POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Enhancing Energy Flexibility in Electric Vehicle Charging Stations using Reinforcement Learning

Tesi di Laurea Magistrale in
Automation and Control Engineering - Ingegneria dell'Automazione

Author: **Antonio Alejandro Aslan Suarez**

Student ID: 10835761
Advisor: Prof. Fredy Ruiz Palacios
Co-advisors: Cesar Diaz Londono
Academic Year: 2024-25

Abstract

As the world shifts towards cleaner energy, the integration of renewable energy sources (RES) poses new challenges for grid stability due to its variable nature. This increased dependence on RES has led to the concept of grid flexibility as a means to maintain equilibrium between supply and demand. Electric Vehicle (EV) charging stations offer a promising avenue to enhance grid flexibility by dynamically adjusting their energy consumption. However, most existing literature focuses on optimizing EV charging to minimize costs rather than maximizing flexibility.

This thesis proposes a reinforcement learning (RL)-based framework to optimize EV charging stations, emphasizing the role of flexibility. By incorporating flexibility into the EV charging model, various RL methods are trained and evaluated using a customized simulation environment. This simulator, built on top of EV2Gym, has been enhanced to include economical rewards when providing flexibility. The performance of the selected RL solutions is compared with Model Predictive Control (MPC) approaches, focusing on cost minimization and flexibility maximization.

The results demonstrate that RL agents can achieve a balance between economic viability and enhanced flexibility, although cannot outperform MPC when forecasted data about EV arrival and electricity prices are completely accurate. These findings indicate that RL may offer a viable alternative for managing EV charging stations when the forecast of said data is uncertain or not available.

Keywords: Reinforcement Learning, Electric Vehicle, Charging Stations, Grid Flexibility

Abstract in lingua italiana

Con il passaggio globale verso un'energia più pulita, l'integrazione delle fonti di energia rinnovabile introduce nuove sfide per la stabilità della rete a causa della loro natura variabile. Questa dipendenza dalle fonti rinnovabile ha messo in evidenza la necessità di flessibilità della rete per mantenere un equilibrio tra domanda e offerta. Le stazioni di ricarica per veicoli elettrici (EV) rappresentano una promettente soluzione al fine di migliorare la flessibilità della rete, regolando dinamicamente il loro consumo energetico. Tuttavia, gli studi esistente si concentrano principalmente sull'ottimizzazione della ricarica dei veicoli elettrici per ridurre i costi, piuttosto che massimizzare la flessibilità.

Questa tesi propone un framework basato sull'apprendimento per rinforzo (RL) per ottimizzare le stazioni di ricarica per EV, con un'enfasi sulla flessibilità. Incorporando la flessibilità nel modello di ricarica EV, sono stati allenati e valutati diversi metodi di RL tramite un ambiente di simulazione personalizzato. Questo simulatore, basato su EV2Gym, è stato migliorato per includere ricompense economiche per la fornitura di flessibilità. Le soluzioni RL selezionate sono state confrontate con approcci di Model Predictive Control (MPC), concentrandosi sia sulla riduzione dei costi sia sulla massimizzazione della flessibilità.

I risultati dimostrano che gli agenti RL possono bilanciare la redditività economica con una maggiore flessibilità; tuttavia, non superano l'MPC quando sono disponibili previsioni precise sugli arrivi degli EV e sui prezzi dell'energia. Questi risultati suggeriscono che l'RL offre un'alternativa valida per la gestione delle stazioni di ricarica EV in scenari in cui le previsioni dei dati siano incerte o non disponibili.
Parole chiave: Apprendimento per Rinforzo, Veicoli Elettrici, Stazioni di Ricarica, Flessibilità della Rete.

Contents

1 | Introduction

As an effort to minimize the impact of global warming, electricity generation is increasingly shifting from fossil fuel power plants to renewable energy sources (RES) such as wind, solar and hydropower [1]. While these sources provide a cleaner and more sustainable solution, they suffer from a main drawback intrinsic to their nature. Which is that power generated by RES cannot be scheduled. This could cause a mismatch between supply and demand on the power grid which, if not dealt with, can lead to grid instability and voltage fluctuations. Thus, the concept of flexibility is introduced as the capability of the grid to maintain this delicate balance between power generation and consumption [2, 3]. On one hand, grid flexibility can be enhanced on the supply side, as thermal, nuclear and hydroelectric plants can adjust their power outputs in cases of demand fluctuations or sudden drops in RES power output. On the other hand, flexibility can also be provided on the demand side by using variable loads or power storage methods such as batteries or hydroelectric plants.

Another side effect of this shift towards sustainability has been the ever-increasing demand for Electric Vehicles (EVs) [4] and thus charging stations, which are controlled by aggregators. Thankfully, by tweaking the power supplied when charging an EV, aggregators can smooth demand curves and thus provide flexibility. One single home-grade charger may not have an impact on grid flexibility, but an aggregator that controls several Direct Current Fast Charging (DCFC) stations which have a reserved power capacity of between 50 and 350kW [5] could aid significantly in maintaining grid balance.

Previously, aggregators relied on simple heuristic strategies such as charging As Fast As Possible (AFAP) or Round Robin (RR) for managing charging stations under parking time constraints. But thanks to the increased availability of future data for electricity prices, aggregators are starting to incorporate more elaborate strategies such as Model Predictive Control (MPC) which finds the most profitable charging trajectory while respecting said time constraints. In fact, in cases where grid conditions can be forecasted

and flexibility is remunerated, MPCs can be extended to minimize charging costs while maximizing flexibility.

Over the last years, the use of Reinforcement Learning (RL) methods for minimizing charging costs has gained significant traction, as these methods could handle scenarios where grid data in highly uncertain or unavailable. However, almost none of the published literature considers flexibility for profits maximization as no standard simulator that includes flexibility has been established yet.

1.1. Thesis Purpose

Due to the absence in literature of RL solutions that consider flexibility in the EV charging problem, the main objective of this thesis shall be to formulate the EV charging problem including flexibility, so that various RL methods can be used to find a strategy that minimizes operating costs and maximizes flexibility at the same time. For this reason, a simulator that includes flexibility should be created to train the various RL agents. Finally, after the RL solutions are developed, they should be compared to the current state of the art solutions to prove their efficacy.

1.2. Thesis Overview

This chapter has been devoted to introducing the grid flexibility problem and justifying the purpose behind this thesis. Chapter 2 will cover the state of the art methods currently used for EV dispatch, along with a more in-depth explanation of the concept of flexibility. As the aim of the thesis is to provide an RL solution, the entirety of chapter 3 will be devoted to providing the mathematical principles behind any general RL algorithm and the four RL solutions will be deeply explained. As said algorithms need a simulator for training, chapter 4 shall cover how an existing simulator has been extended to accommodate for flexibility simulation. In chapter 5, the final results of the various RL methods are presented and discussed, along with a comparison with MPC. Finally, chapter 6 will provide the conclusions that can be drawn from the results and a road-map will be presented with some key points that can guide future development in this area.

2 | State of the Art

Thanks to the increased availability of grid information, charging strategies are rapidly evolving from simple heuristics to more sophisticated approaches that use said data to identify and predict the most profitable charging trajectory at each time instant.

Due to the relative simplicity of the EV charging model, optimization-based control strategies such as MPC can be used for the EV dispatch problem. These strategies solve a numerical optimization problem at each time step to minimize the operating cost of the aggregator over the prediction horizon (PH). Moreover, MPC can be extended to cases were vehicle-to-grid (V2G) power injection is allowed [6] or cases in which regulating grid's flexibility is rewarded.

Although optimization-based methods always find the best solution, they rely on accurate data regarding arrival and departure times for EVs and electricity prices, which in some circumstances may not be reliable or available. For this reason, model-free methods [7–9] based on RL are being studied as a promising alternative as they can be trained on simulated scenarios using real world data and find strategies that do not rely on said forecasts.

2.1. Economic MPC

The economic MPC (eMPC) [10] is a strategy used by aggregators that takes advantage of future knowledge to find the most profitable charging trajectory by solving a constrained minimization problem at every time step. The optimality of this method is highly dependent on the fidelity of the charging dynamics model and the accuracy of the future values.

2.1.1. MPC Fundamentals

Considering that a model of the system (plant model) is available, MPC uses forecast values to predict future trajectories of the states over the PH and then chooses to apply the

input for the trajectory that minimizes a cost function $J(\cdot)$, which is normally dependent on the control input u and states x for setpoint tracking. Figure 2.1 illustrates that while the calculated input trajectory spans only for a certain number of time steps (which is normally known as control horizon or Hc), the evolution of the states is calculated until the end of the PH, ensuring that said states stays within acceptable values even when no input is being applied.

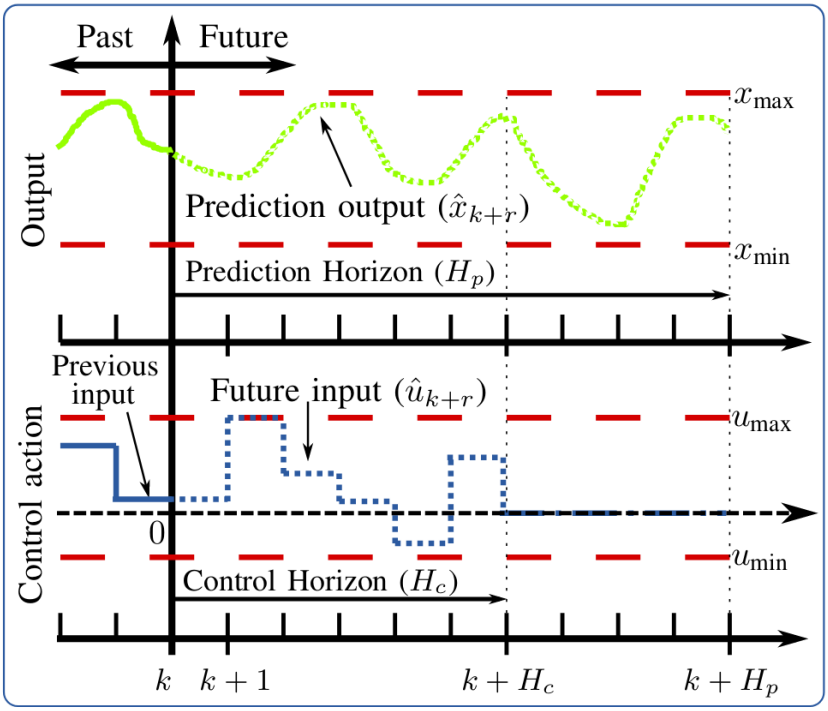

Figure 2.1: Temporal evolution followed by MPC [10]

System definition and prediction

As previously said, MPC relies on having an accurate representation of the plant to generate the trajectories. In this case, a linear time invariant (LTI) system will be considered as in (2.1) where x_k represents the state vector, u_k represents the input vector and y_k collects the measured outputs, which in this case equal to the states as the model is strictly proper with $C = I$. It is worth mentioning that MPC can be extended to cases where the system representation is non-linear. This will not be covered as subsection 2.1.2 shows how the EV charging dynamics can be modeled using an LTI system.

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k \end{aligned} \tag{2.1}$$

Taking system (2.1) with system matrices A, B and C, the evolution of the system can be predicted until PH using the matrix equation $\hat{x} = \mathbb{A}x_{k_0} + \mathbb{G}u$ where $\hat{x}$ a is matrix containing the state evolution until PH, x_{k0} being the initial state vector and d u containing the free control input evolution until Hc. Matrices $\mathbb{A}$ and $\mathbb{G}$ are constructed according to (2.2). Using this approach, the future evolution of the states can be predicted and optimized using u.

$$\mathbb{A} = \begin{bmatrix} A \\ A^2 \\ \vdots \\ A^{H_c} \\ A^{H_c+1} \\ \vdots \\ A^{PH} \end{bmatrix}, \quad \mathbb{G} = \begin{bmatrix} B & 0 & 0 & \cdots & 0 \\ AB & B & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A^{H_c-1}B & A^{H_c-2}B & \cdots & \cdots & B \\ A^{H_c}B & A^{H_c-1}B & \cdots & \cdots & AB \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A^{PH-1}B & A^{PH-2}B & \cdots & \cdots & A^{PH-H_c}B \end{bmatrix} \tag{2.2}$$

Objective function definition

Considering a generic case of set-point tracking, the controller's objective should be to minimize the error between the state and the reference $(\hat{x} - z)$ while minimizing the use of the control signal. Said objective can be achieved by minimizing a quadratic cost function like the one shown in (2.3), where Q is a diagonal positive semi-definite matrix that sets the penalty for output tracking while R is also diagonal but positive definite, penalizing the control input effort or usage. It should be noted that, as both the states and control inputs shall be between acceptable and-or feasible ranges, the final minimization problem is always constrained.

$$\begin{aligned} \min_{u} \quad & J(u) = (\hat{x} - z)^T Q(\hat{x} - z) + u^T R u \\ \text{s.t.} \quad & x_{\min} \leq x \leq x_{\max} \\ & u_{\min} \leq u \leq u_{\max} \end{aligned} \tag{2.3}$$

2.1.2. Aplication to EV charging case

To apply the MPC strategy to the EV charging station problem, first it is required to formulate the problem and develop the system model. The scheduling problem considers a set I of charging stations that serve a set J of EVs over a 24 hour period. The state variable considered by the controller is the state of charge (SoC) of EV j connected to

charger i at time step k ($SoC_{j,k} = x_{i,k}$), which can be controlled by the output or decision variables that represent the power delivered to charger i ($P_{i,k}$). To solve the scheduling problem, the controller also has available forecast information regarding electricity prices and information about incoming EVs arrival time and SoC, departure time, etc. The notation for said information can be found in table 2.1.

When a EV is connected to a charger, its SoC can be modeled by equation (2.4). Here an auxiliary binary variable $\xi_{i,k}$ represents if there is an EV connected to charger j at time k. This variable is used to update the state variable when an EV arrives ($k = a_j$) or when it leaves ($k = d_j$). So basically, the equation is updating the state variable to the SoC of the EV at time of arrival using condition 1, then while the EV is connected, its SoC is updated using the energy injected by the charger during the time step duration, calculated as $P_{i,k}\Delta t$. And finally, when the EV departs, the state variable is set to zero using condition 3.

$$x_{i,k+1} = \begin{cases} SoC_{j,a_j} & \text{if } \xi_{i,k} = 1 \text{ and } k = a_j, \\ x_{i,k} + P_{i,k}\Delta t & \text{if } \xi_{i,k} = 1 \text{ and } a_j < k < d_j, \\ 0 & \text{if } \xi_{i,k} = 0 \text{ or } k = d_j. \end{cases} \tag{2.4}$$

Now as the problem has been correctly defined and the plant model is obtained, the minimization problem for the economic MPC is presented in the following set of equations. The cost function to be minimized is shown in (2.5), and it calculates the total economic cost of the power consumed by the charging stations throughout the control horizon, considering the specific electricity prices at each time instant. In contrast to (2.3), this function does not contain any state variables, as there is no need for reference tracking in this problem.

To ensure that the trajectory followed by the eMPC complies with the charger's dynamics, they are added as a constraint to the problem (2.6). Constrains (2.11) and (2.10) ensure that the max power of the charger and maximum SoC of the EV are respected. And finally (2.7) ensures that EV j is fully charged at the time of departure d_j.

Type	Name	Symbol	Units
Independent variable	Time slot	k	-
State variable	State of Charge in charger i	$x_{i,k}$	kWh
Output Variables	Power delivered by the station	$P_{T,k}$	kW
	State of Charge in EV$_j$	$SoC_{j,k}$	kWh
Decision Variables	Power delivered to charger i	$P_{i,k}$	kW
Parameters	Battery capacity in EV$_j$	C_j	kWh
	Electric vehicle j	j	-
	Prediction horizon	H	h
	Number of EV battery chargers	I	-
	Number of EVs	J	-
	Actual SoC in EV$_j$ at a_j	SoC_{j,a_j}	kWh
	Actual SoC in EV$_j$ at d_j	SoC_{j,d_j}	kWh
	EV$_j$ arrival time	a_j	h
	Energy Price	c_k	€/kWh
	EV$_j$ departure time	d_j	h
	Time for charging at maximum power	d_m	min
	Maximum power for all chargers	$P_{\max}$	kW
	Schedule of charger i	$\xi_{i,k}$	{0,1}
	Sampling time	Δt	min

Table 2.1: Table of Variables and Parameters for the EV scheduling problem

$$\min_{u_{i,k}} \quad \Delta t \sum_{k=0}^{H_p-1} \left(c_k \sum_{i=1}^{I} u_{i,k} \right) \tag{2.5}$$

$$\text{s.t.} \quad x_{i,k+1} = \begin{cases} SoC_{j,a_j} & \text{if } \xi_{i,k} = 1 \text{ and } k = a_j, \\ x_{i,k} + P_{i,k}\,\Delta t & \text{if } \xi_{i,k} = 1 \text{ and } a_j < k < d_j, \\ 0 & \text{if } \xi_{i,k} = 0 \text{ or } k = d_j \end{cases} \tag{2.6}$$

$$x_{i,d_j} = x_{i,\max} \tag{2.7}$$

$$x_{i,a_j} = SoC_j \tag{2.8}$$

$$x_{i,\max} = C_j \tag{2.9}$$

$$0 \leq u_{i,k} \leq P_{\max} \tag{2.10}$$

$$0 \leq x_{i,k} \leq x_{i,\max} \tag{2.11}$$

By finding the minimum solution of this problem, the eMPC ensures that the selected trajectory will be optimal while satisfying charging requirements, meaning that all EVs will be charged to 100% in their corresponding parking times. By repeating the process at every time step, the trajectories may be adjusted if there is any change with respect to the forecasted value.

2.2. Optimal Control with minimum Cost and maximum Flexibility

The Optimal Control with minimum Cost and maximum Flexibility (OCCF) [11] can be built on top of the eMPC strategy to be used in situations where providing flexibility capacity to the grid is rewarded economically.

2.2.1. Flexiblity definition

As introduced in chapter 1, flexibility on the demand side can be understood as the capacity to adjust the power consumption of a load to smooth any mismatch between power generation and demand in the power grid. This capacity shall be considered in **both directions**. Meaning that the load shall be able to draw more power from the grid, known as **Upwards Flexibility** (U^F) , or lower its power consumption when required, which is known as **Downwards Flexibility** (L^F). Therefore, the total flexibility capacity of an aggregator for a given time step k can be considered as the sum of both upwards and downwards flexibilities that all charging stations managed by the aggregators could supply, as shown in (2.12). It should be mentioned that EV charging stations could also provide flexibility when V2G is enabled by injecting power into the grid, but this is out of the scope of this case study.

$$F_k = \sum_{i=1}^{I} F_{i,k} = \sum_{i=1}^{I} \left(U_{i,k}^F + L_{i,k}^F\right), \quad \forall k = 1, 2, \ldots, K, \tag{2.12}$$

The previous definitions for both types of flexibilities can be used to obtain its values for a single charging station. With Equation (2.13), the upwards flexibility can be defined as the capacity of the charging station to increase power consumption, meaning that it shall be the difference between the maximum power and instant power output of the station. On the other hand, as downwards flexibility should be the capacity of the station to lower its power consumption, it can be defined with just the power injected into the EV, as shown in (2.14). However, to provide flexibility in both cases, an EV shall be connected to the charger ($\xi_{i,k} = 1$) and said EV shall not be fully charged. Another condition shall be that the time to charge at maximum power shall be lower than departure time ($k < d_m$), as if this condition is not satisfied, the charger is obligated to charge at maximum power and can't increase or lower its power consumption.

$$U_{i,k}^{F} = \begin{cases} P_{i,\max} - P_{i,k} & \text{if } \xi_{i,k} = 1 \text{ and } k < d_m, \\ 0 & \text{if } x_{i,k} = x_{i,\max} \text{ or } x_{i,k} = 0 \text{ or } k \geq d_m. \end{cases} \tag{2.13}$$

$$L_{i,k}^{F} = \begin{cases} P_{i,k} & \text{if } \xi_{i,k} = 1 \text{ and } k < d_m, \\ 0 & \text{if } x_{i,k} = x_{i,\max} \text{ or } x_{i,k} = 0 \text{ or } k \geq d_m. \end{cases} \tag{2.14}$$

2.2.2. OCCF Minimization problem

As the scheduling problem has already been constructed for the eMPC, flexibility can be maximized by just extending the model presented in subsection 2.1.2. Using this approach, upwards and downwards flexibilities are added to the cost function (2.15) with a negative sign, as they should be maximized. The flexibility multiplier $0 < \pi_F \leq 1$ implies that flexibility is rewarded with a fraction of the electricity cost. Meaning that, as π_F approaches 1, the OCCF will find trajectories that maximize flexibility as they reduce the operating cost of the aggregator.

