate separatamente. Inoltre, non può essere con certezza indicata come una soluzione migliore rispetto all'utilizzo di una singola *Q-table* per la sola trazione. Per analizzare i risultati ottenuti in frenata è stata scritta una nuova porzione di codice di post-processamento dei dati, che permette di calcolare la variazione di SOC in ogni intervallo di tempo di frenata del ciclo. Considerando le due prove a β=0.9 analizzate in questo capitolo ci si aspetterebbe di vedere che il recupero del SOC in frenata sia maggiore per la prova con due *Q-table*. Tuttavia, si nota in Figura 7.4 è chiaro come si ottenga una variazione di SOC maggiore e quindi una prestazione migliore in frenata per la prova con singola *Q-table* per la sola trazione.

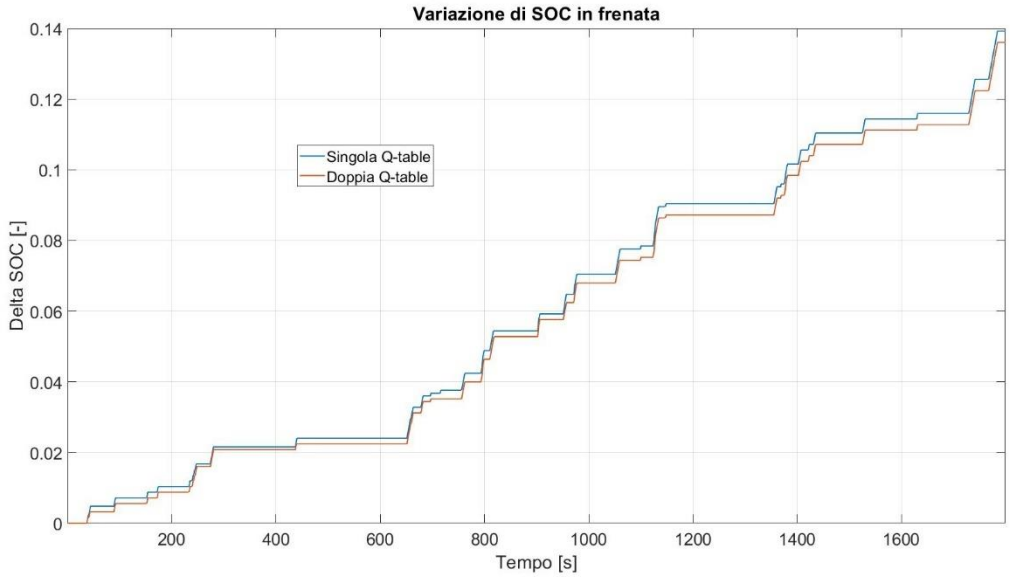

Figura 7.4 Confronto della variazione di SOC in frenata per prove a singola e doppia Q-table

Al fine di risolvere il problema relativo alle prestazioni dell'algoritmo in frenata si è provato ad utilizzare una funzione di *reward* apposita per la frenata e un $\gamma_{frenata}$ molto basso, ma con scarsi risultati.

7.3 Aggiunta di PF per la frenata

Un altro modo provato per gestire la frenata è l'aggiunta di uno o più PF di frenata. Per far ciò è stato necessario effettuare modifiche del codice, cambiando il numero di azioni e le dimensioni della *Q-table*. Vengono proposte, in questa sezione, due soluzioni differenti per gestire la frenata in questo modo: aggiunta di un PF per la frenata rigenerativa e aggiunta di due PF per la frenata rigenerativa e non rigenerativa.

7.3.1 Aggiunta di un PF per la frenata rigenerativa

In questa sottosezione si tratta quindi dell'aggiunta di un PF di frenata rigenerativa. Ciò comporta un aumento del numero di azioni pari a $N_{GB} \cdot N_{ES}$ dove N_{GB} è il numero di marce e N_{ES} è il numero di stati di accensione dell'ICE, sempre pari a due. La *Q-table* assume di conseguenza di dimensioni maggiori: il numero degli stati rimane quindi invariato rispetto a prima, mentre il numero di azioni è aumentato. È inoltre stato necessario cambiare le condizioni di *feasibility* e le conseguenti matrici delle *feasibility*. In ogni intervallo di tempo di frenata vengono considerate come *feasible* solo quelle azioni riferite al PF di frenata rigenerativa. Le altre azioni di PF puramente elettrico sono considerate *feasible* solo per gli intervalli di tempo di trazione.

Con la singola *Q-table* per la sola trazione, nella fase di aggiornamento della *Q-table* le azioni a' comprendono tutte le azioni presenti nella *Q-table* per quello stato, senza considerare le azioni *feasible* nell'istante di tempo riferito allo stato successivo:

$$Q(s,a) = Q(s,a) + \alpha \left[r + \gamma \max_{a'} \big(Q(s',a') \big) - Q(s,a) \right]$$

Applicare ciò a questa nuova *Q-table* di frenata ha portato a risultati pessimi. Ciò è dovuto al fatto che i valori di ogni *Q(s,a)* vengono sovrastimati proprio a causa del termine $\max_{a'} \big(Q(s',a') \big)$. Avendo infatti aggiunto il PF in più riferito alla sola frenata rigenerativa ed essendo il *reward* ricevuto in frenata in generale maggiore di quello ricevuto in trazione, per uno stato generico *s*, l'azione per cui il valore di *Q(s,a)* è massimo è un'azione di frenata. Di conseguenza, indipendentemente dallo stato di trazione o

frenata nell'intervallo di tempo successivo, $\max_{a'}\left(Q(s',a')\right)$ è riferito a un'azione a' di frenata. Intuitivamente, è come se l'agente ogni volta che debba scegliere un'azione all'istante di tempo *t*, si aspetti che nell'istante di tempo successivo ci sia frenata quando in realtà ciò non è vero. L'algoritmo tende di conseguenza a sovrastimare tutti i valori delle *Q* e le aspettative dei risultati: concretamente si generano tanti episodi esclusi. Per questa ragione bisogna prendere in considerazione soluzioni differenti. Sono state valutate quattro possibili soluzioni per la risoluzione del problema.

7.3.1.1 Diverse soluzioni di tentativo

La prima soluzione, la più logica, comporta l'utilizzo delle matrici delle *feasibility* per vedere quali siano le azioni a' realmente *feasible* nell'istante di tempo successivo. In questo caso quindi per lo stato successivo s' non vengono più considerate tutte le azioni presenti nella *Q-table*, ma solamente quelle fisicamente valide. Modificando in questo modo l'apprendimento dell'agente, è stato possibile ridurre il numero di episodi esclusi. Tuttavia, i risultati sono peggiori rispetto a quelli ottenuti con la singola *Q-table* per la sola trazione. Ciò è dovuto al fatto che i valori delle *Q* contenute nella *Q-table*, riportata in Figura 7.5 vengono fortemente sovrastimati. Per gli stati vicini a SOC pari a 0.6, i valori raggiunti dalle *Q(s,a)* per PF in trazione sono maggiori di 199.9, un valore troppo elevato che sarebbe corretto solo se si ottenesse una media dei *reward* nell'episodio finale maggiore di 1.99, cosa che non avviene.

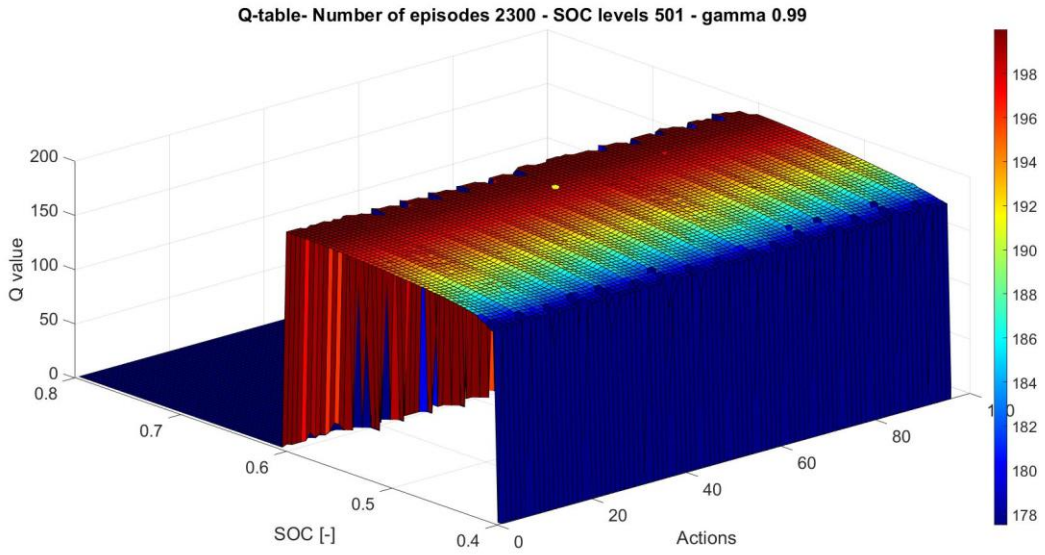

Figura 7.5 Q-table con PF frenata rigenerativa – previsione azioni future

La sovrastima dei risultati è dovuta nuovamente al termine $\max_{a'}\left(Q(s',a')\right)$. Il problema è in realtà molto simile a quello visto nel caso precedente, nonostante esso venga limitato molto grazie all'utilizzo delle sole azioni fisicamente valide per l'istante di tempo successivo. Esso si presenta solo in particolari condizioni. Immaginando di trovarsi all'istante di tempo *t* di trazione, con uno stato *s* dell'ambiente e scelta dell'azione *a* e che l'istante successivo *t'* sia di frenata: il termine *Q(s,a)* viene aggiornato prendendo il massimo valore di $Q(s',a')$ tra le azioni a' di frenata. Come spiegato precedentemente, i valori delle *Q* per azioni di frenata sono maggiori rispetto a quelli delle azioni di trazione. Il problema è che può succedere che in un istante di tempo futuro o in uno tra gli episodi seguenti, si ripresenti lo stato *s* con la scelta dell'azione *a*, ma l'istante di tempo successivo non sia di frenata, bensì di trazione. Il valore di *Q(s,a)* prima di quest'ultimo aggiornamento è stato sovrastimato. La sovrastima del valore delle *Q* deriva quindi da quegli intervalli di tempo di trazione seguiti da un intervallo di tempo di frenata. È vero che, viceversa, gli intervalli di tempo di frenata seguiti da un intervallo di tempo di trazione portano a una sottostima del valore di *Q(s,a)* riferito all'azione *a* di frenata scelta, ma questo fenomeno è nascosto dal fatto che, come azione futura, viene scelta l'azione *a'* che massimizza $Q(s',a')$, in cui si

considera quindi un $Q(s', a')$ sovrastimato. Di conseguenza, anche se ci si trova in un istante di tempo di trazione seguito da frenata, non verrà mai scelto come $Q(s', a')$ un valore sottostimato di $Q(s', a')$.

La seconda soluzione tentata consiste nel considerare come azioni a' tutte le azioni riferite a PF di frenata se l'istante di tempo successivo è di frenata, oppure tutte le azioni riferite a PF di trazione se l'istante di tempo successivo è di trazione. Non vengono quindi prese solamente le azioni fisicamente valide bensì un numero maggiore di azioni. Questa soluzione porta a risultati molto simili alla prima, con sovrastima dei valori delle *Q*. Viene riportata la *Q-table* ottenuta per evidenziare come piccole variazioni dell'allenamento portino a *Q-table* visibilmente diverse:

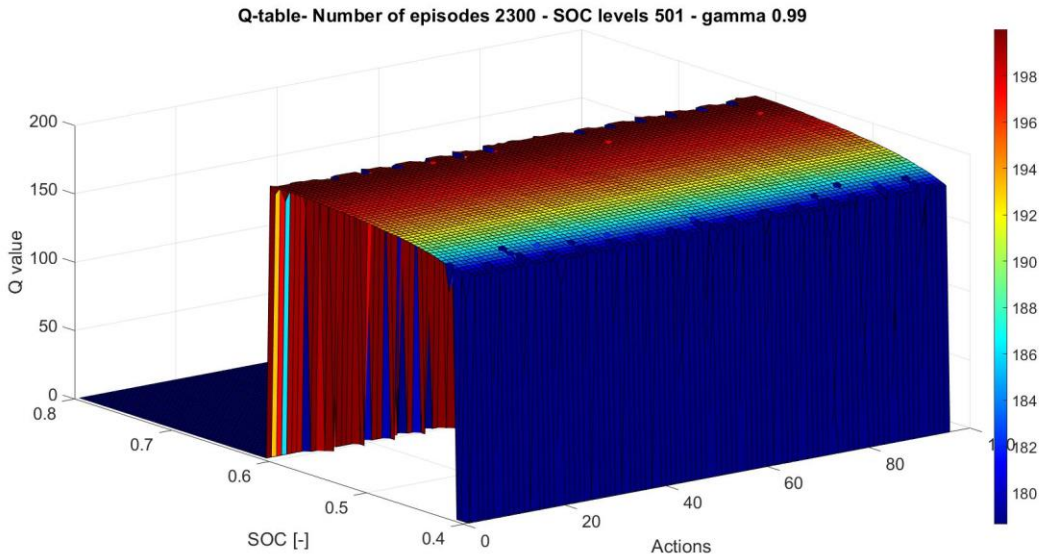

Figura 7.6 Q-table con PF frenata rigenerativa – previsione della sola trazione o frenata

La terza soluzione consiste nel seguire lo stesso ragionamento utilizzato per la doppia *Q-table*. Per ogni istante di tempo *t* di trazione si considera l'istante successivo come fosse un istante di trazione e di conseguenza si prendono come azioni a' solo quelle legate a PF di trazione; invece, per ogni istante *t* di frenata si considera l'istante successivo come fosse anch'esso di frenata e si prendono come azioni a' solo quelle legate a PF di frenata. Dal punto di vista dell'agente, di conseguenza, ad ogni istante di trazione segue un altro istante di trazione e a ogni istante di frenata segue un altro istante di frenata. La *Q-table* ottenuta presenta la grossa differenza rispetto alle due precedenti di avere dei valori di *Q* per le azioni del PF di frenata nettamente più elevati rispetto alle azioni riferite ai PF di trazione. Come prevedibile i risultati ottenuti con questa soluzione sono molto simili a quelli ottenuti con la doppia *Q-table*.

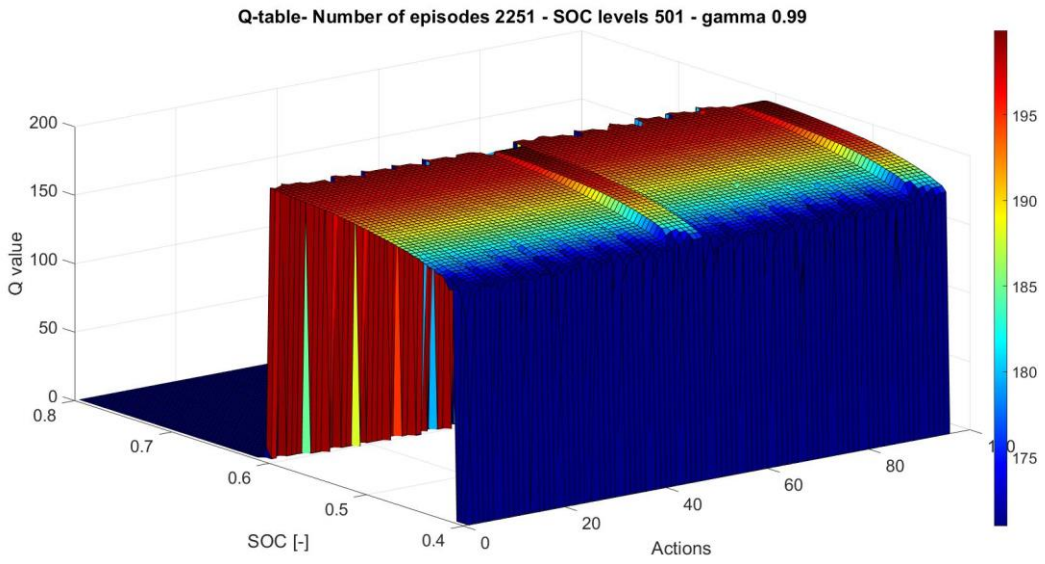

Figura 7.7 Q-table con PF frenata rigenerativa – equivalente a doppia Q-table

Infine, la quarta soluzione è stata pensata al fine di evitare la sovrastima dei valori della Q. Si è pensato infatti di utilizzare la matrice delle *feasibility* per le azioni future solo per gli istanti di tempo presenti di frenata. Per ogni istante di tempo t di trazione vengono considerate come azioni a' solo quelle riferite a PF di trazione, mentre per ogni instante di tempo t di frenata vengono considerate come a' solamente quelle azioni fisicamente valide. Come si può vedere dalla Figura 7.8, i valori di Q riferiti ad azioni di frenata sono maggiori rispetto a quelli di trazione ma comunque minori rispetto al caso precedente. La riduzione di questi valori di Q è dovuta a quegli istanti temporali di frenata seguiti da istanti successivi di trazione.

