Control of a hybrid marine propulsion plant with Reinforcement Learning

Georgios Tsiknias

Diploma Thesis

Postgraduate studies: Automation Systems
National Technical University of Athens

Supervisor: Assistant Prof. George Papalambrou

Committee Member : Assistant Prof. K. Tzafestas
Committee Member : Prof. K. Kyriakopoulos

June 2021

Acknowledgments

This work has been carried out at the Laboratory of Marine Engineering (LME) at the School of Naval Architecture and Marine Engineering of the National Technical University of Athens, under the supervision of Assistant Professor George Papalambrou.

I would first like to thank my thesis supervisor Assistant Professor George Papalambrou for giving me the chance and motivation to work on this topic. I would like to thank him for his trust and patience, continuous support and immense knowledge. His guidance helped me in all the time working on this thesis.

I would like thank PhD candidate Vasileio Karystinos for providing the opportunity to work with the full-scale hybrid diesel-electric marine propulsion powertrain of LME. His assistance and work is considered vital in the path to correct implementation of the theory on such a promising topic of Reinforcement Learning, coming for his great experience on designed control systems.

I am also sincerely grateful to my family, for the unceasing encouragement, support, motivation and attention throughout my studies. This work is dedicated to them as a thank you for their contribution.

I also place on record, my sense of gratitude to one and all, who directly or indirectly, have lent their hand in this venture. This accomplishment would not have been possible without them. Thank you.

Abstract

In this thesis, the implementation of reinforcement learning applications in a hybrid diesel-electric marine power plant is investigated. Initially, modeling procedure of the components of the power plant is been presented. For each component (engine, electric motor/generator, battery), a models was introduced from previous studies [1]. The aim is to find RL methods and set-ups which are accurate and computationally efficient, so that the agent would be able to solve the optimization problem in real time. Moreover, different environment formulations where also reviewed in order to set up a reliable simulation for the RL controller-agent.

Reinforcement Learning control is a sophisticated machine learning control method which can handle nonlinear multi-variable problems with constraints by solving the optimization problem of minimizing an objective function over a finite horizon. The developed algorithms were evaluated regarding the performance with simulations in a virtual hybrid diesel-electric set-up in Julia Pluto environment and Matlab RL Designer.

Finally, the performance of the developed trained agent-controllers was simulated and verified on working cycles with the modeled hybrid propulsion plant HIPPO-2. The simulations were conducted for various load profiles and state transitions, and the agents-controllers were evaluated regarding the their ability to track the contextually reference and satisfy the predefined constraints.

List of Figures

1.1 Wärtsilä HY-2 Diesel-Mechanical Configuration. 15

2.1 Elements of basic RL problem. 19
2.2 Concept of Transition function-Agent in a certain and uncertain environment. 20
2.3 Concept of Q-state. 21
2.4 Probability distribution for Continuous Action. 26
2.5 Elements of an RL Environment. 28

3.1 The HIPPO-2 hybrid diesel-electric testbed of LME. 30
3.2 HIPPO-2 speed and torque outputs. 31
3.3 Loading diagram of CAT C9.3. 32
3.4 Relation between air fuel ratio and emissions. 34

4.1 Q and DQN methods. 40
4.2 Target and Predicton NN. 42
4.3 Approximator NN. 44
4.4 Episode achievements cost and reward function. 47
4.5 Parallel control SOC-SOCtrack 10 last episodes. 48
4.6 Parallel control SE-SEtrack last episode. 49
4.7 Parallel control u_Tice-u_Tel last episode. 50
4.8 Episode achievements cost and reward function in Double action. 54
4.9 Double action control SE-SOC 10 last episodes. 55
4.10 Double action control SE-SEtrack last episode. 56
4.11 Double action control SOC-SOCtrack last episode. 56
4.12 Double action control u_Tice-u_Tel last episode. 57

5.1 Basic policy gradient state to action. 60
5.2 The Actor–Critic PPO algorithm process. 61
5.3 Actor NN. 62
5.4 Critic NN. 62
5.5 Reward function in Double action PPO. 64
5.6 Episode achievements cost function in Double action. 65
5.7 Double action control SE-SOC 10 last episodes PPO. 66
5.8 Double action control SE-SEtrack last episode PPO. 67
5.9 Double action control SOC-SOCtrack last episode PPO. 67
5.10 Double action control u_Tice-u_Tel last episode PPO. 68
5.11 Speed Utice and NOx emission production. 70
5.12 Possible Utice and Utel couples on every Speed value. 70
5.13 Reward function in Double action PPO NOx control. 71
5.14 Episode achievements cost function in Double action NOx control. 72
5.15 Double action control SE-SOC-NOx 10 last episodes PPO. 73

5.16 Double action NOx control SE-SEtrack last episode PPO. 74
5.17 Double action NOx control SOC-SOCtrack last episode PPO. 74
5.18 Double action control NOx-NOxtrack last episode PPO. 75
5.19 Double action NOx control u_Tice-u_Tel last episode PPO. 76

6.1 Relation between Tload and SE in different constant. 78
6.2 Episode achievements cost function and reward in Double action DQN. . . 80
6.3 Double action control simulation SE-SOC 5 last episodes DQN 81
6.4 Double action control simulation SE-SEtrack last episode DQN. 82
6.5 Double action control simulation SOC-SOCtrack last episode DQN. 82
6.6 Double action control simulation u_Tice-u_Tel last episode DQN. 83
6.7 Rewards in Double action PPO Simulation for 5 episodes. 84
6.8 Episode achievements cost function in Double action PPO Simulation. . . . 85
6.9 Double action control simulation SE-SOC 5 last episodes PPO. 86
6.10 Double action control simulation SE-SEtrack last episode PPO. 87
6.11 Double action control simulation SOC-SOCtrack last episode PPO. 87
6.12 Double action control simulation u_Tice-u_Tel last episode PPO. 88
6.13 Rewards in Double action NOx control PPO Simulation for 5 episodes. . . . 89
6.14 Simulation episode achievements and cost function Double action NOx control. 90
6.15 Double action NOx control simulation SE-SOC 5 last episodes PPO. 91
6.16 Double action NOx control simulation SE-SEtrack last episode PPO. 92
6.17 Double action NOx control simulation SOC-SOCtrack last episode PPO. . . 92
6.18 Double action NOx control simulation u_Tice-u_Tel last episode PPO. . . . 93
6.19 Double action control simulation NOx total cycle reduction last episode PPO. 94

List of Tables

2.1 RL methods 23

3.1 States in case 1&2 36
3.2 Parameters in case 1&2 37

4.1 Action space parallel DQN 45
4.2 Action space DQN 51

Contents

List of Figures **7**

List of Tables **9**

1 Introduction **13**
1.1 Problem Formulation . . . 13
1.2 Previous work . . . 15

2 Continuous dynamic systems and RL methods **17**
2.1 Reinforcement learning review . . . 18
2.1.1 Markov Decision Process Structure . . . 18
2.1.1.1 State . . . 19
2.1.1.2 Action . . . 19
2.1.1.3 Transition . . . 19
2.1.1.4 Reward Function . . . 20
2.1.1.5 Policy . . . 20
2.1.1.6 Optimality Criteria and Discounting . . . 21
2.1.1.7 Concept of Q-State . . . 21
2.1.1.8 Value Functions and Bellman Equation . . . 21
2.1.1.9 Online and Offline MDP . . . 22
2.1.1.10 Model-Base and Model-Free Learning . . . 22
2.2 Reinforcement Learning algorithms . . . 23
2.2.1 Q-Learning Algorithm and Basic DQN . . . 23
2.2.2 The Actor-Critic and PPO algorithm . . . 24
2.2.3 Discrete and continuous Action Space . . . 25
2.3 Dynamic systems and RL set-up . . . 27
2.3.1 RL Environment . . . 27
2.3.2 Observation, Action and Reward signals . . . 27

3 Propulsion Plant Description and Modeling **29**
3.1 HIPPO-2 Experimental Test-Bed . . . 30
3.1.1 HIPPO-2 Integration . . . 30
3.2 System components and Modeling . . . 32
3.2.1 Diesel engine . . . 32
3.2.2 Electric Motor/Generator . . . 34
3.2.3 The Battery . . . 35
3.2.4 HIPPO-2 Modeling of dynamics . . . 35
3.2.5 Formulation and Restrictions . . . 35

4 Basic DQN algorithm and Application **39**
4.1 Basic DQN principles . . . 40
4.2 The Julia environment . . . 43
4.3 Parallel PD Speed and State of Charge Tracking Control . . . 44
4.3.1 Training Parallel Control . . . 46
4.4 Double action Speed and State of Charge Tracking Control Case 1 . . . 50
4.4.1 Training Case 1 . . . 52

5 PPO algorithm and Application **59**
5.1 PPO principles . . . 60
5.2 The Matlab RL Designer environment . . . 61
5.3 Double action Speed and State of Charge Tracking Control Case 1 . . . 63
5.3.1 Training Case 1 . . . 64
5.4 Double action Speed and NOx Control Case 2 . . . 69
5.4.1 Training Case 2 . . . 71

6 Simulations and Results **77**
6.1 Basic DQN and Julia set-up and PPO Matlab Simulation Case 1 . . . 78
6.1.1 Basic DQN Simulation Case 1 . . . 78
6.1.2 PPO Simulation Case 1 . . . 83
6.2 PPO Matlab Simulation Case 2 . . . 89

7 Conclusions and Future Work **95**
7.1 Conclusions . . . 95
7.2 Potential for Future Work . . . 96

Bibliography **97**

Chapter 1

Introduction

1.1 Problem Formulation

Over the past two decades, the demand for generality and advancements in control theory over non-linear systems and dynamic systems of great uncertainty, stimulated the interest on machine learning approaches for this purpose. Practices of learning theory have been developed from the 20th century although they have failed over the years to contribute great due to the lack of technological and computational support that they demand. Reinforcement learning (RL) or self learning technology has its origin from subjects like statistics, control theory and psychology. It has a very long history, until the late 80s and early 90s that reinforcement learning technology obtains the wide research and application in some fields such as artificial intelligence(AI), machine learning, automatic control [2]. RL is one of the techniques in artificial intelligence (AI) which is usually considered as a goal-directed method for solving problems in uncertain and dynamic environments. RL refers to a category of unsupervised machine learning algorithms that seek to maximize a reward signal. Supervised learning methods utilizes examples of correct action, but RL methods learns by trying many actions and learns which of those actions produce the most reward. Reinforcement learning is an important machine learning method, as an online learning technology, which combined with the advancements of Neural Network techniques, consists the widely known Deep Reinforcement Learning [3]. For ease, every time RL is mentioned will be considered as it refers to Deep RL.

Now, within the wider international debate on climate change, industries are on a great pressure to reduce their environmental impact. In this context, there are requirements for marine industry to reduce emissions of greenhouse gases, most notably carbon dioxide although other exhaust gases and the components, such as the nitrogen oxides (NOx), are also included. Furthermore, a regulatory framework is implemented by the IMO, MARPOL Annex VI, in order to enforce the above reductions, by setting operational and design limitations. For instance, engines manufactured after 2011 and with output power over 130 kW required to limit their specific NOx emissions. In environmentally "sensitive" sea regions, even lower limits apply. Moreover, the vessel design efficiency is also examined, via the EEDI index. This efficiency depends on the total CO_2 related emissions, the ship would emit in order to complete the required transportation work. Indicatively, by 2030 the ships which will be constructed are required to have 30 % reduced EEDI than 2013 [4].

According to [5] several promising technologies have been proposed. Some of them aim to decrease the power demand, (i.e. by optimizing the efficiency of the hull and propeller) and others to increase the efficiency of power plant itself. Regarding the second manner, a lot of new recent technologies promise to reduce both emissions and fuel consumption. A number of these efforts, are related with optimizing the existing diesel engines perfor-

mance regarding the emissions, directly (e.g. the EGR), or indirectly (e.g SCR). Moreover, alternative fuels (e.g. LNG and bio-fuels) and renewable sources of power have been also proposed. Furthermore, advances in battery technologies regarding their capacitance and efficiency, have already made possible the first fully battery depended ships, employing both high energy efficiency and zero emissions. However, battery cost and limited capacitance still pose barriers which have to be overcome. An interesting solution which aims to combine the proven availability and operational efficiency of conventional propulsion manner, and the benefits of novel technologies is Hybrid Propulsion and Energy Conversion. Hybrid propulsion is an option where one or more modes of powering the ship can be utilized to optimize performance for economic, environmental or operational reasons. A common hybrid configuration is that the different powering modes feed a common electrical bus bar from which power can be drawn for various purposes. This, however, is not necessarily the case since many examples of mechanical linkages between independent power sources have been designed and operated in ships, both past and present [5]. The key factor in order to achieve respectable higher efficiency is the control strategy. For instance, studies have shown that a 10-35% fuel and emission reduction is possible in battery deployment and intelligent use of DC configurations by implemented appropriate control strategies [4].

Keeping in mind the needs and the thresholds of each technology RL could prove itself a key tool, first in obtaining a good performance on speed and tracking control of an engine-motor, and second of optimizing the complex solution of a cost function capable of representing the emission-efficiency problem of such nonlinear systems. In this work, *hybrid diesel-electric ship propulsion* is examined form the point of the implemented tracking control strategy, with respect to effectively controlling a double action plat system and further applying that for minimizing the produced emissions during cycles with transient loads. In particular, control via Reinforcement Learning methods configuration for the closed-loop control of a parallel Hybrid diesel-electric powertrain is investigated. The main purpose is to develop and examine an efficient control scheme which (1) applies desired tracking control over the speed and the state of charge of a battery in a Hybrid system and (2) reduces fuel consumption and NOx emissions for the Hybrid powertrain HIPPO-2 of Laboratory of Marine Engineering (LME). Two control schemes were followed. The first employs indirect control of the engine by regulating the electric torque, in order to follow specified trajectories over the State of Charge of the battery of the system. In the second scheme, both engine and electric motor are controlled directly from the RL agent so as the rotational shaft Speed, State of charge and additionally NOx emission production to follow a predefined reference, in respect to certain constraints. In the last case, the aim is to reduce the engine dynamics in order to reduce the NOx propagation, over a cycle of state transitions in the engine operation.

In accordance to the previous, the potential of RL applications is examined, with the use of algorithms in the discrete and continuous action space, with several representations of the model of the system and several trials on engine and motor control. The implementation of these has been developed in Julia Pluto ReinforcementLearning.jl package alongside the Matlab RL-Designer environment, both of which have been examined with their own limitations through the process. Extended description follows to the rest chapters.

1.2 Previous work

Over the recent years, there is an increasing number of hybrid marine applications. These systems have extra degrees of freedom leading to increased complexity of the propulsion system. However, the control implementations for these systems are based mostly in traditional control strategies, e.g. fixed combinators curves, fixed frequency generators, rule based control of batteries etc. [4]. Research have shown, though, that conservative control strategies which apply to advance architectures, will probably lead to a insignificant fuel and emission reduction. However, control strategies progress regarding marine applications, is not so advanced. Although, strategies with significant increase efficiency have been proposed, most of them lack of impact analysis.

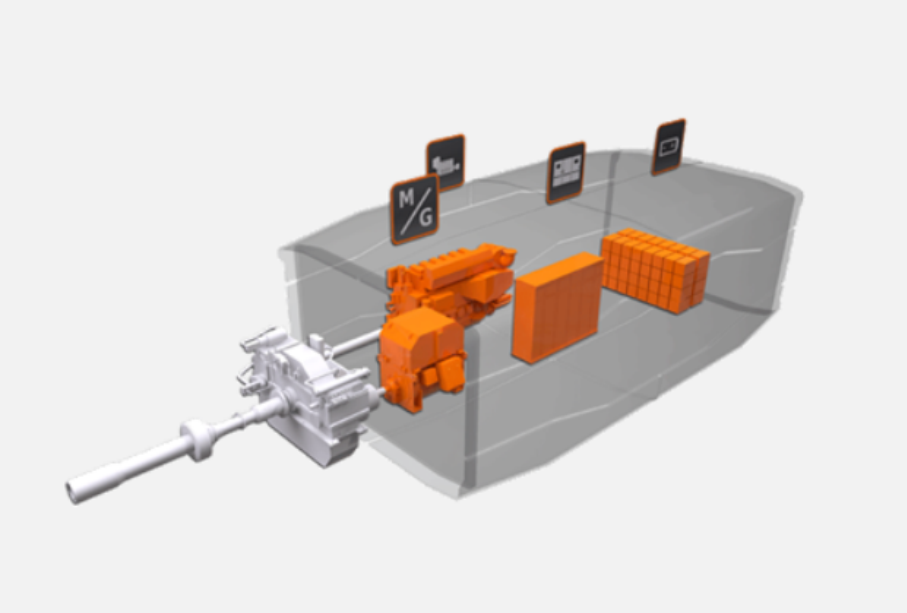

Figure 1.1: Wärtsilä HY-2 Diesel-Mechanical Configuration.

Hybrid propulsion, can be reviewed in accordance with the implemented configuration. There are applications which the main propulsive system is conventional one, such as a diesel engine, the electric motor is connected to the main shaft line. The main idea is that diesel engine is supposed to provide the propulsive power in higher speeds and loads, in which the efficiency is greater. Additionally the electric Motor would provide the propulsive power in lower speeds, in which the the diesel efficiency is significantly lower. In [6], an interesting application is suggested for a naval frigate application in which the the electrical part serves as shaft generator in partial loads, increasing in this way the efficiency by covering the electric supply demand directly from the engine with a higher efficiency. The implemented control strategy was speed and voltage droop control, combining field oriented control for the electric machines. An another interesting application which refers to parallel operation of the electric motor and diesel engine is presented in [7], which the motor assists the engine so as to maintain a specific air-fuel ratio λ reference. In that way, during transient operations, the thermal loading of the engine is decreased, and consequently the NOx emissions drops. In [8], the above is implemented via Model Predictive Control.

At the same time some RL studies have been proposing a control strategy for hybrid electrical vehicle, consisting of a decision-making agent, trained on different test drives with Reinforcement Learning. For these, the Proximal Policy Optimization method was applied. The strategy controls the torque-split between the combustion engine and electric motor, the power of an electrically heated catalyst and internal engine measures. The strategy achieved a fuel reduction of 3.1 % averaged over the test data set [9].

Thesis structure

The structure of the thesis is as follows: in chapter 2 a short review of the RL theory and the experimental facility is presented. In chapter 3, a model of the HIPPO 2 testbed plant, that was used on the development of the study, is presented. In chapter 4, a brief theory review of DQN in Julia alongside set-up techniques are illustrated, and training strategies and results from the applications are provided. In chapter 5, control design and training results with the use of PPO in Matlab are presented. Results from simulations testings of the agent-controllers on HIPPO 2 testbed plant are shown in chapter 6. Finally, the conclusions of this work are presented in chapter 7.

Chapter 2

Continuous dynamic systems and RL methods

In this chapter, literature applications on continuous dynamic systems is presented. Furthermore, a fundamental review on the Reinforcement learning background is developed in order to help the reader with a better insight on the field. Reinforcement learning has been tested in continuous dynamic systems, promising to give solutions to nonlinear discontinuities and uncertainties, from applications for tutorial purposes like the inverted pendulum or crane stabilization to real world robotic applications, self driving cars and energy consumption management systems. Some of the autonomous driving tasks where reinforcement learning could be applied include trajectory optimization, motion planning, dynamic pathing, controller optimization, and scenario-based learning policies for multi-part complex tasks.

2.1 Reinforcement learning review

Reinforcement Learning (RL) is the training of machine learning models to make a sequence of decisions. The agent learns to achieve a goal in an uncertain, potentially complex environment. In reinforcement learning, artificial intelligence faces a game-like situation. The computer employs trial and error to come up with a solution to the problem [10]. To get the machine to do what the programmer wants, artificial intelligence gets either rewards or penalties for the actions it performs. Its goal is to maximize the total score. Although the designer sets the reward policy–that is, the rules of the game–he gives the model no hints or suggestions for how to solve the game. It's up to the model to figure out how to perform the task to maximize the reward, starting from totally random trials and finishing with sophisticated tactics and superhuman skills. By leveraging the power of search and many trials, reinforcement learning is currently the most effective way to hint the machine's creativity. In contrast to human beings, artificial intelligence can gather experience from thousands of parallel gameplays if a reinforcement learning algorithm is run on sufficiently robust computer infrastructure.

The main concentration is the uncertain searching for real dynamical systems, for instance, it is expecting that a robot learning to find its path without crashing with other objects in an environment or an autonomous car learning to set its speed in various situations. In certain searching, it is considered that the environment does not contain any noise and hence, the result of each action will be clear. However, in real environment the result of each action could be a probabilistic affair. The result of each action can correspond to a probabilistic function which is called transition function or dynamic of the system. If the dynamic of the system be a certain function, in other words every state is a discreet mathematically known information, the problem can be dealt as a model-based, and if the dynamic of the system is an unknown-uncertain function, the problem will be a model-free RL. This discrimination is the first branch of decisions over the type of agent to be used for a specific problem.

The formulation of the problem in RL performs with Markov Decision Process (MDP). In other words, the agent, which can be any from an activator-motor to a robot or an autonomous vehicle, can perceive the problem of uncertain searching using MDPs. Basic reinforcement is modeled as a Markov decision process (MDP):

- a set of environment and agent states, S;
- a set of actions, A, of the agent;
- $P_a(s, s') = Pr(s_{t+1} = s' | s_t = s, a_{t+1} = a)$ is the probability of transition (at time t) from state s to state s' under action a.
- $R_a(s, s')$ is the immediate reward after transition from s to s' with action a.

The probability that the process moves into its new state s' is influenced by the chosen action. Specifically, it is given by the state transition function $P_a(s, s')$. Thus, the next state s' depends on the current state s and the decision maker's action a. But given s and a, it is conditionally independent of all previous states and actions; in other words, the state transitions of an MDP satisfy the Markov property.

Basic elemets of an RL based formulation over a problem is presented below.2.1

2.1.1 Markov Decision Process Structure

As stated before, the elements of the RL can be formulized using the Markov Decision Process (MDP) framework. This section describes components such as states, actions,

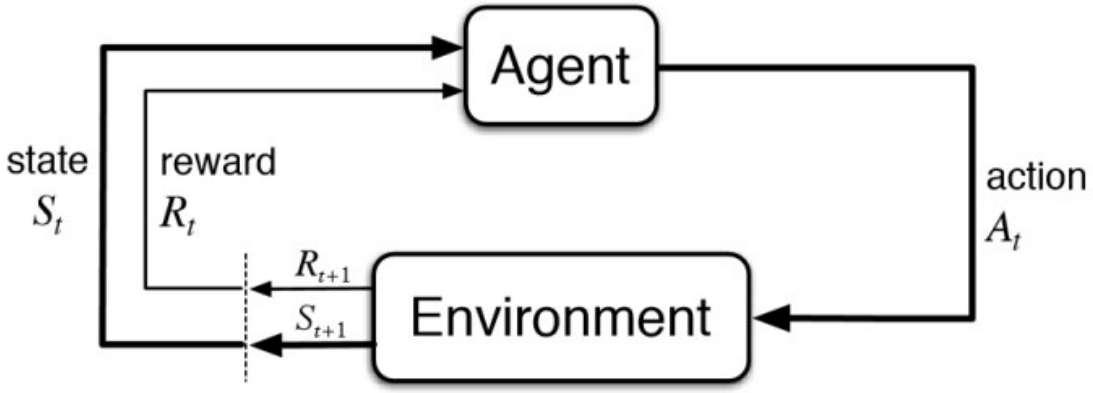

Figure 2.1: Elements of basic RL problem.

rewards and policies, as well as the goals of learning using different kinds of optimality criteria. MDPs are extensively described in [11]. MDPs consist of states, actions, transitions between states and a reward function definition.

2.1.1.1 State

The set of environmental states S is defined as the finite set $s^1, \ldots, s^N$ where the size of the state space is N, i.e.—s— = N. A state is a unique characterization of all that is important in a state of the problem that is modeled.

2.1.1.2 Action

The set of actions A is defined as the finite set $\alpha_1, \ldots, \alpha_K$ where the size of the action space is K, i.e. —A— = K. Actions can be used to control the system state. The set of actions that can be applied in some particular state, s ∈ S, is denoted A(s), where A(s) ⊂ A. In some systems, not all actions can be applied in every state, but in general we will assume that A(s) = A for all s ∈ S

2.1.1.3 Transition

By applying action a ∈ A in a state, s ∈ S, the system makes a transition from s to a new state s' ∈ S, based on a probability distribution over the set of possible transitions. The transition function T is defined as T: S×A×S → [0,1], i.e. the probability of ending up in state s' after doing action, a in state s is denoted T(s, α, s'). It is required that for all actions a, and all states s and s', T(s, α, s') ≥ 0 and T(s, α, s') ≤ 1. Furthermore, for all states s and actions a,Σ T(s, α, s') s'∈ S = 1 , T defines a proper probability distribution over possible next states. Instead of a precondition function, it is also possible to set T(s, α, s') = 0 for all states s' ∈ S. if a is not applicable in s. For talking about the order in which actions occur, we will define a discrete global clock, t = 1, 2, ... Using this, the notation s_t denotes the state at time t and s_{t+1} denotes the state at time t + 1. This enables to compare different states (and actions) occurring ordered in time during interaction. In some references, the transition functions are called dynamic of the system.2.2

The system being controlled is Markovian if the result of an action does not depend on the previous actions and visited states (history), but only depends on the current state, i.e. $P(s_{t+1}|s_t, \alpha_t, s_{t-1}, \alpha_{t-1}, \ldots) = P(s_{t+1}|s_t, \alpha_t) = T(s_t, \alpha_t, s_{t+1})$

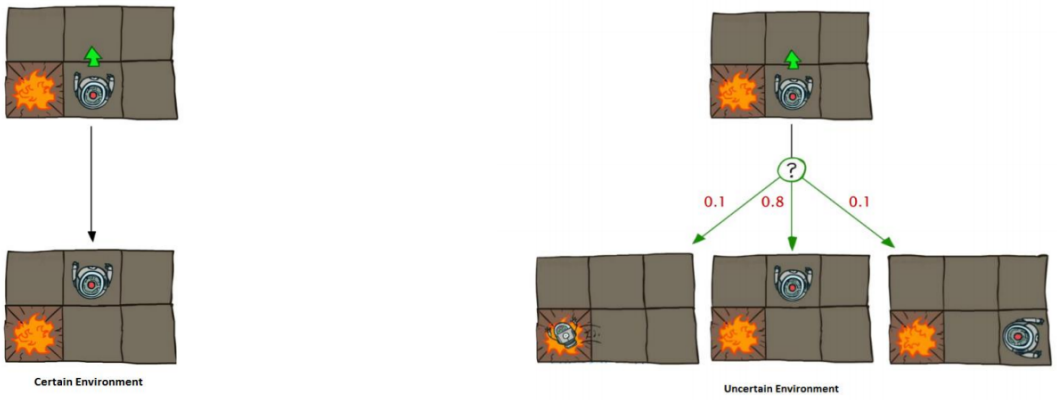

Figure 2.2: Concept of Transition function-Agent in a certain and uncertain environment.