Concerning the set of constrains, when considering flexibilities definitions (2.13) and (2.14), one can express the upwards flexibility as $U^F = P_{\max} - L^F$ and thus the downward flexibility becomes $L^F = P_{\max} - U^F$. Meaning that both flexibilities can be added into the single expression from (2.20), where the binary variable $\xi_{i,k}$ ensures that the constraint is only active when there is an EV connected to charger i. Finally, constrains (2.23) and (2.24) set the upper and lower bounds for both flexibilities.

$$\min_{u_{i,k}} \quad \Delta t \sum_{k=0}^{H_p-1} \left(c_k \sum_{i=1}^{I} u_{i,k} - c_k \pi_F \sum_{i=1}^{I} U_{i,k}^F + L_{i,k}^F \right) \tag{2.15}$$

$$\text{s.t.} \quad x_{i,k+1} = \begin{cases} SoC_{j,a_j} & \text{if } \xi_{i,k} = 1 \text{ and } k = a_j, \\ x_{i,k} + P_{i,k}\,\Delta t & \text{if } \xi_{i,k} = 1 \text{ and } a_j < k < d_j, \\ 0 & \text{if } \xi_{i,k} = 0 \text{ or } k = d_j \end{cases} \tag{2.16}$$

$$x_{i,d_j} = x_{i,\max} \tag{2.17}$$

$$x_{i,a_j} = SoC_j \tag{2.18}$$

$$x_{i,\max} = C_j \tag{2.19}$$

$$L_{i,k}^F \leq u_{i,k} \leq \xi_{i,k}(P_{\max} - U_{i,k}^F) \tag{2.20}$$

$$0 \leq u_{i,k} \leq P_{\max} \tag{2.21}$$

$$0 \leq x_{i,k} \leq x_{i,\max} \tag{2.22}$$

$$0 \leq U_{i,k}^F \leq P_{\max} \tag{2.23}$$

$$0 \leq L_{i,k}^F \leq P_{\max} \tag{2.24}$$

This problem formulation ensures that the objective minimizes the operating cost of the aggregator by minimizing energy cost and maximizing flexibility. Figure 2.2 compares the savings of both eMPC strategy and OCCF with respect to AFAP, in which one can see even when taking the most conservative scenario ($\pi_F = 0.1$), OCCF can double the mean saving that eMPC can achieve while also providing more flexibility to the grid.

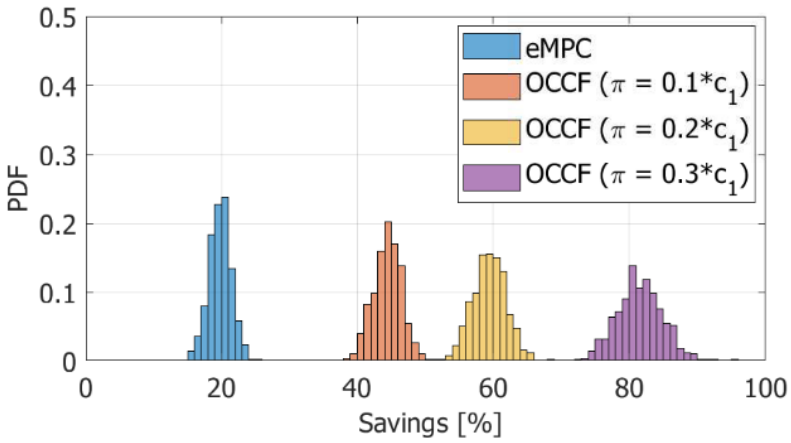

Figure 2.2: Overall saving with respect to AFAP strategy when considering 500 randomly simulated EV arrival scenarios [11]

These results highlight that incorporating more complex scheduling methods not only provides savings for both consumers and maintainers of EV charging stations, they can also solve the critical problem of flexibility as the energy mix of most countries is transitioning to RES. However, EV charging strategy development should continue beyond MPCs and explore newer and more promising methods that do not rely so heavily on forecasted data.

3 | Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning that deals with how agents can learn to make decisions by interacting with an environment. And unlike supervised learning, where models learn from a set of labeled examples, RL is centered around the concept of trial and error, where an agent explores actions and learns from the consequences in the form of rewards or penalties.

RL is not a new concept, in fact, most of the theory behind it was developed in the late 1950s when Richard Bellman introduced the principle of dynamic programming [12] as part of its research in optimal control theory. Bellman's key contribution was the Bellman Equation, which formalized the process of breaking down decision-making problems into simpler sub-problems, ultimately leading to the concept of an optimal policy. While dynamic programming provided the theoretical foundation for decision-making in sequential tasks, it required full knowledge of the environment's dynamics, which limited its practical applicability.

In the 1980s, RL began to evolve into a more practical framework with the development of algorithms like Temporal Difference (TD) learning and Q-learning, allowing agents to learn from experience without requiring a model of the environment. However, it was not until the last decade, particularly with the introduction of the Deep Q-Network (DQN), that RL truly advanced. DQN successfully combined Q-learning with deep neural networks, enabling RL to handle high-dimensional state spaces, such as those encountered in complex video games. This breakthrough paved the way for more sophisticated algorithms, including Deep Deterministic Policy Gradient (DDPG), Trust Region Policy Optimization (TRPO), and Proximal Policy Optimization (PPO). These methods enhanced stability and can be applied in complex and continuous action spaces.

After this brief introduction, the previously mentioned methods will be explained in detail but not before an explanation on the key mathematical concepts behind any RL algorithm.

3.1. Preliminaries

Any RL problem can be represented as a closed-loop system as shown in Figure 3.1. At each time step, the agent selects an action A_t based on a policy and the current state S_t, then the environment returns the next state S_{t+1} and a reward R_{t+1}.

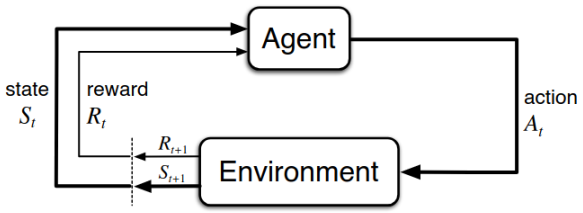

Figure 3.1: Agent-Environment interaction [13]

The environment can be modeled as a Markov Decision Process (MDP) [14], which is characterized by the tuple (s, a, P) where $s \in \mathcal{S}$ represents the set of all possible states and $a \in \mathcal{A}$ denotes the set of actions available to the agent. The transition dynamics are governed by $P(s', r \mid s, a)$, which defines the probability that, when taking action a in state s, the environment will transition to the next state s' and the agent will receive a reward r.

3.1.1. Agent and policies

The agent selects actions based on its past experience and a given policy. The objective of any agent is to maximize the cumulative reward, as defined in (3.1). This metric is influenced by the parameter $\gamma \in (0, 1]$, known as the **discount factor**. This parameter **balances immediate and future rewards in the decision-making process**. A lower value of γ makes future rewards less appealing for the agent, encouraging a focus on immediate outcomes.

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{3.1}$$

To maximize this reward, agents update and follow a policy. Policies can be divided into two categories:

- **Stochastic policies** $\pi(a|s)$: Probability of selecting action a when in state s, these policies are exploratory by nature.
- **Deterministic policies** $q(s)$: Maps a state directly to the action that will yield

the maximum reward as in (3.4) . Also called **greedy policies**.

Exploration vs. Exploitation

At first, one may think that using a deterministic policy is the best option for an agent, as it always selects the action with the highest reward. But as the agent does not have full knowledge of the system dynamics, it may get stuck selecting sub-optimal actions as it has not yet discovered that there are better ones.

To tackle this problem, agents normally use ϵ**-greedy policies** which achieve the right balance between these two behaviors. When using said policies, agents will normally select the action with the highest reward, meaning that they are **exploiting** the knowledge that they have of the system's dynamics. But there is a ϵ % chance of selecting a random **exploratory action**.

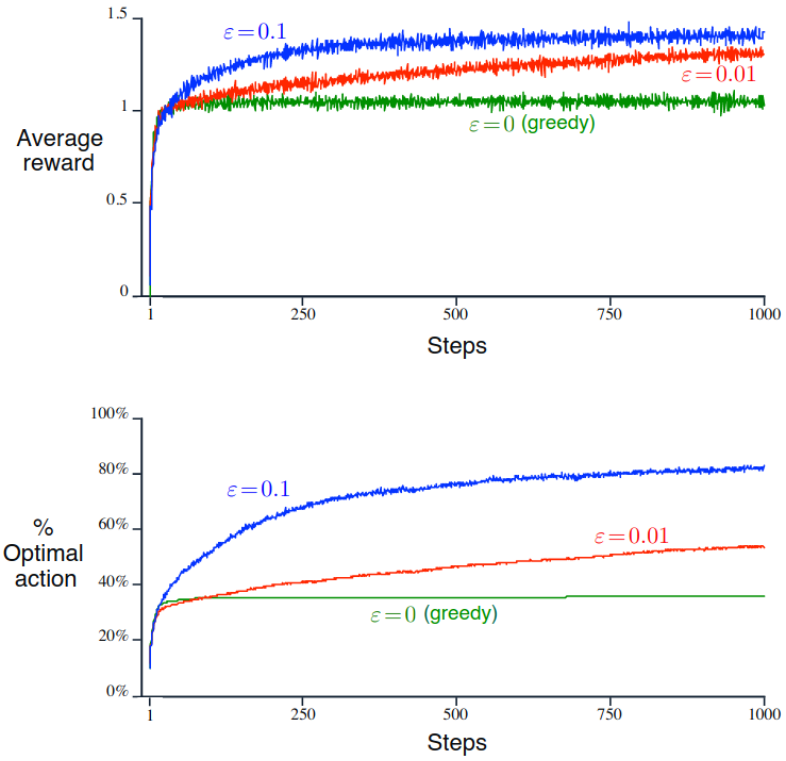

Figure 3.2: Average performance ϵ-greedy methods in a 10-armed bandit environment with different probabilistic distribution [13]

In Figure 3.2 it can be seen how using a higher value for ϵ improves both the average reward and the optimal action selection. The problem with using ϵ-greedy policies (or stochastic policies in general) is that they **cannot be optimal by nature**, due to the

probability of selecting the exploratory action. This will be explaining in further detail in Section 3.2 .

3.1.2. Value Functions

Value functions are used by the agent to estimate how good it is to be in a certain state or to take a specific action, as they provide a mathematical estimation of future rewards. This stems from the fact that they are defined as the expected discounted return for the agent starting at state s and following policy π.

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s']] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_\pi(s')] \quad \text{for all } s \in S \end{aligned} \tag{3.2}$$

The equation shown in (3.2) is the definition of the **state-value function**. It quantifies the *value* of the state as the product of:

- The probability of choosing action a according to policy $\pi(a|s)$.
- The probability of landing in next state s' with reward r.
- The reward plus the state-value function of the next state (with discount γ).

Analogously, one can write the **state-action function** $q_\pi(s, a)$ which also accounts for selecting action a as in (3.3). The key of writing value-functions in such recursive way is that if the MPDs dynamics are known, starting from one state one can compute the value function for the rest of states, and once this is done, the greedy policy can be constructed according to (3.4), this is known as **policy improvement**.

$$q_\pi(s, a) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid S_t = s, A_t = a \right] \tag{3.3}$$

$$\pi'(a|s) = \arg\max_a q_\pi(s, a) \tag{3.4}$$

As π has changed for its improved version π', the value functions shall be recomputed for this newer policy, which is known as **policy evaluation**. This iterative process is known

as **Policy iteration** [15] and in Figure 3.3 it can be see that after a finite number of iterations, this method will converge to the optimal policy π^* and value function v^*.

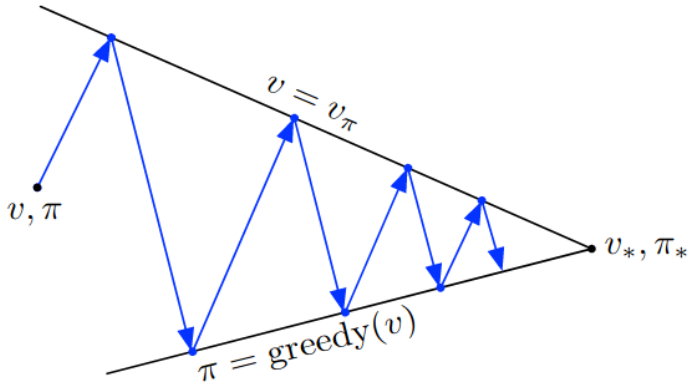

Figure 3.3: Visual representation of the policy iteration method. First the current policy is evaluated by computing its value function, and then it is improved by taking the greedy action [13].

3.1.3. Temporal Difference Learning

The Policy Iteration method previously described converges to an optimal policy, but it can only be applied if the environment's dynamics are fully-known. As this is rarely the case, the best alternative is to estimate and update the value functions using the experiences gained during exploration.

This is the main idea behind Temporal Difference (TD) methods [16]. They exploit the definition of the return being $G_t = R_{t+1} + \gamma V(S_{t+1})$ to make an update based on the difference between the expected return (which is in part estimated) and the current estimation of the value function as shown in (3.5). This difference is referred to as **TD error** δ_t and is multiplied by a learning rate α to set the update rule for $V(S_t)$. It can be proven that, even when starting at arbitrary values, the value functions converges to the real one in a finite number of iterations.

$$\begin{aligned} V(S_t) &\leftarrow V(S_t) + \alpha[G_t - V(S_t)] \\ V(S_t) &\leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \end{aligned} \tag{3.5}$$

Using this method, one would get the value function for the environment without needing

knowledge of the MPD dynamics. And once the value function has converged, an optimal policy can be derived. An alternative solution is the SARSA algorithm [17, 18], which estimates both action-values using the TD error as shown in (3.6) and follow an ϵ-greedy policy to get a working agent.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{3.6}$$

To achieve faster convergence to the (semi) optimal policy, Q-learning [19] uses the same principle as SARSA but uses the **greedy action** for the update rule as in (3.7) to directly estimate the optimal action value function $Q^*(s, a)$. Q-learning is said to be an off-policy method, because while following a stochastic policy, the update rule takes the optimal action. This types of methods will be discussed in subsection 3.2.3 .

$$Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha\left[R_{t+1} + \gamma \max_a Q\left(S_{t+1}, a\right) - Q\left(S_t, A_t\right)\right] \tag{3.7}$$

3.1.4. Function approximation

The two previously described methods only work in tabular scenarios, that is, where the state space is perfectly discretized. A solution for applying these methods when dealing with a continuous state space is just to discretize the space, effectively turning it into a tabular scenario. This solution leads to two problems, first the memory requirements grow exponentially the smaller the discretization is, and secondly, visiting each one of the states several times until the value function converges may be impossible.

For this reason, the best solution is to construct a function approximator $Q_\pi(s, a) \approx \hat{Q}(s, a, \theta)$ which depends on the feature vector θ so that, even if the agent does not visit every state, it can generalize and get an approximate solution for state-action pairs that has not yet visited.

Figure 3.4 shows how a neural network (NN) with weights θ can be used to predict the state-action values. NNs are chosen for this purpose as they are known of being capable of approximating any continuous function [20]. The feature vector is chosen to minimize the square TD prediction error (3.8) using stochastic gradient descend (SGD).

$$L_i(\theta_i) = \mathbb{E}\left[\left(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}; \theta_{i-1}) - Q(s, a; \theta_i)\right)^2\right] \tag{3.8}$$

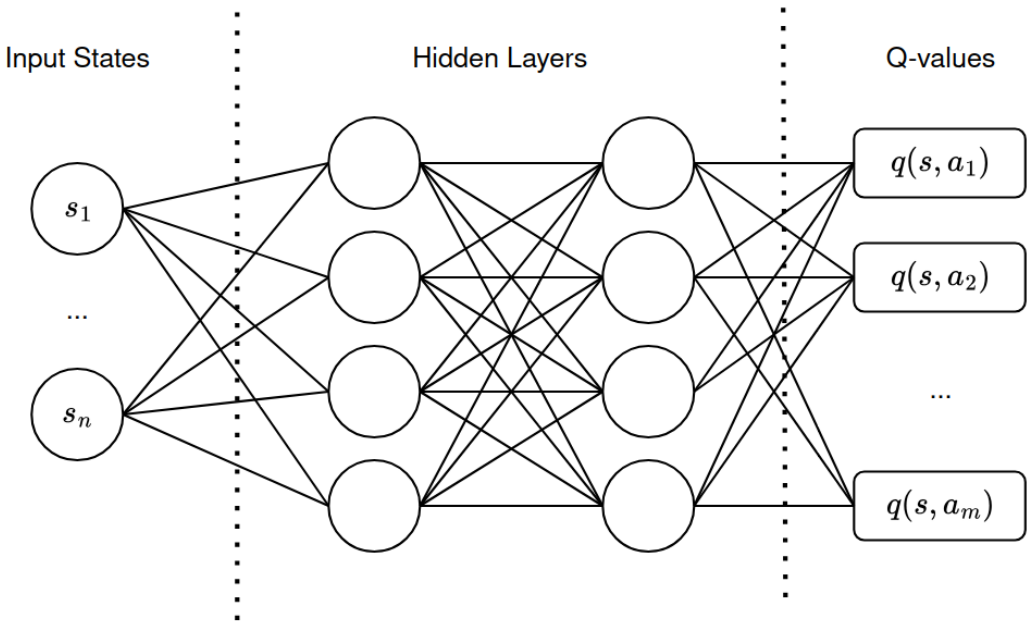

Figure 3.4: NN approximating the Q-values from continuous states $s_1, \ldots, s_n$ using two hidden interconnected layers

This was the first proposed solution, but it suffered stability issues during training due to the fact that the samples are correlated (which is known to cause problems when using SGD) and because the weight vector is being changed at each iteration (the Q-value changes drastically even for the same state-action pair). Deep Q-networks [21] introduced some changes to be the first effective Q-learning algorithm that could use NNs. First, samples are stored and retrieved randomly for a replay buffer to reduce correlation, and secondly, it uses a target network for the computation of the Q-values. This target is a copy of the current network with weights w_i which is not changed every iteration, instead it is updated at every 500 iteration ($\theta_i = w_i$) as shown in (3.9). This enhances stability during training.

$$L_i(\theta_i) = \mathbb{E}\left[\left(r_{t+1} + \gamma \max_a Q(s_{t+1}, a; w_i) - Q(s, a; \theta_i)\right)^2\right] \tag{3.9}$$

Although DQN has no problems handling large continuous observation spaces such as 84x84 RGB images, it cannot deal with continuous action spaces. Nevertheless, the contributions made by this work inspired most of the methods covered in latter sections, specially DDPG which will be covered in Section 3.3.

3.2. Policy Gradient Methods

All the methods described on section 3.1 can be categorized under the umbrella term **action-value methods**, which learn the action value and then use them to determine the action to select. On the other hand, there exists another family of methods which directly learn a parameterized policy (depending on a parameter vector) without the need of action-value estimation. These types of methods are referred to as **policy gradient methods**, meaning that they update the policy parameter vector in the direction that maximizes a performance index, as can be seen in (3.10) . This index is normally chosen to be the **expected discounted reward** when following policy π_θ ($\eta(\theta)$). Taking a close look at this metric in (3.11) and comparing it to equation (3.2) it can be seen that the performance metric can also be expressed as the value function when following said policy ($J(\theta) = v_\pi(s)$).

$$\theta_{t+1} \leftarrow \theta_t + \alpha \nabla J(\theta_t) \tag{3.10}$$

$$J(\theta) = \eta(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right] \tag{3.11}$$

These methods are normally preferred in control problems as they have no issue with handling continuous action spaces. Also, another improvement is that these methods can be adapted to use deterministic (optimal) policies, whereas action-value methods should always use stochastic ones.