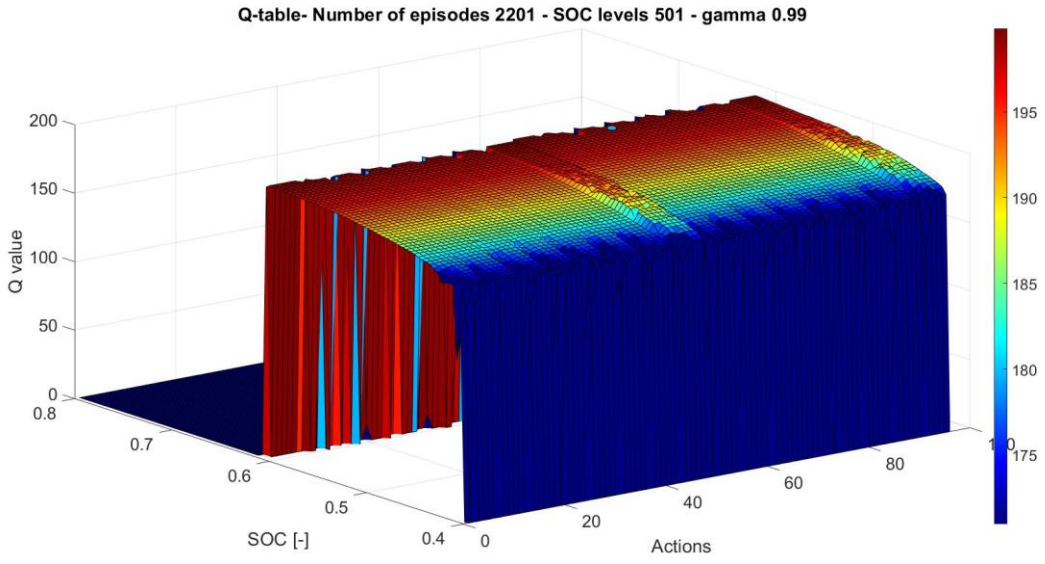

Figura 7.8 Q-table con PF frenata rigenerativa –previsione azioni future solo in frenata

7.3.1.2 Confronto tra le diverse soluzioni

Vengono ora confrontati i valori di SOC e FC ottenuti con le diverse soluzioni tentate considerando un β=0.5. Si nota che la terza e quarta soluzione consentono di ottenere un SOC finale molto maggiore che

compensa l'FC maggiore rispetto alle altre. Confrontando le somme dei *reward* ottenuti si vede che le prove migliori sono nettamente le ultime due.

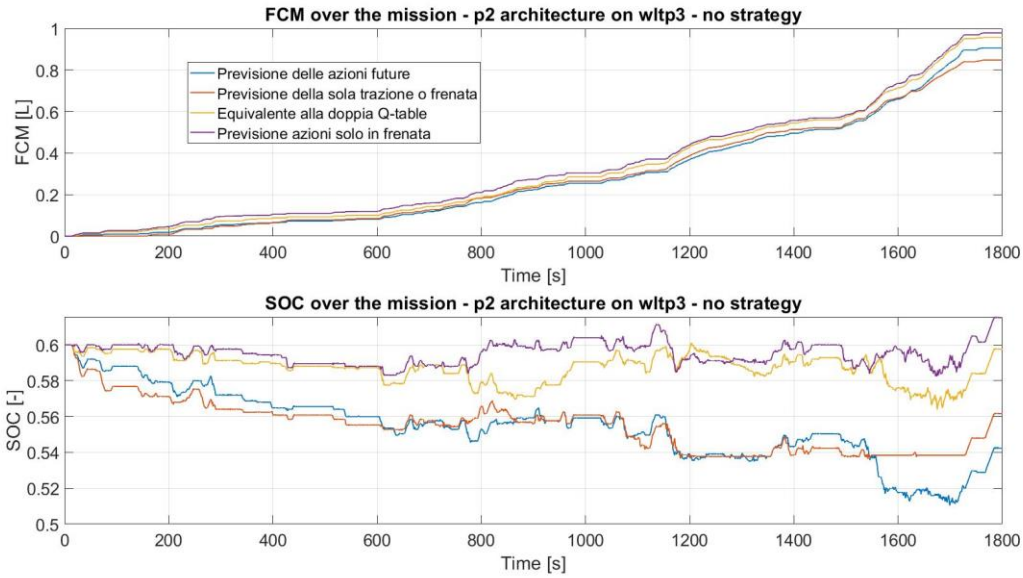

Figura 7.9 Confronto di FC e SOC per le diverse soluzioni con PF frenata rigenerativa per β=0.5

Tabella 7.2 Somma dei reward per diverse soluzioni con PF frenata rigenerativa per β=0.5

Prova considerata	**Somma dei *reward***
Previsione delle azioni future	3490
Previsione della sola trazione o frenata	3497.3
Equivalente alla doppia *Q-table*	3537.5
Previsione azioni solo in frenata	3538.2

Analizzando i risultati dell'allenamento per le quattro soluzioni proposte si può notare come la prima e la seconda forniscano risultati estremamente simili; stessa cosa per la terza e la quarta. Il confronto avviene quindi per comodità tra prove svolte utilizzando la prima e la quarta soluzione. In primo luogo, è possibile notare il differente andamento dei *discounted return* plottati in Figura 7.10. Il *discounted return* finale ottenuto nella prima immagine, riferita alla seconda soluzione, è minore rispetto a quello della seconda immagine riferito alla quarta soluzione; inoltre, per la quarta soluzione il *discounted return* sale più in velocemente fino ad arrivare a convergenza negli ultimi episodi.

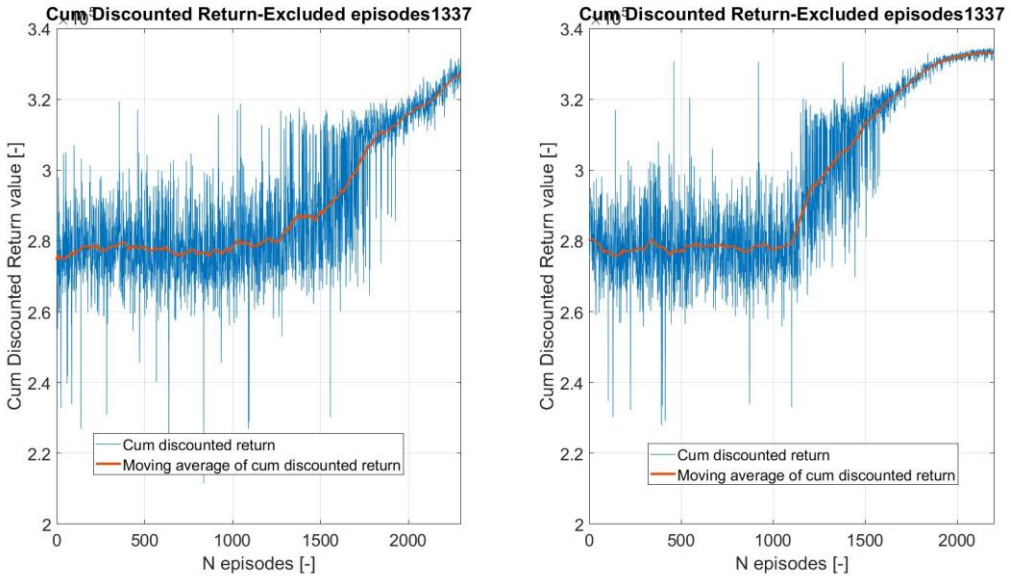

Figura 7.10 Confronto dei discounted return per le diverse soluzioni con PF *frenata rigenerativa*

In seguito, viene mostrato il confronto di FC e SOC ottenuti per prove a β=0.1 e β=0.9 con le diverse soluzioni. In questo caso non è necessario neanche confrontare le somme dei *reward* in quanto si ottengono risultati migliori sia per quanto riguarda il SOC finale che per quanto riguarda l'FC. Per la prova con β=0.9 l'utilizzo della previsione delle azioni future porta addirittura a non far concludere l'episodio finale.

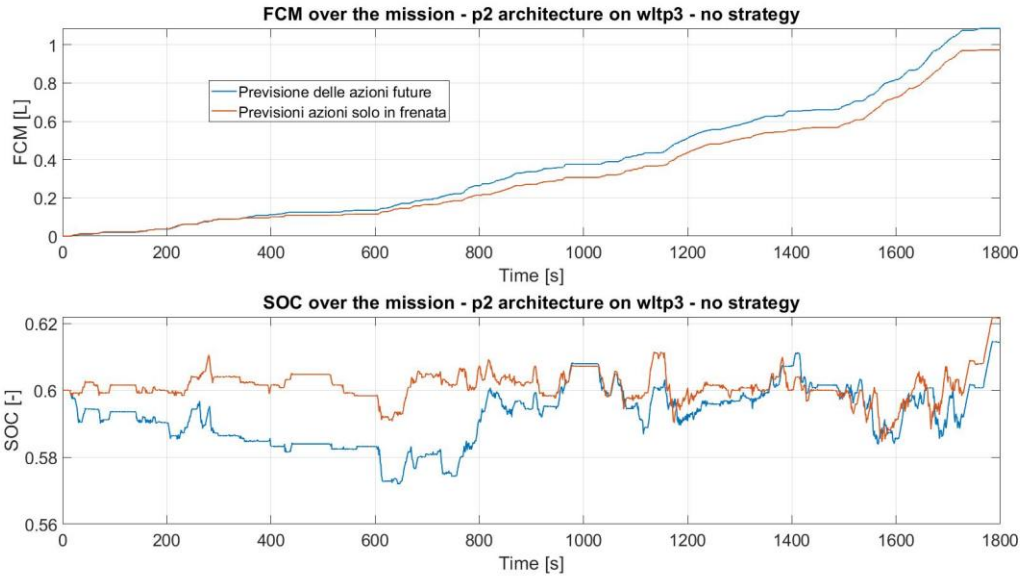

Figura 7.11 Confronto di FC e SOC per le diverse soluzioni con PF frenata rigenerativa per β=0.1

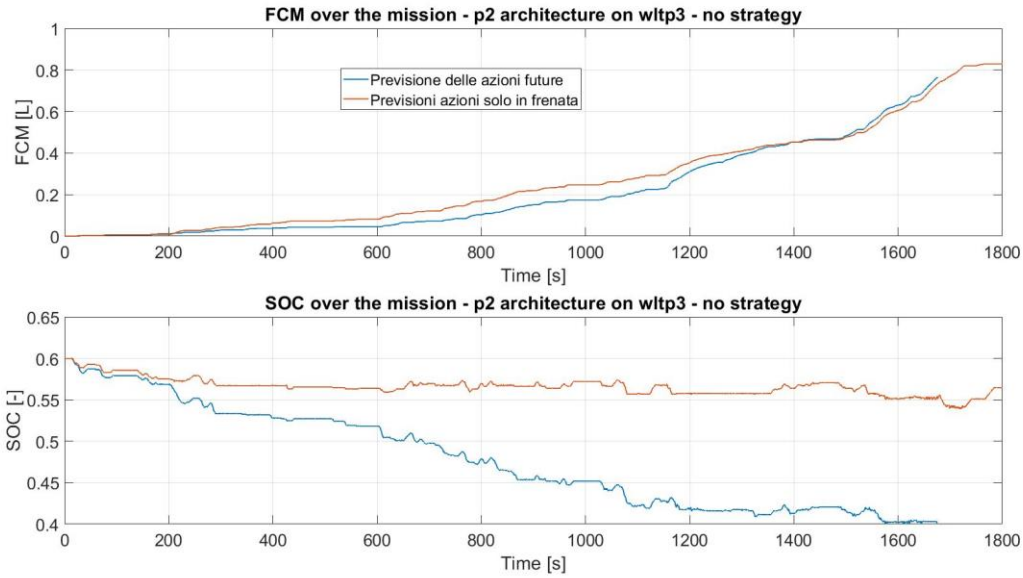

Figura 7.12 Confronto di FC e SOC per le diverse soluzioni con PF frenata rigenerativa per β=0.9

In conclusione, si sono ottenuti risultati buoni, confrontabili, ma non migliori delle prove con l'utilizzo della doppia *Q-table*. Come prima, non si può quindi concludere che l'aggiunta di un PF di frenata rigenerativa porti a risultati migliori rispetto all'utilizzo della singola *Q-table* per la sola trazione.

7.3.2 Aggiunta di due PF per frenata rigenerativa e non rigenerativa

È stata anche provata una soluzione differente che consiste nell'aggiunta di due PF per la frenata: uno puramente elettrico che realizza la frenata rigenerativa, l'altro di frenata non rigenerativa. Questa soluzione è stata introdotta per uno scopo fisico collegato al SOC, ma leggermente diverso da quello inizialmente definito: far arrivare il SOC finale più vicino possibile al SOC iniziale. Per questa ragione le prove effettuate con questa versione dell'agente sono tutte svolte a β=0.1: con β bassi si presenta spesso la situazione di avere un SOC finale molto maggiore di quello iniziale. La minimizzazione del consumo di carburante rimane l'altro obiettivo fisico che si vuole raggiungere. Dal punto di vista dell'algoritmo, il PF per la frenata non rigenerativa è stato gestito in questo modo: è stato imposto un PF puramente termico per convenzione e la frenata viene in questo caso garantita interamente dall'impianto frenante. Questa modifica porta all'aggiunta di un numero di azioni pari a $2 \cdot N_{GB} \cdot N_{ES}$ poiché vengono aggiunti due PF: di conseguenza, la *Q-table* utilizzata ha dimensioni maggiori. Anche in questo caso è necessario cambiare le matrici delle *feasibility*, rendendo *feasible* per intervalli di tempo di frenata solo azioni legate ai due PF di frenata. Sono comunque state sviluppate due versione alternative di questo algoritmo. Nella prima versione possono essere prese azioni a di frenata non rigenerativa in ogni istante di tempo di frenata; nella seconda versione la frenata non rigenerativa al tempo t può essere utilizzata solamente se il SOC presente a tale tempo t è maggiore del valore iniziale di SOC. Si è infatti pensato fosse inutile allenare l'algoritmo a decidere se utilizzare la frenata rigenerativa o meno per un SOC basso: il *reward* ottenuto sarà sempre maggiore o uguale utilizzando la frenata rigenerativa piuttosto che l'altra modalità di frenata per tali valori di SOC. Vengono confrontati i risultati ottenuti con le due versioni con i medesimi parametri per verificare che ciò sia vero. Come si può notare con la seconda versione si ottengono un consumo di carburante minore della prima e un SOC finale maggiore e molto vicino al SOC iniziale di 0.6. Inoltre, il numero di episodi esclusi per la prima versione è più elevato rispetto alla seconda: ciò è ragionevole in quanto nella prima vengono prese azioni di frenata non rigenerativa che non portano ad aumento del SOC. La riduzione del numero di episodi esclusi permette di avere un allenamento migliore per la seconda prova con la quale si ottiene una maggiore somma dei *reward*. È comunque interessante notare come anche nella prima prova l'agente a fine allenamento impari che per SOC<0.6 è sempre più opportuna la frenata rigenerativa.