The idea of Markovian dynamics is that the current state, s gives enough information to make an optimal decision; it is not important which states and actions preceded s. Another way of saying this, is that if you select an action a, the probability distribution over next states is the same as the last time you tried this action in the same state.

2.1.1.4 Reward Function

The reward function specifies rewards for being in a state, or doing some action in a state. The state reward function is defined as R: S $\rightarrow$ R, and it specifies the reward obtained in states. However, two other definitions exist. One can define either R: S $\times$ A $\rightarrow$ R or R: S $\times$ A $\times$ S $\rightarrow$ R.The first one gives rewards for performing an action in a state, and the second gives rewards for particular transitions between states. Throughout this study we will mainly use R(s, a, s'). The reward function is an important part of the MDP that specifies implicitly the goal of learning. For example, in episodic tasks such as in the games Tic-Tac-Toe and chess, one can assign all states in which the agent has won a positive reward value, all states in which the agent loses a negative reward value and a zero reward value in all states where the final outcome of the game is a draw. The goal of the agent is to reach positive valued states, which means winning the game. Thus, the reward function is used to give direction in which way the system, i.e. the MDP, should be controlled. The transition function T and the reward function R together define the model of the MDP.

2.1.1.5 Policy

Given an MDP S, A, T, R, a policy is a computable function that outputs for each state, s $\in$ S an action a $\in$A (or a $\in$A(s)). Formally, a deterministic policy π is a function defined as π : S $\rightarrow$ A. It is also possible to define a stochastic policy as π : S $\times$ A $\rightarrow$ [0,1] such that for each state, s $\in$ S, it holds that π(s, α) $\geq$ 0 and $\Sigma\alpha\in$A π(s, α) = 1.

Application of a policy to an MDP is done in the following way. First, a start state s_0 from the initial state distribution I is generated. Then, the policy π suggest the action $\alpha_0 = \pi(s_0)$ and this action is performed. Based on the transition function T and reward function R, a transition is made to state s_1, with probability $T(s_0, \alpha_0, s_1)$ and a reward $r_0 = R(s_0, \alpha_0, s_1)$ is received. This process continues, producing $s_0, \alpha_0, r_0, s_1, \alpha_1, r_1, s_2, \alpha_2, \ldots$. If the task is episodic, the process ends in state s_goal and is restarted in a new state drawn from I. If the task is continuing, the sequence of states can be extended indefinitely. The policy is part of the agent and its aim is to control the environment modelled as an

MDP. A fixed policy induces a stationary transition distribution over the MDP which can be transformed into a Markov system ⟨S', T'⟩ where S' = S and T'(s, s') =T(s, α, s') whenever π(s) = α.

2.1.1.6 Optimality Criteria and Discounting

In certain environments, we are looking for a chain of actions, however, in uncertain environments we should find a policy. As it is mentioned in previous section, policy is a function from states to actions. A policy is optimal if it gives us maximum benefit, or in other words, optimal policy refers to the best action to be taken at each state, for maximum rewards over time. An optimal policy is defined as follows [12]: $\pi* : S \rightarrow A$

2.1.1.7 Concept of Q-State

If the agent is in the state s and choose the action a and perform the action a, it will go to state s'. Instead of that, we can create what's called a q-table or matrix that follows the shape of [state, action] and we initialize the values to zero. We can then update and store our q-values after an episode. This q-table becomes a reference table for our agent to select the best action based on the q-value. So, if the agent just choose the action a, it will go to an imaginary state which call Q-state and it is graphically shown as a green circle below. This concept could be beneficial for RL algorithms, especially in Temporal Difference (TD) learning and Q-Learning [13].2.3

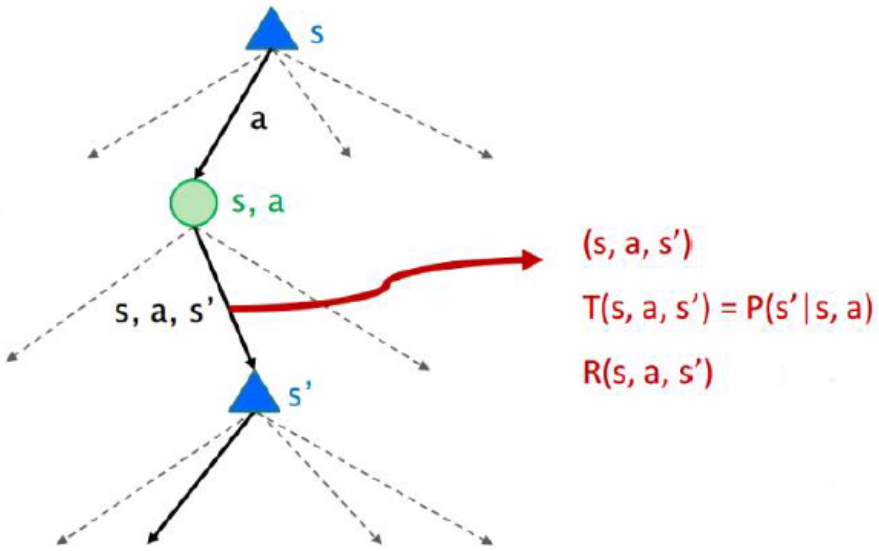

Figure 2.3: Concept of Q-state.

2.1.1.8 Value Functions and Bellman Equation

In the preceding sections we have used the terms MDPs and optimality criteria that can be useful for learning optimal policies. In this section, value functions will be defined, which are a way to link the optimality criteria to policies. Most learning algorithms for MDPs compute optimal policies by learning value functions. A value function represents

an estimate on how good it is for the agent to be in a certain state (or how good it is to perform a certain action in that state). The notion of how good is expressed in terms of an optimality criterion, i.e. in terms of the expected return. Value functions are defined for particular policies.

The value of a state, s under policy π, denoted V^{π}(s) is the expected return when starting in s and following π hereafter. We will use the infinite-horizon, discounted model in this section, such that this can be expressed as:

$$V^{\pi}(s) = E_{\pi}\sum_{k=0}^{\inf}(\gamma^{k} r_{t+k}|s_t = s) \tag{2.1.1}$$

A similar state-action value function Q: S $\times$ A $\rightarrow$ R can be defined as the expected return starting from state, s, taking action a and thereafter following policy

$$\pi : Q\pi(s,\alpha) = E_{\pi}\sum_{k=0}^{\inf}(\gamma^{k} r_{t+k}|s_t = s, \alpha_t = \alpha) \tag{2.1.2}$$

One fundamental property of value functions is that they satisfy certain recursive properties.For any policy π and any state, s the expression in Equation 2.1.1 can recursively be defined in terms of a so-called Bellman Equation Bellman (1957):

$$\begin{aligned} V^{\pi}(s) &= E_{\pi} r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...|s_t = s \\ &= E_{\pi} r_t + \gamma V^{\pi}(s_{t+1})|s_t = s \\ &= \Sigma T(s,\pi(s),s')(R(s,\alpha,s') + \gamma V^{\pi}(s')) \end{aligned} \tag{2.1.3}$$

It denotes that the expected value of state is defined in terms of the immediate reward and values of possible next states weighted by their transition probabilities, and additionally a discount factor. V^{π} is the unique solution for this set of equations. Note that multiple policies can have the same value function, but for a given policy π, V^{π} is unique. The goal for any given MDP is to find a best policy, i.e. the policy that receives the most reward. This means maximizing the value function of Equation 2.1.4, for all states s $\in$S. An optimal policy, denoted $\pi*$, is such that $V^{\pi} * (s) \geq V^{\pi}(s)$ for all s $\in$ S and all policies π. It can be proven that the optimal solution $V^{*} = V^{\pi}*$ satisfies the following Equation:

$$V^{*}(s) = \max_{\alpha \in A} \sum_{s' \in S} T(s,\alpha,s')(R(s,\alpha,s') + \gamma V^{*}(s'))s' \in S \tag{2.1.4}$$

This expression is called the Bellman optimality equation. It states that the value of a state under an optimal policy must be equal to the expected return for the best action in that state.

2.1.1.9 Online and Offline MDP

In a MDP problem if the transition function and the policy are definite, the problem will be an offline MDP. In an online MDP, the transition and policy functions are not definite and the agent should learn them by interacting in the environment. In fact, by performing various episodes the agent tries to learn these functions.

2.1.1.10 Model-Base and Model-Free Learning

In model-based learning, the transition function is an unknown function and the agent by gaining experiences tries to estimate the transition function. If T(s,α,s') is the transition

function of an specific environment, the estimated transition function will be shown as T'(s,α,s'). The reward function is estimating as well. The estimated reward function is showing as R'(s,α,s').

In model-free learning methods, instead of the transition function to be estimated, the value of each state is estimated. In this study, our concentration is on the model-free algorithms for solving MDP problems. The model-free algorithms are classifying in passive and active methods which refer to the estimation of the value of the states with a known or unknown policy. A table of the most common developed Rl methods is provided below. [14]. 2.1

RL methods					
Algorithm	Description	Policy	Action Space	State Space	Operator
Monte Carlo	Every visit to Monte Carlo	Either	Discrete	Discrete	Sample-means
Q-learning	State–action–reward –state	Off-policy	Discrete	Discrete	Q-value
SARSA	State–action–reward –state–action	On-policy	Discrete	Discrete	Q-value
DQN	Deep Q Network	Off-policy	Discrete	Continuous	Q-value
DDPG	Deep Deterministic Policy Gradient	Off-policy	Continuous	Continuous	Q-value
A3C	Asynchronous Advantage Actor-Critic Algorithm	On-policy	Continuous	Continuous	Advantage
PPO	Proximal Policy Optimization	On-policy	Continuous	Continuous	Advantage
TD3	Twin Delayed Deep Deterministic Policy Gradient	Off-policy	Continuous	Continuous	Q-value

Table 2.1: RL methods

2.2 Reinforcement Learning algorithms

For the purposes of this study, Basic DQN and PPO algorithms were used, further description of which will follow on the next sections.

2.2.1 Q-Learning Algorithm and Basic DQN

In this section, the concepts of Q-learning algorithm and Basic DQN for finding the optimal policy is presented. By using Q-learning, finding the suitable action will be performed without using the dynamic of the system. Q-learning is an active learning method in which the agent has the freedom of taking action is each state. Accordingly, the following assumptions are considering in Q-learning algorithm:

- The transition function T(s,α,s') is unknown.
- The reward function R(s,α,s') is unknown.
- The agent chooses each action autonomously.

Q-Learning is based on the notion of a Q-function. The Q-function (a.k.a the state-action value function) of a policy π,$Q(s, \alpha)$, measures the expected return or discounted sum of rewards obtained from state s by taking action α first and following policy π thereafter. We define the optimal Q-function $Q^*(s, \alpha)$ as the maximum return that can be obtained starting from observation s , taking action α and following the optimal policy thereafter. The optimal Q-function obeys the following Bellman optimality equation:

$$Q^*(s, \alpha) = Er + \gamma \max_{\alpha'} Q^*(s', \alpha') \tag{2.2.1}$$

This means that the maximum return from state s and action α is the sum of the immediate reward r and the return (discounted by γ) obtained by following the optimal policy thereafter until the end of the episode (i.e., the maximum reward from the next state s'). The expectation is computed both over the distribution of immediate rewards r and possible next states s'. The basic idea behind Q-Learning is to use the Bellman optimality equation as an iterative update [15]

$$Q_{i+1}(s, \alpha) < -Er + \gamma \max_{\alpha'} Q_i(s', \alpha') asi- > \inf Q- > Q* \tag{2.2.2}$$

Now, for most problems, it is impractical to represent the Q-function as a table containing values for each combination of s and α. Instead, we train a function approximator, such as a deep neural network with parameters θ, to estimate the Q-values, i.e.$Q(s, \alpha; \theta) \approx Q(s, \alpha)$. This can done by minimizing the following loss at each step i:

$$L_i(\theta_i) < -E_{s,a,r,s' \ \rho(.)}(y_i - Q(s, \alpha; \theta_i))^2 where y_i = r + \gamma \max_{\alpha'} Q(s', \alpha'; \theta_{i-1}) \tag{2.2.3}$$

Here, y_i is called the TD (temporal difference) target, and $y_i - Q$ is called the TD error. ρ represents the behaviour distribution, the distribution over transitions s, α, r, s' collected from the environment. Note that the parameters from the previous iteration θ_i are fixed and not updated. In practice we use a snapshot of the network parameters from a few iterations ago instead of the last iteration. This copy is called the target network.

Q-Learning is an off-policy algorithm that learns about the greedy policy $\alpha = \max_\alpha Q(s, \alpha; \theta)$ while using a different behaviour policy for acting in the environment/collecting data. This behaviour policy is usually an ϵ-greedy policy that selects the greedy action with probability 1-ϵ and a random action with probability ϵ to ensure good coverage of the state-action space.

To avoid computing the full expectation in the DQN loss, we can minimize it using stochastic gradient descent. If the loss is computed using just the last transition s, α, r, s', this reduces to standard Q-Learning.

2.2.2 The Actor-Critic and PPO algorithm

PPO stands for Proximal Policy Optimization. In order to understand the methodology of the algorithm we need to define a few things about the policy as a function of states to actions. The objective is to always get the highest rewards. To be able to do so we must define a function that collects these rewards and work to optimize it in order to maximize those rewards. Another equally important point, is that we can do this using a neural network, which, if so, transforms the problem into finding the set of weights θ of the neural network that helps us maximize the objective function. A direct approach to the optimization problem is the so called Policy Gradient, using gradient decent [16]. So starting from this fundamental principle we define J(θ) as the expected rewards R(s, a) that we get in every time step going from zero to the end of the episode T, following a policy $\pi(\theta)$. If we define the episode as a trajectory τ going from t=0 to T, then the

expected rewards is the sum of all possible trajectories of the probability that τ is selected according to θ, times the return of this trajectory R(τ). This leads us to the following equation:

$$J(\theta) = E[\sum_{t=0}^{T} R(s_t, \alpha_t); \pi_\theta] = \sum_t P(\tau; \theta) R(\tau) \tag{2.2.4}$$

in which we could apply gradient decent to search for the extremum (maximum or minimum) and conclude to equation:

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(\alpha_t | s_t) R(\tau^i) \tag{2.2.5}$$

where m is the number of episodes (here called trajectories) executed, π is a policy parametrized by θ, which means when θ varies the policy will be affected. Remember that the policy gives the probability of taking a certain action when the agent is at in a certain state. τ^i is the ith episode or trajectory executed. R(τ^i) is the return (total rewards) of the trajectory τ^i. T is the number of steps in the trajectory τ^i.

Now with a few necessary mathematical modifications on the R for algorithmic reasons, we can include the notion of actor and critic in the optimization of the objective function. Actor–critic algorithms learn both policies and value functions. The 'actor' is the component that learns policies, and the 'critic' is the component that learns about whatever policy is currently being followed by the actor in order to 'criticize' the actor's action choices. So, by modifying 2.2.5 we can get:

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(\alpha_t | s_t)(Q(s_t, \alpha_t) - V(s_t)) \tag{2.2.6}$$

This strangely look similar to Q(s, a) which is the value of action a taken at state s, and V(s) which is the value of the state, or the average of all rewards (caused by all actions taken) at state s. Here its more obvious to find the origin of those actor and critic approaches. These can also be represented by Deep Neural Networks.

Now the PPO algorithm uses that methodology but it clips the objective function to avoid areas that would lead to an excessively large policy update [16].

2.2.3 Discrete and continuous Action Space

The case of discrete or continuous action space is mostly restricted-defined form the architecture of each algorithm 2.1. In our study, as mentioned above, the DQN, capable of producing discrete action space, and PPO algorithm, capable of producing continuous action space, are chosen. In both cases the action is being produced by Neural Networks, the methodology of which is valuable in determining the set-up of a problem.

For the case of DQN, Deep Q-Learning replaces the regular Q-table with a neural network. Rather than mapping a state-action pair to a q-value, a neural network maps input states to (action, Q-value) pairs. One of the interesting things about Deep Q-Learning is that the learning process uses 2 neural networks. These networks have the same architecture but different weights. Every N steps, the weights from the main network are copied to the target network. Using both of these networks leads to more stability in the learning process and helps the algorithm to learn more effectively. Both the Main model and the Target model map input states to output actions. These output actions actually represent the model's predicted Q-value. In this case, the action that has the largest predicted Q-value is the best known action at that state.

For the case of PPO, Proximal Policy Optimization is using Actor-Critic DNNs. Now, there is the potential of generating discrete actions with a policy network (Actor) as the NN generates signals which pass through a softhmax function to give probabilities between 0 and 1 which all add up to 1. This is then used as a probability distribution to pick a random action guaranteeing exploration and the more certain the NN becomes the more we exploit and the less we explore. In our case, continuous action tasks rather than discrete probabilities take floating point inputs in a certain range, where (-1,1) is recommended.2.4 In this case, in order to facilitate exploration we are sampling values

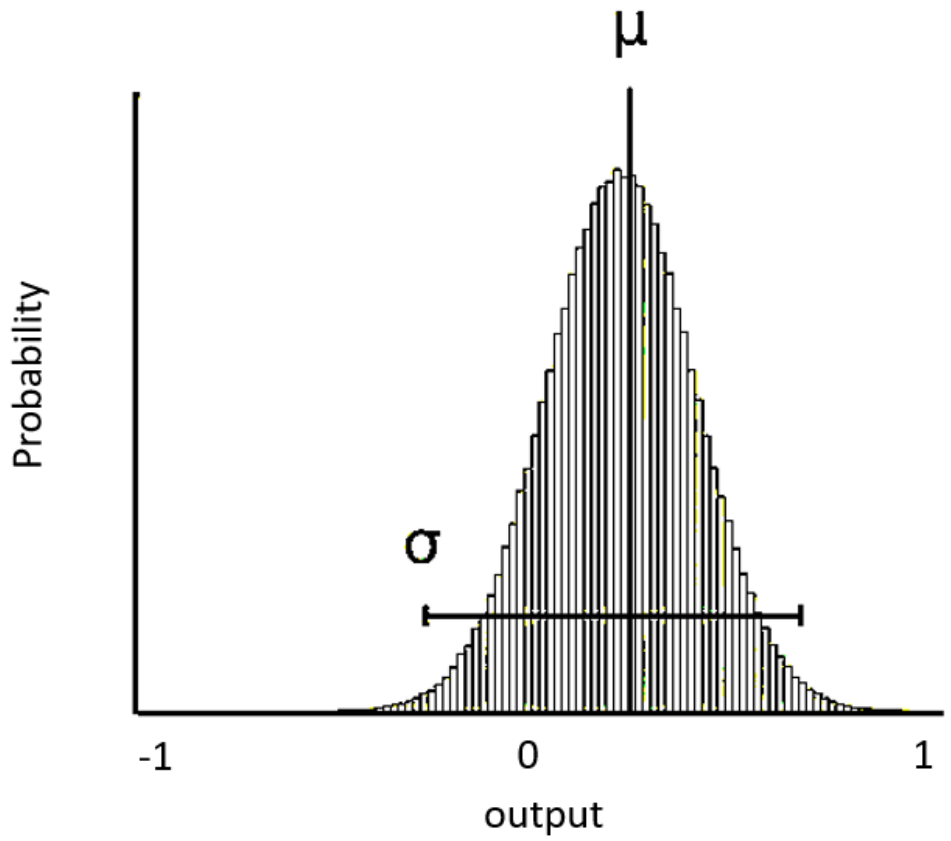

Figure 2.4: Probability distribution for Continuous Action.

from a normal probability distribution. Our policy network will have two outputs instead of one, μ and σ. μ is simply the mean of the probability distribution, the values that are sampled are going to be centered around the mean and σ corresponds to standard deviation (how far from the mean, the values could be). As the NN gets more certain about the output of the values of the actions σ gets smaller and smaller which means exploitation dominates over exploration.

In the next section, I will present useful set-up tactics of a dynamic problem for Deep RL, as issues of NN signal saturation, inadequate observation and reward function set-up can cause great difficulty and confusion for the developer and result in de-convergence over any kind of solution.

2.3 Dynamic systems and RL set-up

In this section some set-up guides are presented. Most of the work that is needed from the developer is on the proper set-up of an experiment, as it can usually result in unwanted situations of saturation and de-convergence, due to badly transferred signals or inadequate information on the typical (s,a,r,s') loop. Dynamic systems consists of differential equations which are called the model of the system in which a control input is constantly present, to maintain or track a given reference, usually in the most optimal way. The optimality could be usually addressed in the construction of a cost function representing the states of the system to be controlled and optimized. This cost function minima would then result in to an optimized solution. As discussed in the first chapters 2.1, RL architecture consists of an environment, an observation (object), an agent, an action (object), a reward (float) and the done signal(boolean). Each one part of the architecture, and its proper handling is analyzed above.

2.3.1 RL Environment

In the context of reinforcement learning, an environment can be seen as an interactive problem, that needs to be solved in the best way possible. To quantify the success, a reward function is defined within the environment. The agent can see the so called observations that give information about the current state of the environment. It can then take a specific action, which will return the observation and the scalar reward of the next environment's state. The agent's goal is to maximize the sum of rewards achieved in a limited number of steps or episodes. From a technical standpoint, there are many different ways build an environment. This consists of the differential equations that describe the system, which are solved in a traditional manner of computational iterations. Every step corresponds to a time division, and each episode into a time sample of solution. For every step, the reward function is calculated which should correspond to a goal, and usually shall be connected with a minimization of a cost function suitable to represent that goal. The accumulative rewards of every step consists of the efficiency of the process and should converge to a maximum value with time-steps and episodes given. In order to reduce unwanted computations, an "is done" termination signal shall be generated when a state or a value is out of some boundaries. All these elements are user defined inside the environment agenta.2.5

2.3.2 Observation, Action and Reward signals

The observation is the feedback given from the environment back to the agent or the neural network. It is really the only thing, the agent can see to derive it's next action. More importantly, the agent does not have a memory. It's decision will solely be based on the observation of the current state. Defining suitable observation is essential to achieve good training results. Uniqueness of Observations is in mathematical terms, to approach the model as a deterministic function f that calculates the actions [a] based on the observations [o]. Its useful to think what a human observer or math calculator would need in order to accomplice the goal of the training. For example, if two different situations or states require two different actions for success, then their respective observation have to be different too. Only when observations differ, the agent can produce two different actions.

Now, before we evaluate the different sets of observations, actions and rewards, is useful to keep in mind, that the neural networks does not know the meaning or context of the defined observations. However, it doesn't have to. The goal of machine learning in general is to find numeric correlation between observations and successful actions. For this, the

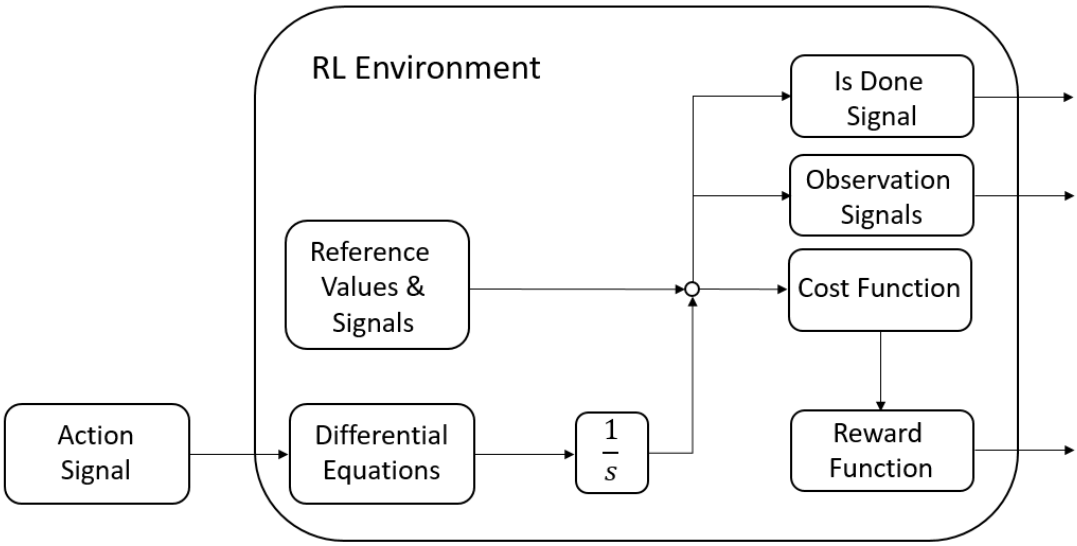

Figure 2.5: Elements of an RL Environment.

context of the data is irrelevant. Thus, Normalization of signals is of great importance as it greatly helps to make sure that the value range of each observation is [-1,1] or [0,1]. This procedure is not mandatory, however, most neural networks will benefit from normalized values. This is, because most neural networks have inverse tangent functions at the end of their calculation, and face difficulty in handling great ranges of signal values, causing saturation. In that case, the normalized value range is more suitable numerically. One simple way of applying linear normalization in a range [a,b] over a know range [xmin,xmax] is given below:2.3.1

$$x' = (b - a)\frac{x - minx}{maxx - minx} + a \tag{2.3.1}$$

The calculation over any signal including the reward, shall be done with the normalized data.

In the next chapter, I will introduce the HIPPO-2 Experimental Test-Bed, which consists the main experiment over of which the study was developed.