3.2.1. Policy Gradient Theorem

For using the update rule in (3.10), it is required to get an expression for the gradient of the policy with respect to $J(\theta)$. Thankfully, the policy gradient theorem [22] gives the contribution of a single state to the gradient of the performance index as can be seen in (3.12).

$$\nabla J(\theta) = \mathbb{E}_\pi \left[\sum_a Q_\pi(S_t, a) \nabla \pi(a|S_t, \theta) \right]. \tag{3.12}$$

Using this contribution, the REINFORCE algorithm [23] replaces a generic action a by a sampled action $A_t \in \pi$ to make the expression dependent on the Q-values, then it takes advantage of the definition of said value function (3.3) to make the gradient dependent

on the return as can be seen in (3.13).

$$\begin{aligned}\nabla J(\theta) &= \mathbb{E}_\pi \left[\sum_a \pi(a|S_t,\theta) Q_\pi(S_t,a) \frac{\nabla \pi(a|S_t,\theta)}{\pi(a|S_t,\theta)} \right] \\ &= \mathbb{E}_\pi \left[Q_\pi(S_t,A_t) \frac{\nabla \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)} \right] \\ &= \mathbb{E}_\pi \left[G_t \frac{\nabla \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)} \right]\end{aligned} \tag{3.13}$$

This algorithm first generates a trajectory using the Monte Carlo method [24] to gather the states, actions and return at each time step and then uses the update rule from (3.14) to perform gradient ascend on the performance index.

$$\theta_{t+1} \doteq \theta_t + \alpha G_t \frac{\nabla \pi(A_t|S_t,\theta_t)}{\pi(A_t|S_t,\theta_t)}. \tag{3.14}$$

3.2.2. Actor-critic methods

One drawback of Monte Carlo methods like REINFORCE is that they require waiting until the end of an episode before performing any policy optimization. Fortunately, the TD error can be integrated into policy gradient methods, allowing for gradient ascent updates at each time step, thus speeding up convergence. Actor-critic methods [25, 26] address this by replacing the full return used in REINFORCE by the one-step return. Equation (3.15) outlines the update rule for the actor, which uses the parameter vector θ to define the policy.

$$\begin{aligned}\theta_{t+1} &\doteq \theta_t + \alpha \left(G_{t:t+1} - \hat{Q}(S_t,A_t,\mathbf{w}) \right) \frac{\nabla \pi(A_t|S_t,\theta_t)}{\pi(A_t|S_t,\theta_t)} \\ &= \theta_t + \alpha \left(R_{t+1} + \gamma \hat{Q}(S_{t+1},A_{t+1},\mathbf{w}) - \hat{Q}(S_t,A_t,\mathbf{w}) \right) \frac{\nabla \pi(A_t|S_t,\theta_t)}{\pi(A_t|S_t,\theta_t)} \\ &= \theta_t + \alpha \delta_t \frac{\nabla \pi(A_t|S_t,\theta_t)}{\pi(A_t|S_t,\theta_t)}\end{aligned} \tag{3.15}$$

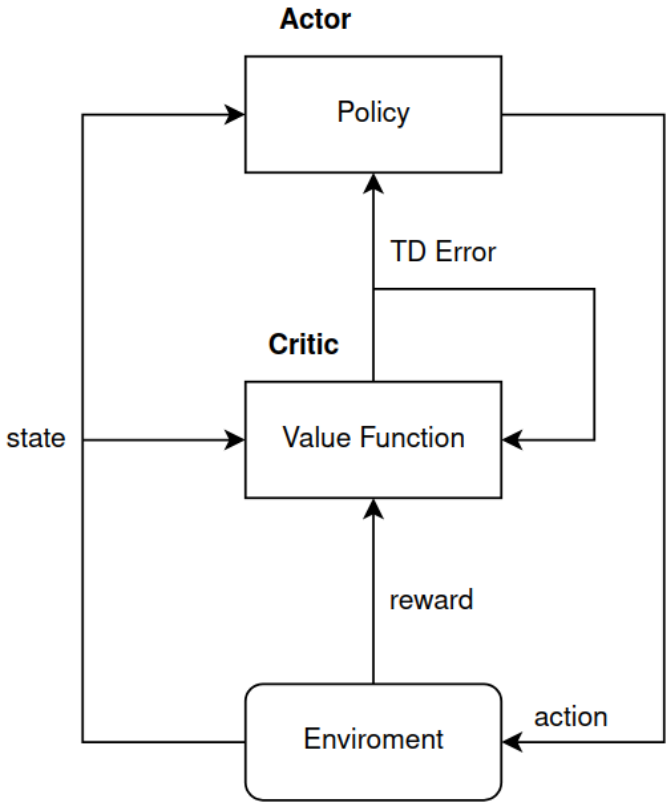

Figure 3.5: Diagram showing the working principle behind any actor critic method: the critic uses the state and the reward returned by the environment to generate the TD error. This error is used to develop a better approximation of the value function and to update the policy.

In this approach, estimating δ_t is required to perform the update. This is done by the critic as can be seen in Figure 3.5 . The critic uses vector w to estimate the value function $\hat{Q}(s, a, w) \approx Q_\pi(s, a)$ and calculate the TD error, which is used to update itself and the actor. Vector w can be adjusted using one of the TD methods seen in subsection 3.1.3 such as SARSA or Q-learning when dealing with a tabular case, or function approximation in a continuous one. In such case, SGD shall be used to reduce the mean squared error from (3.16).

$$e(w) = \mathbb{E}_\pi \left[\left(\hat{Q}(s, a, w) - Q_\pi(s, a) \right)^2 \right] \tag{3.16}$$

3.2.3. Off-policy methods

To get one step closer towards optimal policies, the concept of off-policy methods shall be introduced. Previously, ϵ-greedy policies were always used to maintain exploration despite the fact that they are intrinsically suboptimal. Now, two completely different policies can be considered: first, a **behavioral policy** $\beta(a|s)$ is used to encourage exploration, which is purely stochastic and normally set to a uniform distribution between actions. Then a **target policy** $\pi(a|s)$ is constructed using the sampled data from the behavioral policy. This policy shall be greedy with respect to action-values, as in (3.4).

As now the samples are not coming from the policy that is being optimized, an **importance sampling ratio** shall be used. This ratio is set to the probability of selecting a state-action pair under π over the probability of taking the same pair in β ($\frac{\pi_\theta(a|s)}{\beta(a|s)}$). Equation (3.17) shows how adding the sampling ratio gives an expression of the gradient with respect to $J(\pi_\theta)$ even when the samples are taken from β. This trick is used to derive an effective off-policy actor-critic algorithm [27] which laid the groundwork for the algorithm presented in the next section.

$$\nabla J(\pi_\theta) = \mathbb{E}_\beta \left[\frac{\pi_\theta(\mathbf{a}|\mathbf{S_t})}{\beta(\mathbf{a}|\mathbf{S_t})} \sum_a Q_\pi(S_t, a) \nabla \pi_\theta(a|S_t) \right] \tag{3.17}$$

3.3. Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient [28] or DDPG for short is an algorithm that combines the tricks incorporated by DQN with the techniques seen in section 3.2. This off-policy actor-critic algorithm is capable of deriving deterministic policies which can be used in continuous action spaces, that is, the action space is not discrete.

3.3.1. Deterministic Policy Gradient

For handling continuous actions, DDPG considers directly a parameterized deterministic policy ($a_t = \mu_\theta(s_t)$) that shall be optimized. However, the policy gradient theorem used to calculate the gradient of the objective function from (3.12) only works for stochastic policies. Thankfully, [29] derived the expression shown in (3.18) thanks to the introduction of the deterministic policy gradient theorem.

$$\nabla J(\mu_\theta) = \mathbb{E}\left[\nabla_\theta \mu_0(s) \nabla_a Q^\mu(s,a)\Big|_{a=\mu_0(s)}\right] \tag{3.18}$$

This theorem stems from the fact that, when dealing with continuous action spaces, using a greedy policy improvement as in (3.19) becomes problematic, as it requires solving a global maximization problem at each iteration. A less computationally expensive alternative is to move the policy in the direction of the gradient of the Q-values, as expressed in the update rule (3.20).

$$\mu_{k+1}(s) = \arg\max_a Q_\mu(s,a). \tag{3.19}$$

$$\theta^{k+1} = \theta^k + \alpha \mathbb{E}\left[\nabla_\theta Q^{\mu^k}(s, \mu_\theta(s))\right] \tag{3.20}$$

Then, using the chain rule and considering an action sampled by the policy ($a = \mu_k(s)$) the final update rule from (3.21) gives an expression for the gradient of a deterministic policy, which is exactly the expression used to derive the deterministic policy gradient theorem. In fact, it has been proven that the deterministic version of the theorem is a special case of the stochastic one. This was done by modeling the deterministic policy as a stochastic policy with varying variance, then as the variance tends to zero, both gradients converge to equal values ($\lim_{\sigma \downarrow 0} \nabla_\theta J(\pi_{\mu_0,\sigma}) = \nabla_\theta J(\mu_0)$).

$$\theta^{k+1} = \theta^k + \alpha \mathbb{E}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu^k}(s,a)\Big|_{a=\mu_\theta(s)}\right] \tag{3.21}$$

3.3.2. Final algorithm

The DDPG algorithm can be seen as an extension of the DQN algorithm but using a different update rule to account for the fact that it uses a deterministic policy. In reality, this algorithm also incorporates a replay buffer and both the actor and critic use target networks for enhanced stability. A small difference between both works is that, while DQN updates the target every 500 iterations, DDPG updates the targets at each iteration but using a moving average of the weights, stabilizing training even more.

It should be noted that, to maintain exploration while preserving the deterministic nature of the policy, a random action noise is added during training. Meaning that actions taken

are actually $a = \mu_\theta(s) + \mathcal{N}$. This noise is normally set to be Ornstein-Uhlenbeck (OU) Noise [30] as actions are temporally correlated.

3.3.3. TD3

One drawback discovered when using DDPG is that the critic was prone to suffer from an overestimation bias with respect to the Q-values, meaning that, the estimated values are always higher than the real ones. To address this issue, Twin Delayed Deep Deterministic Policy Gradient or TD3 [31] was developed and introduces three main modifications to address this issue:

1. **Double Q-learning:** TD3 incorporates two different critic networks, which are initialized and updated separately. Then, the actor simply takes the lowest value for the two critics to make an update. Effectively reducing overestimation.
2. **Target Smoothing:** TD3 also adds a clipped Gaussian noise to the target action while updating the critic, meaning that while the policy takes action a with its own OU noise, the critic is updated with action $a' = a + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma)$. This prevents the critic from becoming too sensitive to minor changes in the target policy, thus reducing variance of Q-values.
3. **Delayed Policy Updates:** To ensure that the actor is updated using more reliable estimates, its update is done at every two or more iterations, while the critic is still being updated at every iteration.

3.4. Trust Region Policy Optimization

DDPG is not the only method prone to suffering from stability issues during training. In general, all policy gradient methods are known from this drawback. This stems from the fact that the policy is being optimized on **approximations** of both the expected discounted reward and its gradient constructed at (s, a). Naturally, while the approximation may be reasonable at these points, it degrades when moving away from said pair.

Trust Region Policy Optimization (TRPO) [32] interiorize this intrinsic defect of using function approximation and deal with them by applying a trust region (TR) constrain to limit the divergence between old and new policies, thus reducing the risk of sudden performance drops.

3.4.1. Trust Region Methods

When considering numerical optimization methods [33], TR methods achieve the right balance between the fast convergence speeds of quadratic approximation methods such as the Newton method and the stability of using a first-order approximation such as the gradient method.

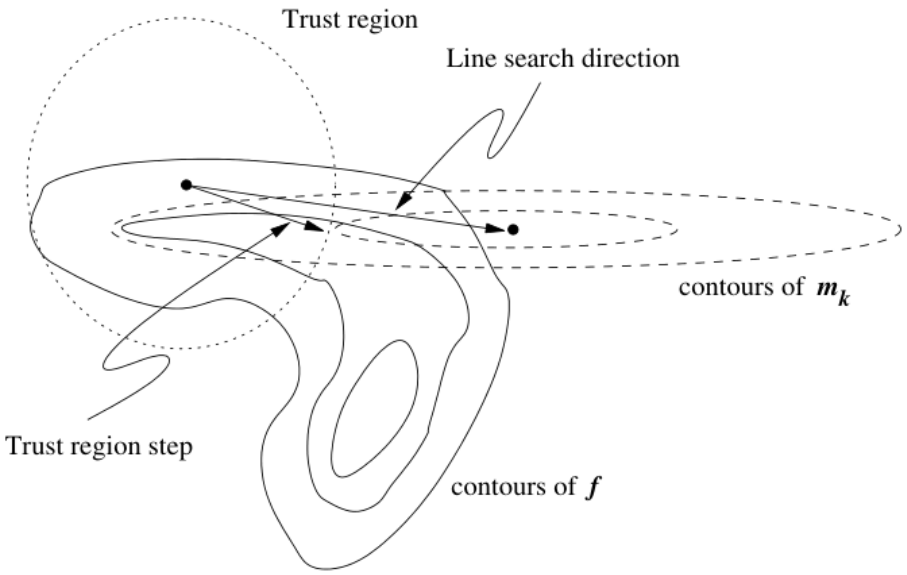

Figure 3.6: Comparison of steps taken between trust region methods and line search methods when minimizing f [33]

Figure 3.6 shows how while line search methods perform the minimization over a single line (the steepest descent direction), TR methods perform the minimization over the whole region. However, the step cannot go over the TR size as the quadratic approximation may not match the objective function anymore. Equation (3.22) shows how this can be achieved by just adding a hard constraint to the minimization problem, where m_k is the quadratic approximation of f at the current point x_k.

$$\min_{p \in \mathbb{R}^n} m_k(p) \quad \text{s.t.} \quad \|p\| \leq \delta_k \tag{3.22}$$

Region size

Choosing a correct value for δ_k is crucial for step effectiveness (which will determine convergence speed). If this value is chosen to be small and conservative, each step will not yield sufficient reduction and the TR method may become slower than just using a

typical line search method. However, if the TR size is chosen to be too large, m_k may not match f and the iteration could result in a failed step, meaning $f(x_k) < f(x_{k+1})$. Fortunately, the TR size can be adjusted iteratively according to the following scheme:

1. Obtain a trial step p_k by solving the constrained sub-problem (3.22).
2. Compute the performance metric ρ_k as the ratio between the predicted and actual reduction.
$$\rho_k = \frac{f(x_k) - f(x_k + p_k)}{m_k(0) - m_k(p_k)} \tag{3.23}$$
3. Update TR size based on ρ_k.
$$\delta_{k+1} = \begin{cases} \frac{1}{4}\delta_k & \text{if } \rho_k < \frac{1}{4} \\ \delta_k & \text{if } \frac{1}{4} \leq \rho_k \leq \frac{3}{4} \\ \min(2\delta_k, \delta_{\max}) & \text{if } \rho_k > \frac{3}{4} \end{cases} \tag{3.24}$$
4. Update the point x_{k+1}
$$x_{k+1} = \begin{cases} x_k + p_k & \text{if step is successful} \\ x_k & \text{else} \end{cases} \tag{3.25}$$

Using this update algorithm, even if the value of δ_k is not chosen properly at the start, it will adapt depending on the performance of the quadratic approximation at each time step.

Adding this TR constraint to the maximization problem is the main idea behind TRPO. However, first a metric should be constructed that measures the difference between the new and old policy.

3.4.2. Conservative Policy Iteration

In [34], Kakade and Landford proposed an useful identity that expresses the expected return of a new policy $\tilde{\pi}$ in terms of its advantage over an old policy π. This relationship is dependent on the visitation frequency $\sum_s \rho_{\tilde{\pi}}(s)$ and the advantage value function, which can be defined as $A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$.

$$\eta(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_{\tilde{\pi}}(s) \sum_a \tilde{\pi}(a|s) A_\pi(s,a) \tag{3.26}$$

Due to the dependence of (3.26) on the trajectory of the new policy $\rho_{\tilde{\pi}}(s)$, a local approximation can be computed based on the trajectory of the old policy as can be seen in (3.27). This approximation can be used for optimizing the policy, as it was proven that, when using parametrized policies $\pi_\theta(a|s)$, it approximates the objective function and its gradient perfectly as $L_{\pi_{\theta_0}}(\pi_{\theta_0}) = \eta(\pi_{\theta_0})$ and $\nabla_\theta L_{\pi_{\theta_0}}(\pi_\theta)\big|_{\theta=\theta_0} = \nabla_\theta \eta(\pi_\theta)\big|_{\theta=\theta_0}$.

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_\pi(s) \sum_a \tilde{\pi}(a|s) A_\pi(s,a) \tag{3.27}$$

After this discovery, the authors proposed an update scheme called **Conservative Policy iteration** (CPI) which provides a lower bound for $\eta(\tilde{\pi})$ shown in (3.28). The dependence of the bound on α^2 implies that any over-aggressive updates on the policy will be penalized, as α is the distance between the old policy π and π', which is the greedy policy with respect to L_π.

$$\eta(\tilde{\pi}) \geq L_{\pi_{\text{old}}}(\tilde{\pi}) - \frac{2\epsilon\gamma}{(1-\gamma)^2}\alpha^2 \tag{3.28}$$

$$\epsilon = \max_s \left| \mathbb{E}_{a\sim\pi'(a|s)}[A_\pi(s,a)] \right| \tag{3.29}$$

$$\tilde{\pi}(a|s) = (1-\alpha)\pi(a|s) + \alpha\pi'(a|s). \tag{3.30}$$

3.4.3. Extension to stochastic policies

Although the previous update scheme was proved to tend to the monotonic improvement of the expected reward, it was only applicable to mixture policies constructed as (3.30). To extend this scheme while maintaining monotonic improvement, a new metric is introduced to measure the distance between policies called KL divergence (3.31).

$$D_{\text{KL}}(\pi \parallel \tilde{\pi}) = \sum_s \sum_a \pi(a|s) \log\left(\frac{\pi(a|s)}{\tilde{\pi}}\right) \tag{3.31}$$

This new measure can substitute α in (3.28) and thus generate a new lower bound (3.32) that is applicable to stochastic policies. The new lower bound uses $C = \frac{4\epsilon\gamma}{(1-\gamma)^2}$ and the maximum KL divergence over all the states.

$$\eta(\tilde{\pi}) \geq L_\pi(\tilde{\pi}) - C D_{\text{KL}}^{\max}(\pi, \tilde{\pi}), \tag{3.32}$$

This scheme can be used for the optimization of stochastic policies, as by maximizing the lower bound, one would be maximizing the expected return of the policy. The problem with this initial approach was that using coefficient C results in insufficient step size and thus slow convergence speed. To tackle this issue, the KL divergence can be removed from the objective function and a TR constrain can be added (3.33) , effectively controlling the divergence of the policy from step to step.

3.4.4. Adaptation for sample-based methods

Adding the TR constraint turns the unconstrained maximization problem (3.32) into the constrained one form (3.33). Also, as the KL divergence is bounded at every state, it is changed with an average obtained by following a trajectory from π.

$$\begin{aligned} &\max_{\tilde{\pi}_\theta} L_\pi(\tilde{\pi}_\theta) \\ &\text{subject to} \quad D_{\mathrm{KL}}^{\rho^\pi}(\pi, \tilde{\pi}_\theta) \leq \delta. \end{aligned} \tag{3.33}$$

However, the local approximation from (3.27) still relies on having exact knowledge on state distributions and advantage values. To make this approach compatible with MC, three changes shall be introduced to the local approximation:

- The visitation frequency $\sum_s \rho_{\tilde{\pi}}(s)$ can be substituted by the expectation $\frac{1}{1-\gamma}\mathbb{E}$, although only the expectation can be left as the discount factor is constant.
- The advantage values can be substituted by Q-values.
- To replace the sum over the actions, an importance sampling estimator can be used. It will depend on the method used to collect samples.

$$\sum_a \tilde{\pi}_\theta(a|s_n) A_\pi(s_n, a) = \mathbb{E}_{a\sim q}\left[\frac{\tilde{\pi}_\theta(a|s_n)}{q(a|s_n)} Q_\pi(s_n, a)\right] \tag{3.34}$$

Adding these slight changes, the contribution of a single state to the local approximation is calculated as (3.34) and the final constrained optimization problem (COP) is presented in (3.35).

$$\begin{aligned} &\max_{\tilde{\pi}_\theta} \mathbb{E}_{s\sim\rho_\pi, a\sim q}\left[\frac{\tilde{\pi}_\theta(a|s)}{q(a|s)} Q_\pi(s, a)\right] \\ &\text{subject to} \quad \mathbb{E}_{s\sim\rho_\pi}\left[D_{\mathrm{KL}}(\pi(\cdot|s)\|\tilde{\pi}_\theta(\cdot|s))\right] \leq \delta. \end{aligned} \tag{3.35}$$

3.4.5. Sample collection methods

To collect the samples from the environment, TRPO can use the two methods shown in Figure 3.7. On the left, the simpler single-path method is shown, in which the agent just evaluates one trajectory at a time. While on the right, the vine method generates several branches starting from state s_n and evaluates the different trajectories based on the different available action. Once a trajectory has been evaluated, the environment is then reset to s_t to evaluate the next branch.