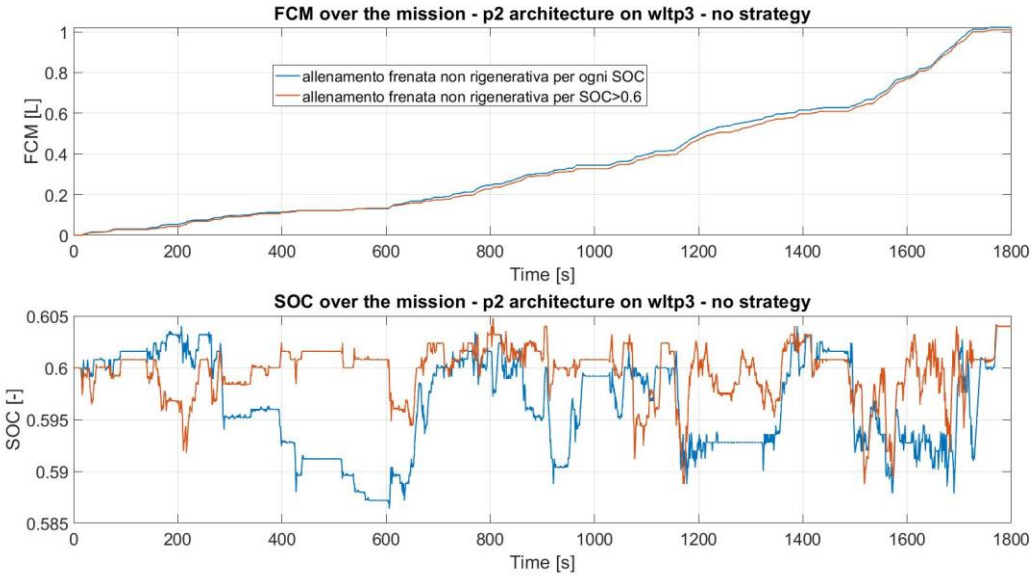

Figura 7.13 Confronto di FC e SOC per le diverse versioni con due PF di frenata

Appurati i migliori risultati della seconda versione dell'agente, le successive prove sono svolte utilizzando quest'ultima. In questo caso, per l'aggiornamento della *Q-table*, sono stati utilizzati il terzo e quarto metodo introdotti nella sezione 7.3. Tuttavia, l'applicazione del metodo equivalente alle due *Q-table* e del metodo con previsione delle azioni future solo per istanti di tempo di frenata, portano a risultati concettualmente diversi per quanto riguarda il SOC. Nella Figura 7.14 si mostra il confronto tra i SOC ottenuti con prove con metodi di aggiornamento della *Q-table* diversi. Nel grafico i tratti di linea colorati di verde si riferiscono alla trazione, i rossi alla frenata e i blu alla condizione di veicolo fermo. Nel subplot superiore dove è utilizzato il metodo equivalente alle due *Q-table*, ogni volta che si ha un intervallo di tempo di frenata con un SOC>0.6 viene sempre scelta la frenata non rigenerativa. Ciò è logico in quanto l'agente prevede che a un istante di tempo di frenata segua sempre un altro istante di tempo di frenata. Al contrario nel secondo subplot dove è usato il metodo con previsione delle azioni future solo per istanti di tempo di frenata, la frenata rigenerativa viene utilizzata anche per valori del SOC maggiori di quello iniziale. In conclusione, è preferibile utilizzare questo metodo con cui si ottengono minore consumo di carburante e un valore del SOC finale molto vicino a quello iniziale.

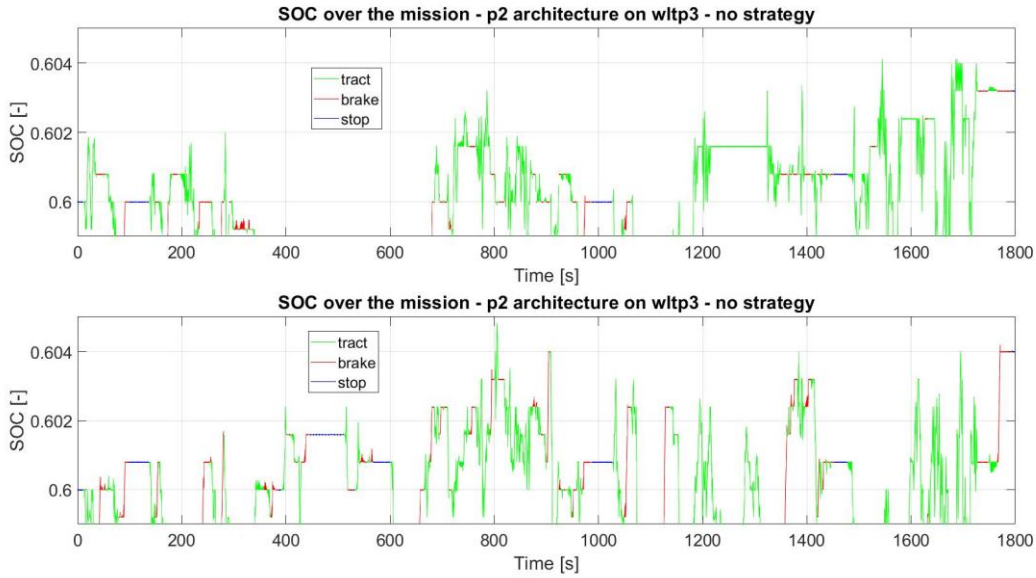

Figura 7.14 Confronto del SOC per prove aggiornate diversamente con due PF di frenata

È necessario sottolineare come conclusione che l'agente descritto in questa sottosezione è meno adatto al raggiungimento degli obiettivi fisici definiti nei capitoli precedenti. A pari parametri e β=0,1, si ottiene infatti un consumo di carburante maggiore e un valore finale del SOC minore, più vicino però al valore iniziale di SOC.

7.4 Aggiunta dello stato trazione-frenata

Un'altra soluzione tentata è stata quella di aggiungere una grandezza differente dal SOC allo stato dell'ambiente. In questo caso non viene utilizzata nessuna azione aggiuntiva: le azioni sono le medesime considerate nell'agente iniziale, senza nessun PF specifico per la frenata. La grossa differenza rispetto agli algoritmi utilizzati precedentemente è l'aggiunta di una variabile booleana, che indica trazione o frenata, allo stato che caratterizza l'ambiente. Considerando di trovarsi in un istante di tempo *t*, tale grandezza assume un valore pari a 1 se nell'istante *t* il veicolo è in trazione; viceversa, è pari a 0 se nell'istante *t* il veicolo si trova in frenata. Anche in questo caso la *Q-table* assume dimensioni maggiori, in quanto il numero di stati possibili raddoppia a pari discretizzazione del SOC. In questa configurazione, a ogni stato corrisponde un preciso valore di SOC discretizzato e l'informazione legata alla condizione di trazione o frenata del veicolo. Questa soluzione è stata pensata per garantire una rappresentazione più appropriata dell'ambiente, avvicinando il sistema a un sistema markoviano. Tuttavia, tale informazione è solamente parziale e non permette una caratterizzazione completa dell'ambiente. Anche in questo caso i valori di *Q(s,a)* vengono aggiornati utilizzando la formula:

$$Q(s,a) = Q(s,a) + \alpha\left[r + \gamma \max_{a'}\left(Q(s',a')\right) - Q(s,a)\right]$$

Si ripresenta il problema della scelta di quali azioni considerare come azioni a': in questo caso, possono essere infatti comprese tra le azioni a' solamente le azioni fisicamente applicabili nell'istante di tempo successivo, oppure tutte le azioni presenti nella *Q-table* riferite allo stato successivo s'.

Viene ora riportata in figura la *Q-table* ottenuta in una prova con questo agente, per sottolineare le differenze e somiglianze rispetto a quelle viste in precedenza. Osservando l'immagine si vedono due regioni separate della *Q-table*, una riferita alla trazione, l'altra riferita alla frenata. Nella seconda si può notare come i valori delle *Q(s,a)* non nulli siano quelli riferiti ad azioni con PF puramente elettrico, in quanto in un istante di tempo di frenata le uniche azioni imposte *feasible* sono quelle che portano a

frenata rigenerativa. Anche in questo caso i valori delle Q ottenuti sono sovrastimati, a causa del medesimo problema incontrato nella sezione 7.3.

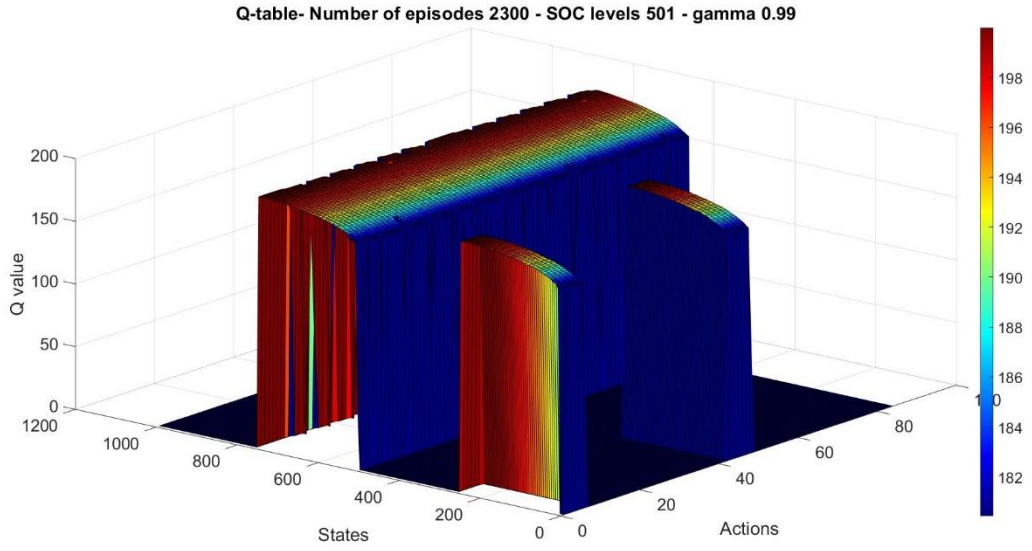

Figura 7.15 Q-table con stato trazione-frenata

Analizzando i risultati ottenuti con questo agente, si osserva una netta somiglianza con il primo e secondo metodo descritto nella sottosezione 7.3.1.1 riferito all'aggiunta del PF frenata rigenerativa. Nonostante non sia a prima vista evidente la similitudine nell'apprendimento tra i due agenti risulta chiara dal seguente ragionamento. Ragionando sulla formula per aggiornare *Q(s,a)*, ogni singolo valore di *Q* all'interno della *Q-table* è associato a una coppia stato *s* e azione *a*. Ciò che conta è quindi la coppia generica *(s,a)* e non il solo stato *s*, né la sola azione *a*. Si consideri che all'istante di tempo *t* si abbia un SOC pari a SOC_t e vi sia trazione. Utilizzando la *Q-table* con due stati (il SOC e il booleano per trazione o frenata), al tempo *t* lo stato *s* contiene al suo interno l'informazione relativa alla trazione e al SOC. Quindi la coppia *(s,a)* è relativa a quel determinato SOC_t e alla trazione. Nel caso di singola *Q-table* con l'aggiunta del PF di frenata rigenerativa in più, al tempo *t* si avrà lo stato *s* che non porta informazioni relative alla trazione o frenata. Però, siccome al tempo *t* vi è trazione, l'azione *a* viene scelta tra le azioni riferite ai PF di trazione e quindi in questo caso è l'azione *a* portare con sé l'informazione della trazione. La coppia *(s,a)* si riferisce quindi a quel determinato SOC_t e alla trazione esattamente come nel caso di *Q-table* con due grandezze di stato. Questo ragionamento vale sia per *Q(s,a)* che per *Q(s',a')*. Quindi l'aggiornamento del valore di *Q(s,a)* avviene nello stesso modo per i due metodi. Per concludere, i due metodi portano a *Q-table* con dimensioni differenti, poiché una ha più stati e l'altra più azioni, ma con lo stesso significato e simili risultati.

7.5 *Q-table* per frenata con stati velocità e potenza

In questa sezione, è invece proposta una soluzione completamente differente rispetto alle precedenti. Si tratta il problema della frenata in modo completamente separato da quello della trazione: si accetta, di conseguenza, di non riuscire a collegare l'apprendimento delle corrette azioni di frenata e trazione. Il ragionamento su cui si basa la creazione di questo agente è molto semplice e ha come obiettivo l'ottimizzazione della scelta di azioni in istanti di tempo di frenata. Si ricorda che in frenata risulta opportuno imporre la frenata rigenerativa, quindi un PF puro elettrico, visti i peggiori risultati ottenuti utilizzando un agente con sia frenata rigenerativa che non rigenerativa. La scelta dell'azione in frenata non è quindi legata alla scelta della distribuzione di potenza tra i due motori, bensì alla sola scelta della marcia. Tuttavia, come stato dell'ambiente è stato fino ad ora considerato solamente il SOC della batteria, che è una grandezza fisica completamente scollegata dalla scelta della marcia. La scelta del

rapporto di trasmissione da utilizzare per la frenata dipende dalla velocità angolare omega e dalla potenza del motore elettrico. È stata quindi analizzata la curva caratteristica velocità angolare-potenza del motore elettrico presente sul veicolo studiato. In Figura 7.16, si può notare come la curva in questione abbia un andamento molto simile a quello teorico, con un primo tratto lineare e poi una seconda parte in cui la potenza rimane quasi costante al variare della velocità angolare. Essendo comunque una caratteristica reale, estratta da dati sperimentali, nel secondo tratto si nota un leggero decremento della potenza. Di conseguenza a ogni valore differente di velocità del motore corrisponde un diverso valore di potenza massima. Nella figura sottostante sono riportati i valori massimi di potenza in valore assoluto, validi quindi sia per la trazione che per la frenata con potenze negative.

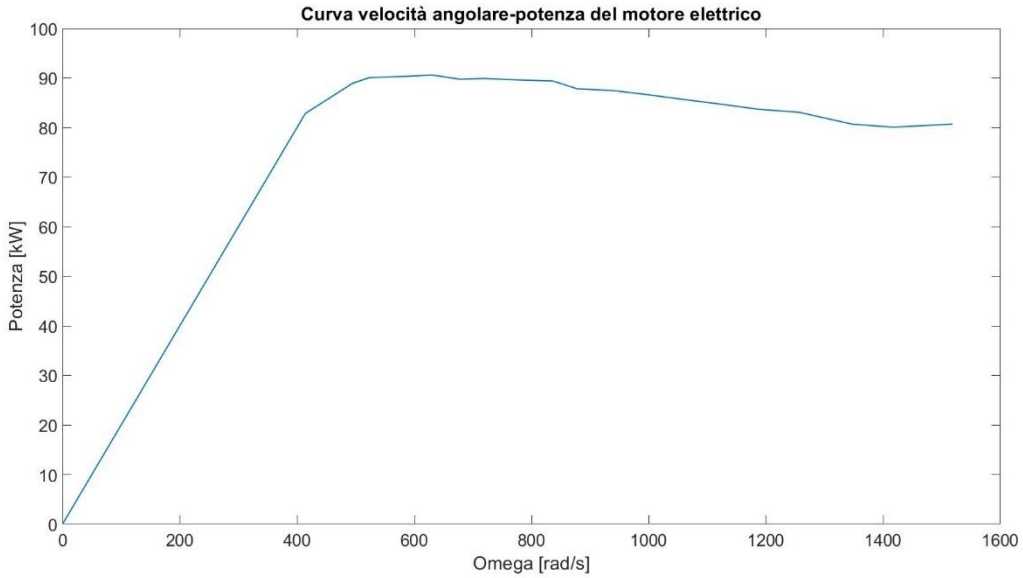

Figura 7.16 Curva caratteristica dell'EM

Ricordando che in frenata la potenza elettrica del motore viene ricevuta e immagazzinata dalla batteria permettendo l'aumento del SOC, l'obiettivo è quello di consentire all'EM di inviare alla batteria la massima potenza possibile in valore assoluto, in frenata. Siccome la scelta della marcia condiziona la velocità angolare del motore, è chiaro che il rapporto di trasmissione scelto influisce indirettamente sull'aumento del SOC in fase di frenata. Scegliendo quindi come stati la velocità e potenza richieste a veicolo, si dovrebbe teoricamente riuscire ad ottenere l'aumento del SOC desiderato. Precisamente in questo capitolo si suppone che l'aumento desiderato del SOC in frenata sia il massimo possibile. Questa ipotesi è quasi sempre vera: l'unica situazione in cui potrebbe non essere corretta è il caso in cui la finestra di SOC considerata sia molto limitata. Ad ogni modo si può affermare che per prove con un β elevato, è sempre la scelta migliore aumentare il più possibile il SOC durante la frenata. Per questa ragione, in questa sezione, vengono analizzate prove a β=0.9.