Chapter 3

Propulsion Plant Description and Modeling

In this chapter, the experimental hybrid powertrain facility *HIPPO-2* where the RL agents were trained, applied and tested, is presented. As mentioned the reinforcement learning methodology, requires to model the above components so as to train the agent-controller, simulate a work cycle and solve the optimization control problem. Here the model schemes for each component are presented.

3.1 HIPPO-2 Experimental Test-Bed

The facility is composed of three major components, *Internal Combustion Engine (ICE)*, *Electric Motor/Generator (EM)* and *Electric Brake (EB)*, which applies the load torque to the system. In addition, *a virtual Battery (B)* is also modeled, which was used in the training and simulation processes. The HIPPO-2 hybrid diesel-electric power plant consists of a internal combustion engine (ICE) in serial connection to an electric motor (EM). In this configuration, the rotational speed of the ICE and the EM are identical and the supplied torques add together to maintain the total torque demand applied by a electric motor brake (EB). In Fig. 3.1 and 3.2 the experimental hybrid powertrain of LME is presented, along with a schematic representation of the speed and torque outputs.

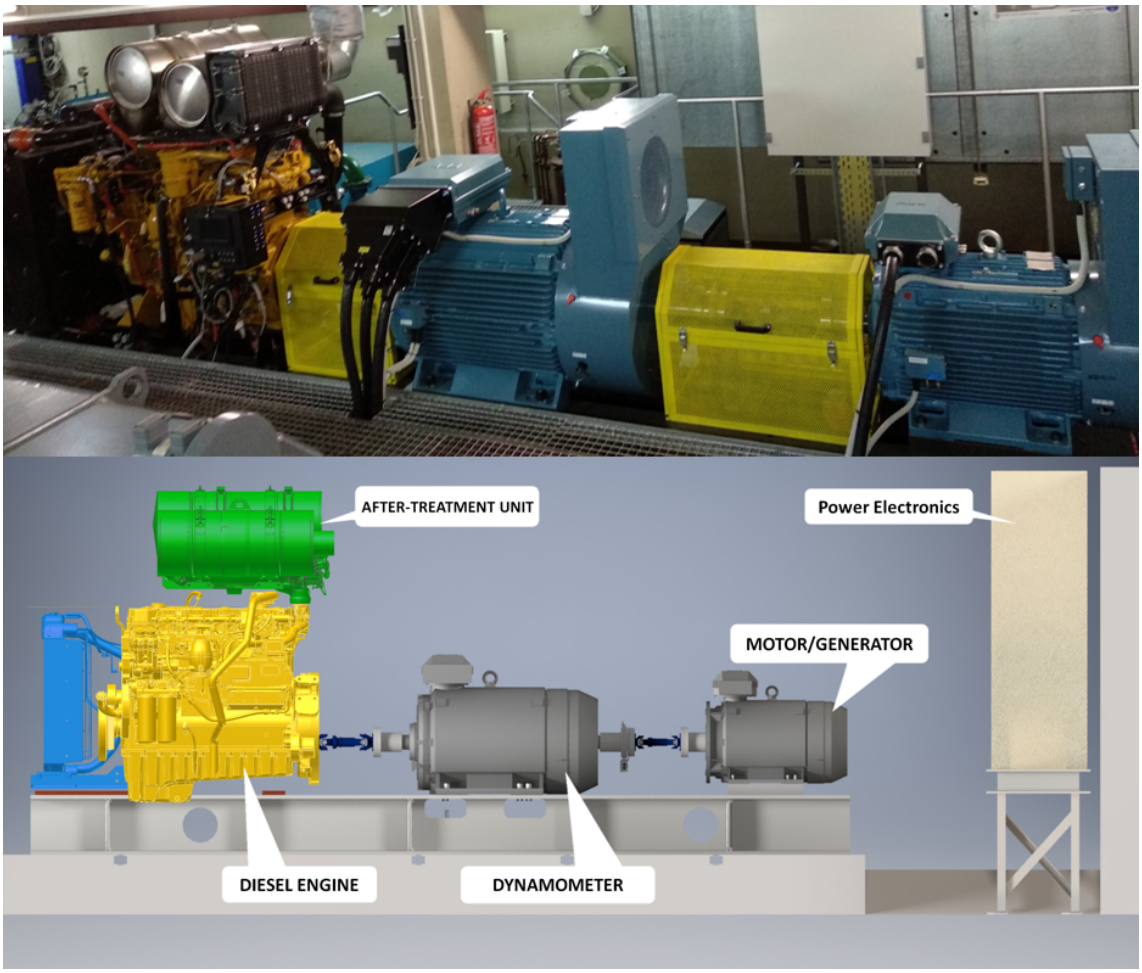

Figure 3.1: The HIPPO-2 hybrid diesel-electric testbed of LME.

3.1.1 HIPPO-2 Integration

The ICE is a turbocharged CATERPILLAR R 6-cylinder 9.3-liter 4-stroke industrial diesel engine, model C9.3 ACERTTM, producing 261 kW at 2200 rpm and maximum torque 1596 Nm at 1400 rpm. The loading diagram of engine is shown in **??** (Rating C). According to the speed reference and the deviation of speed measurement, the electronic control unit (ECU) of the ICE controls the fuel injection in the cylinders in closed loop control, using controller in the form of look-up tables. The engine is designed to meet U.S. EPA Tier 4

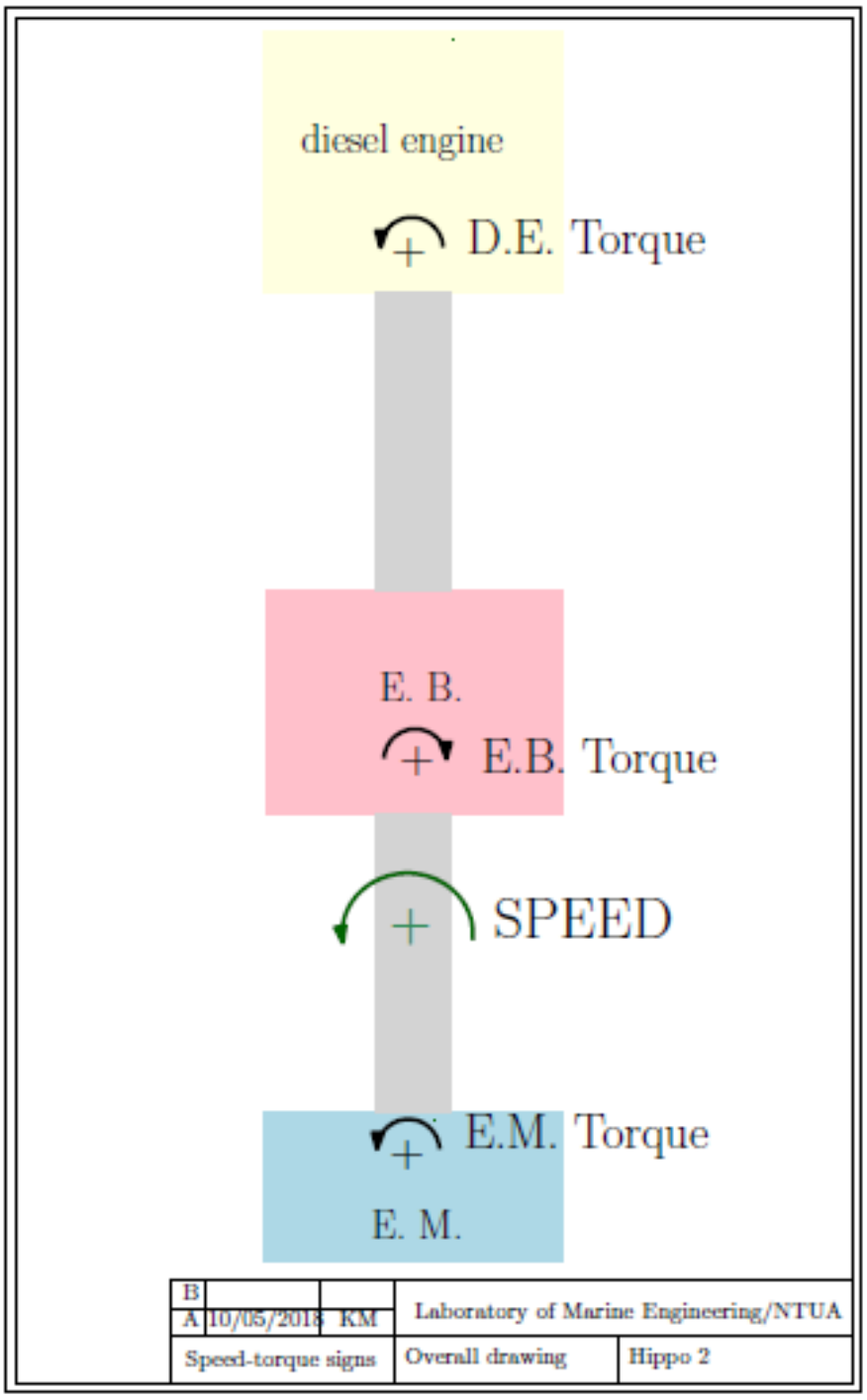

Figure 3.2: HIPPO-2 speed and torque outputs.

Final, EU Stage IV emission standards. Exhaust Gas Recirculation (EGR) and Selective Catalyst Reducer (SCR) systems for NOx reduction, are also incorporated, along with a Diesel Particulate Filter (DPF).

The EM is a standard AC induction 3-phase motor, with a rated power of 90 kW at 1483 rpm, type M3BP 280SMB 4 IMB3/IM1001, manufactured by ABB R. The EM can

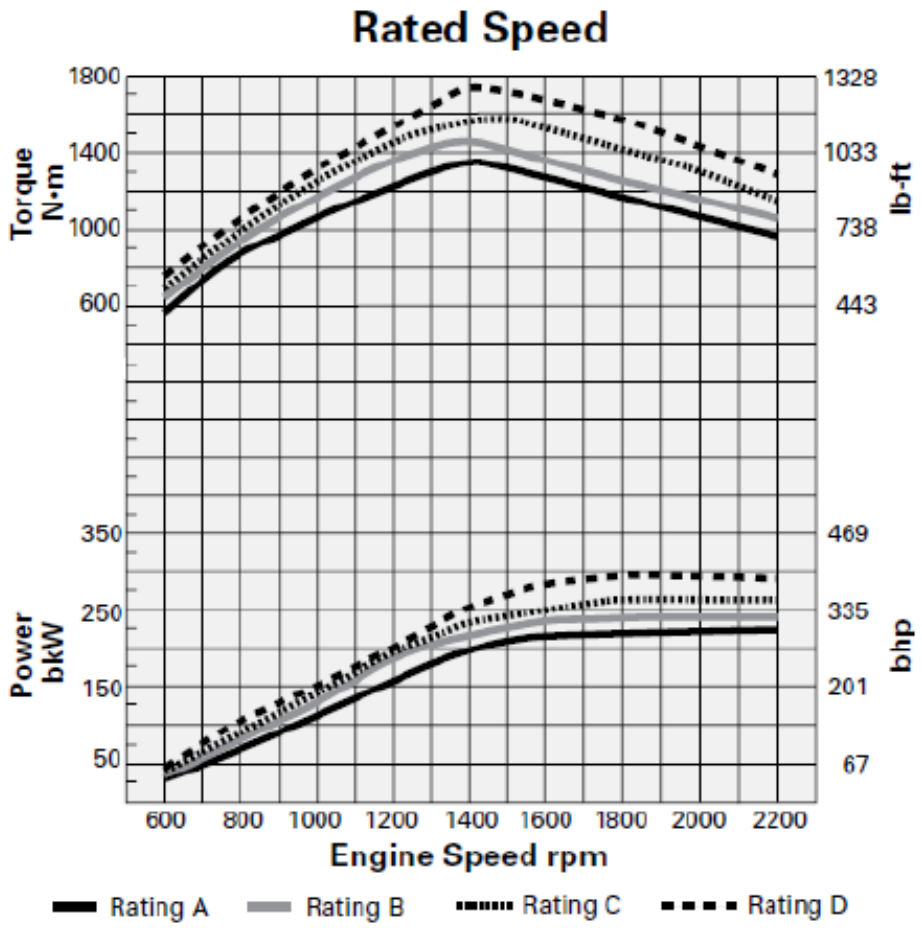

Figure 3.3: Loading diagram of CAT C9.3.

operate both as motor, to assist the engine, and as generator to store energy. The EB is a standard AC induction 3-phase motor manufactured also by ABB R , type M3BP 355SMB 4 IMB3/IM1001, with 315 kW load capacity, operating at 1488 rpm.

The 3 motors are connected in series, thus the operating speed range of HIPPO-2 is from 600 to 2200 rpm, with maximum load of 341 kW (ICE and EM combined power).

3.2 System components and Modeling

In this section a short brief of background analysis and focus is demonstrated along the final HIPPO-2 Modeling of dynamics used to execute the RL algorithms in this study. It is essential to describe the elements of the system and the approach in order to obtain a better understanding over the control philosophy and the construction of the algorithms in respect to the dynamics and the produced cost and reward function.

3.2.1 Diesel engine

The main attributes of diesel engines and consequently the reason of their dominance as the main powering device in industry, over the past century, is their relatively high power/weight ratio and their relatively high thermal efficiency [17]. Four stroke tur-

bocharged Diesels can reach efficiency of approximately 40 %. The two key factors are responsible for the above. The

first is the increased compression ratio. When the working uid is compressed, its temperature rises, leading to increased thermal efficiency. Since, the compressed fluid is consisted only from oxidizer (air), there is no self-ignition problem. The fuel eject and ignites, at the desired crankshaft angle. The second is that diesel engine can operate with lean mixtures of air and fuel in cylinder, such that throttling of the intake air can be completely avoided, something which is possible due to extremely hot air in cylinder. Consequently, the high thermal efficiency is maintained to a certain degree for part load operations.

The operation of diesel engines is associated with two major drawbacks [18]. The first is related to the low power-density they exhibit. This occurs from the fact that the mixture inside the combustion chamber is always lean, and thus less fuel can be burned in atmospheric conditions inside a cylinder. Moreover, engine maximum speed is relatively low due to mechanical limitations. This problem is sufficiently addressed with super or turbocharging the engine, namely, compressing the air before enters the cylinder, allowing more fuel to be burnt, in this way. The second disadvantage refers to issues of the exhaust gas purification. Apart of the ideal products of combustion, which are water (H2O) and carbon dioxide (CO2), several by products are also produced. A part of them are harmful to environment or cause health problems to humans. Therefore, numerous legislation, aiming to reduce the above effects, have been applied since the early 1970s for the automotive, and the late 1990s for marine industry, implementing limits to their concentration in the exhaust gases. The main pollutants that the above limits apply are nitrogen oxides (NOX), unburned hydrocarbons (HC), carbon monoxide (CO) and soot. Key factors for the concentration of above is ratio of air-fuel compered to stoichiometric (λ) and the cylinder temperature. The typical relation of air-fuel ratio and the emissions above is shown in **??**.

Although, Diesel engines have lower raw emissions than Otto , their working principle (i.e. lean operation), exclude the solution of the the three-ways-catalysts, since this system requires stoichiometric air-fuel ratios. Consequently, other technologies have developed, in order to restrict the above emissions, continually lowering fuel consumption and optimizing performance at the same time. These options after-treat the exhaust gas (e.g. Selective Catalyst Reducer - SCR), or affect the operation of engine itself (e.g. Exhaust Gas Recirculation - EGR).

Diesel engine behavior at transients operations

Most of the engine-oriented literature is focused on steady state operation, although transient applications represent a large portion of the engine operating patterns (e.g. maneuvering conditions for ships), or even the majority of operations (e.g. automotive vehicles). In recent years, due to the latest regulatory framework regarding the engine emissions, more attention is given in regard to this operation mode. According to literature [19], during transient loading profiles, gaseous and noise emissions typically exceed their acceptable values following the extreme, non-linear and non-steady-state conditions experienced during dynamic engine operation. For instance, 50 % of NOx emissions from automotive engines during the European Driving Cycle stem from periods of acceleration, whereas instantaneous particulate matter and NOx emissions during load increase transients have been measured to be 1 to 2 orders of magnitude higher than their respective quasi-steady values.

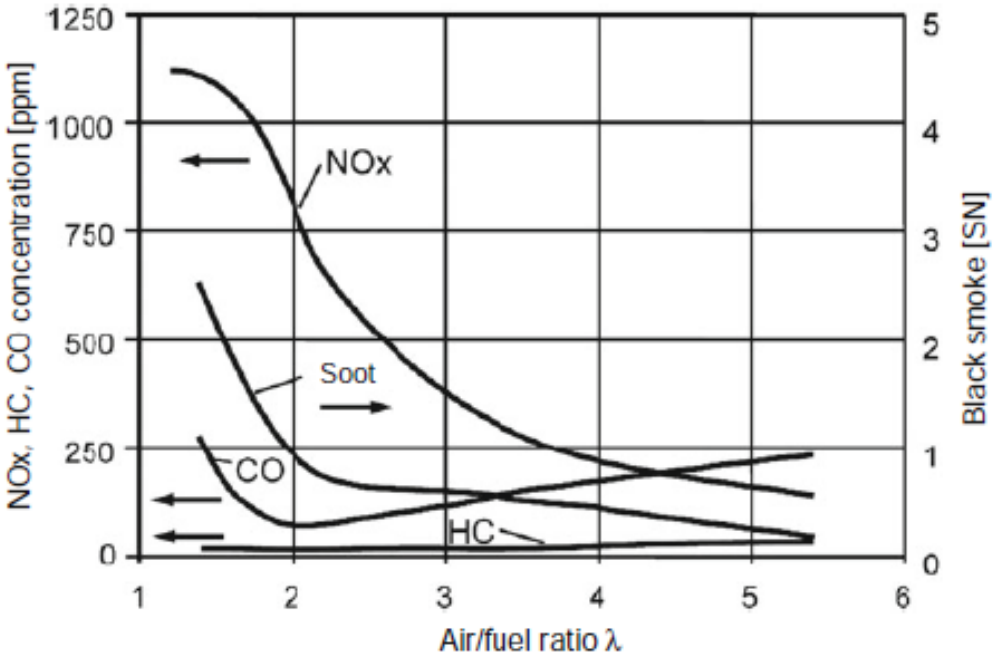

Figure 3.4: Relation between air fuel ratio and emissions.

3.2.2 Electric Motor/Generator

In hybrid propulsion plants, the electric machines are a key component, and usually they are reversible (i.e. the can operate both as motor and generator). The operation of these machines, according to [20], can be distinguished into three modes, one motoring and two generating, which are: (1) to convert the electrical power from the battery into mechanical power to drive the vehicle, (2) to convert the mechanical power from the engine into electrical power to recharge the battery, or (3) to recuperate mechanical power available at the drive train to recharge the battery (regenerative braking). Of course, in a marine hybrid plant the latter is not so common during operation, except maneuvering or special cases (e.g. during propeller ventilation the electric part can absorb and store to battery a portion of kinetic energy in order to reduce over-speeding). Desirable characteristics, for electric machines operating in a hybrid propulsion plant are [20]: high efficiency, low cost, high specific power, good controllability, fault tolerance, low noise, and uniformity of operation (low torque fluctuations).

Control of these motors is conducted with sophisticated electronics (inverters), and various control schemes have been applied in the past, such as the Scalar Control (V/f Control), the Field Oriented Control, Sliding Control Mode (SMC) and the Direct Torque Control (DTC). The industrial Drive which controls the AC motor of HIPPO 2 is using the DTC control scheme. This type of control is based on the mathematical approach of induction machines, and therefore various parameters, such as stator resistance, mutual inductance, saturation co-efficiency, etc. are required. The control variables are output torque and stator magnetic ux. DTC is able to control more accurate and has the fastest response time, does not need feedback devices and has reduced mechanical failure. The disadvantage is that due to the inherent hysteresis of the comparator, higher torque and flux ripple exist [21].

3.2.3 The Battery

Electrochemical batteries are key components of Hybrid Diesel - Electric Ships and Vehicles (HEV), in general. The main function of these devices, in a hybrid propulsion plant, is to transform and store electrical energy in chemical form, and then re-transform it back to electricity, in order to be used by the electric motors, when it is required. Each battery cell is characterized by the maximum power it can provide to the propulsion plant, and the nominal capacity. The

rst refers to the rate of energy a battery can provide to the plant, and it is the product of current and voltage. The latter, defines the amount of electricity a battery can supply, in terms of Coulombs (Ampere-seconds, As) or more often in Ampere-hours, Ah. Also, a dimensionless parameter, the State of Charge (SoC), describes the remaining capacity of the battery, and it is expressed as percentage or fraction of the nominal capacity.3.2.1

$$SoC(t) = \frac{Q(t)}{Q_nom} \tag{3.2.1}$$

3.2.4 HIPPO-2 Modeling of dynamics

In HIPPO-2 experimental testbed, the torque command to CAT C9.3, is given as percentage of the maximum indicated torque. According to the dynamic model which was presented in great detail in thesis [1], the gross torque can be modeled using three components, the net torque, the friction torque and the torque which is absorbed for pumping. Assuming that pumping torque, can be described for the particular operating region, from a 2nd degree polynomial, the relation between the command and the output torque is:3.2.2

$$T_{ICE} = c_1 \cdot uT_{ICE} - (c_2 + c_3 \cdot SE + c_4 \cdot SE^2) \tag{3.2.2}$$

Considering that the engine rating is C, the peak indicated torque according to manufacturer, is $1596Nm$ and since in torque control mode, command input is given to the controller as percentage of the peak net torque, it is considered that c_1 is 15.96 $Nm/\%$. The rest coefficients, were fitted from data which was measured from the hybrid test-bed, via least squares method.

The rotational shaft speed, was modeled via the second Newton's Law. Considering the previous relation, the differential equation which describes the dynamics of the system is non-linear, as follows:3.2.3

$$\frac{d\omega}{dt} = \frac{1}{J}(c_1 \cdot u_{TICE} + c_{EM} \cdot u_{TEL} - T_{LOAD} - (c_2 + c_3 \cdot SE + c_4 \cdot SE^2)) \tag{3.2.3}$$

Where ω is the rotational speed SE at rad/s, c_{EM} is the coefficient which transforms the control command to Nm. According to manufacturer, 100 % command translates to 579.6 Nm. Therefore, c_{EM} was considered to be 5.796 $Nm/\%$. T_{LOAD} refers to the resistant torque on the shaft in respect to the value of the Speed calculated to a constant c_{res}.3.2.4

$$T_{LOAD} = c_{res} \cdot SE \tag{3.2.4}$$

3.2.5 Formulation and Restrictions

The formulation under of which the problem was modeled, in accordance to the previous, includes all the elements and the restrictions due to the physical and functional limits of the system. As mentioned, the problem formulation follows two main discrete cases that where modeled: (1) the speed (RPM) and state of charge (SOC) tracking control over

defined trajectories (2) the speed (RPM) and state of charge (SOC) and NOx emission production control over the specified trajectories.

The states of the system are divided in to the two categories and follow a common formulation shown in table 3.1.

States in cases 1&2				
Variable	Symbol	Description	Range	Units
Controller Inputs case 1&2	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice}$	Engine Torque Rate	[-20,30]	$units/s$
	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel}$	Electric Torque Rate	[-50,50]	Nm/s
Intermediate States case 1&2	u_{Tice}	Engine Torque	[2,100]	%
	u_{Tel}	Electric Torque	[-90,90%]	-
Controlled States case 1&2	SE	Speed	[750,2000]	RPM
	SoC	State of Charge	[20,80%]	-
Extra Controlled States case 2	m_{nox}	NOx production	[0-60]	g/s
	tot_{nox}	NOx production	[0-inf]	g

Table 3.1: States in case 1&2

Now there have been used several parameters with an experimental on the test bed, parametric analysis and literature origin extensively described in [1]. The values of the parameters that where used in both two cases of the system are shown in table 3.2.

Altering the 3.2.2, 3.2.3&3.2.4, the Model of the system is described by the differential equations:3.2.5

$$\frac{\mathrm{d}}{\mathrm{d}t}SE(t) = \frac{60}{2\pi J_{set}}(15.96u_{Tice} + 5.8u_{T}el - T_{load} - (c_1 + c_2SE + c_3SE^2)) \tag{3.2.5}$$

which describes the RPM of the system

$$\frac{\mathrm{d}}{\mathrm{d}t}SoC(t) = -\frac{100}{Q_{nom}2Ri}(Uoc - \sqrt{(Uoc^2) - 4RiPb}) \tag{3.2.6}$$

3.2.6,which describes the State of Charge of the Battery, where

$$Uoc = kV1 + kV2SoC \tag{3.2.7}$$

3.2.7is the Battery Voltage, and

$$Pb = (5.8u_{Tel}(\frac{2\pi SE}{60} + Po)ef^{\frac{2}{1+2.7183^{2u_{Tel}}}-1} \tag{3.2.8}$$

3.2.8is the Electric Motor Consumption. Finally, 3.2.9

$$\begin{aligned} m_{nox} = \frac{1}{1000}(b2_{nox}(a2_{nox4} - \frac{0.5}{1+2.7^{-0.051(SE-1050)}})u_{Tice}^{1.1} \cdot a2_{nox3} \cdot SE \cdot a2_{nox5} \\ - 0.05(\frac{0.04}{1+2.7^{-0.4(u_{Tice}-80)}}\frac{0.4}{1+2.7^{-0.01*(SE-1750)}}u_{Tice} \cdot SE)) \end{aligned} \tag{3.2.9}$$

which describes the NOx production of the system.