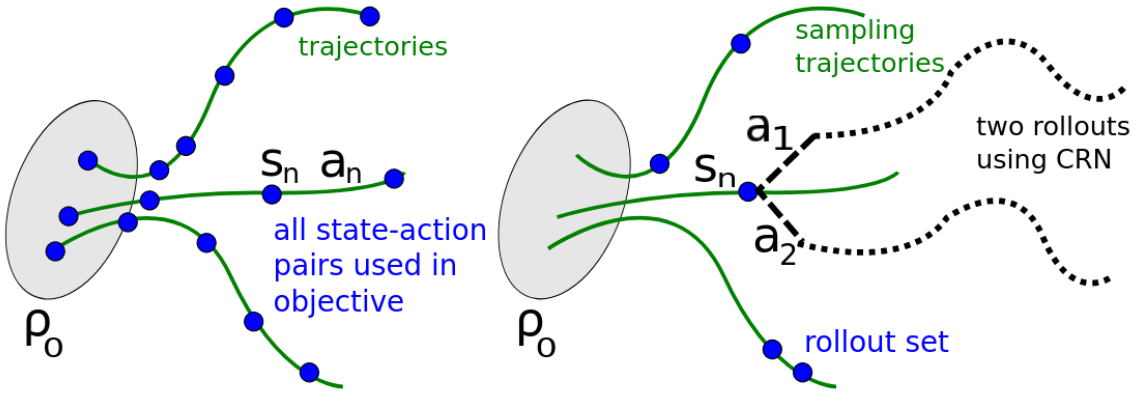

Figure 3.7: Single Path (left) and vine method (right) [32]

The vine method converges much faster when compared to single-path, mainly because the Q-values are evaluated more often. However, it introduces a new level of complexity to the problem, as now the importance sampling ratio from (3.35) should be recalculated when evaluating different branches. On the other hand, when using the single-path, the sampling ratio can just be set to the policy ($q(a|s) = \pi(a|s)$) as this is the only trajectory being followed. Nevertheless, once one of the methods has been selected, the TRPO algorithm can update the policy using the following steps:

1. Collect set of state-action pairs using the MC method.
2. By averaging over samples, the COP from (3.35) can be constructed by replacing expected values by sample averages.
3. Solve the COP to update π_θ.

For solving the COP, the original authors proposed using the conjugate gradient method followed by a line search to be sure that the TR constrain is not violated. However, in most practical application the natural gradient method [35] is used.

3.5. Proximal Policy Optimization

TRPO satisfies its main objective of achieving stable policy updates but at the expense of adding a constraint to the optimization process. For this reason, Proximal Policy Optimization [36] or PPO was created, which instead of constraining the updates, uses a clipped objective to prevent large policy updates. This addition preserves the stability of TRPO while reducing the computational expense of solving a COP at every iteration, as now the optimization process is unconstrained. This balance between stability and lightness makes PPO one of the most used RL methods to date.

3.5.1. Clipped objective function

Both TRPO and PPO based their development on the proposed CPI method explained in subsection 3.4.2. PPO sticks to using the advantage values (estimated advantage values) but still replaces visitation frequency by the expectation and uses the importance sampling introduced by TRPO. Equation (3.36) shows how the local approximation from the CPI shown in (3.27) can be written with said modifications. In the first equation, the term $r_t(\theta)$ denotes the probability ratio between the new and old policy and is used for evaluating the divergence of the policy (so $r_t(\theta_{\text{old}}) = 1$).

$$L^{CPI}(\theta) = \mathbb{E}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}_t\right] = \mathbb{E}\left[r_t(\theta)\hat{A}_t\right]. \tag{3.36}$$

As said previously, performing an optimization on this objective function will lead to large policy changes. Luckily, it can be modified to penalize changes that would move $r_t(\theta)$ from 1. The objective function from (3.37) achieves this goal without adding any constraints by choosing the minimum between the unclipped objective and the objective with a clipped probability ratio. This addition removes any incentive of moving $r_t(\theta)$ outside $[1-\epsilon, 1+\epsilon]$ as it would not reflect any improvement as can be seen in Figure 3.8.

$$L^{CLIP}(\theta) = \mathbb{E}\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]. \tag{3.37}$$

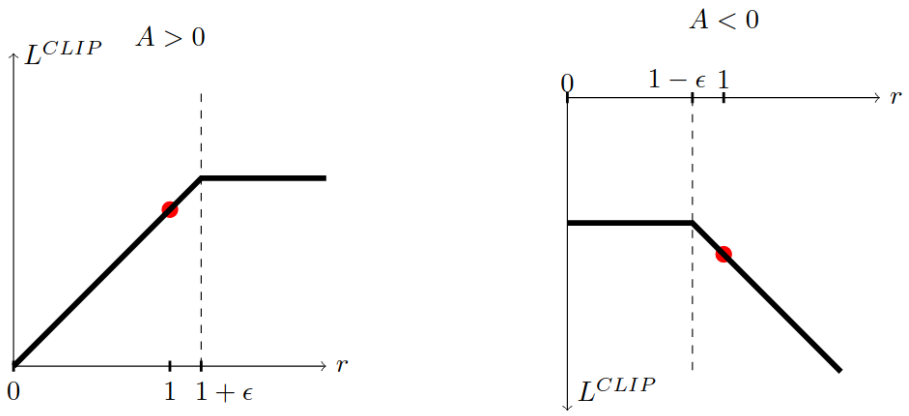

Figure 3.8: Plots representing how function L^{CLIP} discourages updates that lead $r_t(\theta)$ being outside a desired range. This clipping is depend on if the advantage values are positive (left) or negative (right) [36]

Figure 3.9 helps to understand how clipping encourages stable updates, because as the KL divergence between policies grows, L^{CLIP} is being penalized as it is lower when compared to its unclipped version L^{CPI}. Although this clipping discourages divergent updates, it is far from stable when compared to constraining the KL divergence.

3.5.2. KL penalization

In subsection 3.4.3, it is explained how the first idea of the creators of TRPO was to use a penalty coefficient C on the KL divergence as shown in (3.32), but it practice this term leads to slow convergence. However, to penalize divergence even further, PPO maintains a similar penalty coefficient β which it adapted iteratively using a similar method as the one from subsection 3.4.1:

1. Using SGD, optimize the penalized objective from (3.38).
2. Compute $d = \mathbb{E}\left[\mathrm{KL}\left[\pi_{\theta_{\text{old}}}(\cdot|s_t), \pi_\theta(\cdot|s_t)\right]\right]$
   - If $d < 1.5\delta$, then $\beta \leftarrow \frac{\beta}{2}$ (penalize less).
   - If $d > \frac{\delta}{1.5}$, then $\beta \leftarrow 2\beta$ (penalize more).

$$L^{KLPEN}(\theta) = \mathbb{E}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}_t - \beta\,\mathrm{KL}\left[\pi_{\theta_{\text{old}}}(\cdot|s_t), \pi_\theta(\cdot|s_t)\right]\right]. \tag{3.38}$$

Using this scheme, β is updated at each iteration and policy updates with KL divergence higher than δ are rare. Now, the method can produce more stable updated when compared to TRPO by just solving an unconstrained optimization problem, which is less

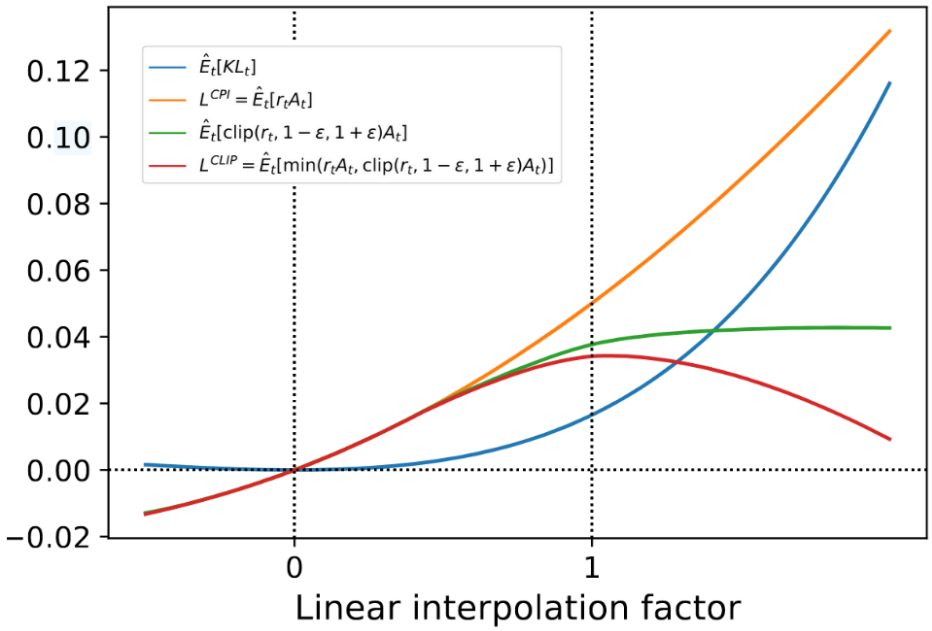

Figure 3.9: Interpolation of functions between old policy parameter θ_{old} and the new policy [36]

computationally expensive than a COP. Moreover, simpler optimizers such as SGD or Adams [37] can be used. It should be noted that a critic should be introduced to compute the advantage estimates $\hat{A}_t$ to construct the problem.

4 | Simulator Environment

When developing a successful RL solution, both the agent and the environment should be treated with the same importance, as if the environment does not provide an accurate representation of the scenario, the agent will yield a policy that will not perform in the real world. Thus, while chapter 3 focused on the theory and implementation of different agents, this chapter will provide an overview of the environment in which said agents will be trained.

The environment selected for this scenario has been EV2Gym [9] for two main reasons: first, it utilizes real-world data to serve as a simulator specifically tailored for training RL agents. To achieve this, the simulator uses the charging dynamics seen in 2.1.2 to model EVs, charging stations and other power distribution systems, ensuring that each simulated episode is as realistic as one can get. Secondly, as an open-source platform written in Python, it enables easy customization, making it possible to modify any aspect of the simulation to suit the thesis's specific needs.

4.1. Simulator Structure

To allow easy customization, EV2Gym follows a modular design, meaning that the different components of the simulation are defined and updated separately, as can be seen in Figure 4.1.

Class structure

It can be seen that the heart of the simulator is located in the *environment* class, which also serves as the main interface with the agent. When the simulation is started, this class loads the simulation parameters from the configuration file and then spawns the different objects (*Transformers, chargers, EVs*). It should be noted that at the initialization phase, all the parameters of the EVs being simulated are defined and checked. Ensuring that the simulated episode is feasible.

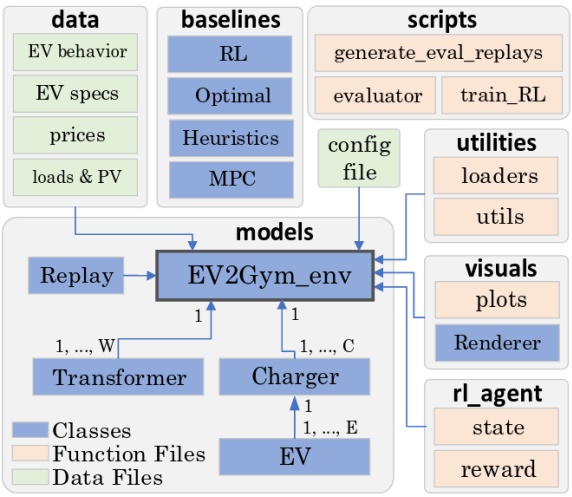

Figure 4.1: Class diagram and functions of the simulator [9]

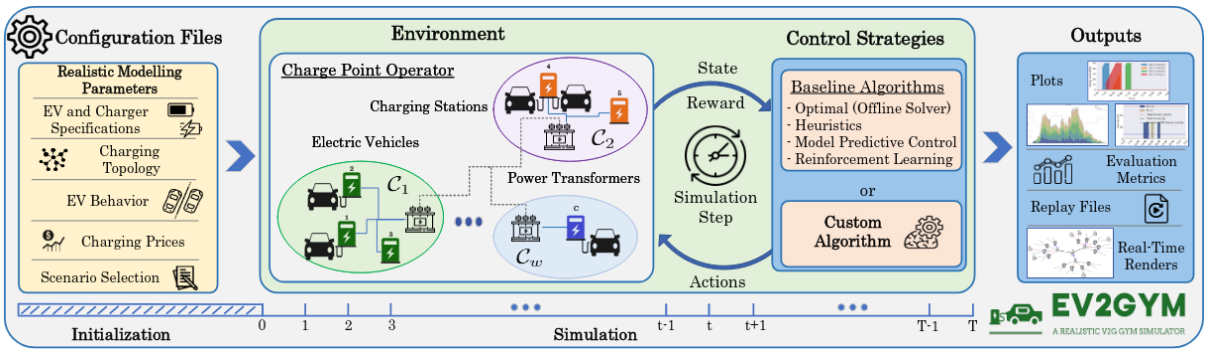

Figure 4.2: Phases and interactions while simulating a single episode in EV2Gym [9].

Simulation flow

After the initialization phase, the simulation is started. From the diagram in Figure 4.2 it can be seen that during this phase, the environment takes the action provided by the agent and uses it to update the different objects of the simulation. As the design is modular, this can be achieved by just calling the *step* method of each class. Once the step is done, the simulator calls both two pre-defined functions explained in subsections 4.2.2 and 4.2.3 to produce the state vector and reward metric that are returned to the agent. The agent-environment interaction is standardized and follows the Gym API [38], ensuring a smooth integration with other pre-existing libraries such as Stable Baselines3 [39] (SB3).

At each time step, the *environment* class checks for a series of termination conditions. Once they are achieved, the simulation is stopped and the agent is notified. Next, functions such as *plots* or *evaluator* are called (if needed). Finally, the simulator calls the *reset*

method to repeat the initialization phase in case that another episode shall be simulated.

4.2. Modifications done

The simulator was created for scenarios in which the agent follows a defined charging trajectory (set-point tracking) to maximize profits in scenarios where vehicle to grid (V2G) power injection is possible. Also, no flexibility is modeled in the simulator. For this reason, almost none of the predefined state and reward functions can be used. During this section, the main modifications made within the code [40] will be presented and justified.

4.2.1. Adding flexibility

Flexibility has been added to the simulator following the guidelines from subsection 2.2.1. That is, upwards and downwards flexibilities are modeled according to the equations (2.13) and (2.14).

To add these new parameters to the code, first a set of new attributes are added to the *environment* class to store the historical data of said flexibilities. Then, a new method was created for said class called *check to add flex*. This function takes the data from a charging station and calculates if the flexibility for said charging station should be added based on three criteria:

- There is one EV connected to the charging station.
- The SoC of the EV connected is not at 100%.
- The time to charge at maximum power is lower than then departure time. This condition check if the charger actually has the option to not charge at maximum power.

Said method is then called when the simulator is updating itself, that is, when calling the *step* method for each *charging station*. The method takes the data from the charging station, calculates the minimum time to charge using the charger and EV parameters, and if the three conditions are met, both flexibilities are added to their corresponding variables for said time step, else a 0 is added.

4.2.2. State Vector

Most of the predefined *state* functions include information regarding set-point tracking. However, creating a new one proved to be a simple task, as it is just required to include the desired environment variables in a 1D vector.

For the profit maximization case, while also considering flexibility, the proposed state vector is shown in Figure 4.3. First, the current time step is included as a number bounded between 0 and 1. This is done to preserve stability as passing large and unbounded numbers can lead to unstable gradients during training. Then, both the aggregated power and flexibility from the last time step are passed as two separate numbers. As the agent has to decide the charging strategy, the price for electricity in €/kWh is passed for a predefined number of steps, which is dependent on the variable PH. Finally, for each charging station, the value for the SoC is passed (bounded between 0 and 1) and the remaining time for the EV at its corresponding charging station. If there is no EV connected to the charger, both variables are set to 0.

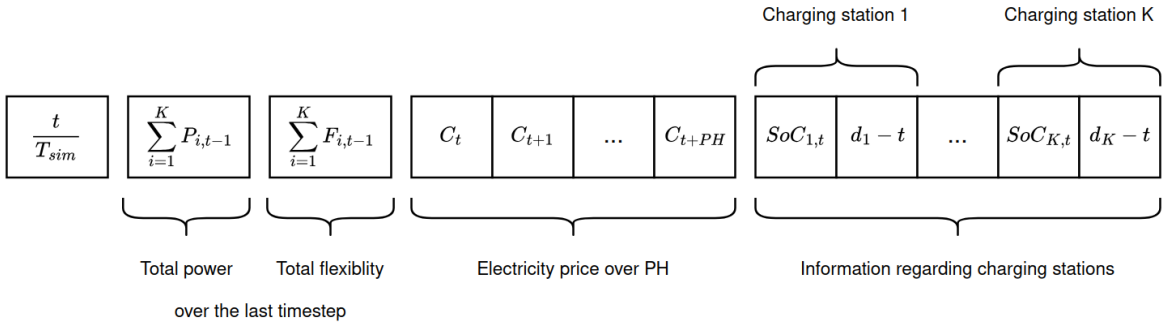

Figure 4.3: State vector passed to the agent at time step t considering an scenario with K charging stations.

4.2.3. Reward function

Correctly modeling the reward function should be one of the main objectives when constructing an environment, as it has a huge weight on the agent's final behavior. As the main goal of the agent shall be profit maximization, the reward can be modeled as the price for energy used to charge the EVs, the profit gained from the current flexibility (considering multiplier π) and user satisfaction reward function or L^{USR} as shown in (4.1). This reward function resembles the objective function from section 2.2 but it is **inverted**. This is because while the MPC aims to minimize the objective function, this reward is

designed to be **maximized**.

$$r(t) = -c_t \sum_{i=1}^{K} P_{i,t} + \pi \cdot c_t \sum_{i=1}^{K} F_{i,t} + L^{USR}(t) \tag{4.1}$$

The user satisfaction reward (USR) is a new addition with respect to the MPC. This function is only called in steps in which an EV is being dispatched (EV leaving its charging station) and it will penalize the agent according to a *score*, calculated as the difference between the final SoC of the EV at time of departure and the target SoC (which in this case is set to 100%). This function needs to be constructed and tuned as, if not added, the optimal policy for the agent would be to just not charge the EVs. In the case of MPC, this condition is imposed as a hard constraint, as can be seen in (2.7) , but as in RL this is not possible, it should be added as a soft constraint.