Chiaramente per creare questo nuovo agente le modifiche apportate al codice sono state maggiori. In primo luogo, è necessario definire una griglia di discretizzazione per ognuna delle due nuove grandezze considerate tra gli stati: velocità e potenza richieste al veicolo. Ogni stato possibile dell'ambiente in frenata corrisponde quindi a un preciso valore di velocità e di potenza. Viene quindi creata una *Q-table* apposita per la frenata, le cui dimensioni dipendono dal passo di discretizzazione scelto per queste due grandezze. Una volta iniziato l'allenamento è necessario per ogni istante temporale ricondurre i valori di velocità e potenza realmente incontrate durante il ciclo di guida ai valori più vicini ad essi delle grandezze discretizzate. In questo modo si riesce a individuare lo stato presente e lo stato successivo. In questo caso, però, lo stato successivo non dipende dall'azione presa, poiché è imposto dal ciclo di guida scelto; la scelta dell'azione influenza solamente il *reward* ottenuto.

Inoltre, si è deciso di imporre una funzione di *reward* e un *discount factor* per la frenata completamente differenti da quelli utilizzati per la trazione. Anche per quanto riguarda questi parametri le scelte

compiute sono state fatte per ottenere dalla frenata la massima variazione di SOC possibile in ogni istate temporale di frenata. Per questa ragione la funzione di *reward* utilizzata è la seguente:

$$r = cost + \frac{SOC(s') - SOC(s)}{k}$$

Dove:

- $cost$ e k sono due costanti positive
- $SOC(s)$ e $SOC(s')$ son i valori del SOC per lo stato presente e per lo stato successivo

Imponendo questa funzione di *reward* è chiaro che si voglia massimizzare proprio la differenza $SOC(s') - SOC(s)$, come è stato detto. La $cost_1$ è utile per garantire che i *reward* siano sempre maggiori di zero: avviene infatti spesso in frenata che la differenza $SOC(s') - SOC(s)$ sia nulla. La $cost_2$ è invece una costante utile per far alzare il basso valore di $SOC(s') - SOC(s)$: essa viene quindi scelta minore di uno. In concomitanza con questa funzione di *reward* è stato utilizzato un *discount factor* bassissimo:

$$\gamma_{frenata} = 0$$

In questo modo l'agente non considera minimamente i *reward* futuri, ma ha come obiettivo solo quello di massimizzare il *reward* che si ottiene immediatamente. Questa è una scelta ragionevole in frenata in quanto l'obiettivo è proprio la massimizzazione della variazione di SOC in ogni istante di tempo. L'utilizzo di un gamma più elevato non sarebbe teoricamente scorretto ma porterebbe ad approssimazioni che potrebbero far peggiorare i risultati finali. A dimostrazione di ciò sono riportati i grafici della variazione di SOC in frenata per due prove a pari parametri, ma con differente $\gamma_{frenata}$. Si nota come con un gamma elevato, pari a 0.9, si ottengano risultati peggiori rispetto all'utilizzo del $\gamma_{frenata}$ nullo. Un risultato analogo è stato ottenuto utilizzando la funzione di *reward* parabola invece che la nuova funzione di *reward* per la frenata.

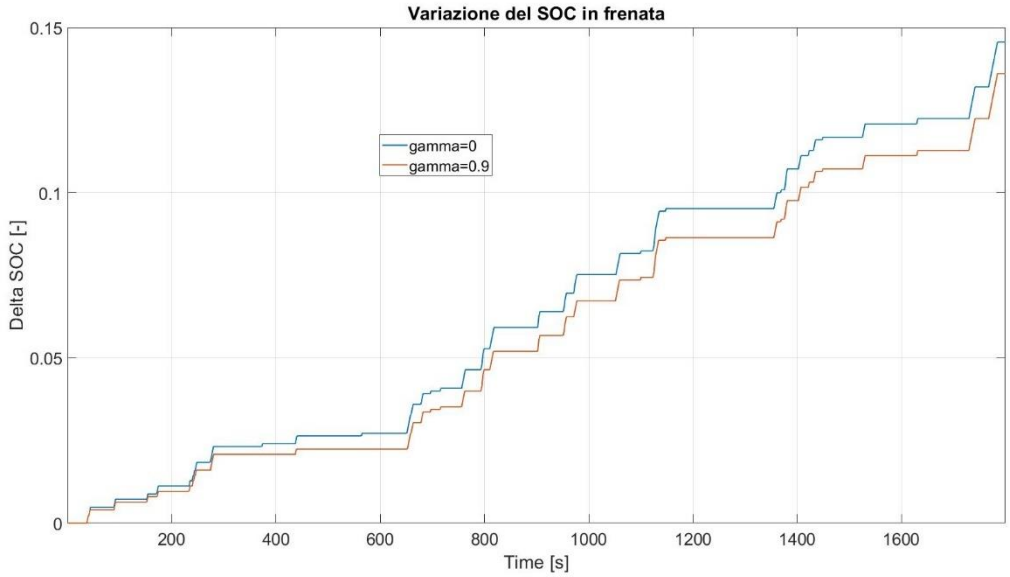

Figura 7.17 Confronto tra le variazioni del SOC in frenata per prove a discount factor diversi

A fine allenamento si ottiene una *Q-table* per la frenata molto differente rispetto a quelle osservate con gli agenti studiati in precedenza. In Figura 7.18, si può osservare una rappresentazione leggermente differente della *Q-table* che permette di apprezzare che solo per alcuni stati i valori della *Q(s,a)* sono diversi al variare dell'azione. Ciò è ragionevole poiché per potenze basse e per determinate velocità del veicolo il cambio di marcia risulta totalmente ininfluente, mentre per potenze più elevate diversi rapporti

di trasmissione fanno variare il SOC in modo differente. Per quanto riguarda la *Q-table* per la trazione e l'andamento del *discounted return*, essi sono molto simili a quelli ottenuti con gli agenti precedenti, dal punto di vista grafico.

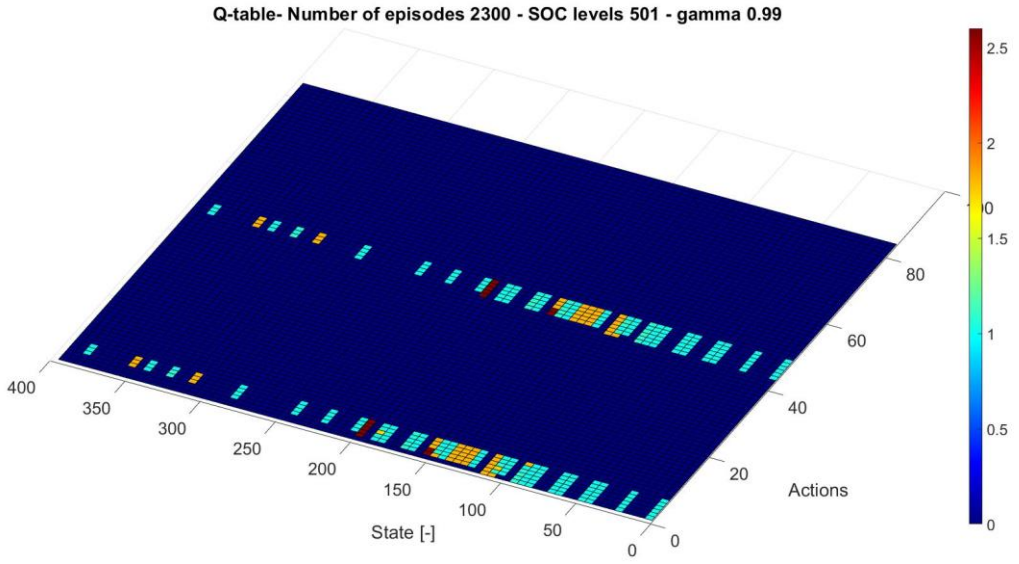

Figura 7.18 Q-table per la frenata con stati potenza e velocità

Ora vengono confrontati i risultati ottenuti con questo agente rispetto a quelli ottenuti con gli agenti visti precedentemente. Vengono quindi considerate tre prove tutte con i medesimi parametri per la trazione, ma agenti differenti: la prima si riferisce all'agente illustrato in questo capitolo, la seconda alla singola *Q-table* per la trazione, e la terza all'agente illustrato nella sezione 7.2 *Q-table* per la frenata. Per verificare che la *Q-table* per la frenata con stati velocità e potenza porti ai risultati desiderati, si confronta in Figura 7.19 la variazione del SOC che si riesce a ottenere in frenata per le tre prove. Si nota che effettivamente la prova che fornisce i migliori risultati da questo punto di vista è proprio la prima.

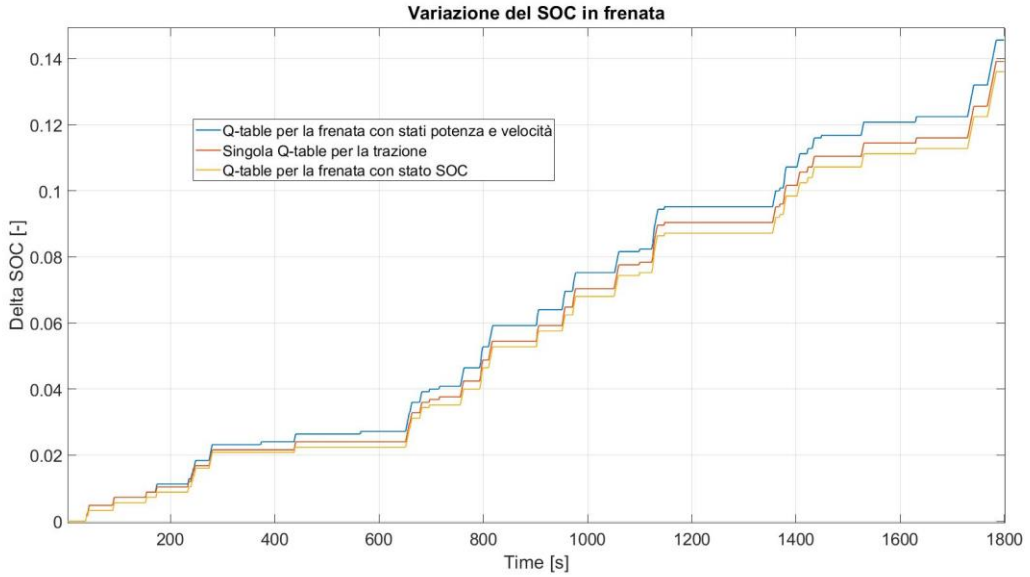

Figura 7.19 Confronto della variazione del SOC in frenata per tre prove con agenti differenti

Confrontando invece i risultati riferiti a tutta la prova e non solo per quanto riguarda la frenata, si può vedere che con il nuovo agente si riesce ad ottenere un FC minore, nonostante il minor SOC finale.

Ricordando che le prove sono state svolte con un β=0.9, questo è un buon risultato. Per avere un confronto quantitativo tra le prove si è dovuto utilizzare la funzione di confronto introdotta nella sezione 6.4.2, a causa della diversa funzione di *reward* utilizzata per la frenata. Le somme dei *reward* equivalenti ottenuti tramite questa funzione sono riportate nella tabella sottostante.

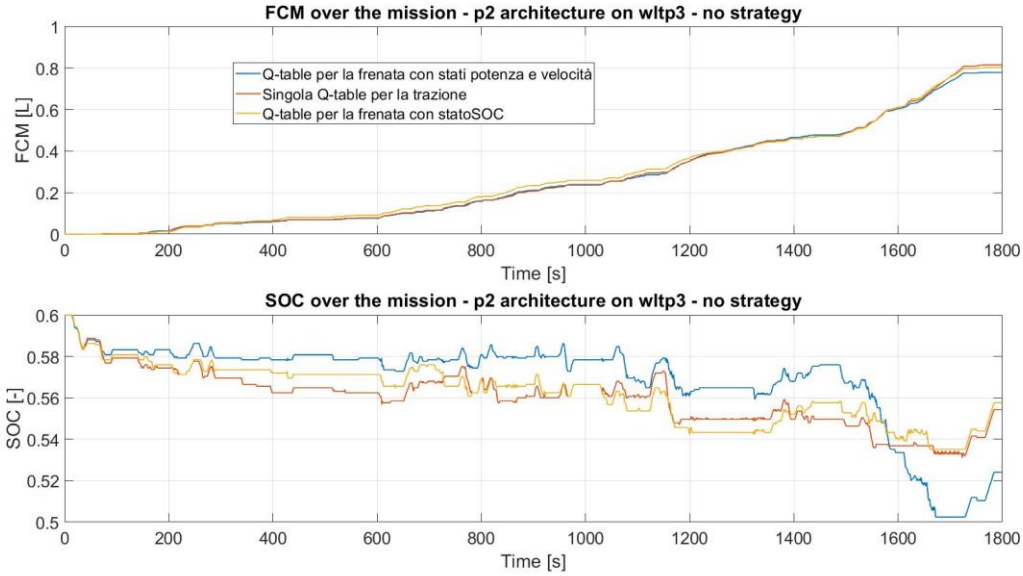

Figura 7.20 Confronto di FC e SOC per prove con stati differenti per la frenata

Tabella 7.3 Somma dei reward di prove con stati differenti per la frenata

Prova considerata	Somma dei *reward*
Q-table per la frenata con stati velocità e potenza	3508.3
Singola *Q-table* per la trazione	3501.8
Q-table per la frenata con stato SOC	3503.8

In conclusione, l'agente analizzato in questa sezione ha portato a ottimi risultati. Durante le fasi di frenata garantisce un recupero del SOC maggiore rispetto a qualsiasi altro agente provato fino a questo momento e in trazione ottiene i medesimi risultati dell'agente con singola *Q-table* per la sola trazione. Di conseguenza, i risultati complessivi ottenuti per prove a β=0.9 con questo agente sono i migliori fino a questo punto del lavoro. Tuttavia, l'apprendimento nella fase di trazione è completamente separato da quello nella fase di frenata.

Visti comunque gli ottimi risultati ottenuti utilizzando come grandezze di stato velocità e potenza richieste dal veicolo, si è deciso di modificare ulteriormente l'agente, cercando di migliorare le prestazioni anche in trazione attraverso l'aggiunta di alcune grandezze agli stati.