States in cases 1&2				
Section	Symbol	Description	Value	Units
Shaft Torque case 1&2	J_{set}	HIPPO 2 mass Inertia	12.047	kgm^2
	Qf	Heating value diesel	$42.9 * 10^3$	kJ/kg
	c_1	Parametric constant	0.0238	-
	c_2	Parametric constant	-0.2343	-
	c_3	Parametric constant	0.1247	-
Battery Coefficients case 1&2	k_{V1}	Constant	1400	-
	k_{V2}	Constant	0.0409	-
	Ri	Constant	2.0480	-
	Q_{nom}	Constant	36000	-
Electric Motor Coefficients case 1&2	ef	Constant	0.9598	-
	Po	Constant	385.178	-
NOx production case 2	$a1_{nox3}$	Parametric constant	0.05344	-
	$a1_{nox4}$	Parametric constant	0.7919	-
	$a1_{nox5}$	Parametric constant	0.8534	-
	$b1_{nox}$	Parametric constant	0.09634	-
	$a2_{nox3}$	Parametric constant	0.3087	-
	$a2_{nox4}$	Parametric constant	0.7927	-
	$a2_{nox5}$	Parametric constant	0.04003	-
	$b2_{nox}$	Parametric constant	0.2358	-

Table 3.2: Parameters in case 1&2

Now, the constrains of the system can be categorised and are presented below:

Engine Torque Command limits
These refer to the potential of the engine to follow up the intermediate torque control input u_{Tice} and are expressed below.3.2.10 3.2.11 3.2.12

$$u_{Ticelimit1} = \frac{1}{15}SE - \frac{2.5}{3} \tag{3.2.10}$$

$$u_{Ticelimit2} = -\frac{1}{50}SE - 120.91 \tag{3.2.11}$$

where

$$u_{Tice} \leq u_{Ticelimit1} \& \leq u_{Ticelimit2} \tag{3.2.12}$$

Battery limits

These refer to the potential of the battery to follow up the intermediate torque control inputs and RPM and are expressed below.3.2.13

$$\frac{Uoc^2}{4Ri} - Pb \geq 1000 \tag{3.2.13}$$

Finally the last thing to be modeled for the specific applications is the cost function. This will be discussed on the next chapters as it depends on more elements such as the observations of the agent and the scenario trial of each training and simulation process.

Chapter 4

Basic DQN algorithm and Application

In this chapter, elements of basic structure for the RL agent Deep Q Network (DQN) in the julia environment which was applied are presented. The presentation focuses on the problem formulation, and the particulars solutions which were implemented in order to solve the problem of control. Furthermore, the basic functions and attributes of the software package ReinforcementLearning.jl which was employed to implement the control and observation schemes are also included. Last, simulation and conclusion over the different set-ups are presented.

4.1 Basic DQN principles

Recent years, many AI laboratories are working on studying deep reinforcement learning (DRL) which is expected to be a core technology in the future. Deep Q-Network (DQN) that is the first deep reinforcement learning method proposed by DeepMind. After the paper was published on Nature in 2015 [22], a lot of research institutes joined this field because deep neural network can empower RL to directly deal with high dimensional states like images, thanks to techniques used in DQN. As mentioned in Chapter 2, Q-learning is a simple yet quite powerful algorithm to create a value "sheet" for our agent. This helps the agent figure out exactly which action to perform. Now, in cases where this value sheet is too long, like an environment with 10,000 states and 1,000 actions per state, this would create a table of 10 million cells, making it computationally intensive. It is pretty clear that we can't infer the Q-value of new states from already explored states. This presents two problems:

- First, the amount of memory required to save and update that table would increase as the number of states increases
- Second, the amount of time required to explore each state to create the required Q-table would be unrealistic

In order to combat that we approximate these Q-values with machine learning models such as a neural network. In deep Q-learning, we use a neural network to approximate the Q-value function. The state is given as the input and the Q-value of all possible actions is generated as the output. The comparison between Q-learning & deep Q-learning is illustrated below:

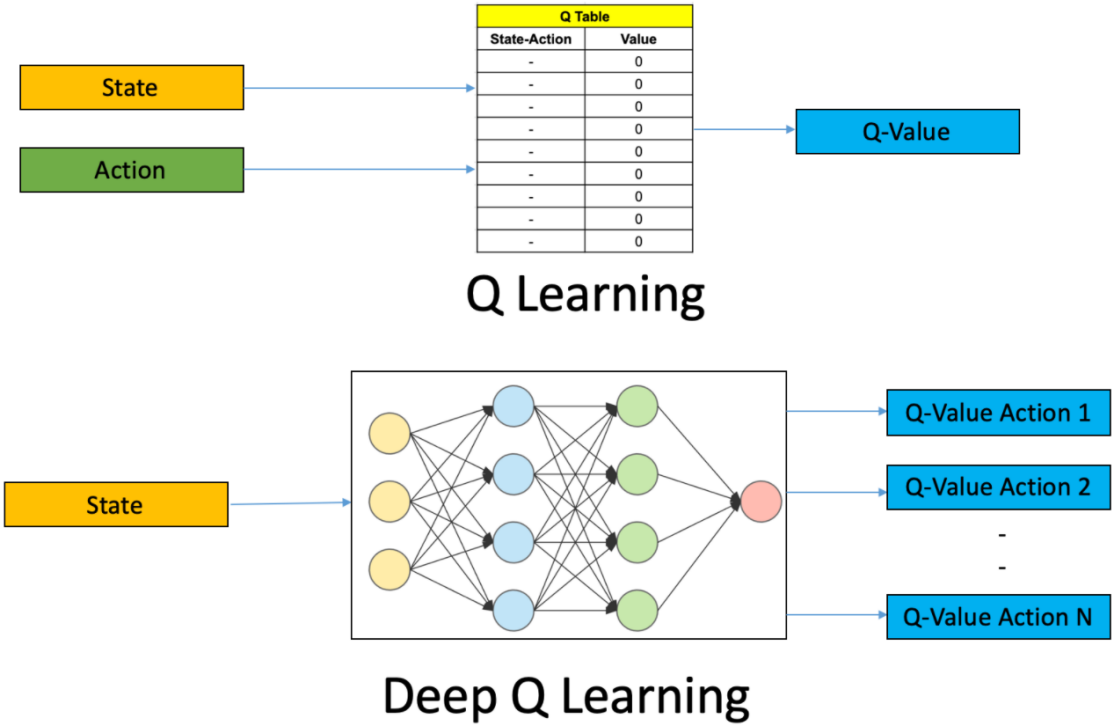

Figure 4.1: Q and DQN methods.

So, the steps involved in reinforcement learning using deep Q-learning network (DQNs) can be summarised:

- All the past experience is stored by the user in memory
- The next action is determined by the maximum output of the Q-network
- The loss function here is mean squared error of the predicted Q-value and the target Q-value – Q*. This is basically a regression problem. However, we do not know the target or actual value here as we are dealing with a reinforcement learning problem. Going back to the Q-value update equation derived fromthe Bellman equation. we have:

All the past experience is stored by the user in memory The next action is determined by the maximum output of the Q-network The loss function here is mean squared error of the predicted Q-value and the target Q-value – Q*. This is basically a regression problem. However, we do not know the target or actual value here as we are dealing with a reinforcement learning problem. Going back to the Q-value update equation derived fromthe Bellman equation. we have: 4.1.1

$$Q(S_t, A_t) < -Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{4.1.1}$$

The section in the brackets represents the target. We can argue that it is predicting its own value, but since R is the unbiased true reward, the network is going to update its gradient using backpropagation to finally converge.

Challenges in Deep RL
So far, We understood how neural networks can help the agent learn the best actions. However, there is a challenge when we examine deep RL. [23]

Algorithm 1: Pseudocode for DQN.

$Initialize Q(s,a) for all pairs (s,a)$
$s = initial state$
$k = 0$
while *convergence is not achieved* **do**
  simulate action a and reach state s'
  if $target = R(s,a,s')$ **then** s' is a terminal state
  else
    $target = R(s,a,s') + \gamma max_{a'} Q_k(s',a')$
  end
  $\theta_{k+1} = \theta_k - \alpha \Delta_\theta E_{s' \ P(s'|s,a)}[(Q_\theta(s,a) - target(s'))^2]|_{\theta=\theta_k} \ s = s'$
end

As we can see in the above code, the target is continuously changing with each iteration. In deep machine learning, the target variable does not change and hence the training is stable, which is just not true for RL. To summarise, we often depend on the policy or value functions in reinforcement learning to sample actions. However, this is frequently changing as we continuously learn what to explore. As training develops, we get to know more about the ground truth values of states and actions and hence, the output is also changing. So, we try to learn to map for a constantly changing input and output. Since the same network is calculating the predicted value and the target value, there could be a lot of divergence between these two. In order to address that demand, instead of using one neural network for learning, we can use two. Further information on this and the structure can also be found here. [24] We could use a separate network to estimate the

target. This target network has the same architecture as the function approximator but with frozen parameters. For every C iterations (a hyperparameter), the parameters from the prediction network are copied to the target network. This leads to more stable training because it keeps the target function fixed (for a while):4.2

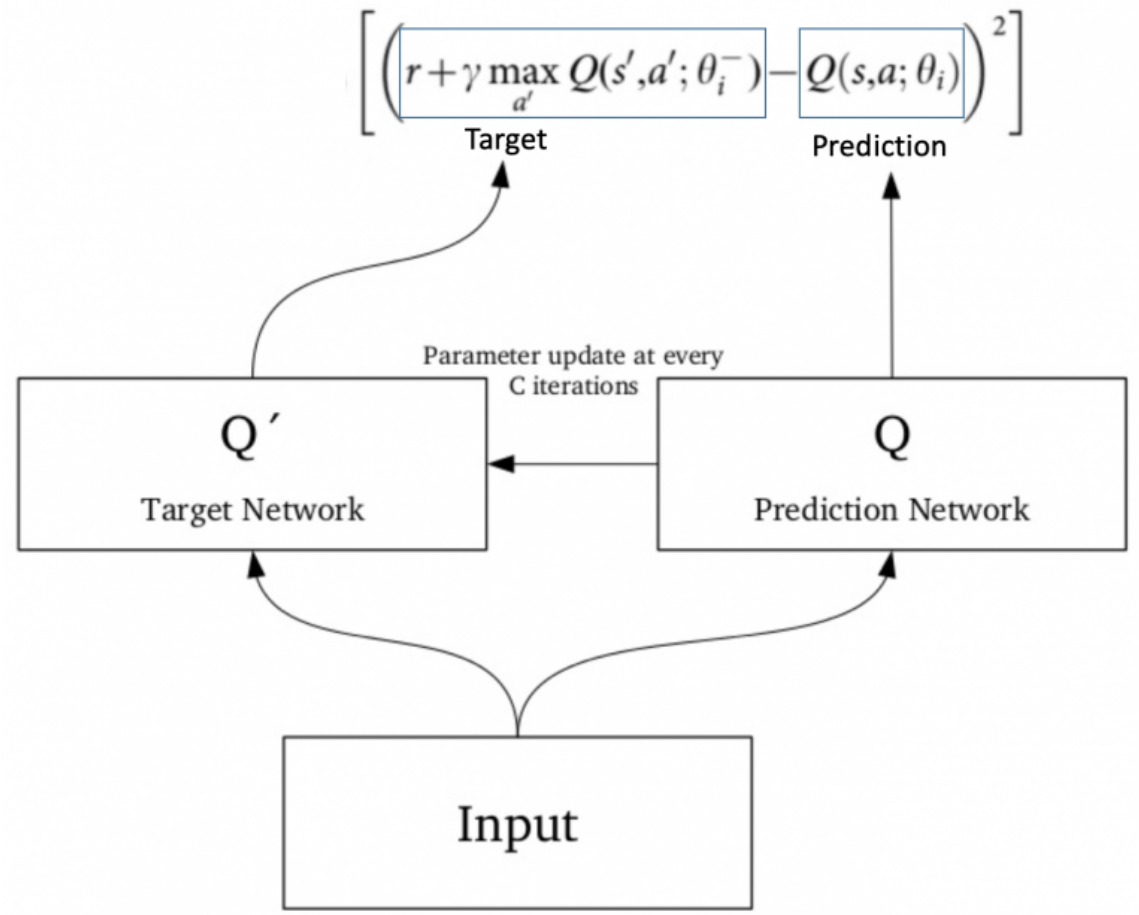

Figure 4.2: Target and Predicton NN.

Last, we need the element of experience. To perform experience replay, we store the agent's experiences. Instead of running Q-learning on state/action pairs as they occur during simulation or the actual experience, the system stores the data discovered for [state, action, reward, next-state] – in a large table. For example, suppose we are trying to build a video game bot where each frame of the game represents a different state. During training, we could sample a random batch of 64 frames from the last 100,000 frames to train our network. This would get us a subset within which the correlation amongst the samples is low and will also provide better sampling efficiency.

Summary in Deep RL

To sum up, the steps involved in a deep Q-network (DQN) are listed below:

- Pre-process and feed the state s to our DQN, which will return the Q-values of all possible actions in the state.
- Select an action using the epsilon-greedy policy. With the probability epsilon, we select a random action a and with probability 1-epsilon, we select an action that has a maximum Q-value, such as $a = argmax(Q(s, a, w))$.

- Perform this action in a state s and move to a new state s' to receive a reward. This state s' is the preprocessed image of the next state. We store this transition in our replay buffer as ¡s,a,r,s'¿.
- Next, sample some random batches of transitions from the replay buffer and calculate the loss.
- It is known that $Loss = (r + \gamma \max_{a'} Q(s', a'; \theta') - Q(s, a; \theta))^2$ which is just the squared difference between target Q and predicted Q.
- Perform gradient descent with respect to our actual network parameters in order to minimize this loss.
- After every C iterations, copy our actual network weights to the target network weights.
- Repeat these steps for M number of episodes.

4.2 The Julia environment

The Julia programming language is a flexible dynamic language, appropriate for scientific and numerical computing, with performance comparable to traditional statically-typed languages. ReinforcementLearning.jl, as the name says, is a package for reinforcement learning research in Julia language [25]. All the development and experimentation over this study have been conducted in Pluto.jl Julia package, which is an interface that enables fast programming and trials.

In this environment there are four core components in a general reinforcement learning experiment set-up:

- Agent.
- Environment.
- Stop Condition.
- Hook.

Each of the components can be designed or used from a library. In respect to the Agent, for our purposes this clearly refers to DQN agent. A set-up of this is actually presented below.

begin

agent = Agent(policy,trajectory)

policy = QBasedPolicy(learner,explorer)

learner = DQNLearner(approximator,target_approximator)

approximator = target_approximator

end

By calling the Agent function, a policy and a trajectory which is used to store transitions between the agent and the environment. The policy is set as the QBasedPolicy

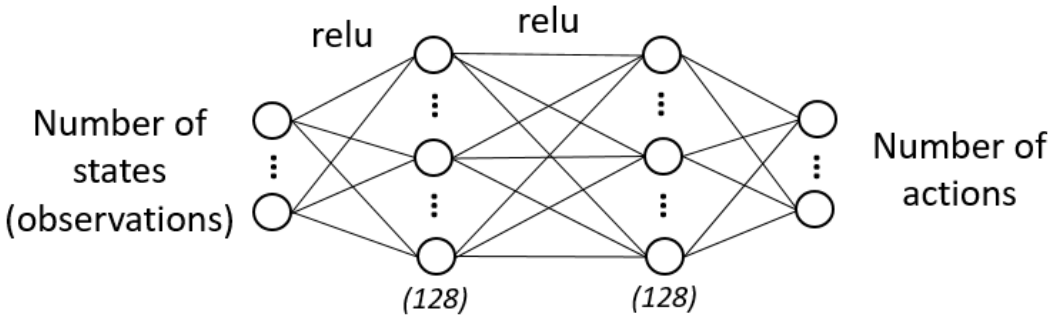

Figure 4.3: Approximator NN.

which requiresa learner and an explorer which is the EpsilonGreedyExplorer with ϵ=0.01 and exponential decay. In the learner function we define a prediction approximator and a target approximator NN where here they have the same chain structure of 3 layers with the number of states as inputs and the number of actions as outputs. All the intermediate nodes are 128 in number and are connected with relu activators.4.3

In respect to the Environment, for our purposes this clearly refers to the model of the system alongside any code suitable for information process and access. First, all the parameters and constants are defined in a structure as given in 3.2. Second, action space and observation space are declared in [-1,1] range. In a function structure by the name Reset, all the elements are initialized for every episode of the training process. Last, In a function structure by the name Env, all the elements alongside the iterations of the differential equations of the system are calculated and stored for every episode of the training process. Once the training is done, the trained agent can be extracted and stored for Simulation use, re-train purposes or actual trial on the physical system.

As for the Stop Condition and Hook, these are typical features of the Julia design which are not extensively developed.

4.3 Parallel PD Speed and State of Charge Tracking Control

In this section, the first set up of Speed and SOC control is presented. As mentioned, the HIPPO system consists of a Diesel engine and an electric motor on which the actions refer to, in respect to the RPM of the system plus the battery charge control. The action space, in respect to that plus the restrictions of discrete definition for the DQN algorithm, consists of 11 sets-couples of the one control variable $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel}$ as control variable from the Agent and $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice}$ as a control variable with a traditional PD controller in %, on Motor and Diesel accordingly.4.1

the PD control variable is as expected 4.3.1

$$u_{Tice} = k1 * errorSE + k2 * \frac{\mathrm{d}}{\mathrm{d}t} errorSE \tag{4.3.1}$$

As mentioned in 2 the observation space should be adequate to make it possible for the agent to create "logic" over a desired set of actions and should be normalized, preferably to a [-1,1] range. As for this case, the observation space consists of 4 input variables shown

Action space			
Numb of Action	Control variable	Numb of Action	Control variable
1	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = 0$	7	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = -10$
2	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = 10$	8	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = -20$
3	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = 20$	9	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = -30$
4	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = 30$	10	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = -40$
5	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = 40$	11	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = -50$
6	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = 50$	-	-

Table 4.1: Action space parallel DQN

below.

$$InputStates = \begin{bmatrix} SOCnorm \\ SOCdotnorm \\ errorSOCnorm \\ errorSOCdotnorm \end{bmatrix} \tag{4.3.2}$$

These refer to SOC, SOCdot, errorSOC and errorSOCdot all normalised under the formula 4.3.5. errorSOC is defined as 4.3.3

$$errorSOC = SOCtrack - SOC \tag{4.3.3}$$

and errorSOCdot is defined as eq:31

$$errorSOCdot = \frac{\mathrm{d}}{\mathrm{d}t} errorSOC \tag{4.3.4}$$

$$Normalisation = \tanh\left(2\frac{InputStates - min}{max - min} - 1\right) \tag{4.3.5}$$

the tanh function is actually used to squeeze closer the boundary values and provide a saturation safety during execution over some possible random oversized numbers.

To continue, we have the reward function. We shall define the cost function from which it is derived, which is set as follows 4.3.6

$$J = K3(errorSOCnorm)^2 + K4 * (errorSOCdotnorm)^2 \tag{4.3.6}$$

where K3=1 and K4=0.001 according to the weight "importance" we need to assign accordingly.

So the reward function for every iteration is defined 4.3.7

$$reward = \frac{K3 + K4}{2} - \sqrt{J} \tag{4.3.7}$$

Last, we define our termination condition, based on elements of the performance. Here this condition is set on the errorSOC=SOCtrack-SOC which should be kept in a range minor to 2, otherwise the episode is terminated earlier than the defined 300 steps of 0.2 sec each, or a.k.a. 60 seconds episode trial.

4.3.1 Training Parallel Control

For the training process we need to define the training scenario and the goal related reward function on the environment. For this case, we use a classic PD controller in the Speed variable (RPM) and a tracking control over a trajectory for the state of charge SOC of the battery, over a 60 second time range. The goal here is to create "logic" over training in multiple random scenarios, to track efficiently any given possible trajectory as a result.

The algorithm that define the trajectory of SOC is presented below:

Algorithm 2: Pseudocode for SOC tracking trajectory.

randomized start for every episode in range (47,53)
$IntSOC = 47 + (53 - 47)rand(1)$ initial randomized values
$RefSOC = 50.0$ reference value
$b = log(RefSOC/IntSOC)$
while *episodes not finished* **do**
  calculate SOC trajectory
  if *t less than 20 sec* **then**
  $SOCtrack = e^{b\frac{t}{20}-b}RefSOC$
  else
    $SOCtrack = RefSOC$
  end
end

Holding the training process for 500 episodes, with the specified reward function and termination condition we have the following results. First, we evalute the achievements in completing each episode in full steps, alongside the cost and reward function accumulation.4.4

What we can observe here is:(1) the training process achieves very early a full step state with the relative cost function and reward following a converging path to a min/max value. (2) There are areas-episodes batches where the convergence appears to be closer to optimum. This can mean that the training process does not constantly result in an optimum solution, since the physical limitations of the system allows a possible convergence, and the stop of the training should be considered to be better in those areas that it gets closer.

Moving further, we can get a clearer view over the performance of the agent on the 490 to 500 episode sample range.4.5 As mentioned, the error difference SOCtratck-SOC mainly contributes in the cost and reward function the bold line corresponds to the last episode.

As for the PD controlled state SE (RPM) the last episode is presented 4.6. Here, the limitation of the PD controlled variable is obvious as it presents with some fluctuation and a constant error from the tracking value unavoidable from fine tuning over the PD's weights. This is also considered as a "destabilization" factor for the optimal training. Last the variables u_{Tice} u_{Tel} that derive from the control variables - PD controller and agent action $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice}$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel}$ are presented below. 4.7 Since the training is complete and we are satisfied with the performance of the agent we can extract the trained agent and use it as a controller in senarios relevant to the training ones. Simulation and trials will follow up in 6.

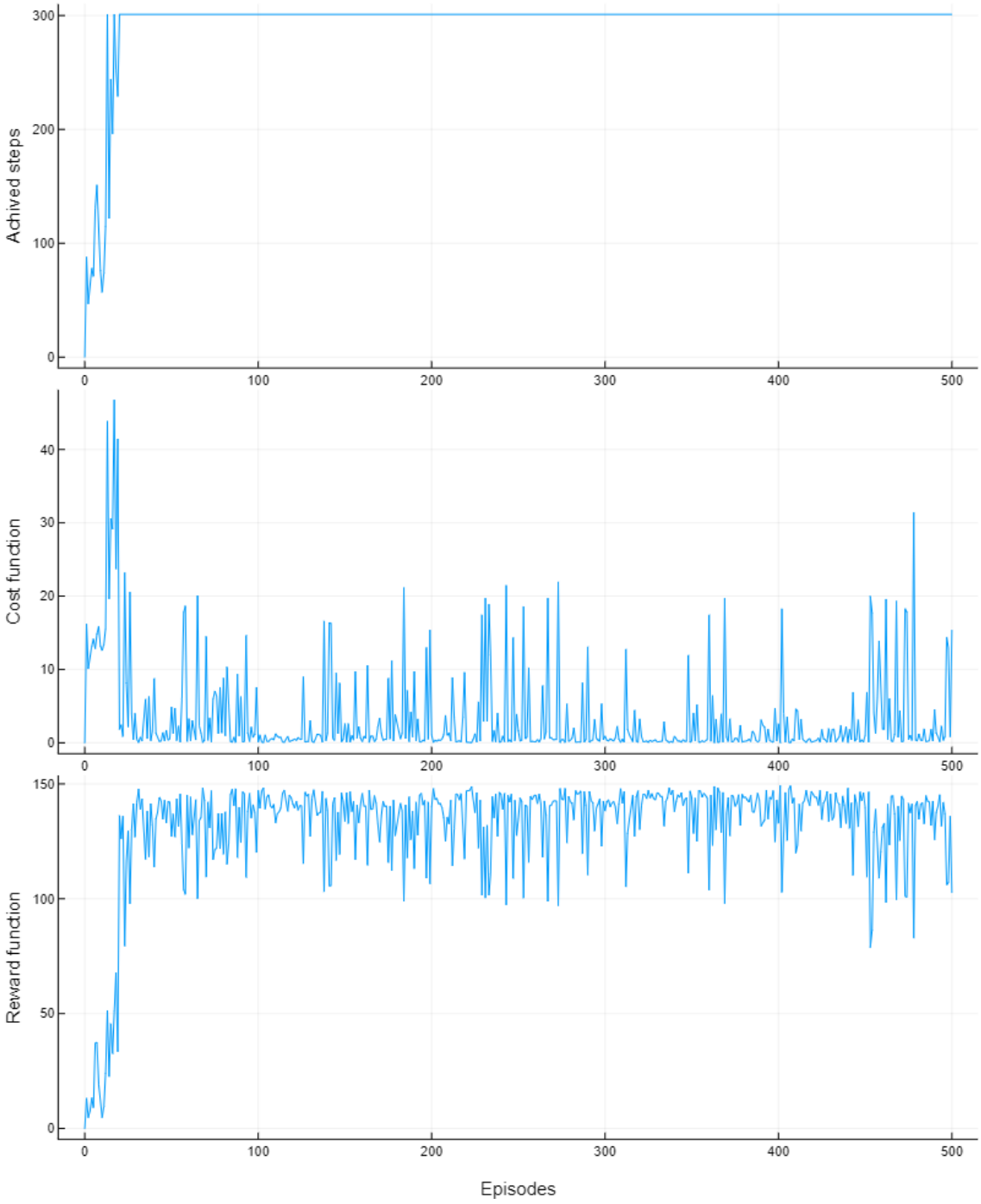

Figure 4.4: Episode achievements cost and reward function.