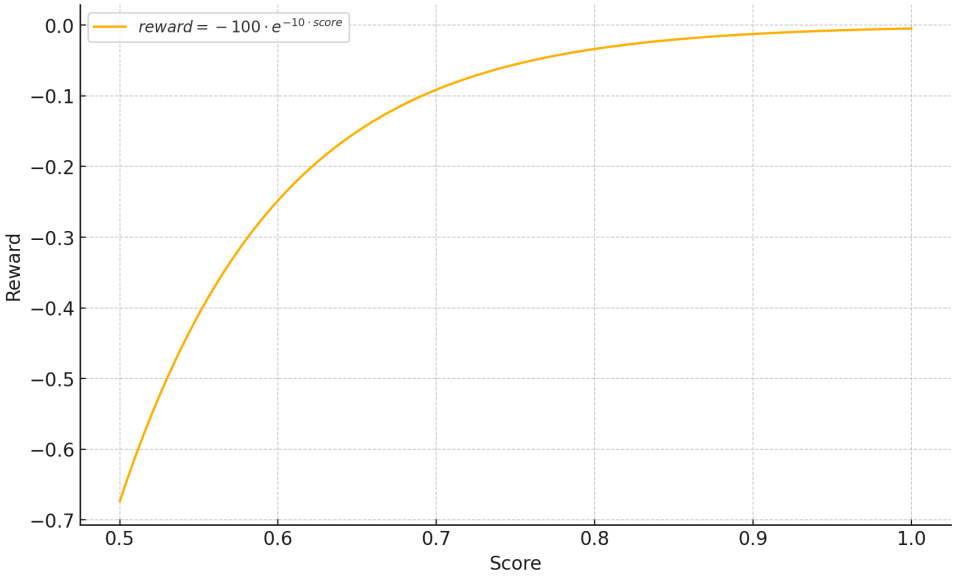

Figure 4.4: Default user satisfaction function provided by EV2Gym

To test the effectiveness of using the default USR provided by EV2Gym, different agents were trained on a simple scenario with just two charging stations and no flexibility reward ($\pi_F = 0$). After training, all of the agents converged to policies in which not charging the EVs was the best option to maximize the reward. Further analysis revealed that, when considering a staring SoC of 70%, a mean price of 0.15 €/kWh, and EVs with a maximum capacity of 50 kWh, charging the EV would yield a reward of -2.25. After looking at Figure 4.4, it is not surprising that the agents converged to said policy, as when considering the same scenario, not charging the EV would lead to a reward of -0.1.

As it was discovered that further penalization is needed, it was decided that to encourage charging, a stronger exponential penalization is needed for SoCs lower than 85%. On the other hand, an exponential reward was added for values greater than 95% to encourage charging over said percentage, else, the optimal strategy would be to just charge at 85%. Equation (4.2) shows the new piece-wise exponential function that was developed.

$$L^{USR}(x) = \begin{cases} -e^{-15(x-0.85)} & \text{if } x < 0.85 \\ 0 & \text{if } 0.85 \leq x < 0.95 \\ e^{15(x-0.95)} & \text{if } x \geq 0.95 \end{cases} \tag{4.2}$$

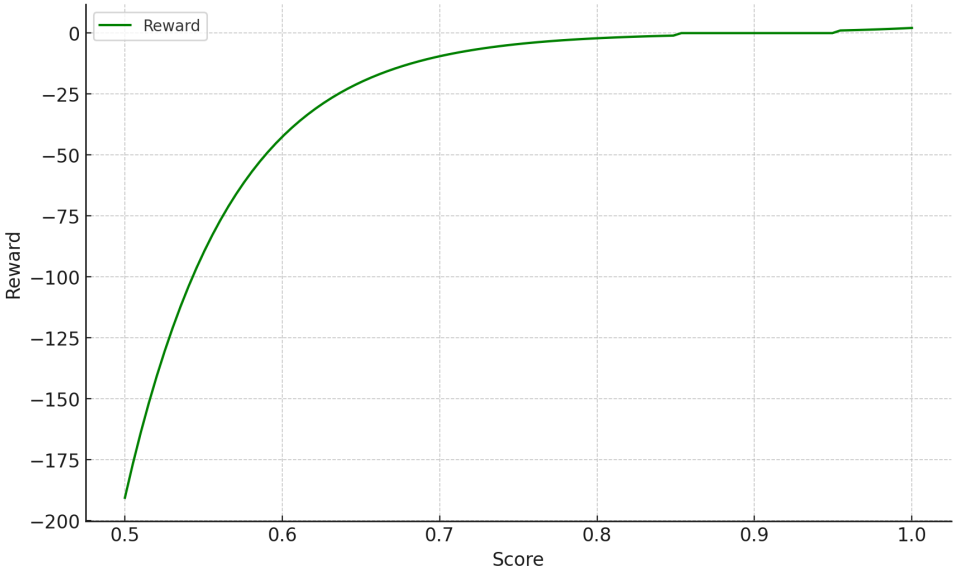

Figure 4.5: Exponential user satisfaction reward function.

The same process of validation was followed for the new exponential function. However, in this case, the agents did not even converge to a fixed policy. Mainly because, as exponential functions were used, agents were penalized too heavily when they did not even knew which type of actions to take, making training extremely unstable.

A final test was carried out, this time using the linear piece-wise function from (4.3) whose plot is shown in Figure 4.6. Using said function proved to be the best option, as while performing poorly from the start, all agents converged to reasonable policies in which the EVs were charged with a final SoC between 90% and 95%. This will be explained with greater detail in chapter 5.

$$L^{USR}(x) = \begin{cases} 100x - 80 & \text{if } x < 0.80 \\ 0 & \text{if } 0.80 \leq x < 0.85 \\ 13.33x - 11.33 & \text{if } x \geq 0.85 \end{cases} \tag{4.3}$$

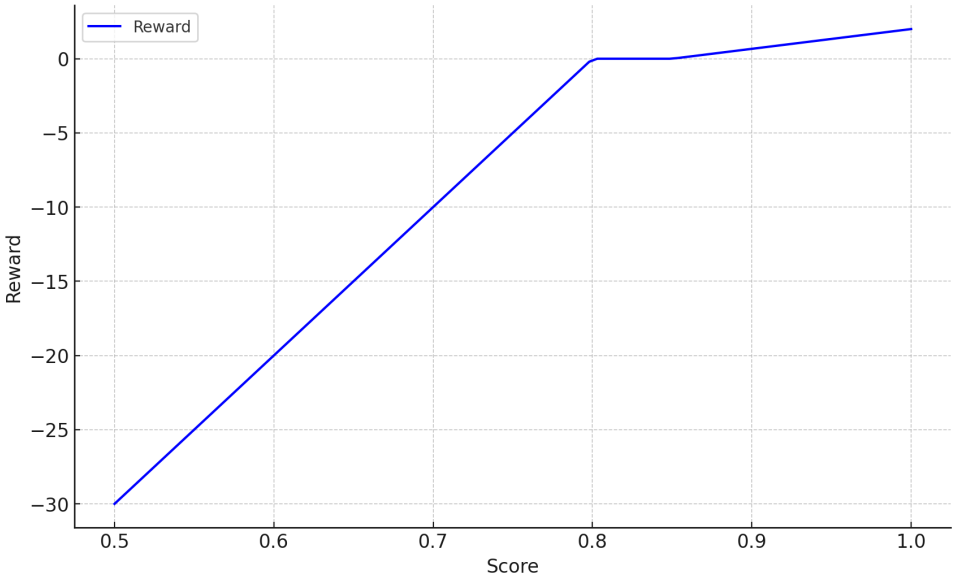

Figure 4.6: Linear user satisfaction reward function.

4.2.4. EV arrival distributions

The simulator uses real-world statistics from different scenarios (public, residential or work) to generate the arrival time, time of stay and initial SoC of each EV that is spawned during simulation. Considering the public scenario, it can be seen from Figure 4.7 that the vast majority of EVs being simulated will start with a relatively high SoC (between 70% and 80%) and a long time of stay of around 4 to 8 hours. However, while this methodology accurately represents a real-world situation, it makes most of the training episodes useless for training. This is because in most events, the EVs arrive almost fully charged (see Figure 4.8), therefore the agent cannot get experience on developing a successful charging strategy.

To get the most training out of each episode, the initial SoC is sampled from an uniform distribution between 40% and 60% and the parking time is sampled from a normal distribution $\max(\mathcal{N}(4, 0.8^2), 2)$ as shown in Figure 4.9. Samples of parking time less than 2 hours are not allowed as it would require charging at maximum power, thus not being helpful in the development of a policy.

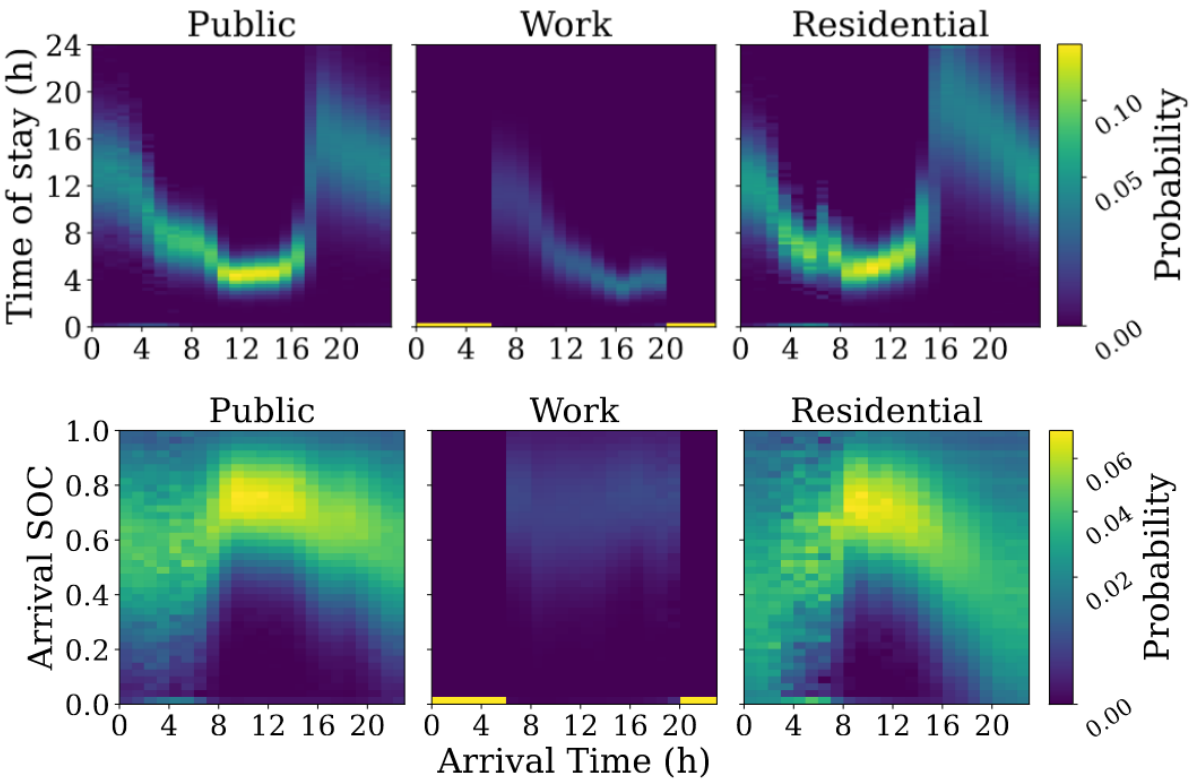

Figure 4.7: Probability density function for the arrival SoC and time of stay depending on the arrival time for the three different scenarios that the simulator offers [9].

Figure 4.10 shows an episode being generated with the previously described modifications. When comparing to Figure 4.8, it is clear that each episode is more useful for training the agent. As the EVs arrive with a reasonable SoC and EVs are being generated per episode, thus speeding up training. It should be noted that agents trained with the modified probabilities shall perform better when evaluated using the default probabilities, as they have been trained in a **pessimistic scenario.**

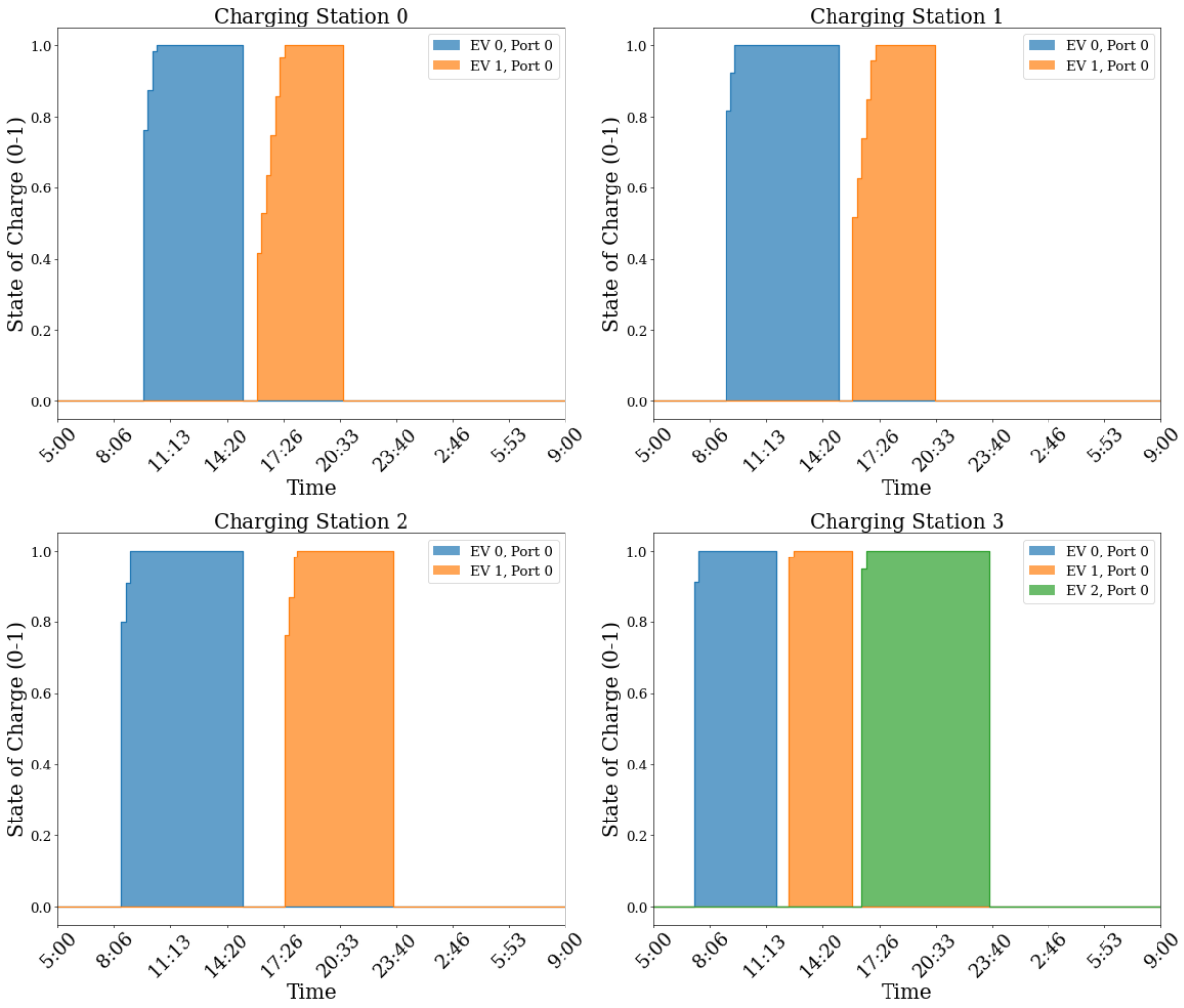

Figure 4.8: Sample episode considering 4 charging stations using AFAP charging strategy with default probability distributions

4.2.5. Matlab integration

EV2Gym comes with its own implementation of an MPC, so developing a solution for section 2.2 is possible inside the simulator. However, for simplicity and to make the process of plotting easier, it was decided to export the simulated episodes into Matlab and run the OCCF code with said data.

To achieve this task, a new method has been added which creates a dictionary with all the useful variables from the simulation and saves them into a .mat file. This method is called when the termination criteria for the episode is met and the option *savemat* has been set to *True* in the configuration of the environment.

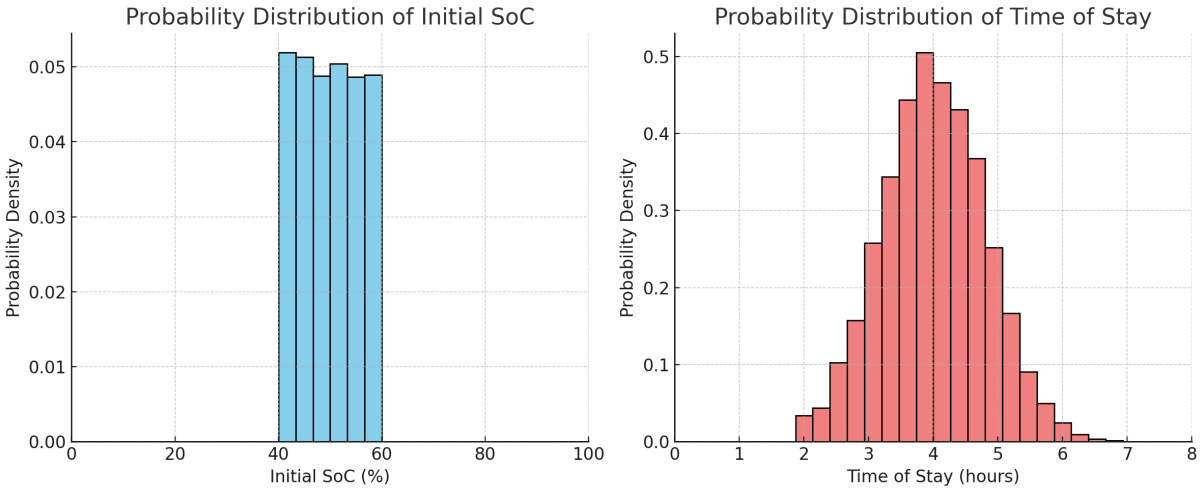

Figure 4.9: Updated probability density functions for initial SoC and time of stay.

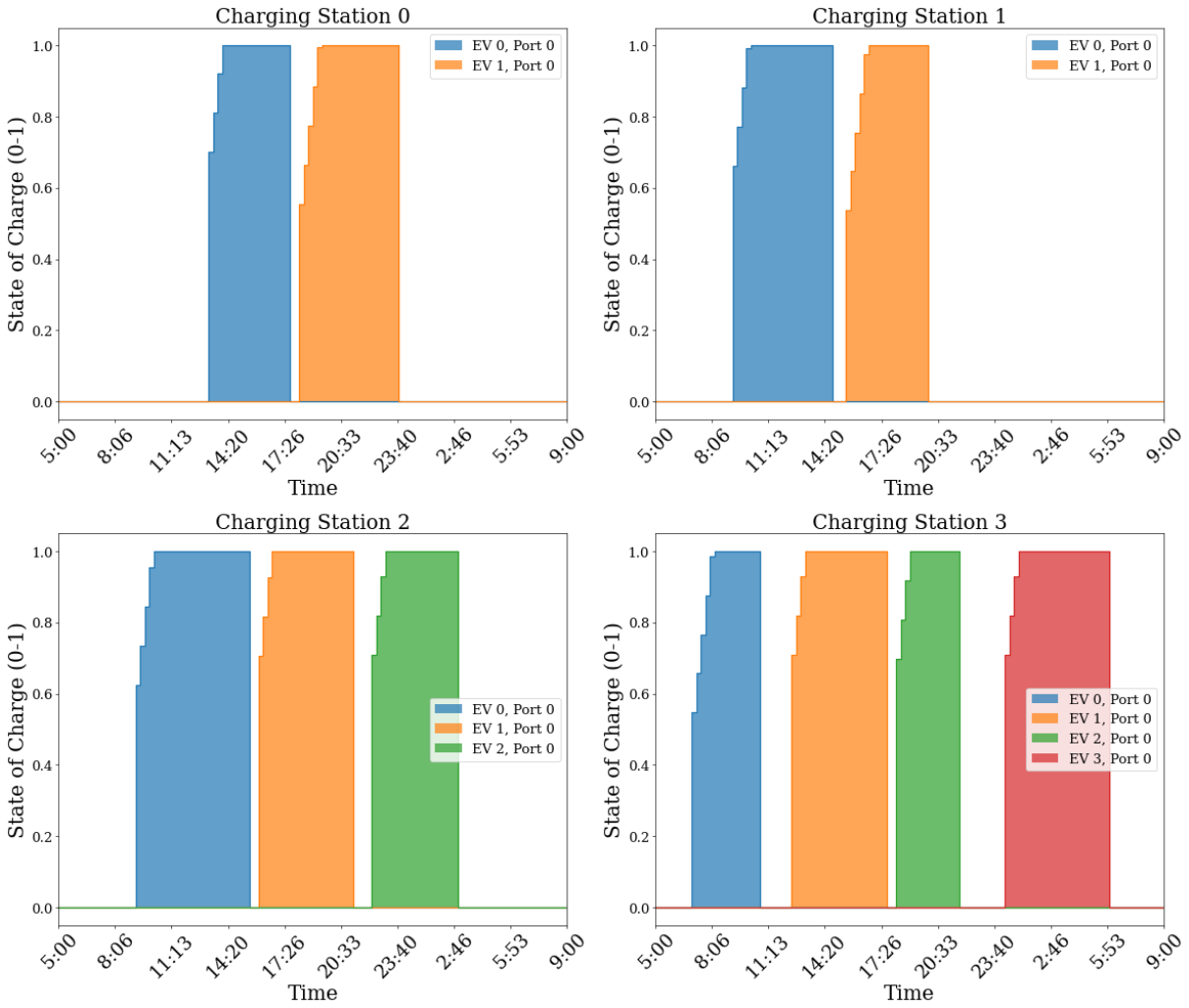

Figure 4.10: Sample episode considering 4 charging stations using AFAP charging strategy with modified probability distributions.