8 *Q-table* a tre stati SOC, velocità e variazione di velocità

L'agente studiato nell'ottavo capitolo è il più complesso di questo lavoro di tesi. In quest'ultimo capitolo si è cercato di utilizzare grandezze facilmente misurabili per un veicolo reale e una discretizzazione adatta a ridurre gli errori di discretizzazione. Vengono quindi considerate tre grandezze di stato: il SOC, la velocità V e la variazione di velocità ΔV del veicolo in un intervallo di tempo. L'aggiunta delle ultime due grandezze permette di caratterizzare l'ambiente in modo molto più preciso e completo e di avvicinarlo a un sistema markoviano. Le informazioni riguardanti velocità e variazione di velocità possono influire molto sia sulla scelta della marcia che sulla scelta del PF. Per quanto riguarda la scelta della marcia è naturale pensare che la conoscenza della velocità del veicolo permetta di posizionarsi all'interno della caratteristica dei motori EM e ICE in un punto di lavoro a maggiore efficienza. La variazione di velocità in un intervallo di tempo è invece stata scelta al posto della potenza richiesta al veicolo per mantenere il problema più vicino possibile a quello reale: è infatti molto più semplice misurare in diretta sul veicolo la variazione di velocità, piuttosto che la potenza istantanea richiesta. Ad ogni modo, anche la variazione di velocità fornisce informazioni importanti ed è direttamente collegabile alla potenza, utilizzando anche l'informazione sulla velocità. Su strada piana, la potenza richiesta al veicolo dipende dai contributi legati al rotolamento, alla resistenza aereodinamica e all'inerzia, essendo nullo il termine legato alla pendenza della strada. I primi due termini sono calcolabili grazie alla conoscenza della velocità, e l'ultimo grazie alla conoscenza della variazione di velocità: di conseguenza con l'utilizzo di queste due grandezze di stato si ha anche un'informazione importante sulla potenza. Tale informazione gioca un ruolo importante per la scelta del PF.

Anche in questo caso si è dovuto implementare un nuovo codice per la creazione di questo agente. In primo luogo, è necessario definire il passo di discretizzazione per ognuna delle grandezze appartenenti allo stato dell'ambiente. Tale scelta risulta essere tutt'altro che banale e influenza molto i risultati finali ottenuti. Ogni stato corrisponde quindi a un valore discretizzato di SOC, velocità e variazione di velocità. Successivamente viene creata la *Q-table* che assume dimensioni molto maggiori rispetto a quelle create in precedenza: il numero di stati presenti è pari al prodotto tra il numero di livelli di discretizzazione di ogni grandezza di stato. In ogni istante temporale dell'allenamento va quindi individuato lo stato presente e quello successivo, trovando i valori discretizzati che più si avvicinano ai valori continui delle tre grandezze di stato. L'agente è in ogni caso capace di gestire sia le fasi di frenata che quelle di trazione. L'aggiornamento dei valori delle Q avviene sempre utilizzando come azioni a' le azioni fisicamente *feasible* per l'intervallo di tempo successivo. La prima scelta importante da compiere riguarda quindi la discretizzazione delle grandezze di stato.

8.1 Discretizzazione delle grandezze caratterizzanti lo stato dell'ambiente

8.1.1 Discretizzazione del SOC

Prima di studiare il passo di discretizzazione del SOC è necessario ripetere che in questo capitolo si è deciso di utilizzare una finestra del SOC e un passo di discretizzazione tali da far sorgere bassi errori energetici dovuti alla discretizzazione, rendendo il problema più simile a quello reale. L'obiettivo teorico e puramente ideale sarebbe quello di ottenere una discretizzazione del SOC tale da far cambiare lo stato di carica della batteria ogni qual volta la potenza della batteria sia diversa da zero. Per ottenere ciò sarebbe però necessario l'utilizzo di variabili continue, cosa non possibile con l'algoritmo di *Q-learning* tabulare utilizzato. Di conseguenza è necessario accettare che ci sia un certo valore minimo di potenza della batteria al di sotto del quale non si hanno variazioni dello stato di carica: bisogna fare in modo che tale valore sia sufficientemente basso da comprendere solo una frazione ridotta di tutte le potenze incontrate negli istanti di tempo del ciclo di guida studiato. Per queste ragioni è stata utilizzata

una finestra del SOC di ampiezza minore, rispetto alla precedente, ed è stata diminuita la capacità della batteria E_{batt}. Considerando la finestra e discretizzazione del SOC utilizzate nei capitoli precedenti la potenza minima della batteria per far avvenire una variazione di SOC risultava troppo elevata. Si procede ora con il calcolo di tale potenza, ricordando che la finestra di SOC utilizzata fino ad ora era compresa tra $0.4 < SOC < 0.8$ quindi con un $\Delta SOC_{max} = 0.4$, la $E_{batt} = 3000\,Wh$ e un numero di valori discretizzati di SOC pari a $N_{SOC} = 500$:

$$E_{batt_{min}} = E_{batt} \cdot \Delta SOC_{max} \cdot N_{SOC}$$

$$P_{batt_{min}} = E_{batt_{min}} \cdot 3600s = 8.64\text{kW}$$

Dove:

- $P_{batt_{min}}\, e\, E_{batt_{min}}$ sono rispettivamente la potenza e l'energia minima necessaria per far variare di un livello di discretizzazione il SOC

Analizzando il ciclo di guida WLTP e utilizzando le matrici delle configurazioni calcolate in precedenza, si nota come il numero di istanti di tempo in cui la potenza della batteria è inferiore al valore di $P_{batt_{min}}$ è molto elevato, pari al 48%: ciò significa che nel 48% degli istanti di tempo in cui ci sarebbe dovuto essere una variazione del SOC, questa non avviene. Per questa ragione la finestra del SOC è stata diminuita tra [0.55-0.65] e la $E_{batt} = 2kWh$. Nonostante sarebbe stato più opportuno dal punto di vista energetico aumentare il numero di livelli di discretizzazione del SOC, questo è stato invece abbassato fino a $N_{SOC} = 250$ solo per ridurre il numero degli stati dell'ambiente, garantendo un allenamento più veloce. Si è ottenuta quindi una potenza minima necessaria a far cambiare il SOC pari a:

$$P_{batt_{min}} = 2.88kW$$

Con tale potenza si riesce a ridurre fino al 12% il numero di istanti di tempo con potenza della batteria non nulla per cui non avviene una variazione del SOC. Come citato, tale valore risulta essere un compromesso tra la fedeltà alle condizioni energetiche reali e i tempi di allenamento dell'agente. Se venisse infatti aumentato il numero di livelli di discretizzazione del SOC, aumenterebbe il numero di stati e sarebbe necessario aumentare il numero di episodi di allenamento. Con l'utilizzo di questo nuovo agente, tuttavia, la fase di allenamento risulta particolarmente lunga, a causa del numero più elevato di grandezze di stato; di conseguenza, aumentare ulteriormente il numero di stati aumentando l'N_{SOC} porterebbe a prove ancora più lunghe.

8.1.2 Discretizzazione della velocità e della variazione di velocità

La scelta della discretizzazione della velocità V e variazione della velocità ΔV non comporta errori energetici nel problema, ma risulta di fondamentale importanza per il corretto apprendimento dell'agente, come si vedrà successivamente nel confronto tra prove a differenti livelli di discretizzazione. Per eseguire correttamente tali scelte sarebbe necessario tenere in considerazione diverse grandezze e le caratteristiche dei due motori. Sicuramente risulta opportuno studiare con attenzione la caratteristica dell'ICE tenendo conto che la scelta della discretizzazione influisce sulla scelta delle marce e di conseguenza sull'efficienza del motore. In questo lavoro si adotta però un metodo approssimativo basato sulla potenza della batteria. In primo luogo, si parte con il calcolo della potenza P_v richiesta al veicolo per ogni valore di V e ΔV compresi tra i valor massimi e minimi di V e ΔV effettivamente presenti nel ciclo studiato. Per farlo è necessario utilizzare le formule presentate nel capitolo 3 riguardanti la resistenza di rotolamento, la resistenza inerziale e la resistenza aereodinamica. Si ottiene di conseguenza il grafico successivo.

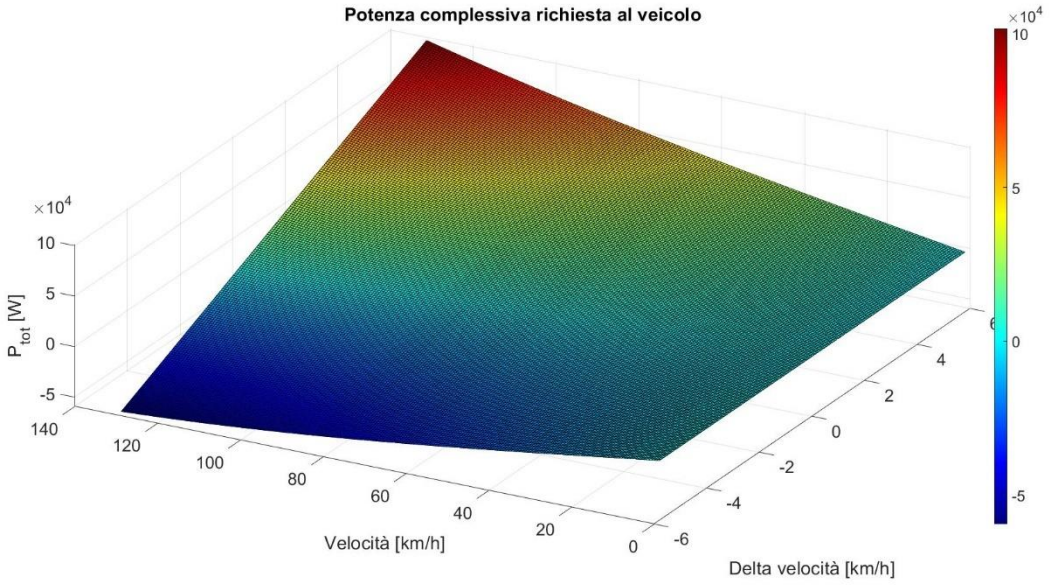

Figura 8.1 Potenza richiesta al veicolo in funzione della velocità e del delta velocità

Successivamente si calcola la potenza richiesta alla batteria a partire dalla potenza del veicolo considerando un PF puro elettrico. Bisogna suddividere il caso della trazione da quello della frenata poiché risulta necessario considerare le efficienze dei componenti intermedi tra ruote e batteria: differenziale η_d, trasmissione del cambio η_c, inverter η_{inv} e motore elettrico η_{EM}. Per semplicità, viene preso un valore realistico, ma costante di ognuna di queste efficienze e viene calcolata la potenza che la batteria dovrebbe fornire in trazione $P_{batt_{trazione}}$ e ricevere in frenata $P_{batt_{frenata}}$ per ogni coppia di V e ΔV.

$$P_{batt_{trazione}} = \frac{P_v}{\eta_d \cdot \eta_c \cdot \eta_{inv} \cdot \eta_{EM}}$$

$$P_{batt_{frenata}} = P_v \cdot \eta_d \cdot \eta_c \cdot \eta_{inv} \cdot \eta_{EM}$$

Ora facendo uso della potenza minima della batteria necessaria a far avvenire una variazione del SOC, si può calcolare per ogni coppia di V e ΔV, il numero $N_{\Delta SOC}$ di livelli di SOC di cui varia quest'ultimo se è applicata la potenza della batteria calcolata. Per la frenata e per la trazione, si ottengono di conseguenza due grafici differenti, riferiti proprio al $N_{\Delta SOC}$ in funzione di V e ΔV. Chiaramente è necessario considerare la potenza massima della batteria, che limita il $N_{\Delta SOC}$ massimo. Come si può notare, il numero di livelli di cui cambia il SOC risulterebbe più elevato per la trazione rispetto alla frenata se non ci fosse il limite imposto dalla potenza massima della batteria. Di conseguenza ammettendo di eseguire una discretizzazione uniforme sia per V che per ΔV, è necessario studiare più approfonditamente la trazione.

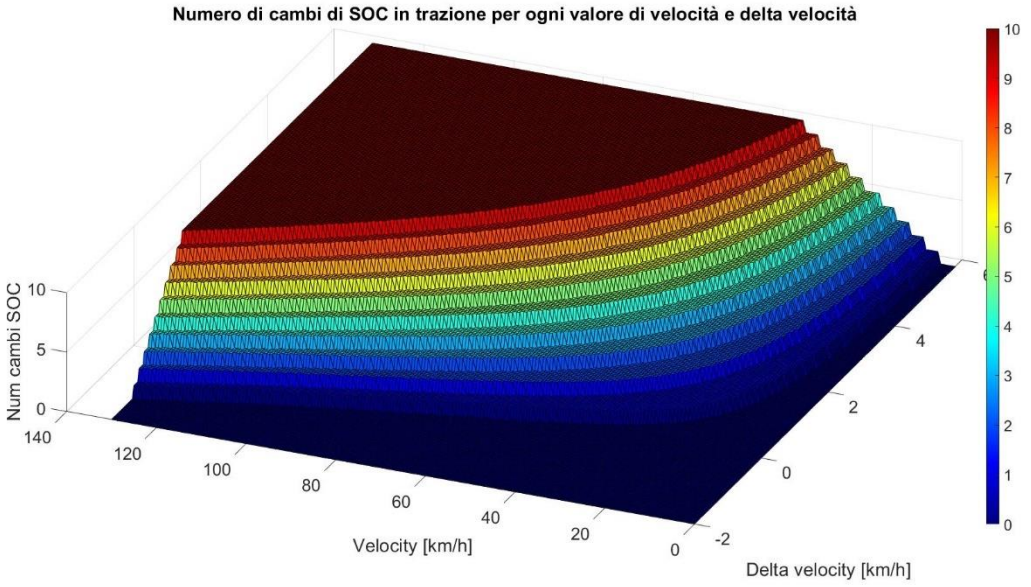

Figura 8.2 Numero di cambi di SOC in trazione

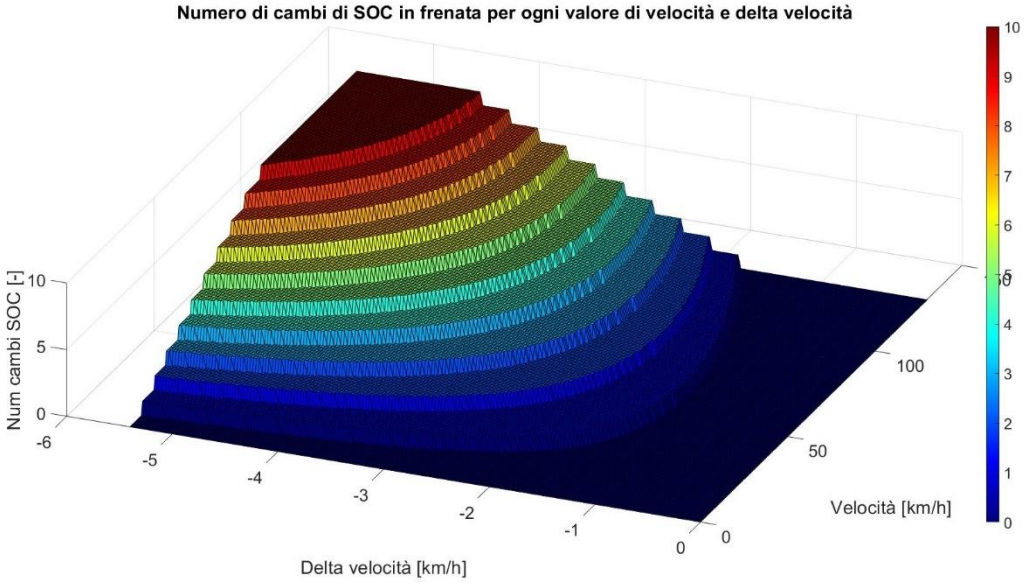

Figura 8.3 Numero di cambi di SOC in frenata

Conoscendo i valori dei V e ΔV del ciclo che il veicolo deve compiere, si sceglie arbitrariamente il livello di discretizzazione delle due grandezze. A questo punto si può vedere direttamente sul grafico se valori di V e ΔV non discretizzati appartenenti allo stesso livello di discretizzazione portano a variazioni di SOC differenti. Siccome il passo di discretizzazione deve essere mantenuto relativamente elevato per ragioni di carattere computazionale, ciò è sempre vero. Bisogna quindi decidere la massima differenza di variazione di livelli di SOC accettabile per coppie di V e ΔV non discretizzati appartenenti allo stesso livello di discretizzazione. Una volta determinato questo valore si può effettivamente determinare la griglia di discretizzazione desiderata. Questo metodo empirico è tuttavia molto legato al particolare ciclo studiato. Viene quindi elaborato un metodo alternativo. Per semplicità si consideri solo la velocità: osservando il grafico, si può vedere il valore minimo della differenza tra due valori di V non discretizzati a pari ΔV per cui si verifica che la differenza tra le variazioni del SOC riferiti a quei due valori di V e

ΔV è pari ad uno. Imponendo il passo di discretizzazione uguale alla minima differenza tra le V trovata in questo modo, si è sicuri di ottenere la miglior discretizzazione possibile seguendo questo ragionamento. Gli stessi passaggi logici possono essere applicati per la scelta del passo di ΔV. Tuttavia, ciò non è applicato a tutte le prove svolte poiché comporterebbe un passo di discretizzazione troppo basso con un conseguente numero di stati e un numero di episodi di allenamento troppo elevati.