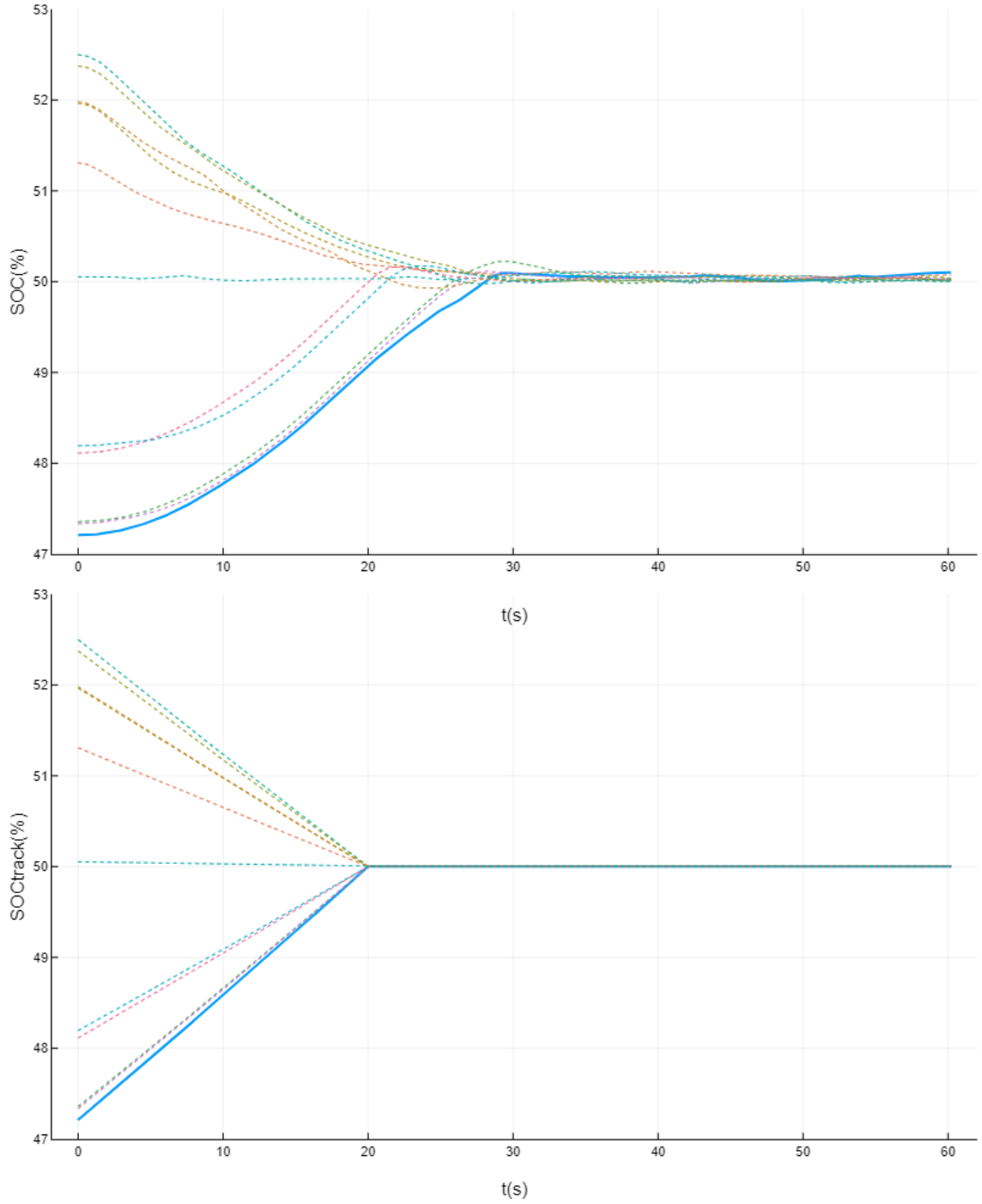

Figure 4.5: Parallel control SOC-SOCtrack 10 last episodes.

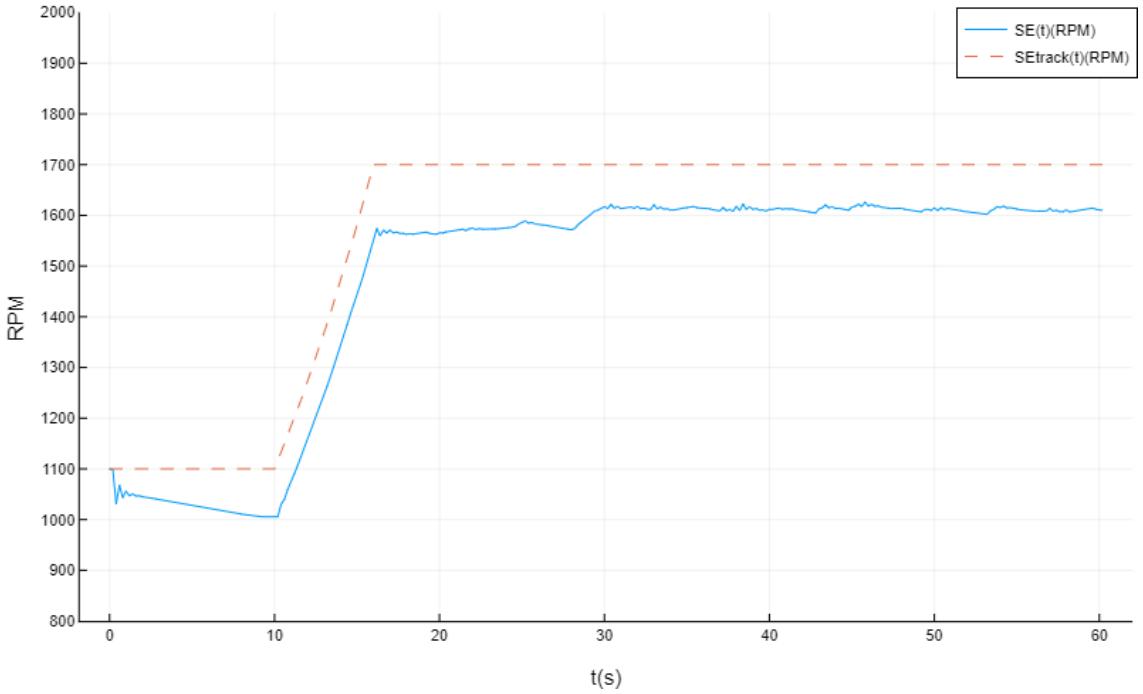

Figure 4.6: Parallel control SE-SEtrack last episode.

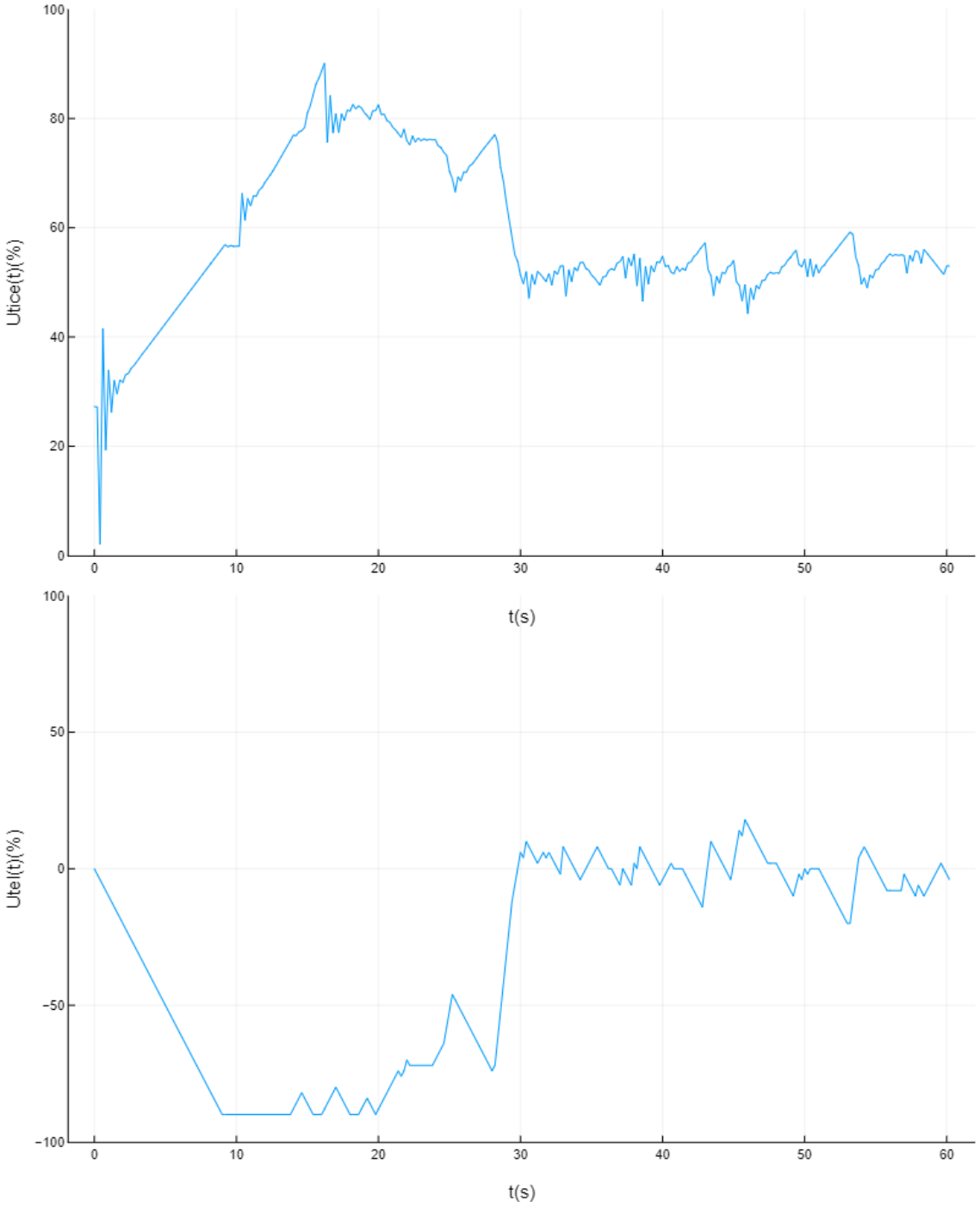

Figure 4.7: Parallel control u_Tice-u_Tel last episode.

4.4 Double action Speed and State of Charge Tracking Control Case 1

In this section, a more advanced set up of Speed and SOC control with double action of the agent is presented. Here, the action space, in respect to that plus the restrictions of discrete definition for the DQN algorithm, consists of 25 sets-couples of the two control variables $\frac{\mathrm{d}}{\mathrm{dt}}u_{Tice}$ and $\frac{\mathrm{d}}{\mathrm{dt}}u_{Tel}$ in (%), on Diesel and Motor accordingly. 4.2

Following the previous, again the observation space should be adequate to make it

Action space			
Numb of Action	Couple control set	Numb of Action	Couple control set
1	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice} = 0$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = 0$	14	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice} = 20$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = -15$
2	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice} = 10$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = 0$	15	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice} = -20$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = -15$
3	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice} = -10$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = 0$	16	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice} = 0$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = 30$
4	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice} = 20$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = 0$	17	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice} = 10$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = 30$
5	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice} = -20$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = 0$	18	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice} = -10$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = 30$
6	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice} = 0$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = 15$	19	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice} = 20$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = 30$
7	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice} = 10$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = 15$	20	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice} = -20$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = 30$
8	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice} = -10$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = 15$	21	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice} = 0$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = -30$
9	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice} = 20$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = 15$	22	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice} = 10$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = -30$
10	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice} = -20$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = 15$	23	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice} = -10$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = -30$
11	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice} = 0$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = -15$	24	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice} = 20$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = -30$
12	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice} = 10$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = -15$	25	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice} = -20$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = -30$
13	$\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice} = -10$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel} = -15$	-	-

Table 4.2: Action space DQN

possible for the agent to create "logic" over a desired set of actions and should be normalized, preferably to a [-1,1] range. As for this case, the observation space consists of 8

input variables shown below.

$$InputStates = \begin{bmatrix} SEnorm \\ SEdotnorm \\ errorSEnorm \\ errorSEdotnorm \\ SOCnorm \\ SOCdotnorm \\ errorSOCnorm \\ errorSOCdotnorm \end{bmatrix} \tag{4.4.1}$$

These refer to SOC, SOCdot, errorSOC and errorSOCdot all normalised under the formula 4.3.5 errorSE is defined as 4.4.2

$$errorSE = SEtrack - SE \tag{4.4.2}$$

and errorSEdot is defined as 4.4.3

$$errorSOCdot = \frac{\mathrm{d}}{\mathrm{d}t} errorSE \tag{4.4.3}$$

the tanh function is used as previous. The cost function is set as follows 4.4.4

$$\begin{aligned} J = K1 * (errorSEnorm)^2 + K2 * (errorSEdotnorm)^2 \\ + K3 * (errorSOCnorm)^2 + K4 * (errorSOCdotnorm)^2 \end{aligned} \tag{4.4.4}$$

where K1=K3=1 and K2=K4=0.001 according to the weight "importance" we need to assign to each state.

So the reward function for every iteration is defined 4.4.5

$$reward = \frac{K1 + K2 + K3 + K4}{2} - \sqrt{J} \tag{4.4.5}$$

Last, we define our termination condition, based on elements of the performance. Here this condition is set on the errorSE which should be kept in a range minor to 100 and errorSOC which should be kept in a range minor to 2, otherwise the episode is terminated earlier than the defined 300 steps of 0.2 sec each, or else 60 seconds episode trial.

4.4.1 Training Case 1

For the training process we need to define the training scenario and the goal related reward function on the environment. For this case, we use a tracking control over a trajectory in the Speed variable (RPM) and a tracking control over a trajectory in state of charge SOC for the battery, over a 60 second time range.

The algorithm that define the trajectory of SE and SOC is presented below:

Algorithm 3: Pseudocode for SE and SOC tracking trajectory.

randomized start for every episode $IntSOC = 48 + (52 - 48)rand(1)$ initial randomized values
$RefSOC = 50.0$ reference value
$b1 = log(RefSOC/IntSOC)$
$RefSE = 900 + (1800 - 900)rand(1)$ initial randomized values
$IntSE = 1200 + (1400 - 1200)rand(1)$ reference randomized values
$TermSE = 1200 + (1400 - 1200)rand(1)$ terminal randomized values
$b2 = log(RefSE/IntSE)$
$b3 = log(TermSE/RefSE)$
while *episodes not finished* **do**
    calculate SE and SOC trajectory
    if *t less than 20 sec* **then**
        $SOCtrack = e^{b\frac{t}{20}-b}RefSOC$
    else
        $SOCtrack = RefSOC$
    end
    if *t less than 10 sec* **then**
        $SEtrack = IntSE$
    else if *t less than 16 sec* **then**
        $SEtrack = e^{b2\frac{t-10}{16-10}-b2}RefSE$
    else if *t less than 40 sec* **then**
        $SEtrack = RefSE$
    else if *t less than 46 sec* **then**
        $SEtrack = e^{b3\frac{t-40}{46-40}-b3}TermSE$
    else
        $SEtrack = RefSE$
    end
end

Holding the training process for 500 episodes, with the specified reward function and termination condition we have the following results. First again, we evaluate the achievements in completing each episode in full steps, alongside the cost and reward function accumulation.4.8

What we can observe here is that the training process achieves again very early a full step state with the relative cost function and reward following a converging path to a min/max value and the convergence appears to be closer and closer to optimum.

Moving further, we can get a clearer view over the performance of the agent on the 490 to 500 episode sample range.4.9

As mentioned, the error difference SEtrack-SE and SOCtratck-SOC mainly contributes in the cost and reward function the bold line corresponds to the last episode. To have a better view on the scale of the error from the tracking value the last episode states to the reference are presented.4.10 & 4.11

Here, we can observe a much better performance and deviation from the tracking value in contrast to the previous combination of PD control. Hence, the double action providing total control over the systme is also considered as a better option for the optimal training.

Last the variables u_{Tice} u_{Tel} that derive from the control variables of the agent action $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice}$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel}$ are presented below. 4.12 Since the training is complete and we are satisfied with the performance of the agent we can extract the trained agent and use it as a controller in senarios relevant to the training ones. Again, simulation and trials will follow up in 6.

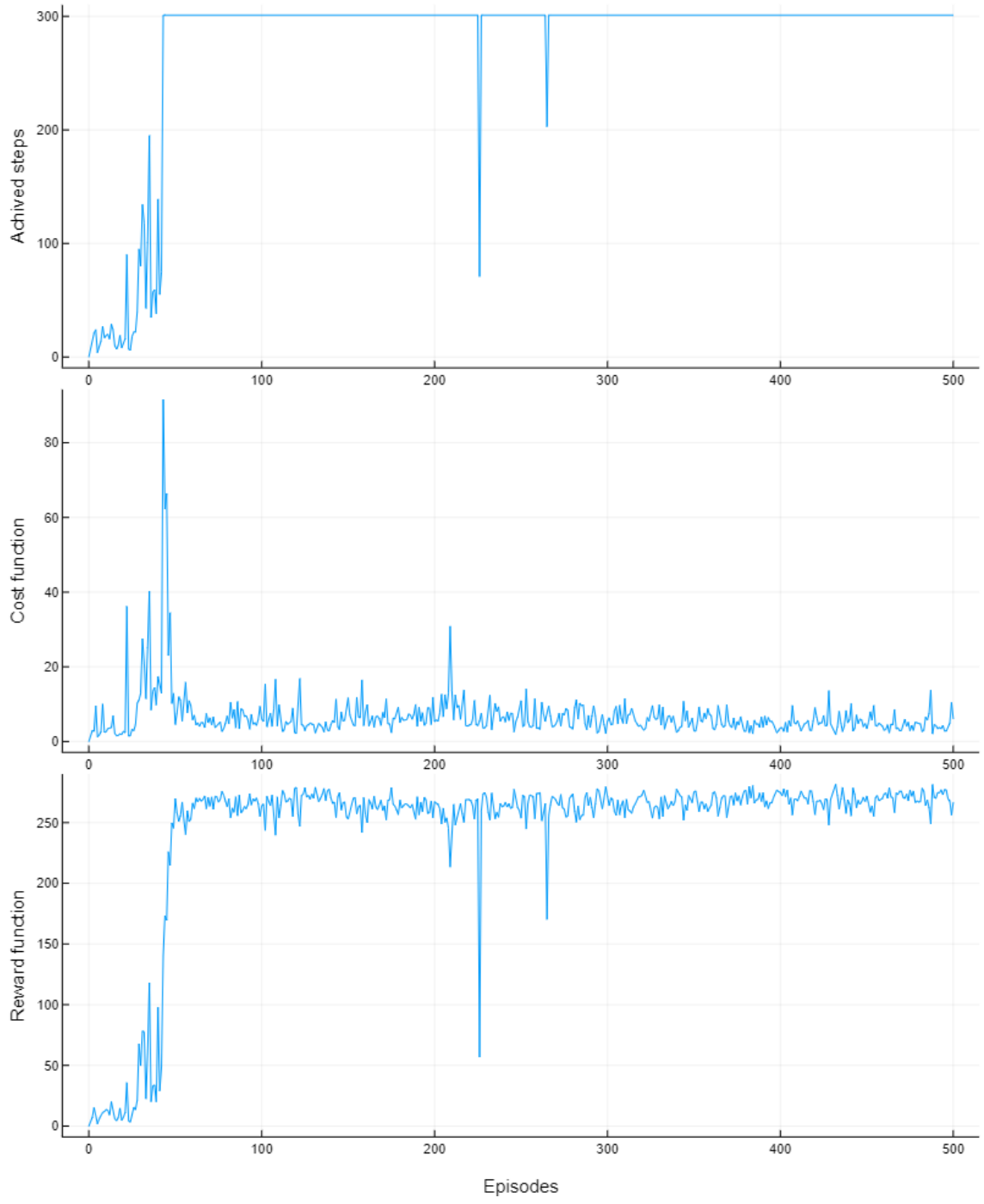

Figure 4.8: Episode achievements cost and reward function in Double action.

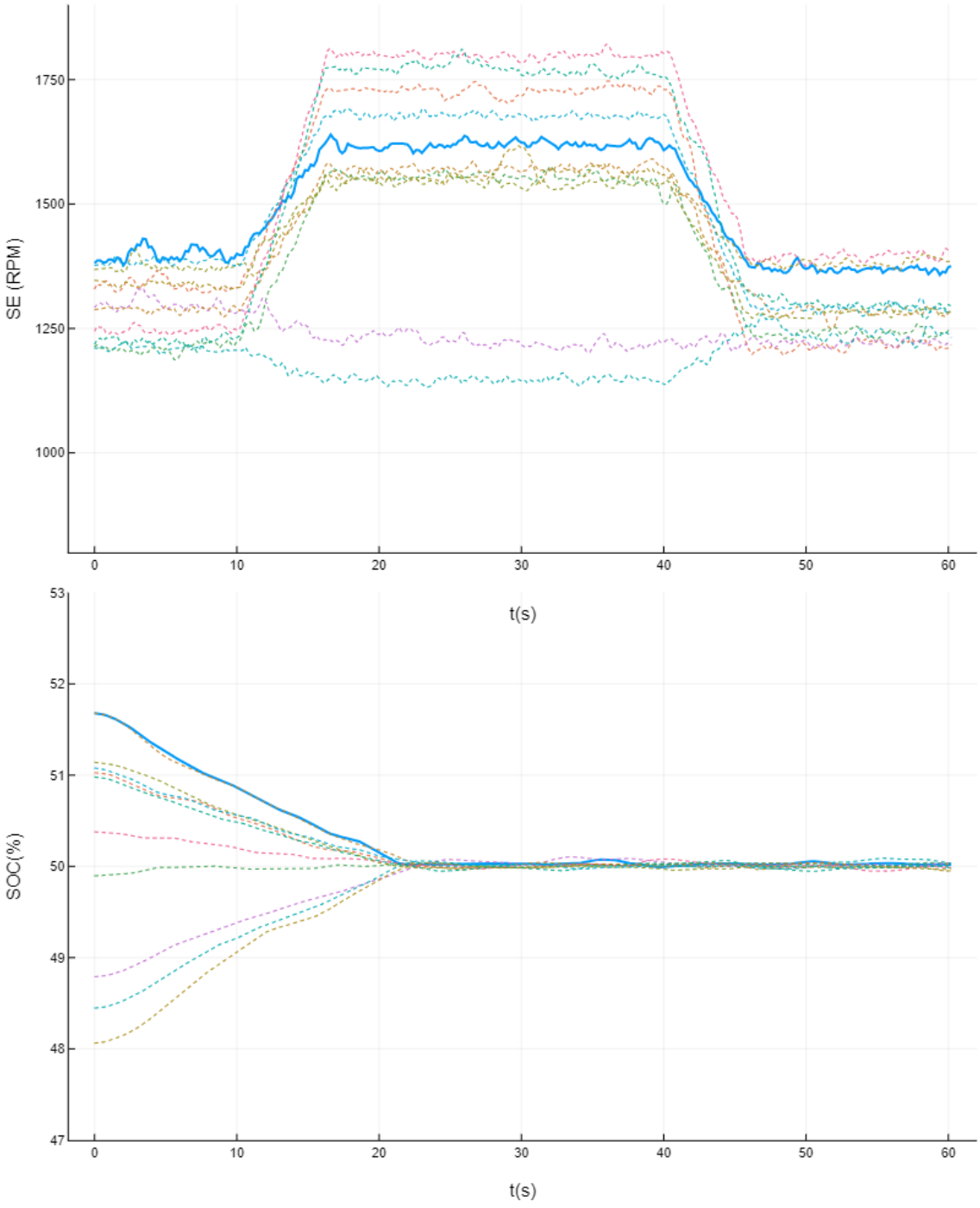

Figure 4.9: Double action control SE-SOC 10 last episodes.

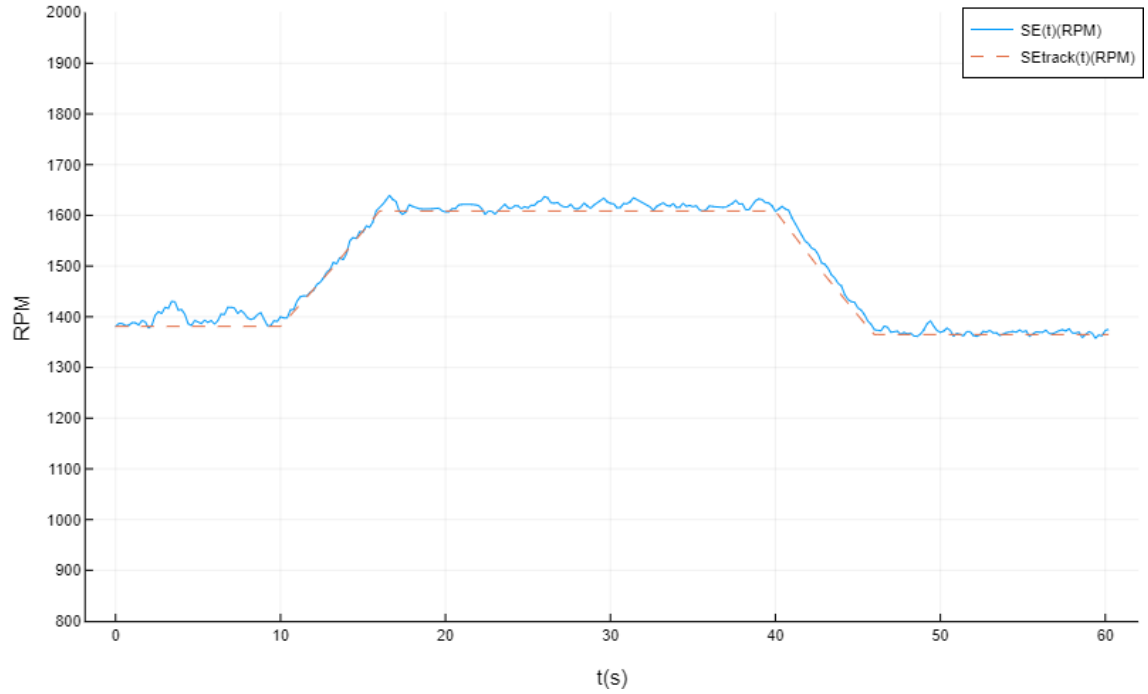

Figure 4.10: Double action control SE-SEtrack last episode.