5 | Experimental results

After having a functional environment, the RL methods presented in chapter 3 can be tested and compared against the optimization-based strategies described in chapter 2. As different RL agents will be compared against MPC strategies using different flexibility multipliers, the analysis shall be separated into different parts: first, the effects that said multiplier has on the results shall be explained and only a few will be selected for the comparison of the different RL agents. After selecting the set of multipliers, two out of four RL methods will be chosen and thus the final analysis will only include the best performing agents against the MPC methods. But before jumping into the results, the methodology for training and comparison shall be explained for reproducibility purposes.

5.1. Experimental setup

All episodes used for training and evaluation are generated by the modified environment from in chapter 4 using the specific parameters presented in table 5.1, which were selected to facilitate the comparison with [11]. Before training even starts, 10 evaluation episodes are generated as they are used to generate the final metrics for each agent. Over said episodes, one evaluation episode is selected for plotting the trajectories shown throughout this chapter.

As one of the objectives of the analysis shall be to understand how the flexibility multiplier affects the final behavior of the agents, every agent is trained with a set of different flexibility multipliers, in this case, they were selected as $\pi = [0.0, 0.1, 0.3, 0.5, 1.0]$.

Simulation Parameter	Value
Time step duration	10 min
Episode lengh	24 hours
Prediction Horizon	6 hours (36 steps)
Number of chargers	15
Simulated scenario	Public
Maximum power per charger	22 kW
EV battery capacity	50 kWh

Table 5.1: Simulator parameters used for training and evaluation.

RL training

Four RL agents were selected for the comparison: DDPG, TD3, PPO and TRPO. Each agent was trained using the previously defined values of π with the default SB3 parameters for 1 million iterations, which proved to be sufficient for all training metrics to stabilize. During training, an evaluation function is called every 2500 iterations to test the agent's performance in deterministic mode (meaning that no exploration is performed) for 5 random episodes. After obtaining the mean reward of the evaluation episodes, the script saves the current agent's weights if the mean reward is the highest achieved so far. Once training has stopped, the weights of the best performing agent are loaded and the agent is evaluated on the initial 10 evaluation episodes, thus obtaining the final metrics.

MPC evaluation

As MPC only solves a minimization problem at each time step, no training is required and the episodes can be evaluated directly. As the evaluation episodes were generated using the *savemat* flag, they can be directly imported into Matlab and evaluated using different flexibility multipliers to obtain the metrics.

5.2. Overall results

The results over the evaluation episodes are presented in the following tables. At a first glance, the main difference between RL methods and MPC lies in the user satisfaction, in which MPC achieved 100% in all episodes. This is expected because, as discussed through 4.2.3, the MPC has a hard constrain which imposes this behavior, while in RL methods this is achieved by a soft constrain. Also, it should be noted that no reward function is shown for the MPC, as it does not use this metric.

Metric	DDPG	TD3	PPO	TRPO	MPC
Profits (€)	-203	-213	-206	-167	-217
Energy cost (€)	-203	-213	-206	-167	-217
Up Flexibility (kWh)	1875	1737	2406	3017	2836
Down Flexibility (kWh)	885	933	910	732	1144
Energy Charged (kWh)	960	1004	981	803	1145
User Satisfaction (%)	93.1	94.8	94.1	86.1	100.0
Reward	-293	-251	-253	-445	-

Table 5.2: Agent performance for $\pi = 0.0$

Metric	DDPG	TD3	PPO	TRPO	MPC
Profits (€)	-166	-173	-125	-139	-133
Energy cost (€)	-223	-228	-206	-214	-224
Up Flexibility (kWh)	1702	1597	2946	2618	3224
Down Flexibility (kWh)	984	1001	918	954	1144
Energy Charged (kWh)	1070	1086	993	1030	1145
User Satisfaction (%)	97.8	98.6	94.3	96.6	100.0
Reward	-125	-112	-135	-107	-

Table 5.3: Agent performance for $\pi = 0.1$

Metric	DDPG	TD3	PPO	TRPO	MPC
Profits (€)	-33	-23	52	45	49
Energy cost (€)	-218	-216	-194	-201	-224
Up Flexibility (kWh)	1964	2084	3052	3008	3227
Down Flexibility (kWh)	951	952	867	903	1144
Energy Charged (kWh)	1042	1040	937	976	1145
User Satisfaction (%)	96.5	96.6	92.5	93.6	100.0
Reward	-22	-4	1	18	-

Table 5.4: Agent performance for $\pi = 0.3$

Metric	DDPG	TD3	PPO	TRPO	MPC
Profits (€)	148	109	232	229	231
Energy cost (€)	-204	-215	-190	-188	-224
Up Flexibility (kWh)	2440	2115	3172	3138	3228
Down Flexibility (kWh)	894	953	845	839	1144
Energy Charged (kWh)	984	1039	917	910	1145
User Satisfaction (%)	94.1	96.8	91.1	90.8	100.0
Reward	96	128	149	130	-

Table 5.5: Agent performance for $\pi = 0.5$

Metric	DDPG	TD3	PPO	TRPO	MPC
Profits (€)	625	512	706	700	715
Energy cost (€)	-157	-194	-161	-163	-239
Up Flexibility (kWh)	3014	2508	3421	3389	3416
Down Flexibility (kWh)	693	844	720	727	1145
Energy Charged (kWh)	763	939	776	789	1145
User Satisfaction (%)	83.8	92.1	84.9	85.3	100.0
Reward	233	409	448	433	-

Table 5.6: Agent performance for $\pi = 1.0$

5.2.1. Flexibility multiplier analysis

Continuing the analysis on the user satisfaction, Figure 5.1 shows the effects that the flexibility multiplier π has on this metric. It can be seen introducing the flexibility multiplier π results in overall higher user satisfaction (especially for $\pi = 0.1$) as the agent has more incentive to charge the EV when electricity prices are lower and does not charge when they are high, as this will generate income in the form of flexibility reward. However, when considering values for $\pi \geq 0.5$, the user satisfaction metric is much worse although the profits (and reward) given to the agent is higher. In these scenarios, agents found that the best strategy is to charge the EV to just 85% not to be penalized according to (4.3) and maximize the passive reward given by the flexibility. This behavior can also be explained by looking at the mean metrics for RL agents (excluding MPC) in Figure 5.2, in which upwards flexibility and profits are considerably higher for $\pi = 1.0$ while energy cost and user satisfaction are lower.

From said figure, one can also see that while higher values of π increase the upwards flexibility, they also decrease the downwards flexibility, which may seem counter-intuitive at first. From its definition in equation (2.14) one can see that while charging the EV, this

flexibility is equal to the power provided by the charger. Meaning that, as the strategies for agents with higher values of π do not charge the EVs as much, the chargers provide less power and thus the downward flexibility is considerably lower. This also explains why the downwards flexibility is equal when considering any value of π for the MPC, as this metric is bounded by the energy that the charger can supply, which is equal to fully charging all EVs.

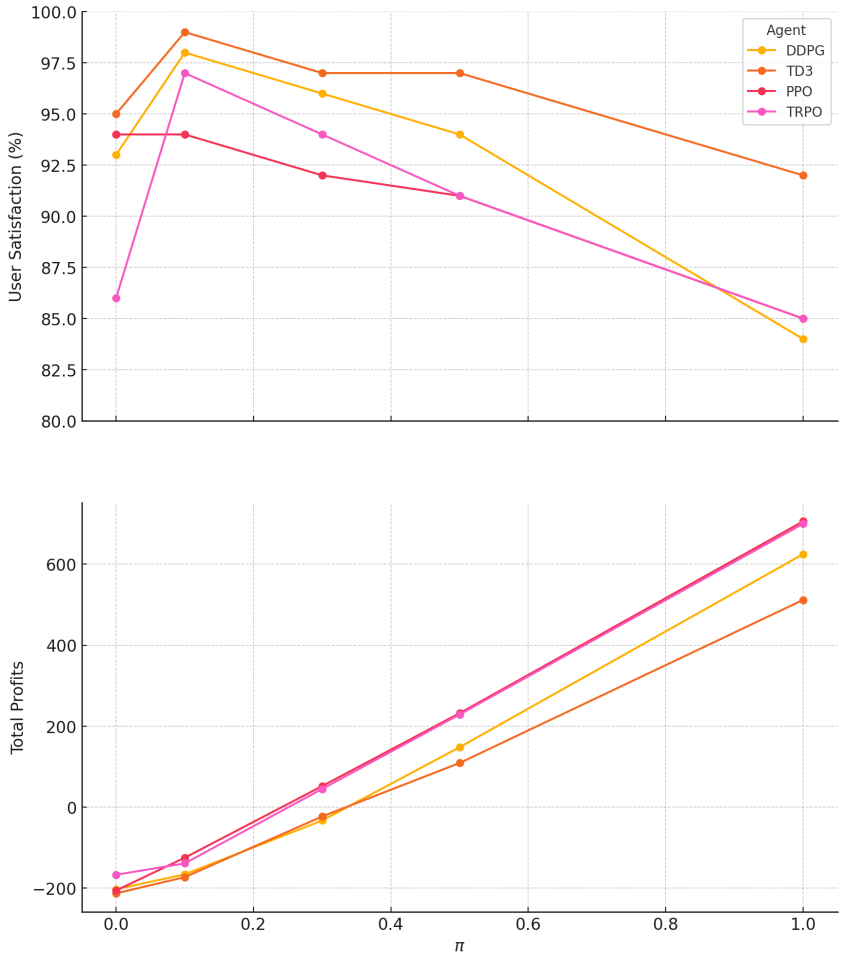

Figure 5.1: User satisfaction and Profits for RL agents depending on flexibility multiplier π

5.2.2. RL agent comparison

Once the analysis with respect to the flexibility multiplier is finished, a comparison between the different RL methods shall be done. For simplicity, this analysis will only include agents trained with values of $\pi = [0.1, 0.3, 0.5]$ as other values resulted in strategies that maximized the reward by not charging the EVs, so they can be discarded from the analysis.

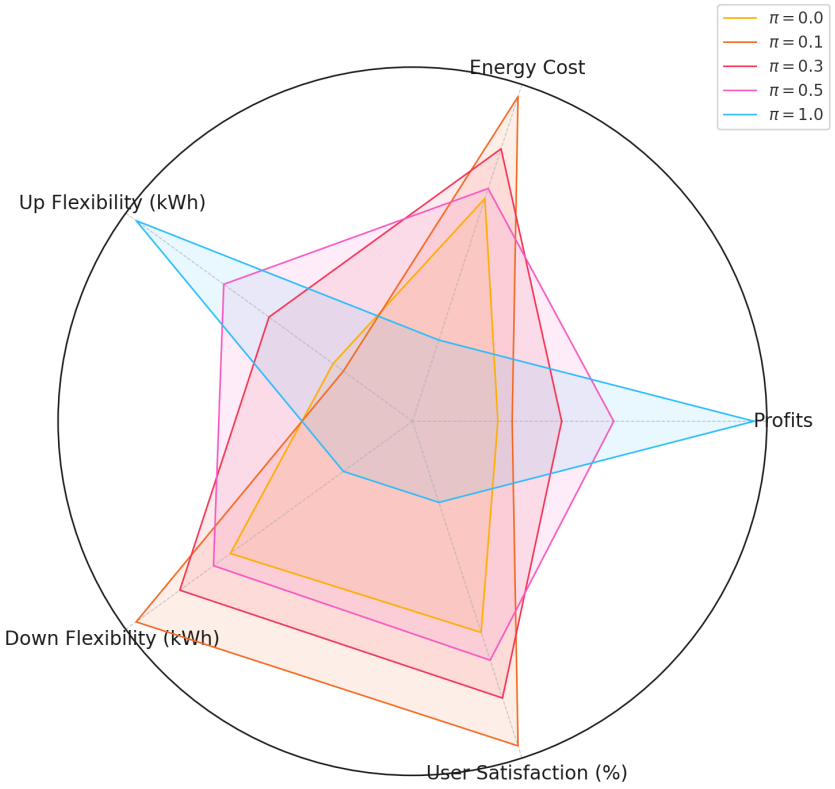

Figure 5.2: Mean performance for RL agents across different values of π

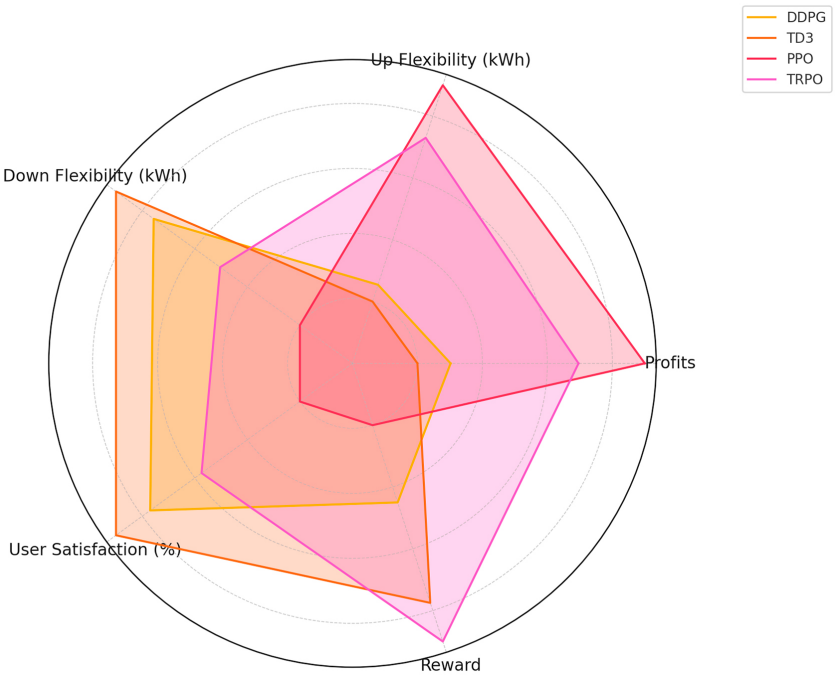

Figure 5.3: RL agents performance metrics for $\pi = 0.1$

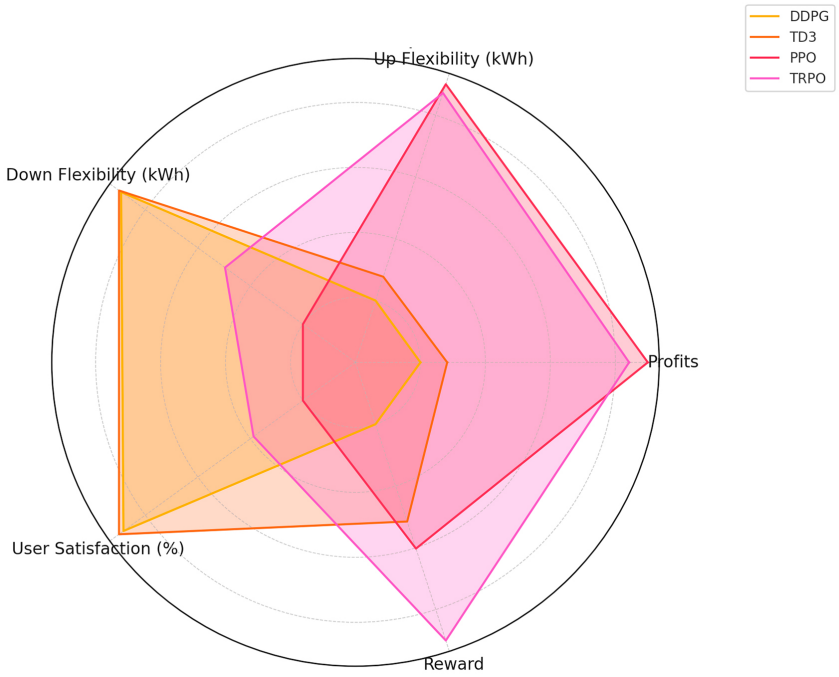

Figure 5.4: RL agents performance metrics for $\pi = 0.3$

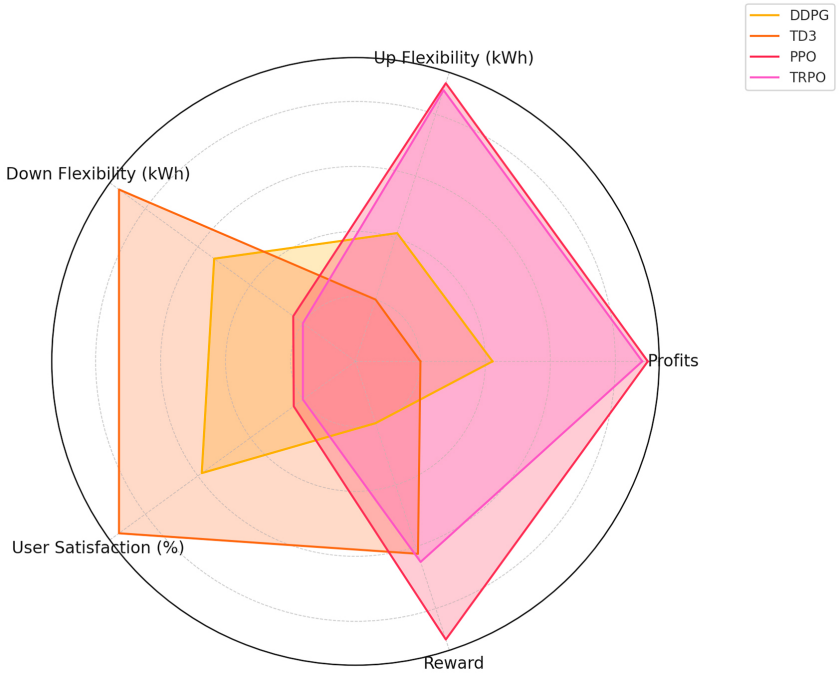

Figure 5.5: RL agents performance metrics for $\pi = 0.5$

From Figures 5.3, 5.4 and 5.5, one can see that agents have developed different strategies to maximize reward. DDPG and TD3 took a strategy that charges the EVs at an acceptable range and thus obtains a higher downward flexibility. On the other hand, PPO

and TRPO performed much better in their main objective (maximizing reward) by not fully charging the EVs and thus spending less on energy and obtaining higher values for upwards flexibility.

Although according to the reward function, the best performing agents should be PPO and TRPO, for the purposes of this thesis DDPG and TD3 are chosen as they developed better charging strategies considering that almost all EVs arrived to the desired SoC as can be seen in Figure 5.6. Also, it is worth comparing the power profiles of said agents in Figure 5.7, where one can see that while DDPG follows a reasonable control strategy to charge the EVs when electricity prices are low, PPO and TRPO use a bang-bang type of control strategy, which is undesired as this can cause negative effects on battery health.