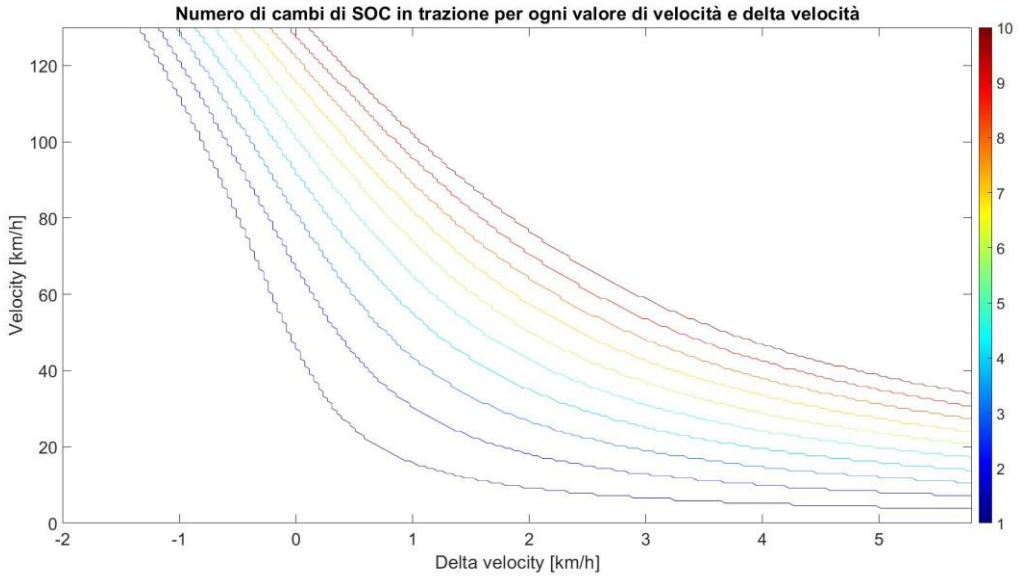

Figura 8.4 Contour del numero di cambi di SOC in trazione

8.2 Andamento dell'allenamento e *Q-table*

Per tutte le prove analizzate l'andamento dell'allenamento risulta ottimo: il *discounted return* ha un andamento crescente e tende a un valore di convergenza, come si noterà nelle sezioni successive. Il grafico 3-D della *Q-table* che si ottiene a fine allenamento ha un aspetto completamente differente rispetto a quelli osservati precedentemente ed è molto meno intuitivo da interpretare. Ogni stato, infatti, non fa riferimento soltanto a un valore del SOC, bensì anche a uno specifico valore di velocità e variazione di velocità. Nella Figura 8.5, vi è la rappresentazione 3-D della *Q-table* ottenuta a fine allenamento di un ciclo *longwltp3* con β=0.5. Come si può notare è impossibile interpretare la *Q-table* con questa rappresentazione e risulta perciò molto utile considerare solo delle porzioni di *Q-table*.

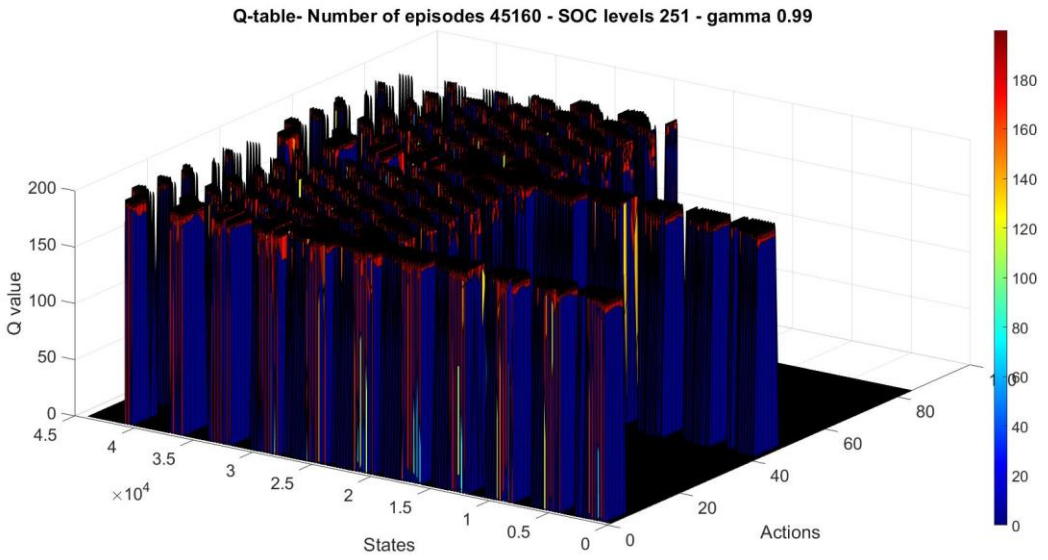

Figura 8.5 Q-table a tre stati: SOC, velocità e delta velocità

Per come è stato impostato l'algoritmo, è possibile isolare una porzione di *Q-table*, ottenendo una rappresentazione molto più facilmente leggibile. Si possono infatti individuare un numero pari a N_{SOC} di stati: ognuno di essi si riferisce al medesimo valore di velocità e variazione di velocità, ma a un differente valore di SOC. Ritroviamo quindi una porzione di *Q-table* molto simile alle *Q-table* ottenute in precedenza. Se si prende in considerazione una velocità e una variazione di velocità che corrispondono alla frenata come in Figura 8.6, la porzione di *Q-table* studiata assume una forma molto simile alla *Q-table* di sola frenata vista nella sezione 7.2. Anche in questo caso gli unici valori della Q diversi da zero sono quelli riferiti ad azioni con PF puramente elettrico; inoltre, tali valori di Q sono sovrastimati come si può vedere dal massimo valore di Q pari a 199.82, ma in misura minore rispetto a quelli di altre prove con previsione delle azioni future.

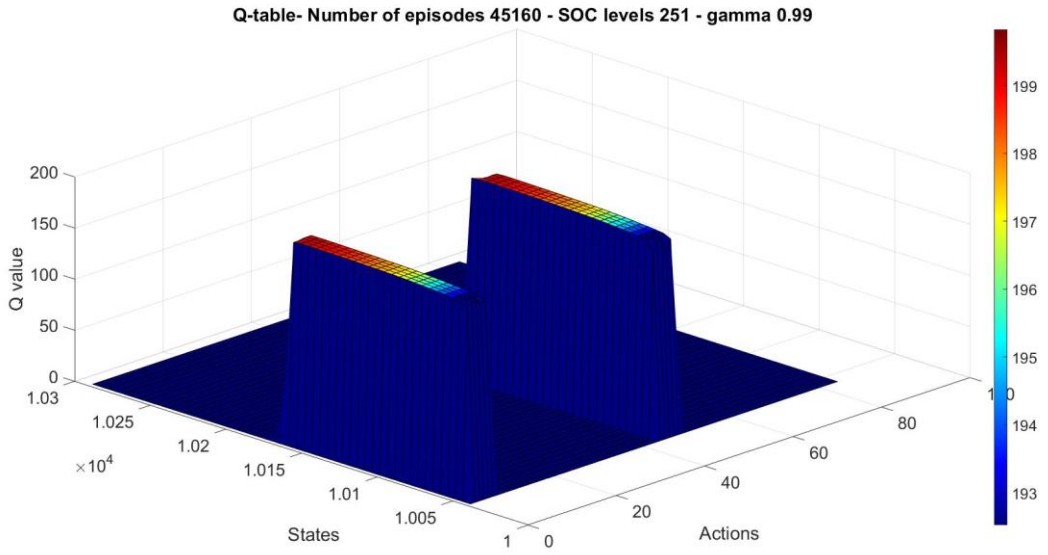

Figura 8.6 Porzione di Q-table riferita a velocità e variazione di velocità in frenata

Se invece viene isolata una porzione di *Q-table* riferita a velocità e variazione di velocità riconducibili a trazione, si nota una grossa differenza rispetto alle *Q-table* di trazione ottenute con altri agenti: solo alcune azioni comportano delle *Q* differenti da zero. Ciò è ragionevole in quanto in questo caso lo stato è collegato alla velocità e per determinati PF e determinate velocità non tutte le marce risultano *feasible*. Si nota che anche per i valori di *Q* riferiti a trazione si raggiungono massimi molto vicini a 200, sintomo di una sovrastima del valore di *Q*.

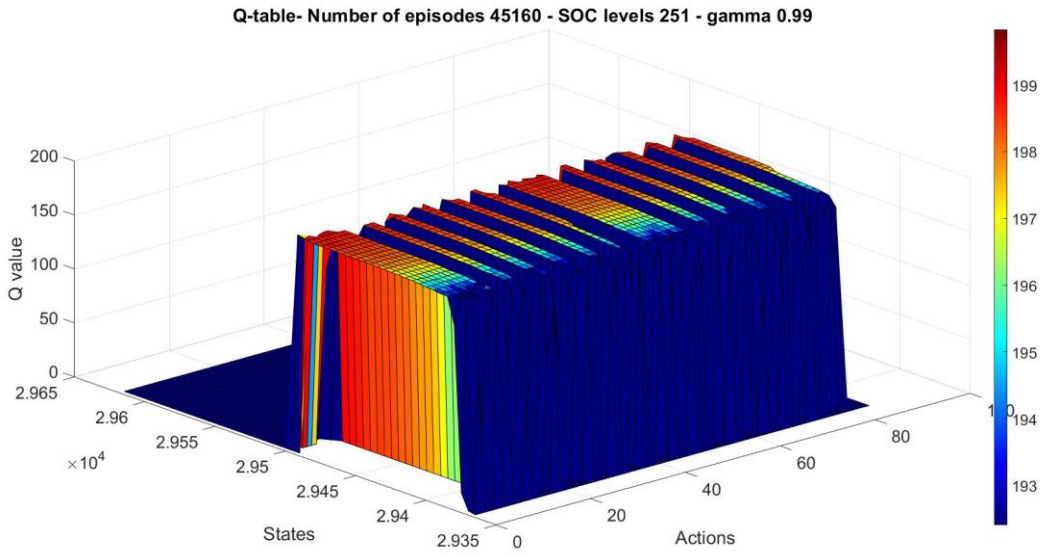

Figura 8.7 Porzione di Q-table riferita a velocità e variazione di velocità in trazione

8.3 Cicli di guida

Come già accennato in precedenza, con questo agente il numero di stati da utilizzare risulta molto più elevato rispetto ai precedenti. Per questa ragione il numero di episodi di allenamento necessario a ottenere un corretto apprendimento dell'agente risulta anch'esso molto elevato. Si sono quindi utilizzati dei cicli di guida più corti per l'allenamento di questo agente, riducendo i tempi di allenamento. In questa sezione vengono quindi riportati i cicli guida utilizzati che sono delle parti del ciclo di guida completo WLTP.

Il ciclo più semplice utilizzato è stato il ciclo *shortwltp3* che comprende i primi 100 secondi del ciclo WLTP. Questo ciclo non presenta condizioni di guida gravose per il veicolo. Avendo una durata pari a un diciottesimo del ciclo completo WLTP garantisce ottimi tempi di allenamento dell'agente. Tuttavia, essendo così breve e con velocità relativamente basse, non porta ad episodi esclusi e risulta quindi molto più semplice l'allenamento dell'agente.

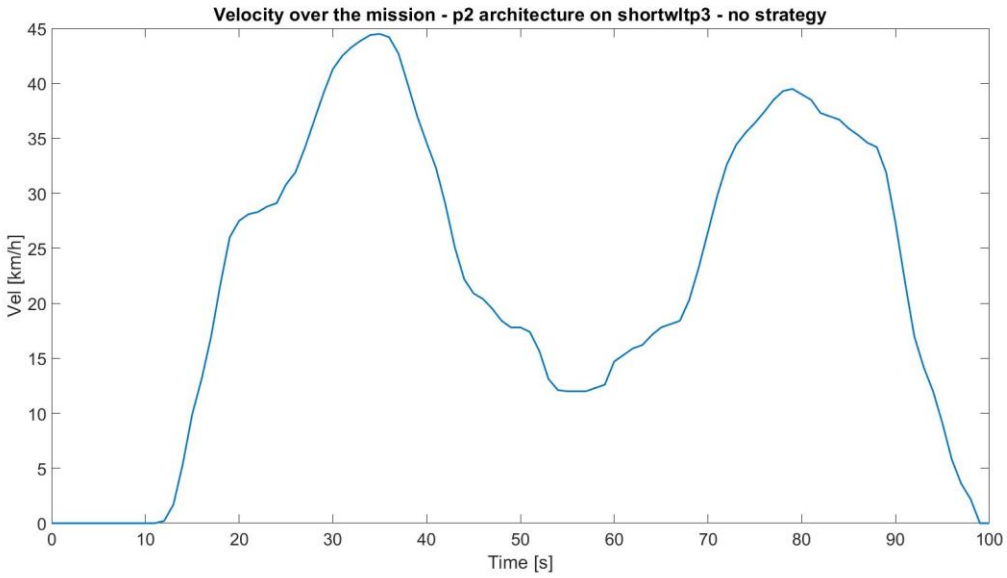

Figura 8.8 Ciclo shortwltp3

Sono stati anche utilizzati altri due cicli più complessi, *midwltp3* e *longwltp3* riportati in seguito. Chiaramente più aumenta la complessità del ciclo, più risulta difficile allenare l'agente e maggiore è il tempo impiegato per completare l'allenamento. Il vantaggio di utilizzare cicli più complessi è quello di poter comprendere le reali potenzialità di un agente di fronte a cicli di guida più vicini a quelli reali.