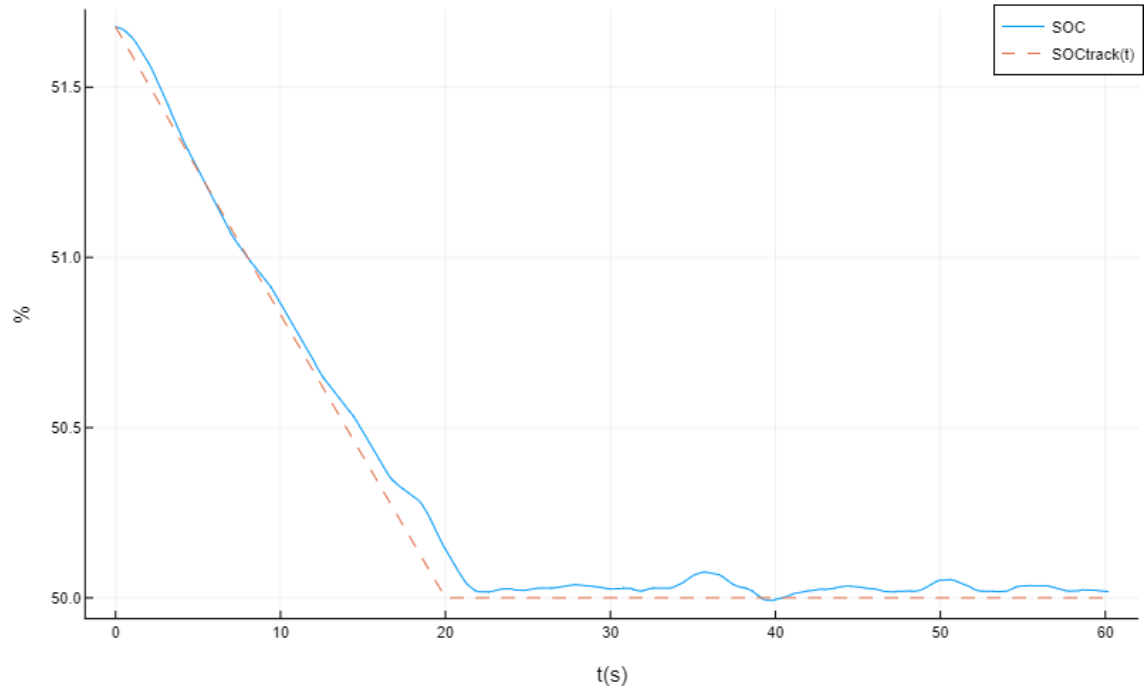

Figure 4.11: Double action control SOC-SOCtrack last episode.

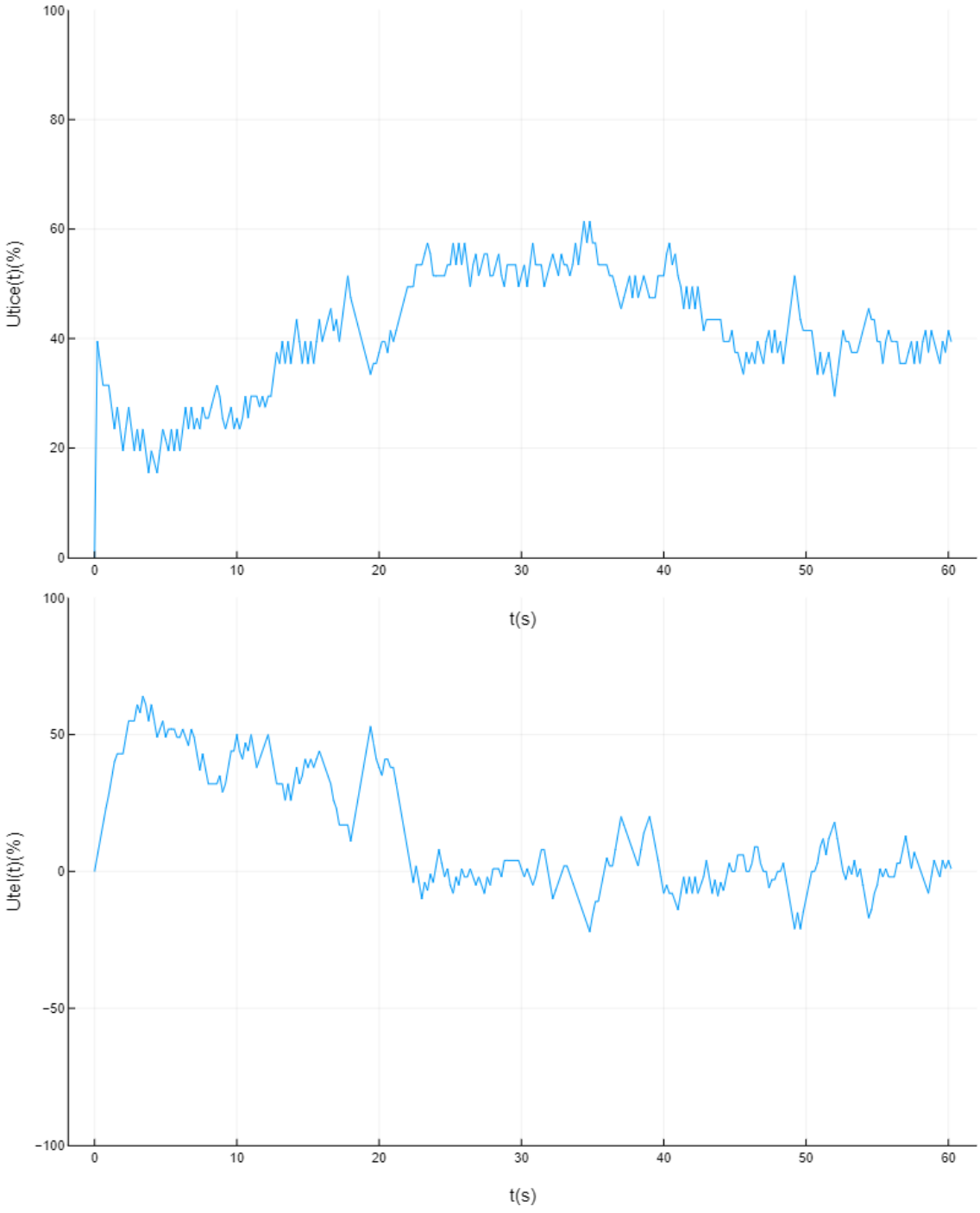

Figure 4.12: Double action control u_Tice-u_Tel last episode.

Chapter 5

PPO algorithm and Application

In this chapter, elements of basic structure for the RL agent Proximal Policy Optimization (PPO) in the MATLAB environment which was applied are presented. The presentation focuses on the problem formulation, and the particulars solutions which were implemented in order to solve the problem. Furthermore, the basic functions and attributes of the software package Reinforcement Learning Designer which was employed to implement the control and observation schemes are also included. Last, simulations and conclusion over the different set-ups are presented.

5.1 PPO principles

In recent years, several different approaches have been proposed for reinforcement learning with neural network function approximators. The search for improvement in developing a method that is scalable (to large models and parallel implementations), data efficient, and robust (i.e., successful on a variety of problems without hyperparameter tuning). For example, Q-learning (with function approximation) fails on many simple problems and is poorly understood, and policy optimization algorithms is relatively complicated, and is not compatible with architectures that include noise (such as dropout) or parameter sharing (between the policy and value function, or with auxiliary tasks). As mentioned in Chapter 2, Policy gradient is an approach to solve reinforcement learning problems. The policy gradient methods target at modeling and optimizing the policy directly. The policy is usually modeled with a parameterized function respect to $\theta, \pi_\theta(a|s)$. The value of the reward (objective) function depends on this policy and then various algorithms can be applied to optimize θ for the best reward. In the PG algorithm, our Agent is also called Actor. Actor has its own strategy π for a specific task. Strategy π can represented by a neural network, and its parameter is θ. Starting from a specific state until the end of the task, it is called a complete episode. At each step, we can get a reward r, and the final reward for a complete task is called R. In this way, for an episode with T moments, the Actor continuously interacts with the environment to form the following sequence τ:5.1 Such a sequence τ is uncertain, because the actions taken by the Actor in different states

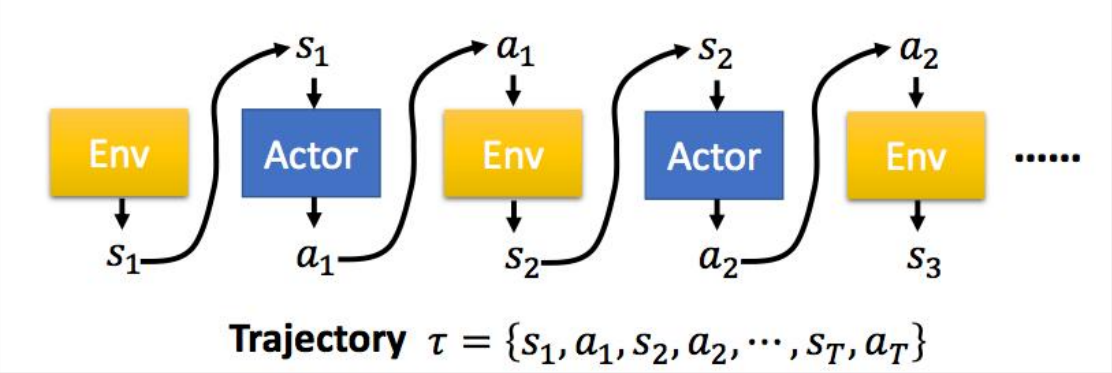

Figure 5.1: Basic policy gradient state to action.

may be different. Our expectation is to adjust the Actor's strategy π to maximize the expected reward, so we have a policy gradient method. The PG method so far uses the same reward for all data in the same episode. In fact, we can change it to be related to s_t and a_t. So if we use the advantage function, namely $Q_\pi(s_t, a_t) - V_\pi(s_t)$. Among them, $Q_\pi(s_t, a_t)$ can be obtained by discounting the reward from the current state to the end of the eposide, and $V_\pi(s_t)$ can be calculated by a critic. Hence, using the formulation introduced in 2, we can have an equation developed from equation 2.2.6 of reward function optimization, to demonstrate the appropriate PPO form.5.1.1

$$J^{CLIP}(\theta) = E[min(r(\theta)A_{\theta_{old}}(s,\alpha)clip(r(\theta)(1-\epsilon, 1+\epsilon))A_{\theta_{old}}(s,\alpha))] \tag{5.1.1}$$

In the above equation, the function clip truncates the policy ratio between the range [1-ϵ, 1+ϵ]. The objective function of PPO takes the minimum value between the original value and the clipped value. The key advantage of Actor–Critic PPO is that a new update of the policy model does not change it too much from the previous policy. It leads to less variance in learning, but ensures smoother policy update and also ensure that the agent

does not go down an unrecoverable path of taking senseless actions. [26] So, we can get the following schematic5.2 To sum up, all the steps are represented in the following algorithm:

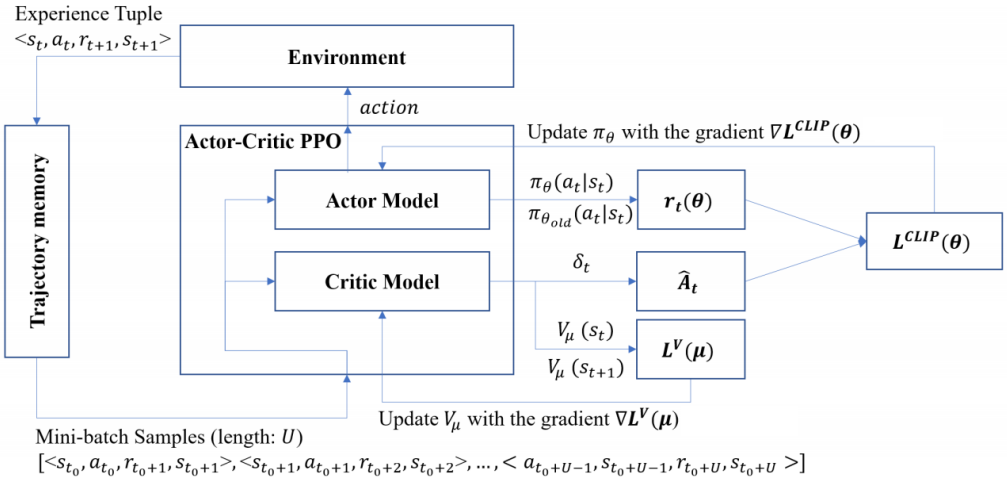

Figure 5.2: The Actor–Critic PPO algorithm process.

Algorithm 4: Pseudocode for PPO.

Initialize policy parameters,θ and value function parameters, ϕ.
while *timesteps not finished* **do**
  -Collect the state,action, prev logprobs, reward in batches from set of trajectories by running actor using the policy $\pi_{\theta_{old}}$.
  -Estimate adavantage value, A_k.
  while *update per iteration not finished* **do**
    -Find the current log probability based on π_θ(Note: in the first iteration always $\pi_{\theta_{old}}$). This is done by running action with new policy π_θ with batch state and batch action from the earlier steps.
    -Find the importance ration $r(\theta)$ = current log probability - prev log probs(Note: as per the equation it is a division, but when you add log to it, we can change the division to substraction.).
    -Compute $r(\theta)A_k$ and clipped ration.
    -Find the actor loss with clipped obejective.
    -Train the actor.
    -Actor policy parameters, θ get updated, we have new policy π_θ.
  end
end

5.2 The Matlab RL Designer environment

The Matlab RL Designer environment is a flexible dynamic app, appropriate for scientific and numerical computing and consists a package for reinforcement learning research in the traditional Matlab interface. All the development and experimentation over this study have been conducted in this, which is an interface that enables fast programming and trials.

In this environment there are four core components in a general reinforcement learning experiment set-up:

- Observation Info.
- Action Info.
- Step Function.
- Reset Function.

Each of the components can be designed or used from a library. In respect to the Agent, for our purposes this clearly refers to continuous action PPO agent. The most important role to the design of an agent is on the design of the Actor and Critic NNs 5.3 5.4

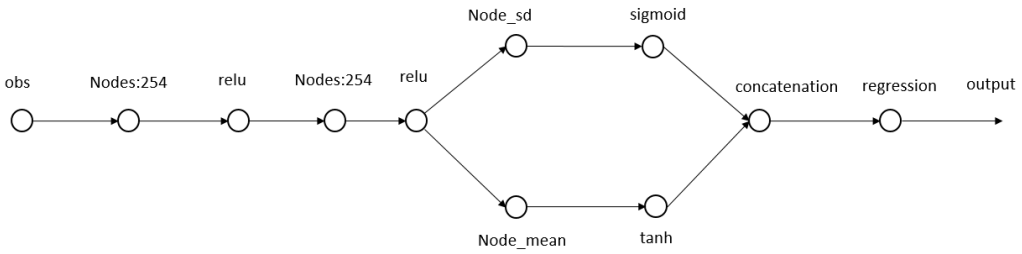

Figure 5.3: Actor NN.

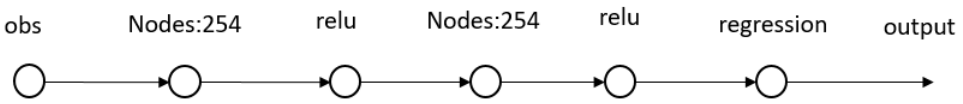

Figure 5.4: Critic NN.

By calling the Agent in the app, importing the NNs to the agent follows along the basic elements of number of steps and episodes and several parametres related to the NN tuning are presented. According to the architecture of the Actor network following 2, the first intermediate nodes are 256 in number and are connected with relu activators, then we have two branches, one corresponding to the probability of each action and the other to standard deviation of it. The first branch follows with a tahn activator to limit the two actions output in range [-1,1], the second with a sigmoid activator to limit the output [0,1] and fight against saturation. The Critic network consists of a single branch straight generation.

In respect to the Step and Reset Function, for our purposes this clearly refers to the model of the system alongside any code suitable for information process and access. First,

all the parameters and constants are defined in a structure as given in 3.2. Second, action space and observation space are declared in [-1,1] range. In the function structure by the name Reset, all the elements are initialized for every episode of the training process. Last, In a function structure by the name Step, all the elements alongside the iterations of the differential equations of the system are calculated and stored for every episode of the training process. Once the training is done, the trained agent can be extracted and stored for Simulation use, re-train purposes or actual trial on the physical system.

5.3 Double action Speed and State of Charge Tracking Control Case 1

In this section, the same as in 4, the more advanced set up of Speed and SOC control with double action of the agent is presented. Here, the action space is defined in continuous space in a range [-1,1], and consists of 2 scaling formulas for the two control variables $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice}$ and $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel}$ in (%), on Diesel and Motor accordingly.

Algorithm 5: Pseudocode for control values.

```
dt = 0.2
while timesteps not finished do
    Import the action values a(1) and a(2) from the Actor NN.
    d/dt u_Tice += 20(min(max(a(1), −1), 1))dt)
    d/dt u_Tel += 30(min(max(a(1), −1), 1))dt)
end
```

the clipping factor is considered supplementary, for tackling slipping and saturation.

Following the previous, again the observation space should be adequate to make it possible for the agent to create "logic" over a desired set of actions and should be normalized, preferably to a [-1,1] range. As its was for DQN, for this case, the observation space also consists of 8 input variables shown below.

$$InputStates = \begin{bmatrix} SEnorm \\ SEdotnorm \\ errorSEnorm \\ errorSEdotnorm \\ SOCnorm \\ SOCdotnorm \\ errorSOCnorm \\ errorSOCdotnorm \end{bmatrix} \tag{5.3.1}$$

These refer to SOC, SOCdot, errorSOC and errorSOCdot all normalised under the formula 4.3.5 errorSE is defined as 5.3.2

$$errorSE = SEtrack - SE \tag{5.3.2}$$

and errorSEdot is defined as 5.3.3

$$errorSOCdot = \frac{\mathrm{d}}{\mathrm{d}t} errorSE \tag{5.3.3}$$

the tanh function is used as previous. The cost function is set as follows 5.3.4

$$\begin{aligned} J = {} & K1 * (errorSEnorm)^2 + K2 * (errorSEdotnorm)^2 \\ & + K3 * (errorSOCnorm)^2 + K4 * (errorSOCdotnorm)^2 \end{aligned} \tag{5.3.4}$$

where K1=K3=1 and K2=K4=0.001 according to the weight "importance" we need to assign to each state.

So the reward function for every iteration is defined 5.3.5

$$reward = \frac{K1 + K2 + K3 + K4}{2} - \sqrt{J} \tag{5.3.5}$$

Last, we define our termination condition, based on elements of the performance. Here this condition is set on the errorSE which should be kept in a range minor to 100 and errorSOC which should be kept in a range minor to 2, otherwise the episode is terminated earlier than the defined 300 steps of 0.2 sec each, or else 60 seconds episode trial. As we mention, all formulation is kept the same, for a potential good comparison.

5.3.1 Training Case 1

For the training process we need to define the training scenario and the goal related reward function on the environment. For this case, we use a tracking control over a trajectory in the Speed variable (RPM) and a tracking control over a trajectory in state of charge SOC for the battery, over a 60 second time range.

The algorithm that define the trajectory of SE and SOC is the same as in the DQN Case 1 trail 4.

Holding the training process for 500 episodes, with the specified reward function and termination condition we have the following results. First again, we evalute the achievements in completing each episode in full steps, alongside the cost and reward function accumulation. 5.5 5.6

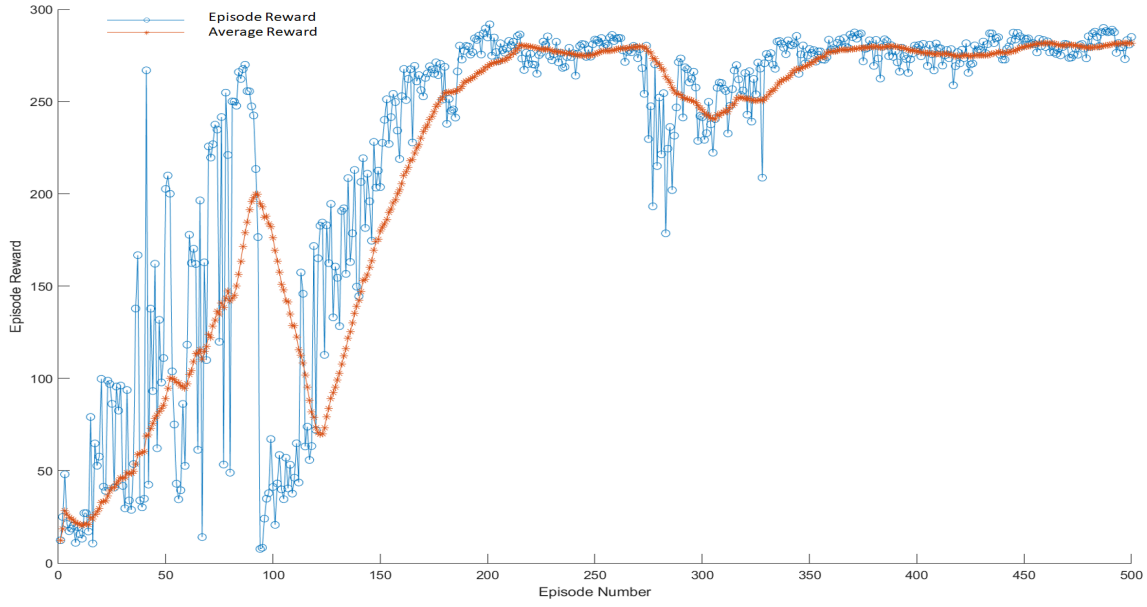

Figure 5.5: Reward function in Double action PPO.

What we can observe here is that the training process achieves again very early a full step state with the relative cost function and reward following a converging path to a min/max value and the convergence appears to be closer and closer to optimum.

Moving further, we can get a clearer view over the performance of the agent on the 490 to 500 episode sample range.5.7 As mentioned, the error difference SEtrack-SE and SOCtratck-SOC mainly contributes in the cost and reward function the boold line corresponds to the last episode. To have a better view on the scale of the error from the

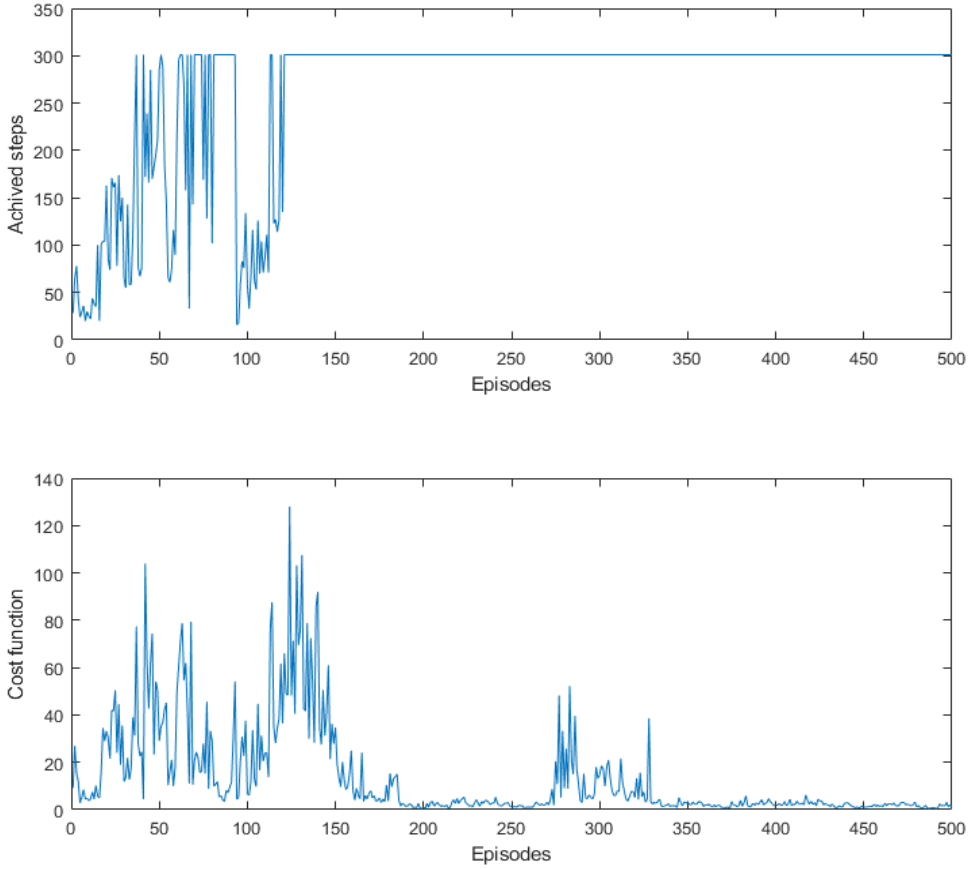

Figure 5.6: Episode achievements cost function in Double action.

tracking value the last episode states to the reference are presented.5.8 & 5.9 Here, we can observe a much better performance and deviation from the tracking value in contrast to the previous combination of PD control. Hence, the double action providing total control over the systme is also considered as a better option for the optimal training. Last the variables u_{Tice} u_{Tel} that derive from the control variables of the agent action $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice}$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel}$ are presented below. 5.10 Since the training is complete and we are satisfied with the performance of the agent we can extract the trained agent and use it as a controller in senarios relevant to the training ones. Again, simulation and trials will follow up in 6.

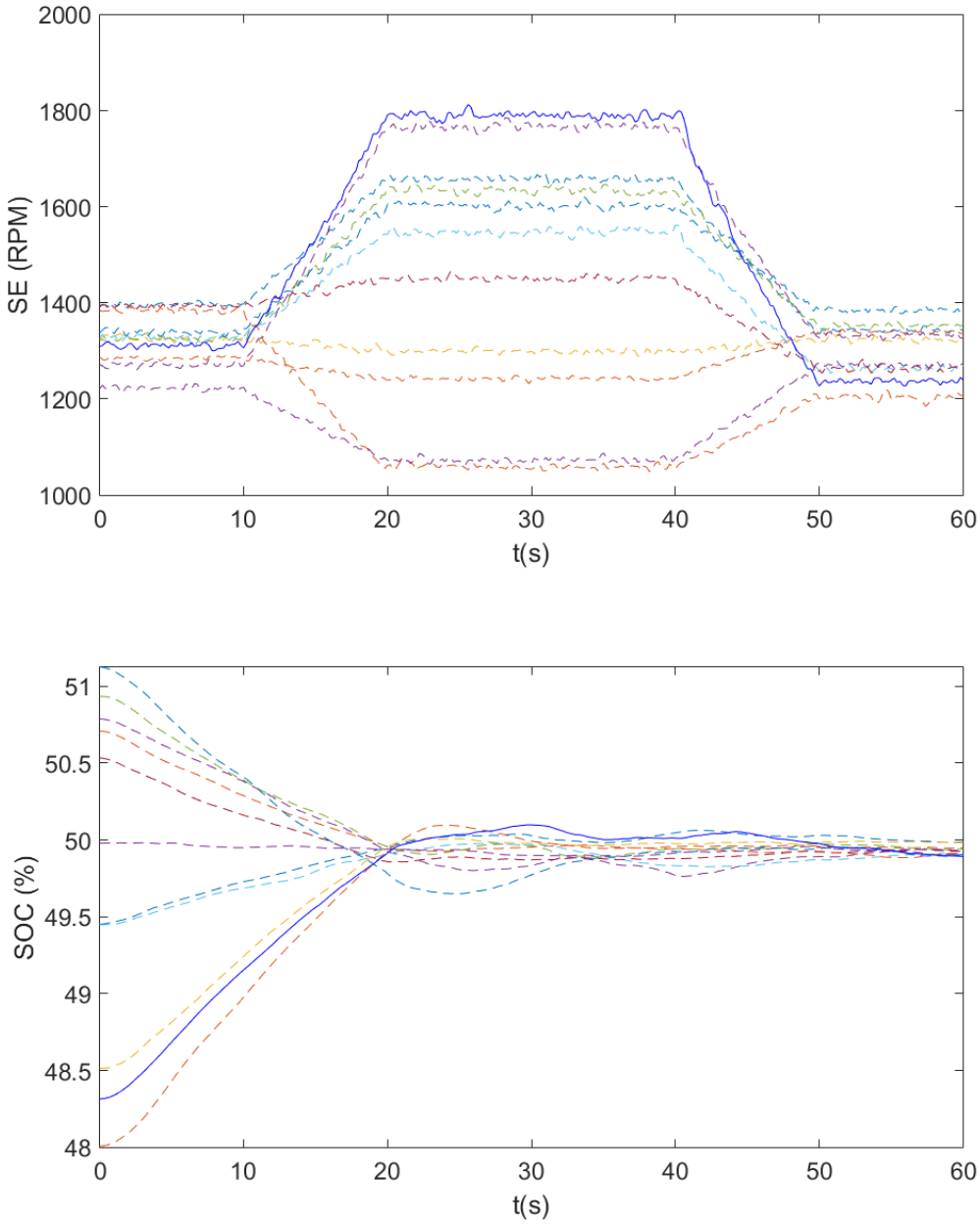

Figure 5.7: Double action control SE-SOC 10 last episodes PPO.