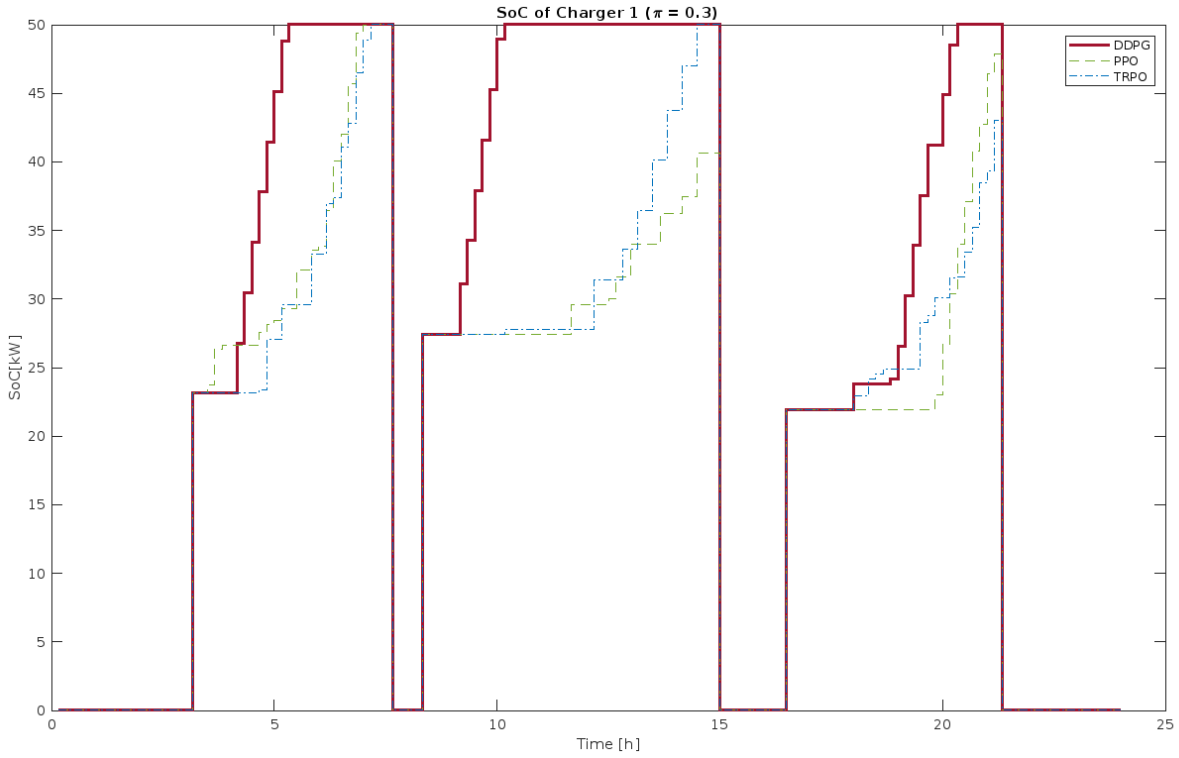

Figure 5.6: SoC evolution of different agents during plotting episode considering $\pi = 0.3$

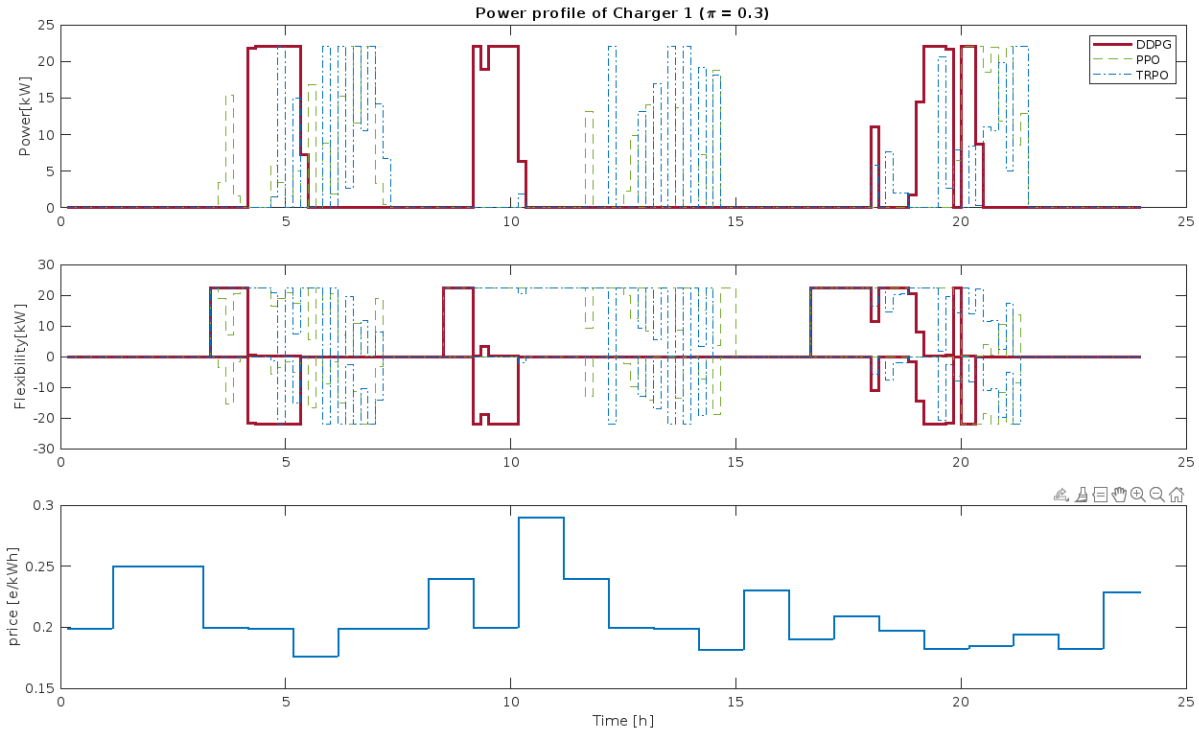

Figure 5.7: Power profile of different agents during plotting episode considering $\pi = 0.3$

5.2.3. RL agents vs. MPC

Now that the analysis has been narrowed down to a couple of agents, a more in-depth comparison can be achieved between RL methods and MPC. This part of the analysis will cover the overall metrics obtained by the agents, while section 5.3 will compare the power profiles of the agents.

The metrics for the final RL agents (DDPG and TD3) along with the MPC have been aggregated in Table 5.7, which has been used to construct Figure 5.8 . Overall, MPC consistently outperformed RL methods across all performance metrics. Which makes sense considering that MPC is solving a minimization problem and taking full advantage of the model's dynamics at each time step. In these cases where the full model is available, it is expected that using a model-free method such as RL will yield worst results.

π	Agent	Profits (€)	Energy Cost (€)	Up Flexibility (kWh)	Down Flexibility (kWh)	User Satisfaction (%)
0.0	DDPG	-203	-203	1875	885	93,1
	TD3	-213	-213	1737	933	94,8
	MPC	-217	-217	2836	1144	100,0
0.1	DDPG	-166	-223	1702	984	97,8
	TD3	-173	-228	1597	1001	98,6
	MPC	-133	-224	3224	1144	100,0
0.3	DDPG	-33	-218	1964	951	96,5
	TD3	-23	-216	2084	952	96,6
	MPC	49	-224	3227	1144	100,0
0.5	DDPG	148	-204	2440	894	94,1
	TD3	109	-215	2115	953	96,8
	MPC	231	-224	3228	1144	100,0
1.0	DDPG	625	-157	3014	693	83,8
	TD3	512	-194	2508	844	92,1
	MPC	715	-239	3418	1144	100,0

Table 5.7: Final RL agents and MPC metrics

As said previously, using a higher flexibility multiplier results in higher upwards flexibility for RL agents. However, it is curious to see that this multiplier does not affect the MPC as much when examining the upwards flexibility. The reasoning behind this will be explained in the next section, where the power profiles of the charging strategies will be examined.

5.3. Episode Analysis

In this section, the charging strategies of the final RL methods will be analyzed and compared with the MPC. For this purpose, a random episode from the evaluation set was chosen and one charger station with three events was selected for plotting. Figure 5.9 shows the different power profiles when there is no flexibility reward, showing that all the methods use a coherent charging strategy in which the EV is charged when the electricity prices are low. However, it can be seen that MPC has no problem charging at maximum power when the lowest price is available, while RL methods may not choose those instants.

As said before, it is surprising to see how little effect the flexibility multiplier has on the upwards flexibility (Figure 5.8). But by looking at the specific power profile in Figure 5.9 it can easily be explained. Although in this case the eMPC is not taking into consideration the flexibility, it is being maximized as the charger is waiting until the EV has to

depart, which coincides with the time at which the price is minimum. When the episode is repeated using a different and flatter price range, the eMPC just charge whenever the electricity prices are lower and thus a lower flexibility is obtained, as can be seen in Figure 5.14. In fact, here one can see that the optimal strategy to maximize flexibility is to charge the EV at a constant power such that the charger remains upward and downward flexible throughout the entirety of the parking time, without any consideration with respect to the price of electricity.

Once the flexibility multiplier is introduced, the behavior of the RL agents starts changing as can be seen in Figure 5.10, as both DDPG and TD3 start to make more clear decisions on when to charge the EV. However, the timing is not perfect as they are not choosing the moments in which electricity prices are lower.

A major improvement can be seen when the flexibility multiplier is set to $\pi = 0.5$, as Figure 5.12 shows that both RL agents are charging when electricity prices are lower and thus they maximize flexibility by chairing only at said instants. Nevertheless, a binary control strategy starts to appear.

Finally, the power profile from Figure 5.13 gives mixed results. On one hand, one can see that TD3 is following a good strategy as it charges just before departure (when electricity prices are lower) and thus provides more flexibility. The charging profile of the TD3 agent resembles the one taken by the eMPC in Figure 5.9. On the other hand, the user satisfaction for both DDPG and TD3 has dropped drastically. In fact, for this specific station, none of the EVs have been charged by the DDPG agent, although from table 5.7 one can see that the mean user satisfaction over all chargers and episodes for the DDPG is still 84%.

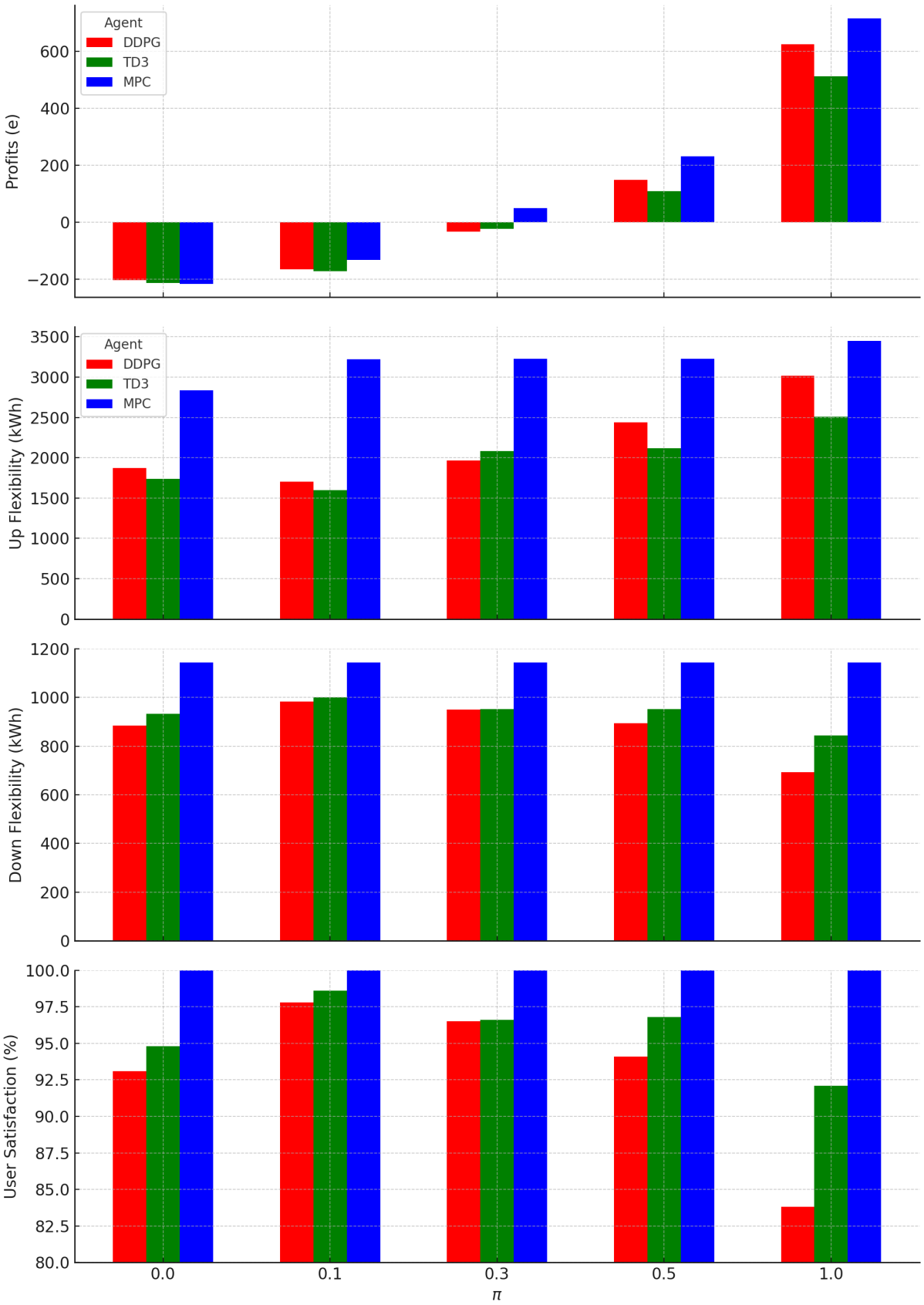

Figure 5.8: Metrics of DDPG, TD3 and MPC across different flexibility multipliers π.

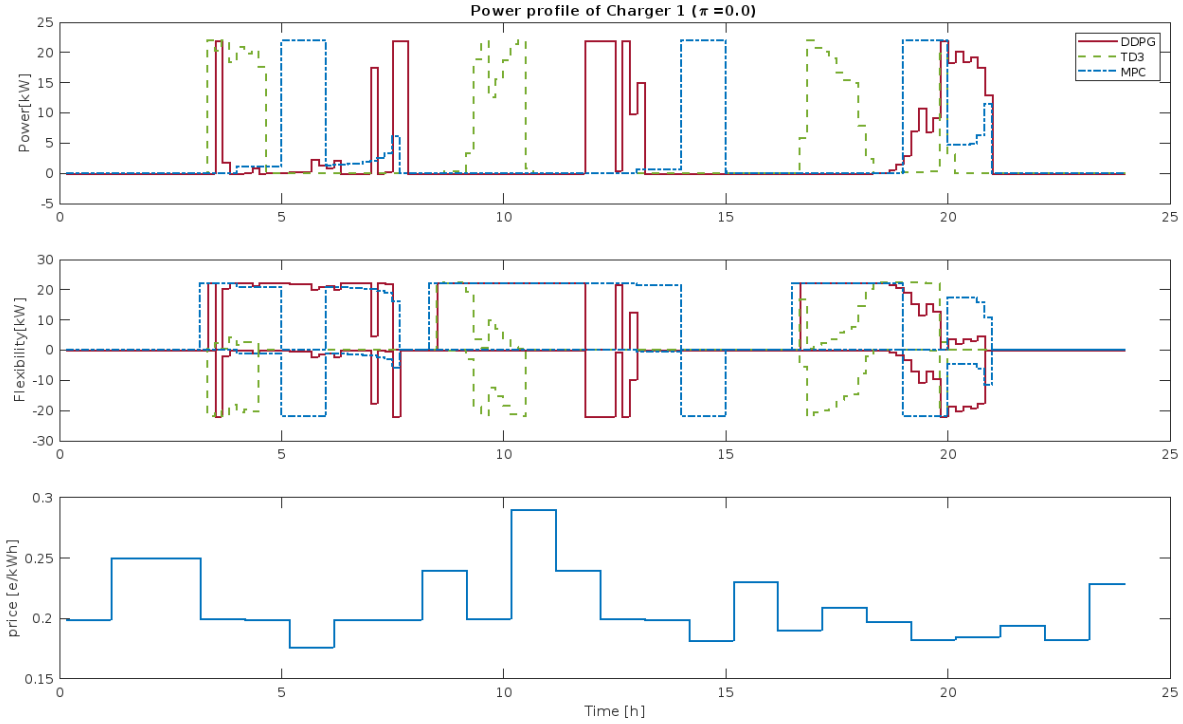

Figure 5.9: Power profile comparison for $\pi = 0.0$.

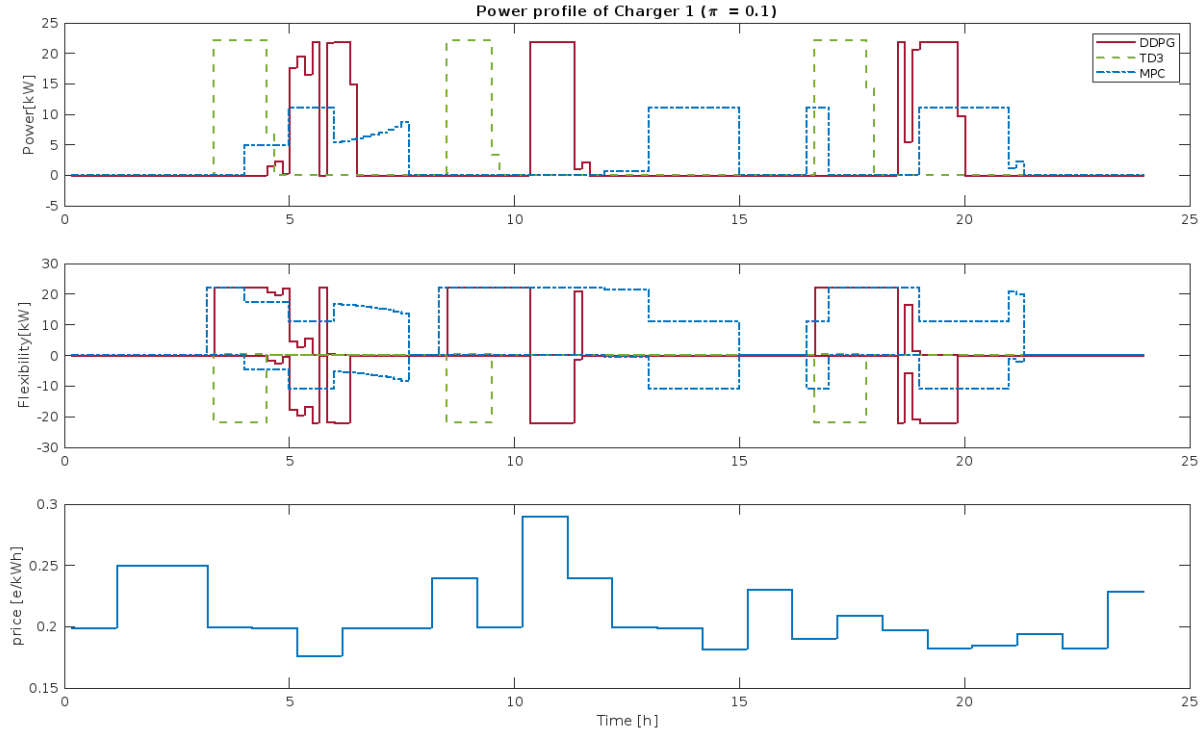

Figure 5.10: Power profile comparison for $\pi = 0.1$.