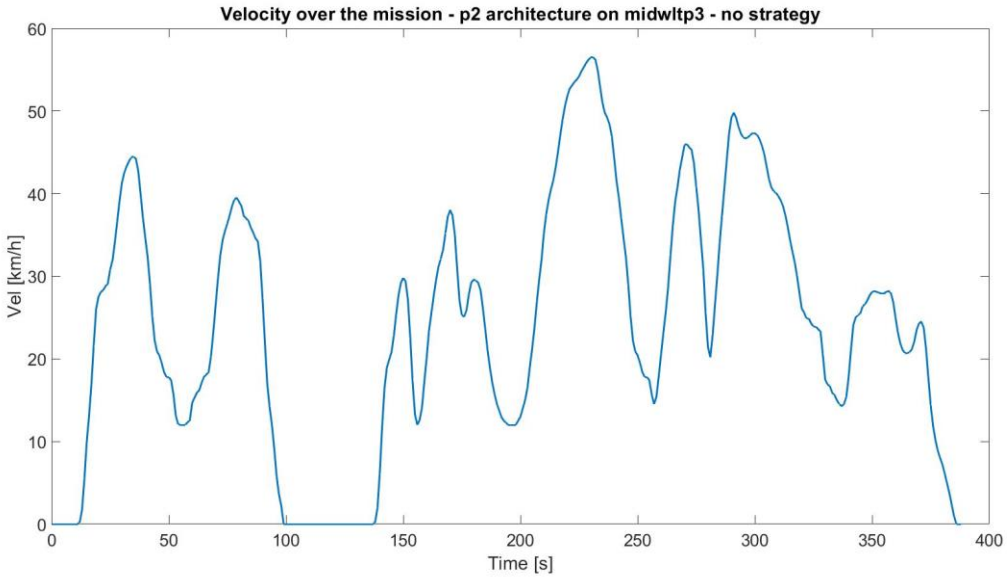

Figura 8.9 Ciclo midwltp3

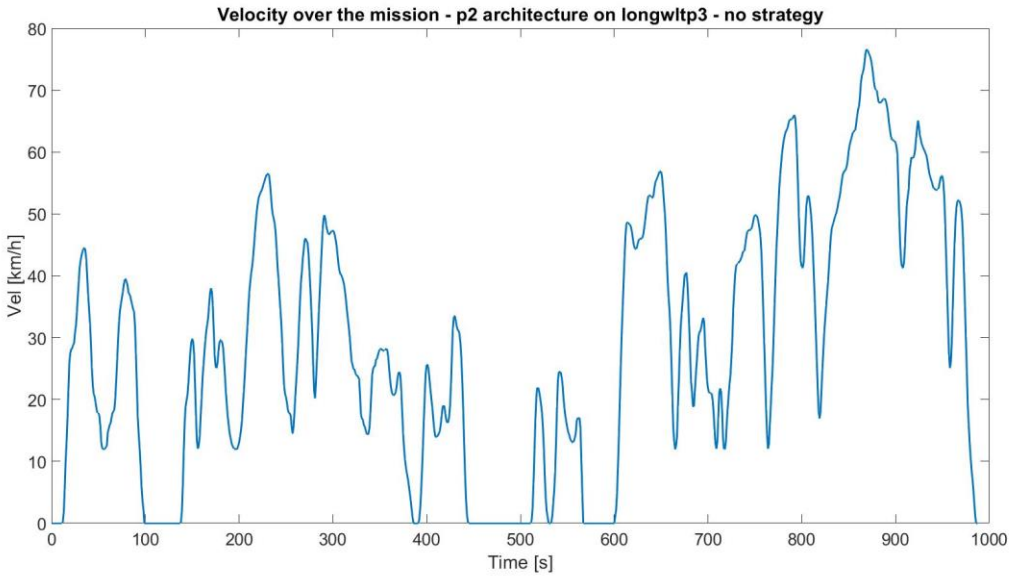

Figura 8.10 Ciclo longwltp3

8.4 Risultati dell'allenamento

I risultati ottenuti dall'allenamento variano molto a seconda del ciclo imposto al veicolo. In questo capitolo viene svolto un confronto tra i risultati ottenuti con diversi livelli di discretizzazione utilizzati a parità di ciclo. Inoltre, si cerca di paragonare i risultati ottenuti con questo agente rispetto a quelli ottenuti con il primo agente utilizzato nel lavoro. Tutte le prove svolte in questo capitolo presentano un β=0.5.

8.4.1 Ciclo *shortwltp3*

Con questo ciclo si sono ottenuti ottimi risultati utilizzando questo agente. Prima di passare al confronto con i risultati ottenuti con l'agente a singola *Q-table* per la sola trazione, si sono svolte delle prove per capire la reale influenza della scelta del passo di discretizzazione sui risultati finali. In primo luogo, è necessario individuare il range entro cui variano la velocità V e variazione di velocità ΔV per questo ciclo:

$$0\ km/h < V < 44.5 km/h$$

$$-4.8\ km/h < \Delta V < 5.4\ km/h$$

Una volta individuati questi valori, è stato utilizzato il metodo descritto in precedenza per determinare il numero ideale di livelli di discretizzazione per la velocità pari a 20 e per la variazione di velocità pari a 17. Essendo un ciclo particolarmente corto è stato possibile provare ad utilizzare un numero di livelli così elevato. Sono state svolte quindi tre prove utilizzando gli stessi parametri, ma differente numero di livelli di discretizzazione per velocità N_V e per variazione di velocità $N_{\Delta V}$: nella prima $N_V = 5$ e $N_{\Delta V} = 9$, nella seconda $N_V = 10$ e $N_{\Delta V} = 9$ e nella terza $N_V = 20$ e $N_{\Delta V} = 17$. Vengono confrontati in Figura 8.11, il consumo di carburante e il SOC ottenuti durante l'episodio finale delle tre prove: si nota come la prova con un numero di livelli di discretizzazione minore presenti risultati peggiori sia in termini di FC che di SOC.

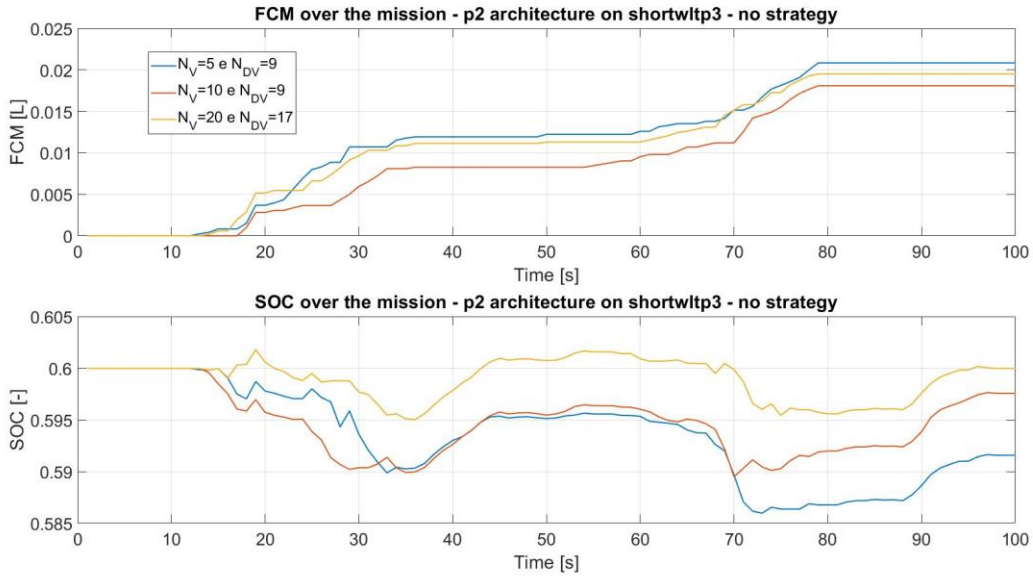

Figura 8.11 Confronto di SOC e FC per diversi livelli di discretizzazione per ciclo shortwlpt3

Anche l'andamento del *discounted return* è molto diverso tra le varie prove e si dimostra nettamente migliore per la prova con il maggior numero di livelli di discretizzazione. Ad ogni modo per la seconda e terza prova si osserva un ottimo andamento dell'allenamento. Nella figura seguente sono confrontati i *discounted return* ottenuti nella prima e terza prova rispettivamente a sinistra e destra nell'immagine. Anche aumentando il numero di episodi di allenamento per la prima prova, non si sono ottenuti risultati migliori per un numero di livelli di discretizzazione così basso.

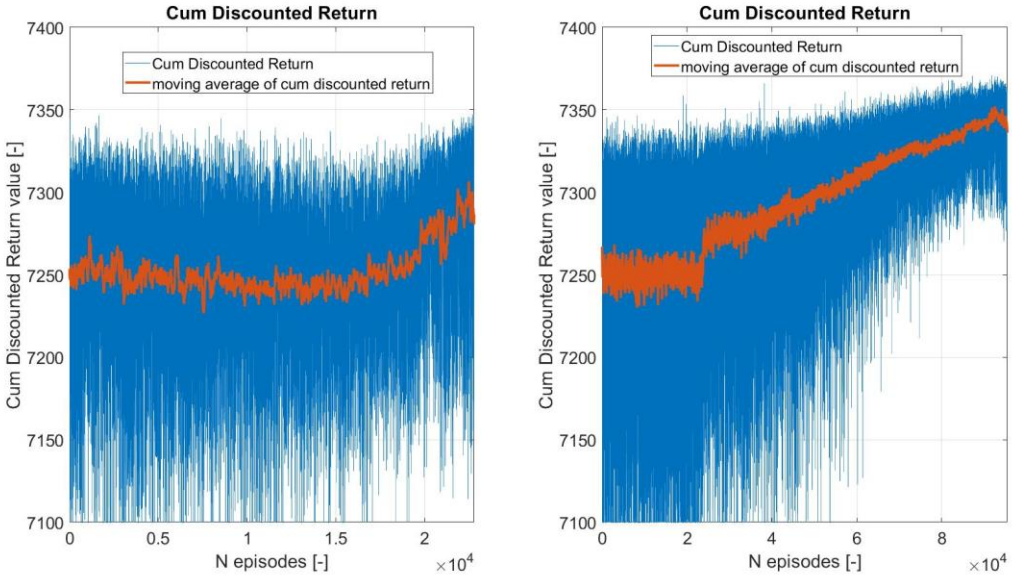

Figura 8.12 Confronto di discounted return per diversi livelli di discretizzazione, ciclo shortwltp3

Confrontando quindi i risultati della prova migliore ottenuta con questo agente con la prova a pari parametri ma con agente con singola *Q-table* per la sola trazione e a un solo stato, si può notare che la prova migliore è quella riferita alla *Q-table* con tre grandezze di stato. Nonostante il SOC finale per tale prova sia minore, è pari al SOC iniziale e il consumo di carburante risulta minore. Anche confrontando la somma dei reward ottenuti si può arrivare alla medesima conclusione.

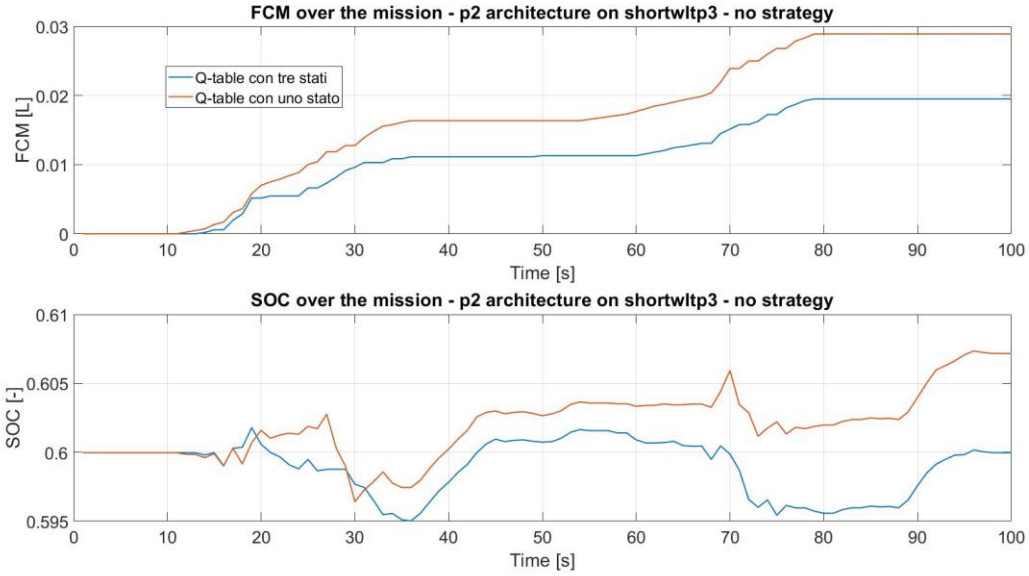

Figura 8.13 Confronto di FC e SOC per diversi agenti per il ciclo shortwltp3

In conclusione, si può affermare che i risultati ottenuti con il nuovo agente risultano davvero ottimi per quanto riguarda il ciclo *shortwltp3*. Tuttavia, essendo questo un ciclo particolarmente semplice, bisogna verificare l'efficacia dell'agente su cicli più complessi.

8.4.2 Ciclo *midwltp3*

Prima di analizzare le prove svolte con questo ciclo, è necessario individuare il range entro cui variano la velocità V e variazione di velocità ΔV per questo ciclo:

$$0\ km/h < V < 56.5 km/h$$

$$-5.6\ km/h < \Delta V < 5.4\ km/h$$

La discretizzazione corretta da applicare in questo caso, ricavata secondo il metodo definito in precedenza, corrisponderebbe ad applicare un $N_V = 22$ e $N_{\Delta V} = 21$. Tuttavia, per questioni di tempo necessario per l'allenamento, la prova è stata svolta con i valori accettabili di $N_V = 12$ e $N_{\Delta V} = 11$.

Nonostante il ciclo *midwltp3* sia un ciclo di complessità maggiore rispetto a quello *shortwltp3*, l'andamento del *discounted return* nelle prove effettuate si dimostra comunque crescente e tende a un valore di convergenza, come mostrato in Figura 8.14. L'allenamento sembra quindi essere efficace.

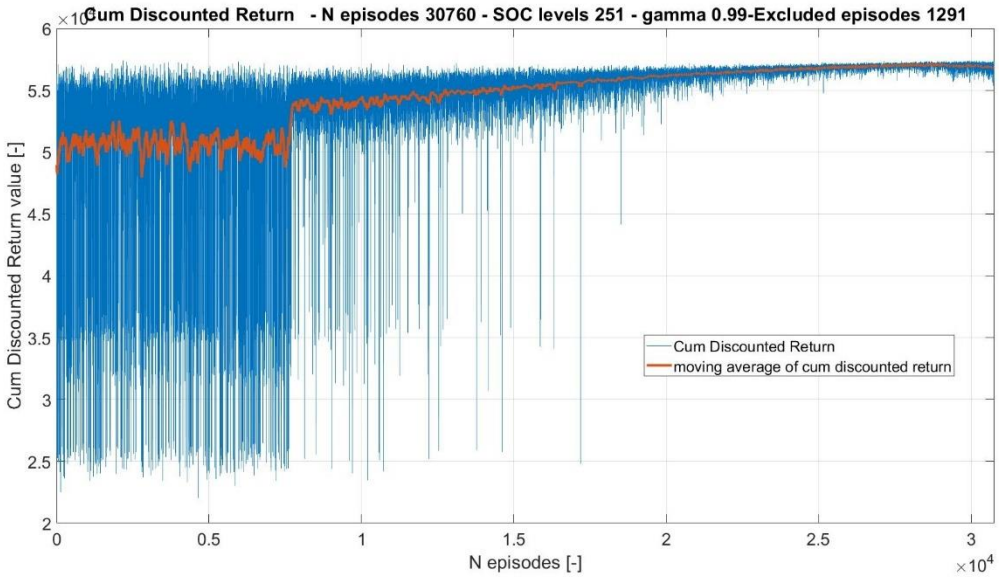

Figura 8.14 Discounted return per una prova per il ciclo midwltp3

Si effettua ora il confronto con una prova svolta con i medesimi parametri, ma con l'agente iniziale in grado di gestire la sola trazione. Si può notare come l'agente descritto in questa sottosezione porti a risultati finali peggiori: il SOC finale risulta inferiore e l'FC maggiore.