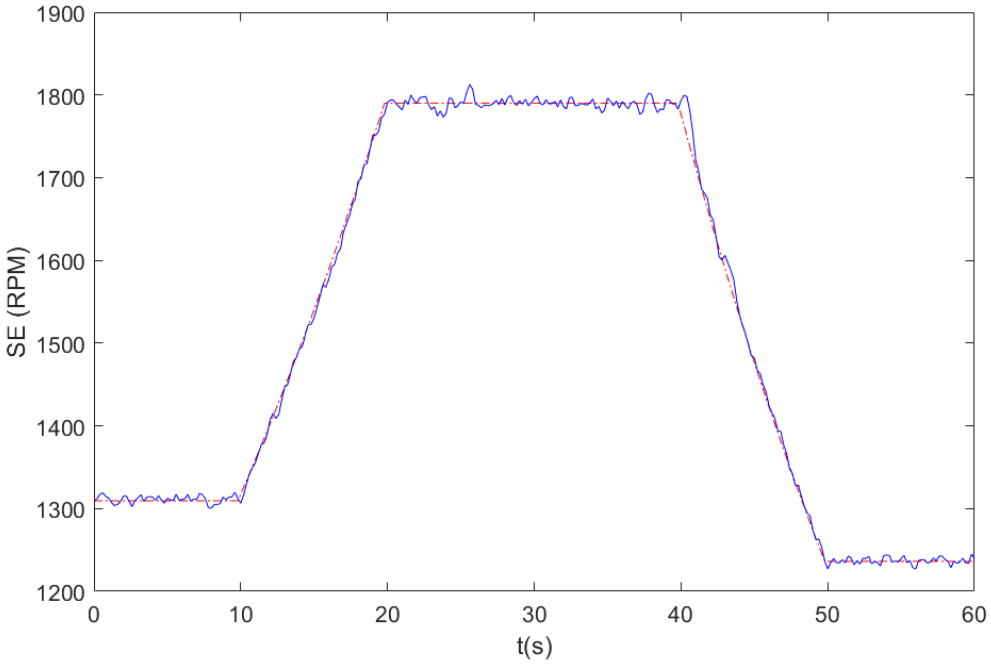

Figure 5.8: Double action control SE-SEtrack last episode PPO.

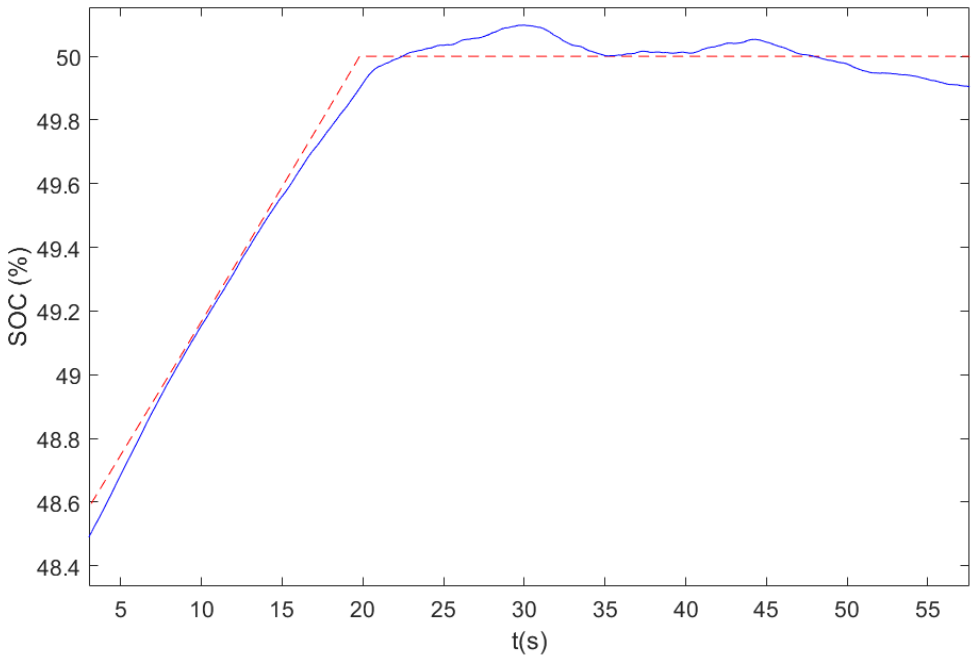

Figure 5.9: Double action control SOC-SOCtrack last episode PPO.

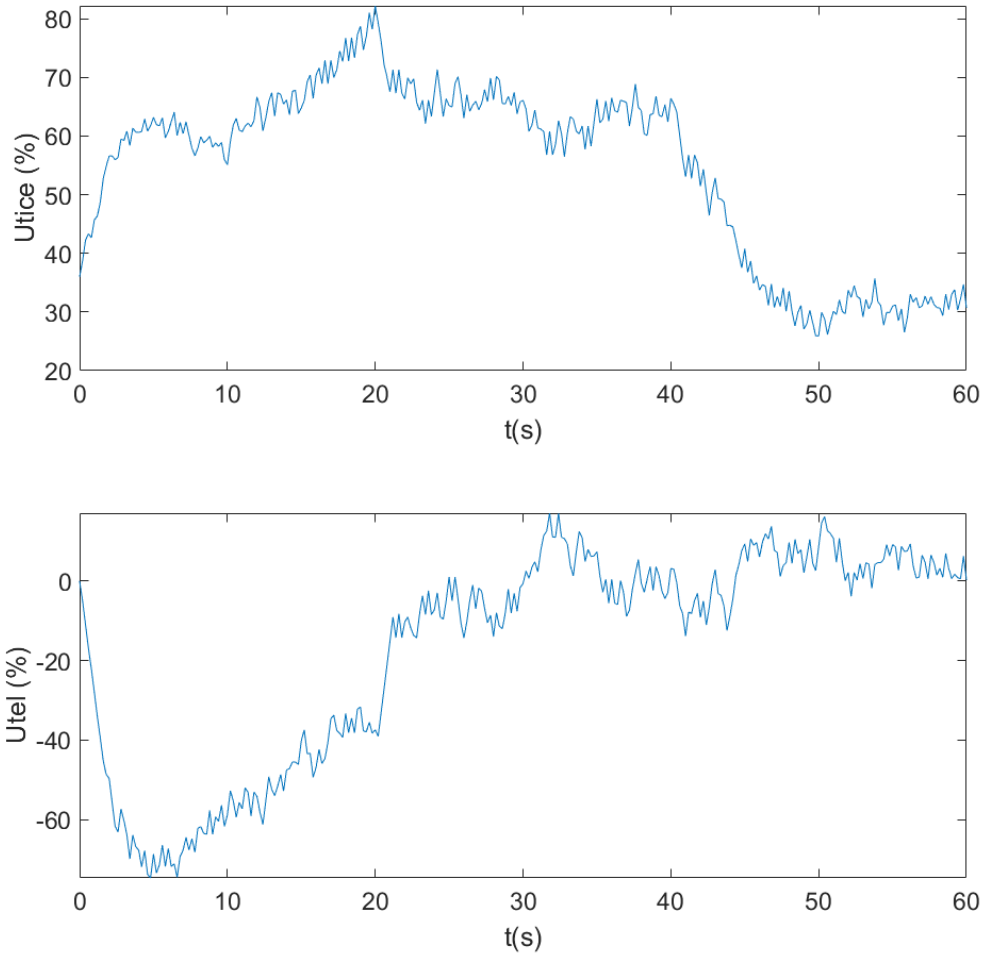

Figure 5.10: Double action control u_Tice-u_Tel last episode PPO.

5.4 Double action Speed and NOx Control Case 2

In this section, the final more advanced set up of Speed and NOx control is introduced with double action of the agent. Here, again the action space is defined in continuous space in a range [-1,1], and consists of 2 scaling formulas for the two control variables $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice}$ and $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel}$ in (%), on Diesel and Motor accordingly, identical to the DQN trial. As mentioned in 3, a model suitable for NOx emission production of the HIPPO2 system is provided **??**. The overall goal here is to have a good trade off between the Diesel engine and the battery-Motor of the system in order to provide a good control on Speed SE (RPM) and at the same time follow an optimal combination of u_{Tice} and u_{Tel} to minimize the NOx in comparison with the single Diesel engine use.

Following the previous, again the observation space should be adequate to make it possible for the agent to create "logic" over a desired set of actions and should be normalized, preferably to a [-1,1] range. As its was for DQN, for this case, the observation space also consists of 9 input variables shown below.

$$InputStates = \begin{bmatrix} SEnorm \\ SEdotnorm \\ errorSEnorm \\ errorSEdotnorm \\ SOCnorm \\ errorSOCnorm \\ mnoxnorm \\ errormnoxnorm \\ totnoxnorm \end{bmatrix} \tag{5.4.1}$$

All are normalised under the formula 4.3.5 errormnox is defined as

$$errormnox = mnoxtrack - mnox \tag{5.4.2}$$

the tanh function is used as previous.

NOx, Speed and Utice-Utel correlation

According to the model 3.2.9, and solving the 3.2.5 for $dSE/dt = 0$, the relation between Speed Utice and NOx emission production $mnox(g/s)$ can be plotted in a diagram to map the changes in these three elements accordingly.5.11 Here we can observe each graph a specific value of the diesel torque command Utice generates and relates the Speed with the NOx. Its fairly easy to observe that NOx production is exploding in low RPMs, something that drastically changes after a specific value of Speed at around 1050 RPMs. Following that observation we can define the *mnoxtrack* in the algorithm below and which is also represented in dashed red color in the previous figure 5.11.

Algorithm 6: Pseudocode for mnoxtrack.

```
dt = 0.2
while timesteps not finished do
    Import the SE values.
    mnoxtrack = (0.14−0.00)∗(min(max((SE−1050), 0), 100)−0)/(100−0)+0.00
end
```

Now the reference value of mnox can be easily altered to obtain a relation between Utice-Utel trade off in to which Speed values the NOx productions shall be battled with the battery use etc.5.12

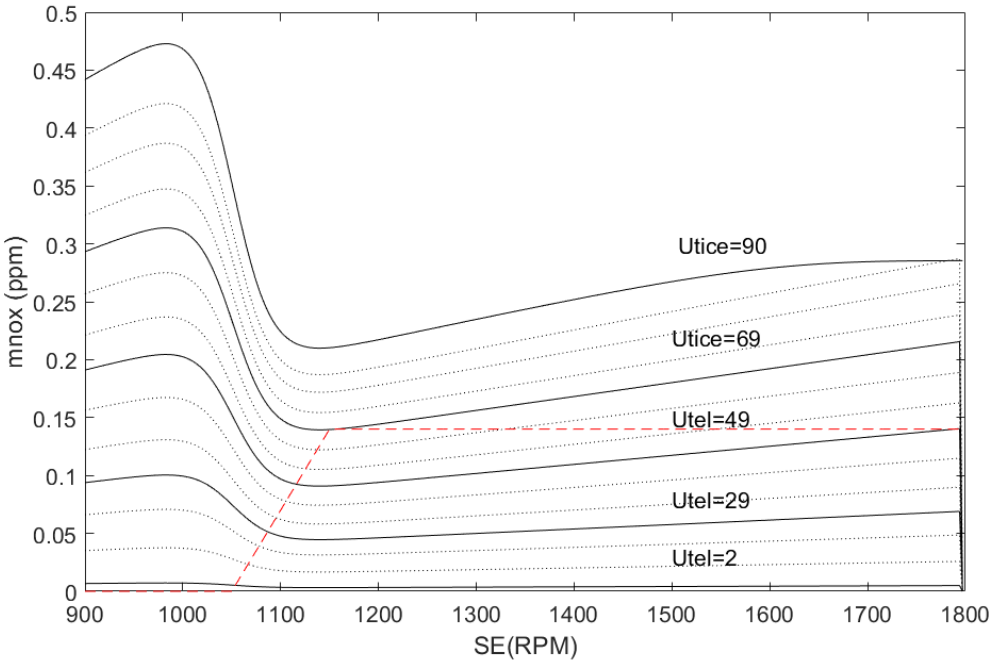

Figure 5.11: Speed Utice and NOx emission production.

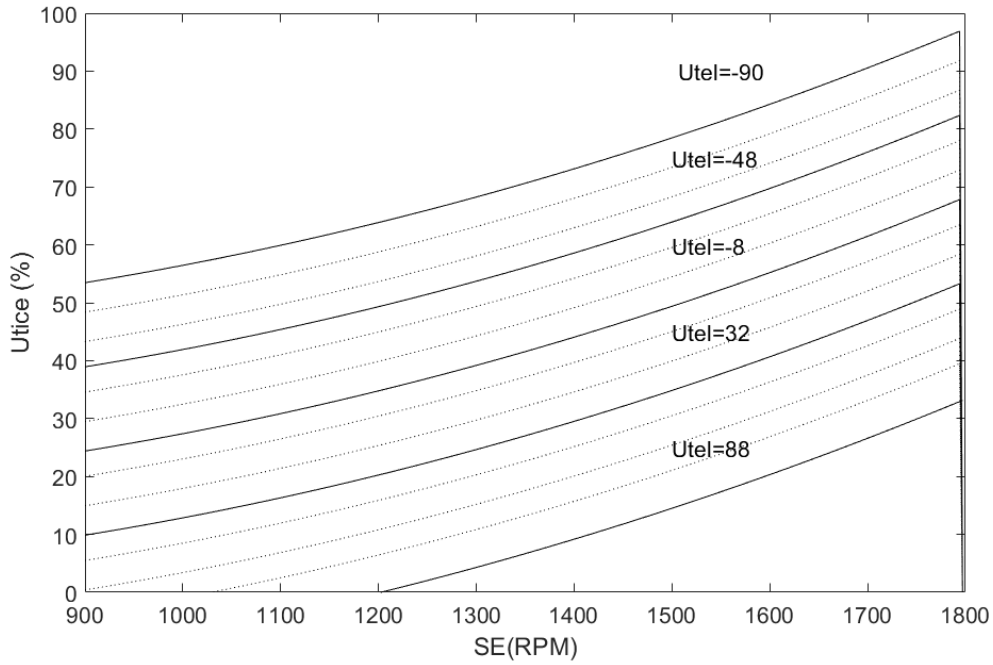

Figure 5.12: Possible Utice and Utel couples on every Speed value.

The cost function is set as follows 5.4.3

$$J = K1 * (errorSEnorm)^2 + K2 * (errorSEdotnorm)^2 + K3 * (errorSOCnorm)^2 + K4 * (errormnoxnorm)^2 \tag{5.4.3}$$

where K1=K3=K4=1 and K2=0.001 according to the weight "importance" we need to

assign to each state.

So the reward function for every iteration is defined 5.4.4

$$reward = \frac{K1 + K2 + K3 + K4}{2} - \sqrt{J} \tag{5.4.4}$$

Last, we define our termination condition, based on elements of the performance. Here this condition is set on the errorSE which should be kept in a range minor to 100 and errorSOC which should be kept in a range minor to 30, otherwise the episode is terminated earlier than the defined 900 steps of 0.2 sec each, or else 120 seconds episode trial.

5.4.1 Training Case 2

For the training process we need to define the training scenario and the goal related reward function on the environment. For this case, we use a tracking control over a trajectory in the Speed variable (RPM), a tracking control over the NOx production mnox, and a "loose" control over the state of charge SOC for the battery to not surpass and error of 30 % over the reference of 50 value of 50 % , over a 120 second time range.

The algorithm that define the trajectory of SE is the same as previous with SOC following the steady reference.

Holding the training process for 500 episodes, with the specified reward function and termination condition we have the following results. First again, we evaluate the achievements in completing each episode in full steps, alongside the cost and reward function accumulation.5.135.14

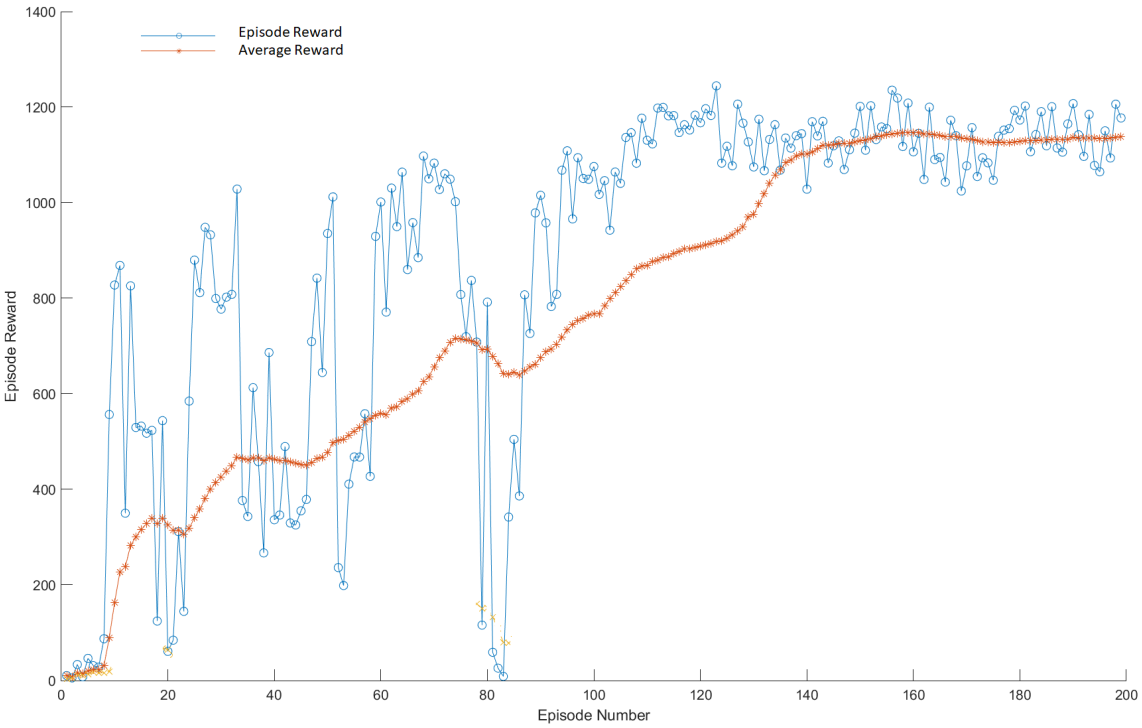

Figure 5.13: Reward function in Double action PPO NOx control.

What we can observe here is that the training process achieves again very early a full step state with the relative cost function and reward following a converging path to a min/max value and the convergence appears to be closer and closer to optimum.

Moving further, we can get a clearer view over the performance of the agent on the 490 to 500 episode sample range.**??**

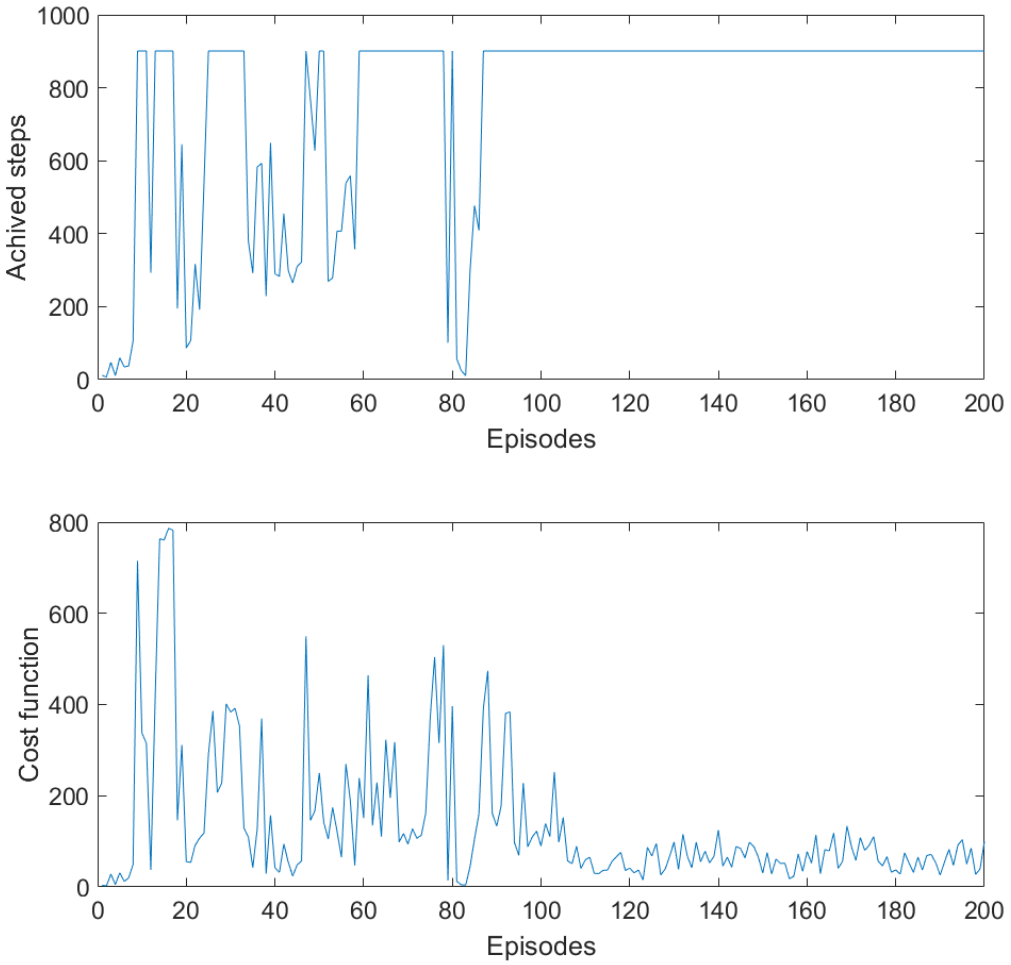

Figure 5.14: Episode achievements cost function in Double action NOx control.

As mentioned, the error difference SEtrack-SE and SOCtratck-SOC mainly contributes in the cost and reward function the bold line corresponds to the last episode. To have a better view on the scale of the error from the tracking value the last episode states to the reference are presented.5.16 & 5.17,5.18.

Here, we can observe a much better performance and deviation from the tracking value in contrast to the previous combination of PD control. Hence, the double action providing total control over the system is also considered as a better option for the optimal training. Last the variables u_{Tice} u_{Tel} that derive from the control variables of the agent action $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice}$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel}$ are presented below. 5.19

Since the training is complete and we are satisfied with the performance of the agent we can extract the trained agent and use it as a controller in scenarios relevant to the training ones. Simulation and trials will follow up in 6.

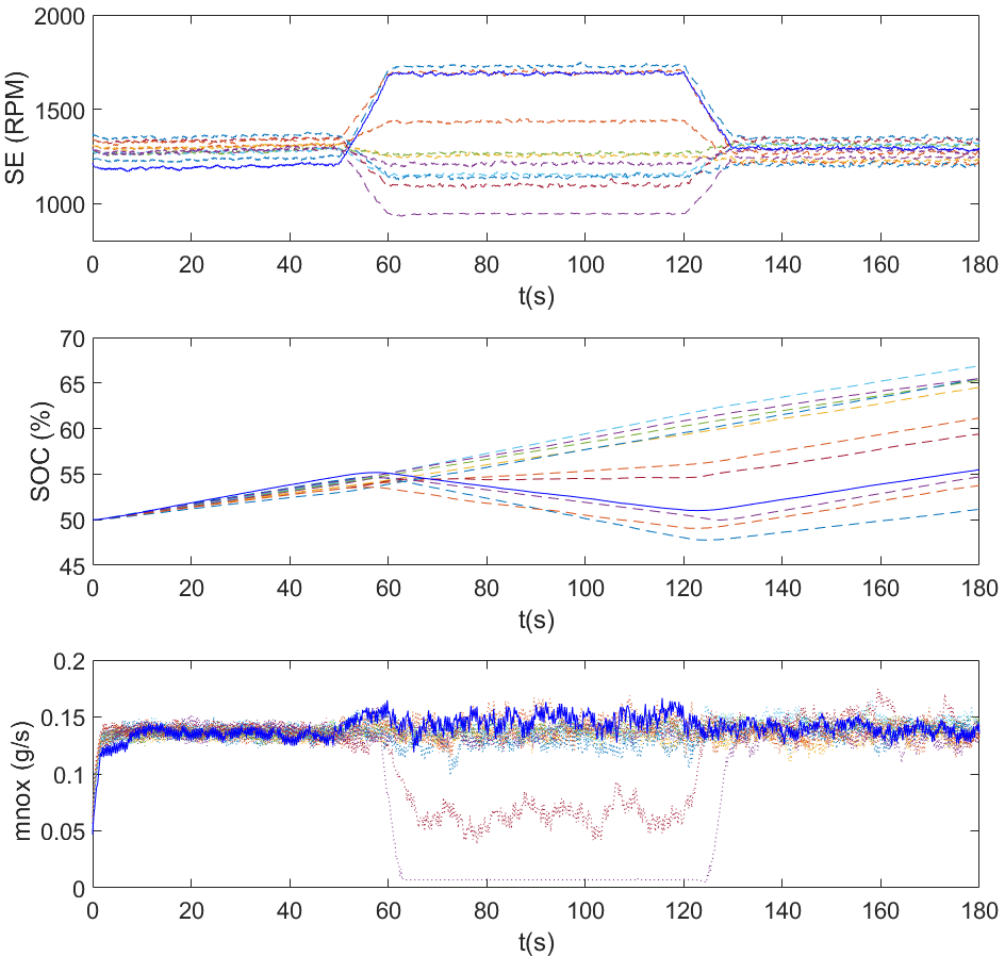

Figure 5.15: Double action control SE-SOC-NOx 10 last episodes PPO.