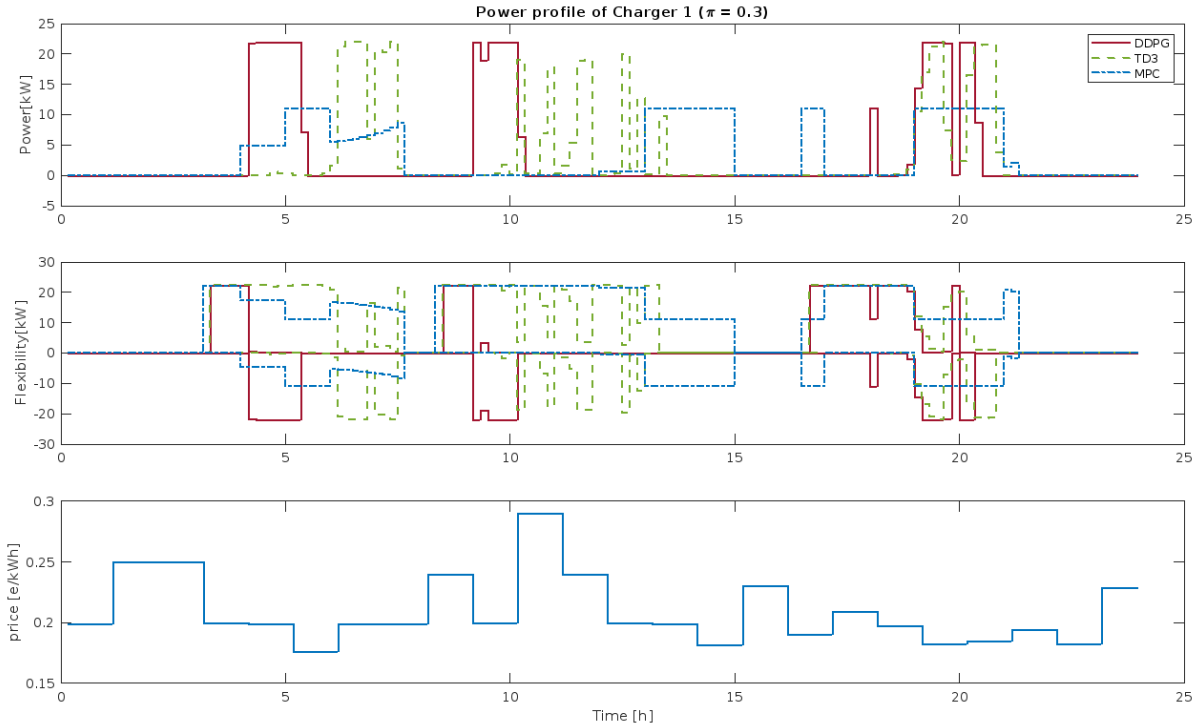

Figure 5.11: Power profile comparison for $\pi = 0.3$.

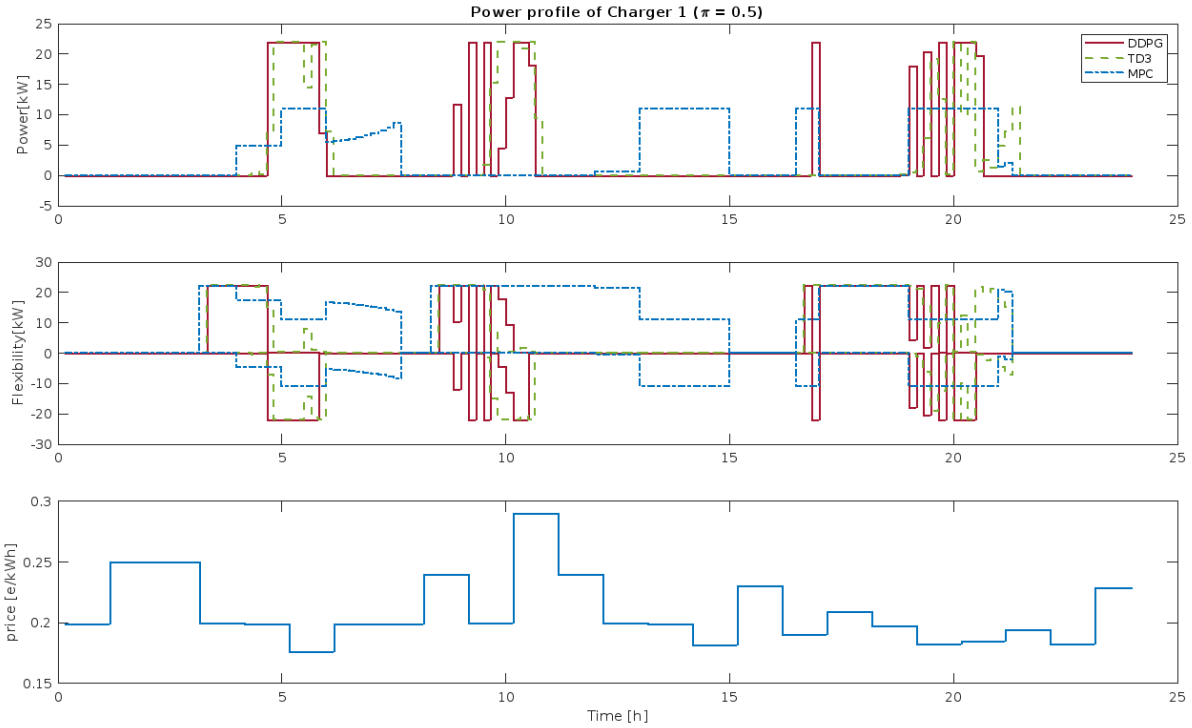

Figure 5.12: Power profile comparison for $\pi = 0.5$.

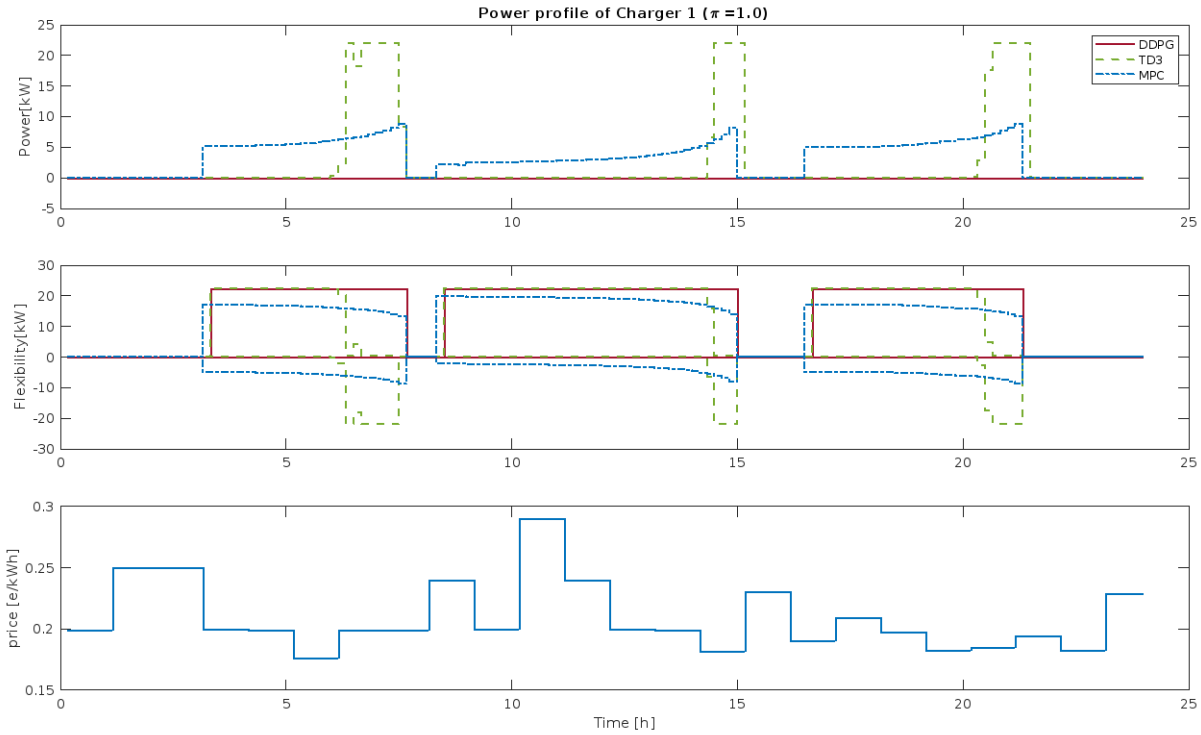

Figure 5.13: Power profile comparison for $\pi = 1.0$.

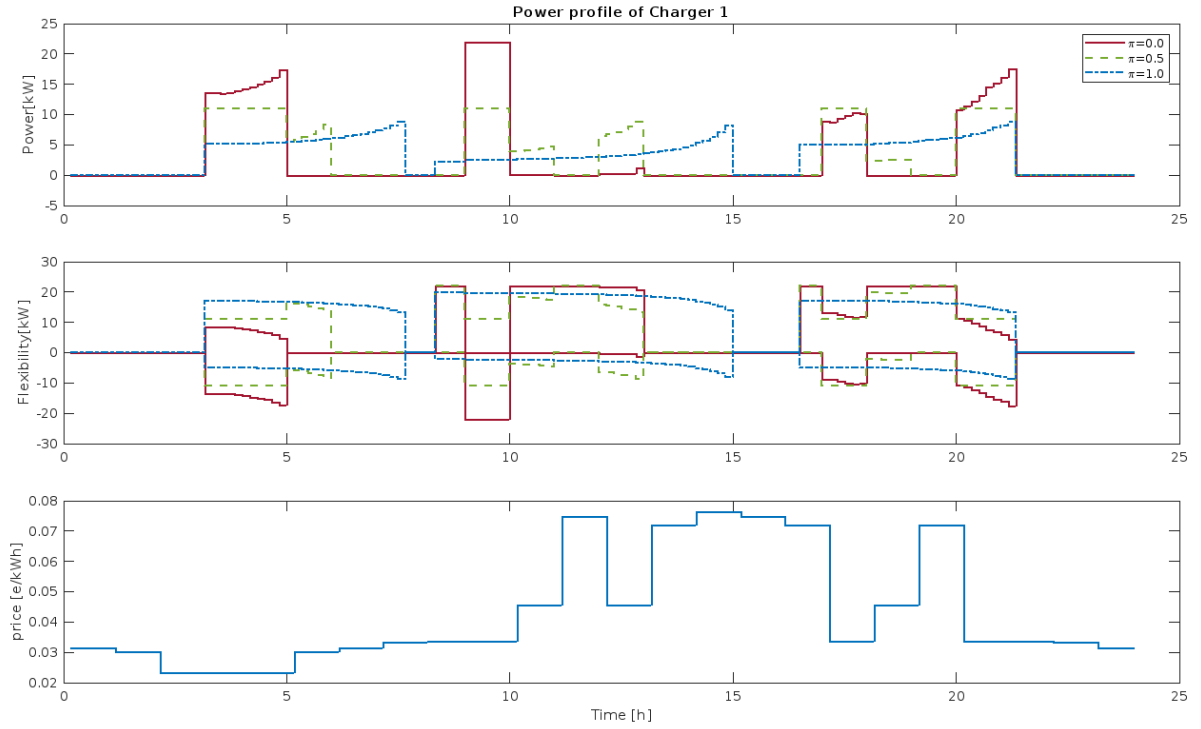

Figure 5.14: Power profile comparison considering MPC with $\pi = [0.0, 0.5, 0.1]$.

6 | Conclusions and Future work

Throughout this thesis, the framework to apply RL methods for the EV charging problem has been developed. And thanks to this, it has been proven that a wide variety of RL methods can be used to manage an aggregator to minimize cost while maximizing upwards and downwards flexibilities. However, the final behavior of the agent will not depend as much on its architecture, but on the reward function on which the agent is trained. This is based on the fact that none of the agents chose to charge the EVs to 100% as the reward for doing so is too weak, and when considering higher profits from flexibilities, this metric drops even further, suggesting that the penalization for not charging shall be scaled with respect to the flexibility multiplier.

When comparing the results of the DDPG and TD3 agents with respect to MPC, it can be concluded that MPC outperforms RL methods when minimizing cost and maximizing flexibility. This is not surprising as when dealing with a simpler problem from which one can develop a plant model, using a method that solves a minimization problem at each iteration will always deliver best results. However, the results delivered by RL are still acceptable and, as such, these methods shall be considered for cases in which flexibility maximization is required but forecasted data is not accurate or available.

To continue the development of RL solutions for flexibility maximization, first it shall be proven that using RL in more complex and real scenarios could result in better performance. This means that agents should be trained without future knowledge of parking times and-or electricity prices so that they develop policies that could predict said values based on past inputs and current time of the day (also depending on the day of the week). In this case, MPC can also be applied but it would require building a NARMA predictor to estimate said parameters. This is a sufficiently complex scenario to truly justify the use of RL.

When constructing this problem, a lot of trial and error would be required to develop

the final reward function. A good starting point would be to take the user satisfaction reward used this thesis but amplifying the reward obtained when charging above 95% to encourage agents to fully charge the EVs and scale the penalization with respect to the flexibility multiplier. Another term should be added to the reward to penalize abrupt changes in power output to discourage the agent into following a binary or bang-bang strategy.

Finally, if the agent should develop strategies without information about parking time and electricity prices, the neural network should be enlarged to develop better function approximation. And due to the momentum that the field of RL currently has, the simulator can be used without any major modifications on techniques that appear in the next years.

Bibliography

[1] International Energy Outlook 2016.

[2] Juan Ma, Vera Silva, Régine Belhomme, Daniel S Kirschen, and Luis F Ochoa. Evaluating and planning flexibility in sustainable power systems. In *2013 IEEE power & energy society general meeting*, pages 1–11. IEEE, 2013.

[3] Eric Hsieh and Robert Anderson. Grid flexibility: The quiet revolution. *The Electricity Journal*, 30(2):1–8, 2017. Publisher: Elsevier.

[4] Siwar Khemakhem, Mouna Rekik, and Lotfi Krichen. A flexible control strategy of plug-in electric vehicles operating in seven modes for smoothing load power curves in smart grid. *Energy*, 118:197–208, January 2017. ISSN 03605442. doi: 10.1016/j.energy.2016.12.039. URL https://linkinghub.elsevier.com/retrieve/pii/S0360544216318448.

[5] U.S. Department of Transportation. EV Charging Speeds | EV Toolkit for Rural Communities, 2024. URL https://www.transportation.gov/rural/ev/toolkit/ev-basics/charging-speeds.

[6] Cesar Diaz-Londono, Stavros Orfanoudakis, Pedro P. Vergara, Peter Palensky, Fredy Ruiz, and Giambattista Gruosso. A Simulation Tool for V2G Enabled Demand Response Based on Model Predictive Control, May 2024. URL http://arxiv.org/abs/2405.11963. arXiv:2405.11963 [eess].

[7] Alexis Pengfei Zhao, Shuangqi Li, Zhengmao Li, Zhaoyu Wang, Xue Fei, Zechun Hu, Mohannad Alhazmi, Xiaohe Yan, Chenye Wu, Shuai Lu, Yue Xiang, and Da Xie. Electric Vehicle Charging Planning: A Complex Systems Perspective. *IEEE Transactions on Smart Grid*, pages 1–1, 2024. ISSN 1949-3053, 1949-3061. doi: 10.1109/TSG.2024.3446859. URL https://ieeexplore.ieee.org/document/10643227/.

[8] Adrian-Petru Surani, Tong Wu, and Anna Scaglione. Competitive Reinforcement Learning for Real-Time Pricing and Scheduling Control in Coupled EV Charging Stations and Power Networks. 2024. doi: 10.24251/HICSS.2023.366. URL http://hdl.handle.net/10125/106749.

[9] Stavros Orfanoudakis, Cesar Diaz-Londono, Yunus E. Yılmaz, Peter Palensky, and Pedro P. Vergara. EV2Gym: A Flexible V2G Simulator for EV Smart Charging Research and Benchmarking, April 2024. URL http://arxiv.org/abs/2404.01849. arXiv:2404.01849 [cs].

[10] Cesar Diaz, Andrea Mazza, Fredy Ruiz, Diego Patino, and Gianfranco Chicco. Understanding Model Predictive Control for Electric Vehicle Charging Dispatch. In *2018 53rd International Universities Power Engineering Conference (UPEC)*, pages 1–6, Glasgow, September 2018. IEEE. ISBN 978-1-5386-2910-9. doi: 10.1109/UPEC.2018.8542050. URL https://ieeexplore.ieee.org/document/8542050/.

[11] Cesar Diaz-Londono, Luigi Colangelo, Fredy Ruiz, Diego Patino, Carlo Novara, and Gianfranco Chicco. Optimal Strategy to Exploit the Flexibility of an Electric Vehicle Charging Station. *Energies*, 12(20):3834, October 2019. ISSN 1996-1073. doi: 10.3390/en12203834. URL https://www.mdpi.com/1996-1073/12/20/3834.

[12] Richard Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515, 1954. ISSN 0273-0979, 1088-9485. doi: 10.1090/S0002-9904-1954-09848-8. URL https://www.ams.org/bull/1954-60-06/S0002-9904-1954-09848-8/.

[13] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning series. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018. ISBN 978-0-262-03924-6.

[14] Finite Markov Decision Process. In *Reinforcement learning: an introduction*, Adaptive computation and machine learning series. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018. ISBN 978-0-262-03924-6.

[15] Elena Pashenkova, Irina Rish, and Rina Dechter. Value iteration and policy iteration algorithms for Markov decision problem. In *AAAI'96: Workshop on Structural Issues in Planning and Temporal Reasoning*, volume 39. Citeseer, 1996.

[16] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, August 1988. ISSN 0885-6125, 1573-0565. doi: 10.1007/BF00115009. URL http://link.springer.com/10.1007/BF00115009.

[17] Temporal Difference Learning. In *Reinforcement learning: an introduction*, Adaptive computation and machine learning series. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018. ISBN 978-0-262-03924-6.

[18] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist*

systems, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[19] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8 (3-4):279–292, May 1992. ISSN 0885-6125, 1573-0565. doi: 10.1007/BF00992698. URL http://link.springer.com/10.1007/BF00992698.

[20] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, December 1989. ISSN 0932-4194, 1435-568X. doi: 10.1007/BF02551274. URL http://link.springer.com/10.1007/BF02551274.

[21] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. 2013. doi: 10.48550/ARXIV.1312.5602. URL https://arxiv.org/abs/1312.5602.

[22] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999. URL https://proceedings.neurips.cc/paper_files/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

[23] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, May 1992. ISSN 0885-6125, 1573-0565. doi: 10.1007/BF00992696. URL http://link.springer.com/10.1007/BF00992696.

[24] Monte Carlo Methods. In *Reinforcement learning: an introduction*, Adaptive computation and machine learning series. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018. ISBN 978-0-262-03924-6.

[25] Policy Gradient Methods. In *Reinforcement learning: an introduction*, Adaptive computation and machine learning series. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018. ISBN 978-0-262-03924-6.

[26] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning, June 2016. URL http://arxiv.org/abs/1602.01783. arXiv:1602.01783 [cs].

[27] Thomas Degris, Martha White, and Richard S. Sutton. Off-Policy Actor-Critic, June 2013. URL http://arxiv.org/abs/1205.4839. arXiv:1205.4839 [cs].

[28] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. 2015. doi: 10.48550/ARXIV.1509.02971. URL https://arxiv.org/abs/1509.02971.

[29] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic Policy Gradient Algorithms.

[30] G. E. Uhlenbeck and L. S. Ornstein. On the Theory of the Brownian Motion. *Physical Review*, 36(5):823–841, September 1930. ISSN 0031-899X. doi: 10.1103/PhysRev.36.823. URL https://link.aps.org/doi/10.1103/PhysRev.36.823.

[31] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods, October 2018. URL http://arxiv.org/abs/1802.09477. arXiv:1802.09477 [cs].

[32] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust Region Policy Optimization, April 2017. URL http://arxiv.org/abs/1502.05477. arXiv:1502.05477 [cs].

[33] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. Springer series in operations research. Springer, New York, 2nd ed edition, 2006. ISBN 978-0-387-30303-1. OCLC: ocm68629100.

[34] Sham Kakade and John Langford. Approximately Optimal Approximate Reinforcement Learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, ICML '02, pages 267–274, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc. ISBN 1-55860-873-7.

[35] R Pascanu. Revisiting natural gradient for deep networks. *arXiv preprint arXiv:1301.3584*, 2013.

[36] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, August 2017. URL http://arxiv.org/abs/1707.06347. arXiv:1707.06347 [cs].

[37] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017. URL http://arxiv.org/abs/1412.6980. arXiv:1412.6980 [cs].

[38] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman,

Jie Tang, and Wojciech Zaremba. OpenAI Gym, June 2016. URL http://arxiv.org/abs/1606.01540. arXiv:1606.01540 [cs].

[39] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

[40] Antonio Aslan. EV2Gym: A Gym Environment for Electric Vehicles, 2023. URL https://github.com/AntonioJojoto/EV2Gym.

List of Figures

List of Tables