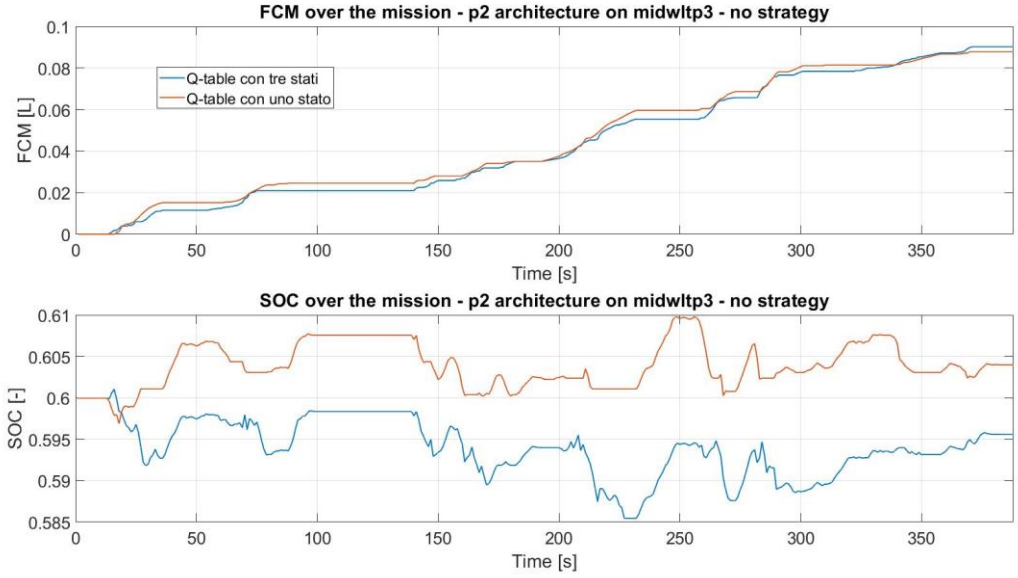

Figura 8.15 Confronto FC e SOC di prove con diversi agenti per il ciclo midwltp3

8.4.3 Ciclo *longwltp3*

Anche per il ciclo *longwltp3* sono stati ottenuti risultati molto simili al ciclo *midwltp3*. Essendo il ciclo ancora più complesso, la discretizzazione ideale avrebbe previsto dei N_V e $N_{\Delta V}$ elevatissimi. È stato quindi necessario ridurli per poter svolgere le prove fino a $N_V = 16$ e $N_{\Delta V} = 11$, in quanto i range entro cui variano la velocità V e variazione di velocità ΔV per questo ciclo sono:

$$0\ km/h < V < 76.6\ km/h$$

$$-5.6\ km/h < \Delta V < 5.8\ km/h$$

L'andamento del *discounted return* è anche in questo caso buono, evidenziando l'apprendimento corretto dell'agente. Tuttavia, i risultati finali ottenuti sono peggiori rispetto a quelli ottenuti con l'agente utilizzato all'inizio del lavoro di tesi, sia per quanto riguarda il SOC, sia per quanto riguarda l'FC.

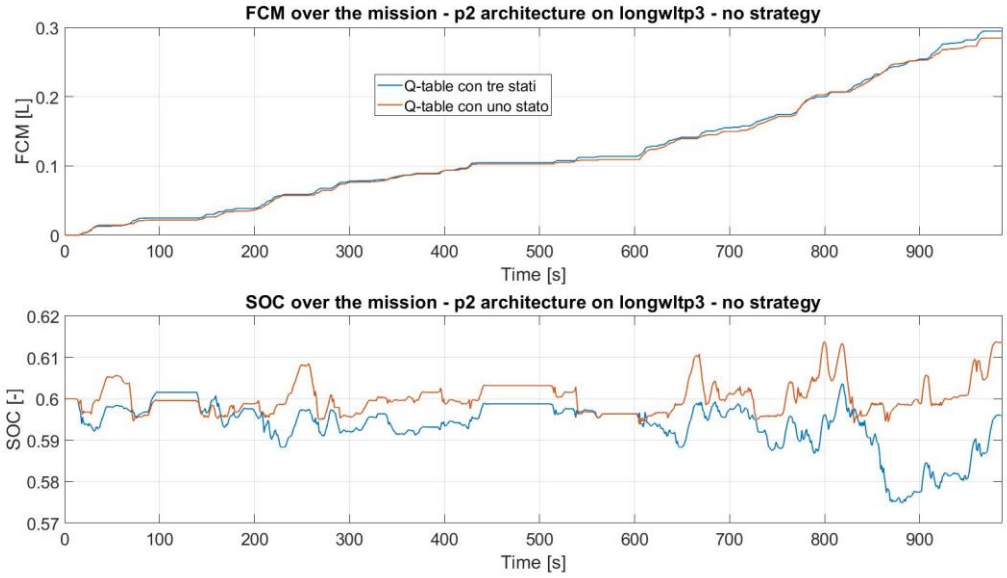

Figura 8.16 Confronto di FC e SOC per prove con diversi agenti per il ciclo longwltp3

In conclusione, l'agente preso in considerazione in questo caso non permette di ottenere ottimi risultati per cicli di guida relativamente complessi, con i passi di discretizzazione e il numero di episodi utilizzati nelle prove mostrate. Tuttavia, bisogna comunque tenere presente come non si possano trarre conclusioni generali riguardo a questo agente basandosi sulle poche prove effettuate. Sarebbe infatti possibile aumentare il numero di livelli di discretizzazione per i cicli più lunghi oppure aumentare il numero di episodi di allenamento facendo variare e probabilmente migliorare molto i risultati, esattamente come è avvenuto per il ciclo *shortwlpt3*. Tuttavia, i lunghissimi tempi di allenamento dell'agente non hanno permesso di tentare prove di questo tipo.

8.5 Variante: aggiornamento della *Q-table* solo in trazione

In questa sezione viene proposta una variante dell'ultimo agente studiato. L'unica differenza apportata riguarda la fase di aggiornamento dei valori di Q. Siccome l'agente precedente porta a una sovrastima di tali valori, si è pensato di aggiornare i valori della *Q-table* solo per intervalli di tempo di trazione come era stato fatto con l'agente utilizzato all'inizio della tesi, mantenendo tuttavia le tre grandezze di stato scelte. In questo modo si ottengono effettivamente valori delle Q minori rispetto a quelli visti con la *Q-table* aggiornata sia in trazione che in frenata e risultati complessivamente migliori a parità di numero di episodi per i cicli più complessi midwltp3 e longwltp3. Un grosso vantaggio dell'utilizzo di questo metodo è legato alla discretizzazione. Aggiornando i valori di Q solo per la trazione, è possibile considerare il limite inferiore della grandezza variazione di velocità vicino a 0km/h. Ciò permette di utilizzare un passo di discretizzazione della variazione di velocità pari a metà del passo che si sarebbe utilizzato aggiornando la *Q-table* sia in trazione che in frenata, a pari numero di livelli di discretizzazione.

Tuttavia, l’utilizzo di questa variante porta con sé lo svantaggio legato alla perdita del collegamento tra trazione e frenata concentrandosi solo sulla fase di trazione. Il problema della frenata non è quindi risolto completamente.

9 Conclusioni

In questo lavoro di tesi è stata studiata la particolare applicazione di un algoritmo di *Q-learning* tabulare al complesso problema del controllo di veicoli ibridi elettrici. Gli obiettivi fisici che si desidera ottenere attraverso l'allenamento dell'agente sono in questo caso la minimizzazione del consumo di carburante (FC) e il controllo dello stato di carica della batteria (SOC). Nella prima parte del lavoro, sono state confrontate differenti funzioni di *reward* atte al raggiungimento di tali obiettivi fisici desiderati. Si è quindi deciso di utilizzare la funzione "Parabola normalizzata" con la quale si sono ottenuti ottimi risultati: essa permette di controllare efficacemente, attraverso un coefficiente di peso β compreso tra zero e uno, quale obiettivo fisico si vuole privilegiare tra i due individuati. È stata quindi svolta un'analisi parametrica del rapporto *exploration-exploitation* e del *learning rate* in funzione del coefficiente β scelto, in modo da definire i valori ottimali di tali parametri. In questa prima parte del lavoro, viene proposta l'idea di utilizzare un *learning rate* decrescente con il numero di episodi: ciò ha permesso di ottenere risultati significativamente migliori nelle prove a β elevati. Per completare il lavoro riguardante l'analisi parametrica, in futuro si potrebbe studiare in modo più approfondito anche l'influenza del *discount factor* sui risultati finali al variare del coefficiente di peso scelto.

Successivamente, sono state aggiunte due funzioni al codice originale. La prima consente di fermare l'allenamento dell'agente quando il *discounted return* giunge a convergenza, permettendo di ottenere una riduzione del tempo di allenamento e migliorando i risultati finali. L'altra funzione è invece molto utile per l'analisi di prove concluse: permette di confrontare due prove svolte con funzioni di reward differenti per determinare quale abbia portato ai risultati migliori.

Nella parte successiva, il lavoro si incentra sull'implementazione di un agente in grado di gestire efficacemente le fasi di frenata. Sono state quindi tentate differenti soluzioni che hanno portato a risultati interessanti. In primo luogo, l'utilizzo di una *Q-table* aggiuntiva aggiornata solo in fasi di frenata ha permesso di ottenere una maggiore somma dei reward rispetto alle prove con singola *Q-table* per determinati β. Anche l'agente caratterizzato da un PF aggiuntivo per la frenata rigenerativa ha fornito simili risultati per due delle configurazioni proposte per questo agente. Tuttavia, non è stato possibile concludere che tali agenti garantiscano sempre risultati migliori a causa delle loro prestazioni non ottimali nelle fasi di frenata, per quanto riguarda la variazione del SOC. L'aggiunta di un PF di frenata rigenerativa e di un PF di frenata non rigenerativa si è dimostrata utile se si vuole raggiungere un SOC finale il più vicino possibile al SOC iniziale, obiettivo leggermente differente da quello definito a inizio lavoro per quanto riguarda il SOC. Infine è stato sviluppato un agente più complesso, dotato di una *Q-table* per la sola frenata in cui si considerano come grandezze di stato dell'ambiente la velocità e la potenza richiesta al veicolo al posto del SOC. Siccome in frenata la scelta dell'azione corrisponde alla sola scelta della marcia e stato di accensione del motore, l'utilizzo di velocità e potenza come grandezze di stato ha portato a ottimi risultati per le fasi di frenata del veicolo: sviluppando una funzione di *reward* apposita per la frenata e utilizzando un *discount factor* nullo è stato possibile ottenere variazioni di SOC in frenata maggiori rispetto agli altri agenti considerati. Lo svantaggio di questo agente è la mancanza di collegamento tra l'apprendimento in trazione e in frenata. In conclusione, si può affermare che nessuno degli agenti proposti risolve completamente il problema relativo alla frenata. Tuttavia, molti di essi hanno portato a risultati interessanti in termini di miglioramento delle prestazioni: in particolare l'ultimo agente descritto ha fornito risultati nettamente migliori per quanto riguarda la fase di frenata.

Avendo notato il netto miglioramento dei risultati grazie all'aggiunta di grandezze di stato differenti dal SOC per caratterizzare l'ambiente, è stato sviluppato un agente che riceve informazioni riguardanti tre grandezze di stato, per gestire sia la fase di trazione che quella di frenata: SOC, velocità e variazione di velocità del veicolo. Ognuna di queste grandezze viene discretizzata secondo una propria griglia di discretizzazione. Mentre il criterio di scelta del passo di discretizzazione del SOC è basato sull'obiettivo di minimizzare gli errori energetici dovuti alla discretizzazione stessa, il passo di discretizzazione delle altre due grandezze viene determinato secondo un metodo approssimativo legato alla discretizzazione scelta per il SOC e allo studio della caratteristica dell'ICE. Si sono quindi svolte prove con un ciclo di guida breve per determinare l'influenza della scelta del passo di discretizzazione e del numero di episodi sui risultati finali ottenuti. Scegliendo un passo di discretizzazione sufficientemente piccolo per velocità

e variazione di velocità, sono stati ottenuti risultati ottimi per prove con cicli brevi: il nuovo agente permette di ottenere risultati migliori dei precedenti. Per cicli lunghi, sono stati scelti passi di discretizzazione nettamente maggiori rispetto a quelli ideali calcolati attraverso il metodo approssimativo introdotto, per questioni legate al tempo di allenamento dell'agente: i conseguenti risultati ottenuti sono peggiori rispetto alle corrispondenti prove svolte con i vecchi agenti. Infine, è stata implementata una versione alternativa dell'agente in grado di apprendere solo in fasi di trazione: essa ha fornito ottimi risultati anche per cicli più lunghi. Come proseguimento naturale del lavoro si potrebbe continuare l'analisi di quest'ultimo agente introdotto, utilizzando per le prove con cicli complessi un passo di discretizzazione minore rispetto a quelli utilizzati e un numero di episodi di allenamento maggiore. I risultati dovrebbero migliorare in modo evidente, proprio come è avvenuto nel ciclo breve.

10 Bibliografia

[1] Consiglio dell'Unione Europea, *Nuova strategia dell'UE in materia di sviluppo sostenibile*, 2006 "@ www.treccani.it." [Online], https://www.treccani.it/enciclopedia/mobilita-sostenibile_%28Lessico-del-XXI-Secolo%29/.

[2] J. Miller, L. Du, and D. Kodjak, "Impacts of world-class vehicle efficiency and emissions regulations in select G20 countries," no. January, 2017.

[3] International Energy Agency (IEA), Global-Ev-Outlook-2020 "@ Www.Iea.Org." https://www.iea.org/reports/global-ev-outlook-2020.

[4] J. P. Torreglosa, P. Garcia-Triviño, D. Vera, and D. A. López-García, "Analyzing the improvements of energy management systems for hybrid electric vehicles using a systematic literature review: How far are these controls from rule-based controls used in commercial vehicles?," *Appl. Sci.*, vol. 10, no. 23, pp. 1–25, 2020, doi: 10.3390/app10238744.

[5] Z. Chen, H. Hu, Y. Wu, Y. Zhang, G. Li, and Y. Liu, "Stochastic model predictive control for energy management of power-split plug-in hybrid electric vehicles based on reinforcement learning," *Energy*, vol. 211, p. 118931, 2020, doi: 10.1016/j.energy.2020.118931.

[6] H. Lee, C. Song, N. Kim, and S. W. Cha, "Comparative Analysis of Energy Management Strategies for HEV: Dynamic Programming and Reinforcement Learning," *IEEE Access*, vol. 8, pp. 67112–67123, 2020, doi: 10.1109/ACCESS.2020.2986373.

[7] Z. Kong, Y. Zou, and T. Liu, "Implementation of real-time energy management strategy based on reinforcement learning for hybrid electric vehicles and simulation validation," *PLoS One*, vol. 12, no. 7, pp. 1–16, 2017, doi: 10.1371/journal.pone.0180491.

[8] Y. Hu, W. Li, K. Xu, T. Zahid, F. Qin, and C. Li, "Energy management strategy for a hybrid electric vehicle based on deep reinforcement learning," *Appl. Sci.*, vol. 8, no. 2, 2018, doi: 10.3390/app8020187.

[9] X. Han, H. He, J. Wu, J. Peng, and Y. Li, "Energy management based on reinforcement learning with double deep Q-learning for a hybrid electric tracked vehicle," *Appl. Energy*, vol. 254, no. August, p. 113708, 2019, doi: 10.1016/j.apenergy.2019.113708.

[10] Y. Wu, H. Tan, J. Peng, H. Zhang, and H. He, "Deep reinforcement learning of energy management with continuous control strategy and traffic information for a series-parallel plug-in hybrid electric bus," *Appl. Energy*, vol. 247, no. March, pp. 454–466, 2019, doi: 10.1016/j.apenergy.2019.04.021.

[11] R. Lian, J. Peng, Y. Wu, H. Tan, and H. Zhang, "Rule-interposing deep reinforcement learning based energy management strategy for power-split hybrid electric vehicle," *Energy*, vol. 197, p. 117297, 2020, doi: 10.1016/j.energy.2020.117297.

[12] H. Tan, H. Zhang, J. Peng, Z. Jiang, and Y. Wu, "Energy management of hybrid electric bus based on deep reinforcement learning in continuous state and action space," *Energy Convers. Manag.*, vol. 195, no. January, pp. 548–560, 2019, doi: 10.1016/j.enconman.2019.05.038.

[13] J. Wu, H. He, J. Peng, Y. Li, and Z. Li, "Continuous reinforcement learning of energy management with deep Q network for a power split hybrid electric bus," *Appl. Energy*, vol. 222, no. January, pp. 799–811, 2018, doi: 10.1016/j.apenergy.2018.03.104.

[14] Andrew Barto, Richard Sutton, *Reinforcement Learning: An Introduction*, vol. 27, no. 9. 1998.

[15] Eitan Gross, "On the Bellman's principle of optimality," 2016, [Online]. https://www.sciencedirect.com/science/article/abs/pii/S037843711630351X?via%3Dihub.

[16] B. Xu *et al.*, "Parametric study on reinforcement learning optimized energy management strategy for a hybrid electric vehicle," *Appl. Energy*, vol. 259, no. August 2019, p. 114200, 2020, doi: 10.1016/j.apenergy.2019.114200.