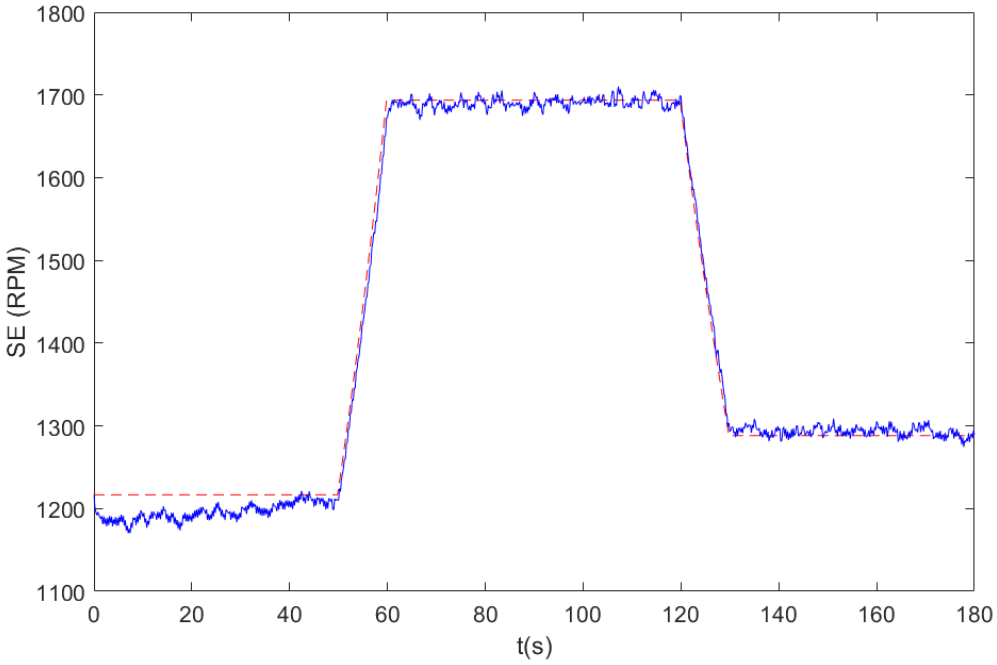

Figure 5.16: Double action NOx control SE-SEtrack last episode PPO.

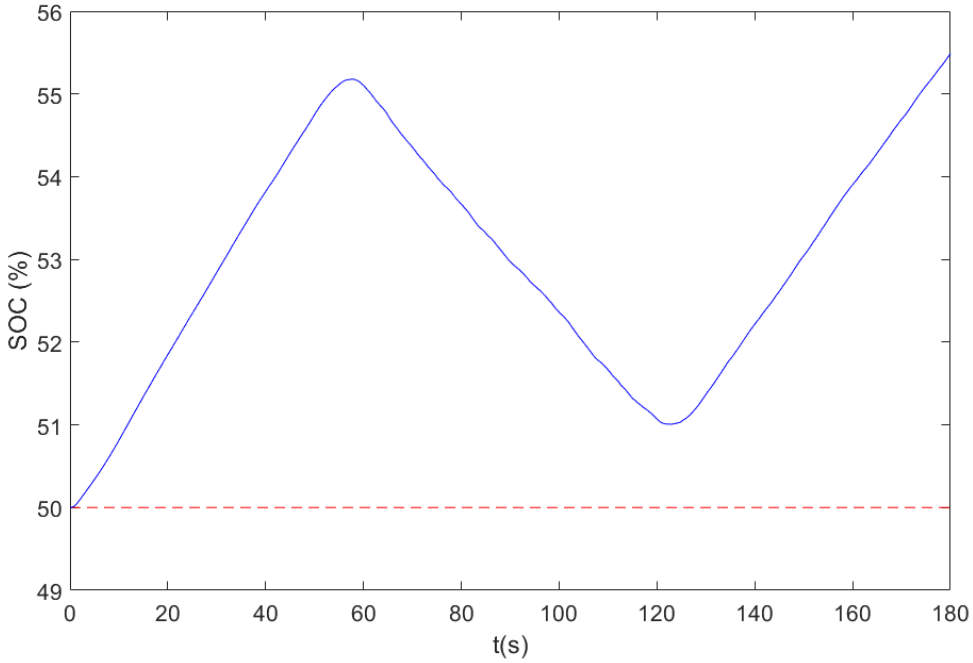

Figure 5.17: Double action NOx control SOC-SOCtrack last episode PPO.

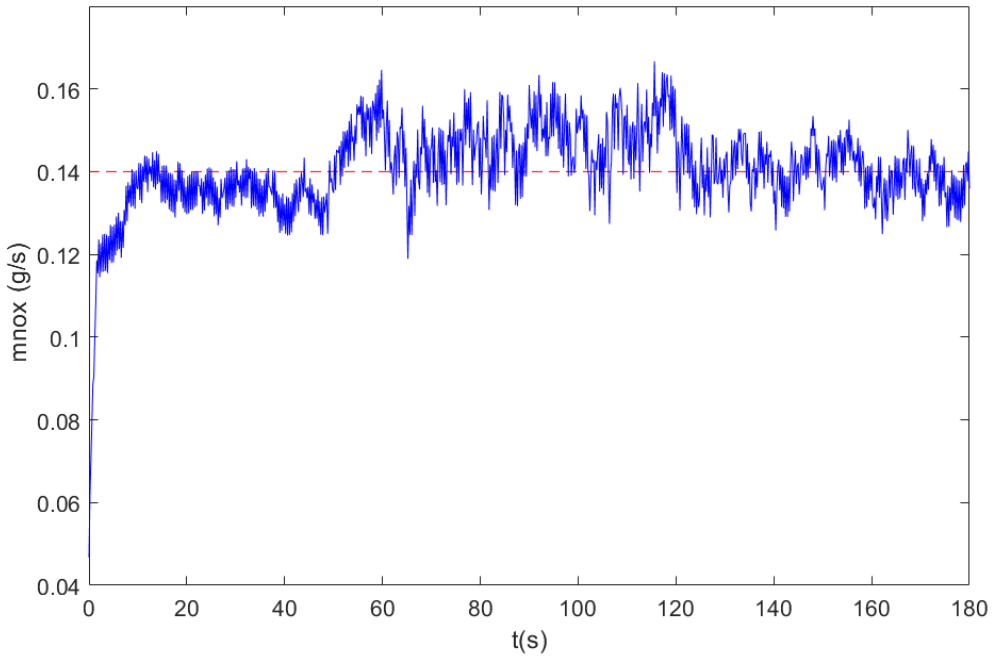

Figure 5.18: Double action control NOx-NOxtrack last episode PPO.

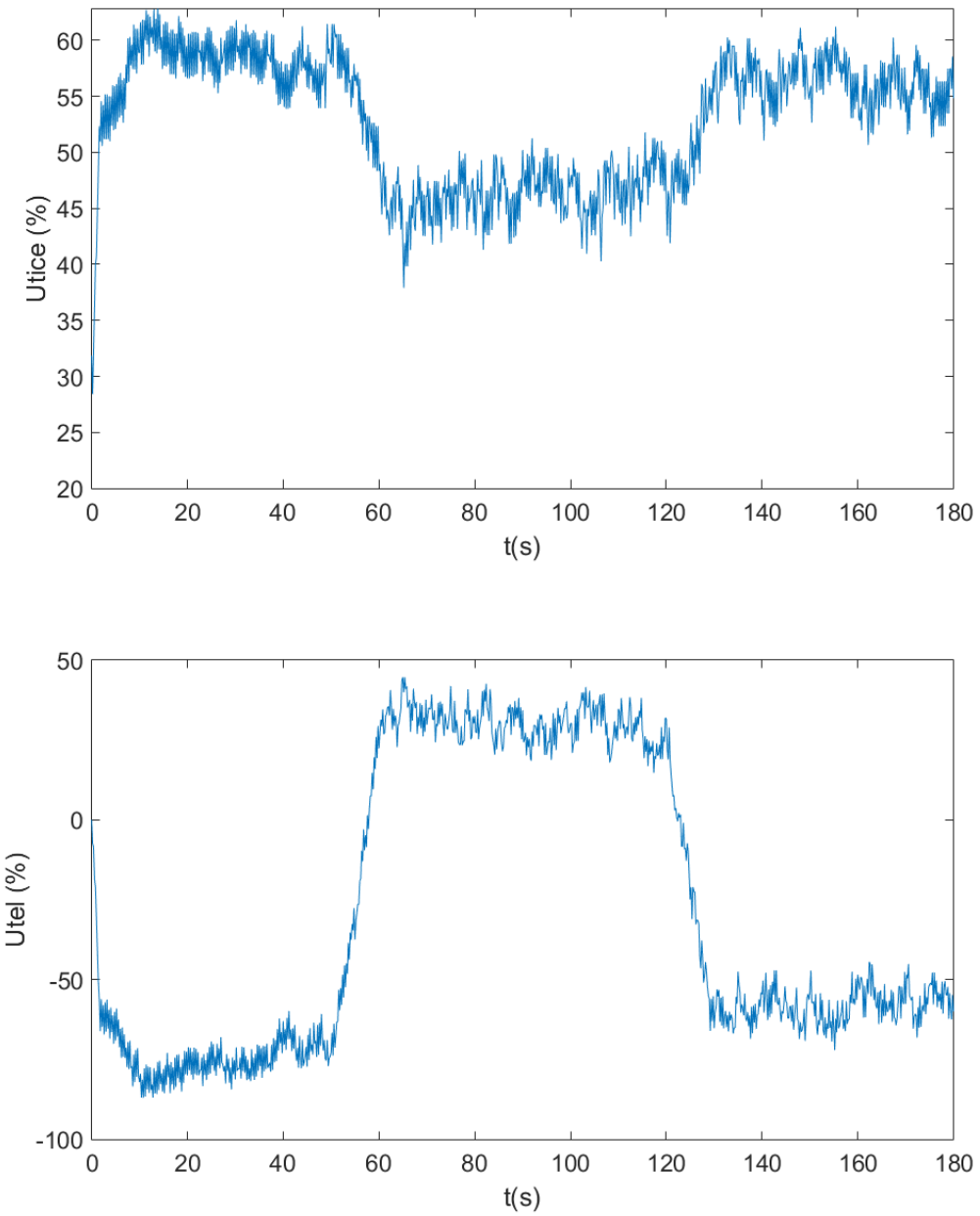

Figure 5.19: Double action NOx control u_Tice-u_Tel last episode PPO.

Chapter 6

Simulations and Results

In this chapter, trials and simulations on scenarios using the previously 4&5 trained agents, for the RL agent Deep Q Network (DQN) and in the Julia environment and the Proximal Policy Optimization (PPO) agent in the MATLAB environment which was applied are presented, compared and discussed. The presentation focuses on the problem formulation, and the particulars training which were implemented in order to create an agent able to perform as a controller in various situations. Furthermore,the generality of the controllers is evaluated and the limitations of each one examined. Last, simulation and conclusion over the different set-ups are presented.

6.1 Basic DQN and Julia set-up and PPO Matlab Simulation Case 1

To sum up, the formulation under of which the problem was modeled, in accordance to the previous, includes all the elements and the restrictions due to the physical and functional limits of the system. As mentioned, the problem formulation follows two main discrete cases that where modeled, here case 1 the speed (RPM) and state of charge (SOC) tracking control over defined trajectories, is ready to be tested over the two appropriately trained agents.

6.1.1 Basic DQN Simulation Case 1

The training scenarios were developed over a specified time of 60 sec episodes, in an approach to cover multi possible states and transitions over the SE and SOC. The most appropriate way of testing it is to simulate a cycle of changes of the states over an amount of time capable to demonstrate resilience and generality, alongside robustness over randomized disturbances. A 50 minutes cycle of state transitions used for this purpose, and is presented below.

In respect to the randomized disturbance a change in 3.2.4 constant c_{res} that corresponds to the potential resistance torque on the shaft, in relation to the Speed.6.1

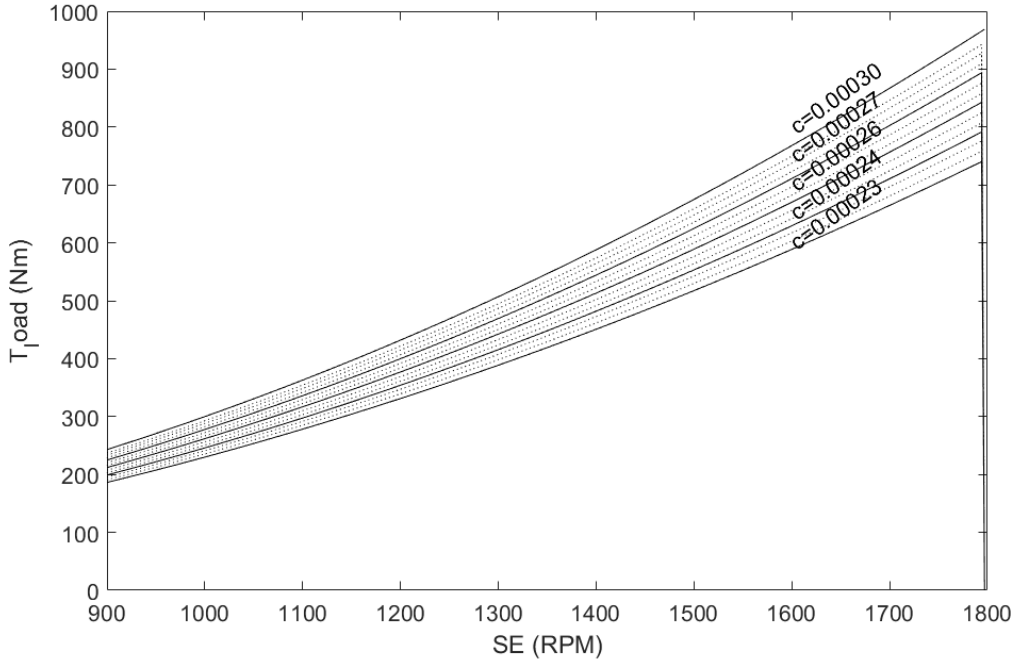

Figure 6.1: Relation between Tload and SE in different constant.

Algorithm 7: Pseudocode for SE and SOC tracking trajectory with disturbance.

randomized start for every episode
$SOCtrack = 50.0 - 10 * sin(5 * pi() * step/15000)$ reference of a long period sinusoidal form
$IntSE = 1100$ starting value
$c_{res} = 0.00025$ initial torque loading constant
while *episodes not finished* **do**
  calculate SE and SOC trajectory
  if *t less than 500 sec* **then**
    $c3 = 0.00025$
  else if *t equal to and greater than 900 sec* **then**
    $c3 = 0.00023 + (0.00030 - 0.00023)rand(1)$
  else if *t equal to and greater than 1100 sec* **then**
    $c3 = 0.00023 + (0.00030 - 0.00023)rand(1)$
  else if *t equal to and greater than 1400 sec* **then**
    $c3 = 0.00023 + (0.00030 - 0.00023)rand(1)$
  else if *t equal to and greater than 2100 sec* **then**
    $c3 = 0.00023 + (0.00030 - 0.00023)rand(1)$
  else if *t equal to and greater than 2500 sec* **then**
    $c3 = 0.00023 + (0.00030 - 0.00023)rand(1)$
  if *t less than 300 sec* **then**
    $SEtrack = IntSE$
  else if *t less than 500 sec* **then**
    $SEtrack = e^{log(1300/1100)\frac{t-10}{16-10} - log(1300/1100)}1300$
  else if *t less than 900 sec* **then**
    $SEtrack = 1300$
  else if *t less than 1100 sec* **then**
    $SEtrack = e^{log(1700/1300)\frac{t-40}{46-40} - log(1700/1300)}1700$
  else if *t less than 1300 sec* **then**
    $SEtrack = 1700$
  else if *t less than 1400 sec* **then**
    $SEtrack = e^{log(900/1700)\frac{t-40}{46-40} - log(900/1700)}900$
  else if *t less than 2000 sec* **then**
    $SEtrack = 900$
  else if *t less than 2100 sec* **then**
    $SEtrack = e^{log(1200/900)\frac{t-40}{46-40} - log(1200/900)}1200$
  else if *t less than 2400 sec* **then**
    $SEtrack = 1200$
  else if *t less than 2500 sec* **then**
    $SEtrack = e^{log(1600/1200)\frac{t-40}{46-40} - log(1600/1200)}1600$
  else
    $SEtrack = 1600$
  end
  $T_{load} = cresSE^2$
end

Holding the simulation process for 5 episodes, with the specified reward function and termination condition from the training we have the following results. First, we can evaluate the achievements in completing each episode in full steps, alongside the cost and

reward function accumulation.6.2

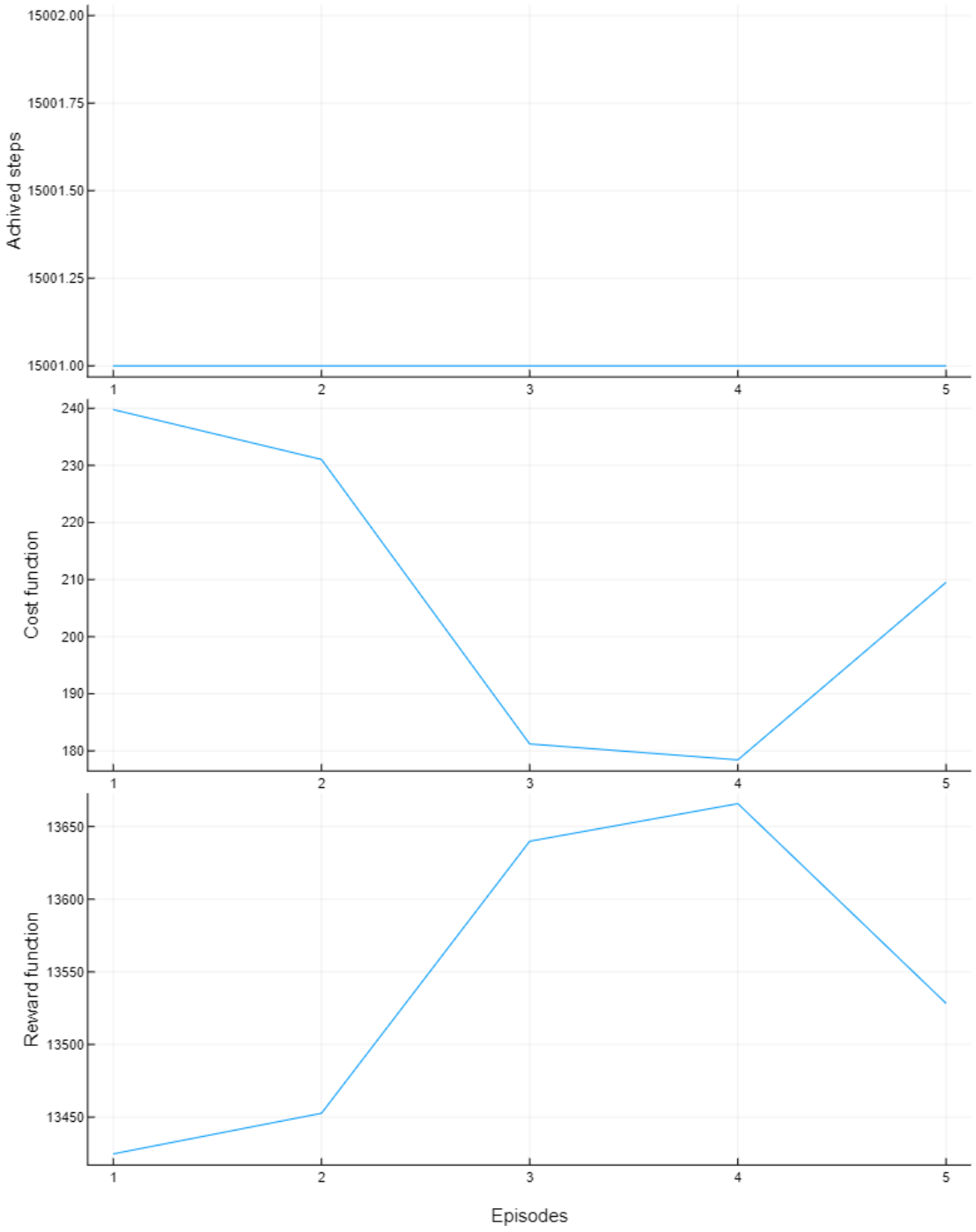

Figure 6.2: Episode achievements cost function and reward in Double action DQN.

What we can observe here is that the simulation process achieves a full step state and indicates very good repeatability over the 5 episodes, with the relative cost function and reward following a converging path to a min/max value and the convergence appears to be close to optimum.

Moving further, we can get a clearer view over the performance of the agent on 5 simulation episodes.6.3 As we can observe here, the simulation results are almost identical to each other. The error difference SEtrack-SE and SOCtratck-SOC mainly contributes

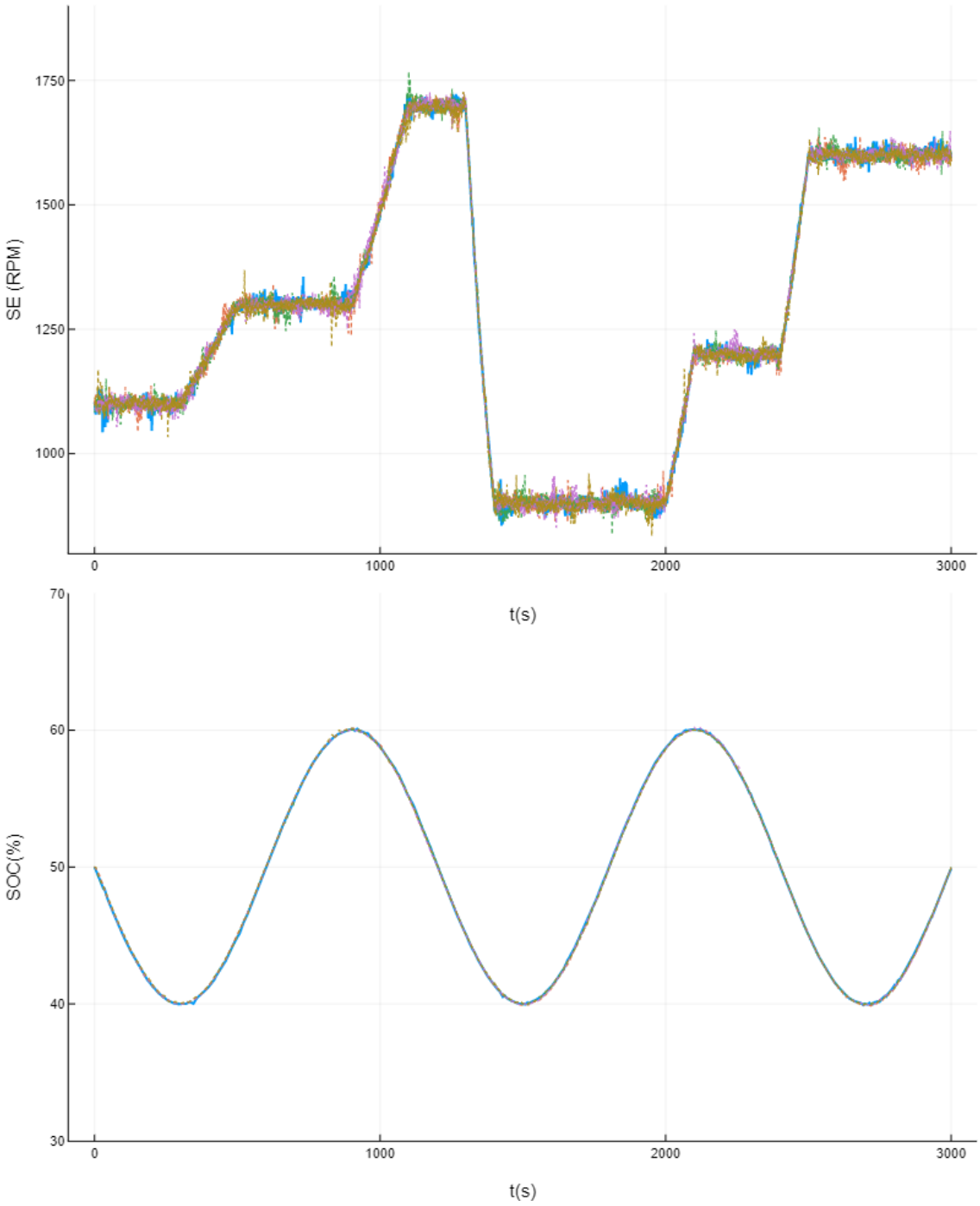

Figure 6.3: Double action control simulation SE-SOC 5 last episodes DQN .

in the cost and reward function the bold line corresponds to the last episode. To have a better view on the scale of the error from the tracking value the last episode states to the reference are presented.6.4 & 6.5 Here, we can observe a very good performance over the Speed tracking and a small but minor deviation from the high points of SOC tracking value.

Last the variables u_{Tice} u_{Tel} that derive from the control variables of the agent action $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice}$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel}$ are presented below. 6.6

It is important to highlight that the disturbance is an element that the agent was not

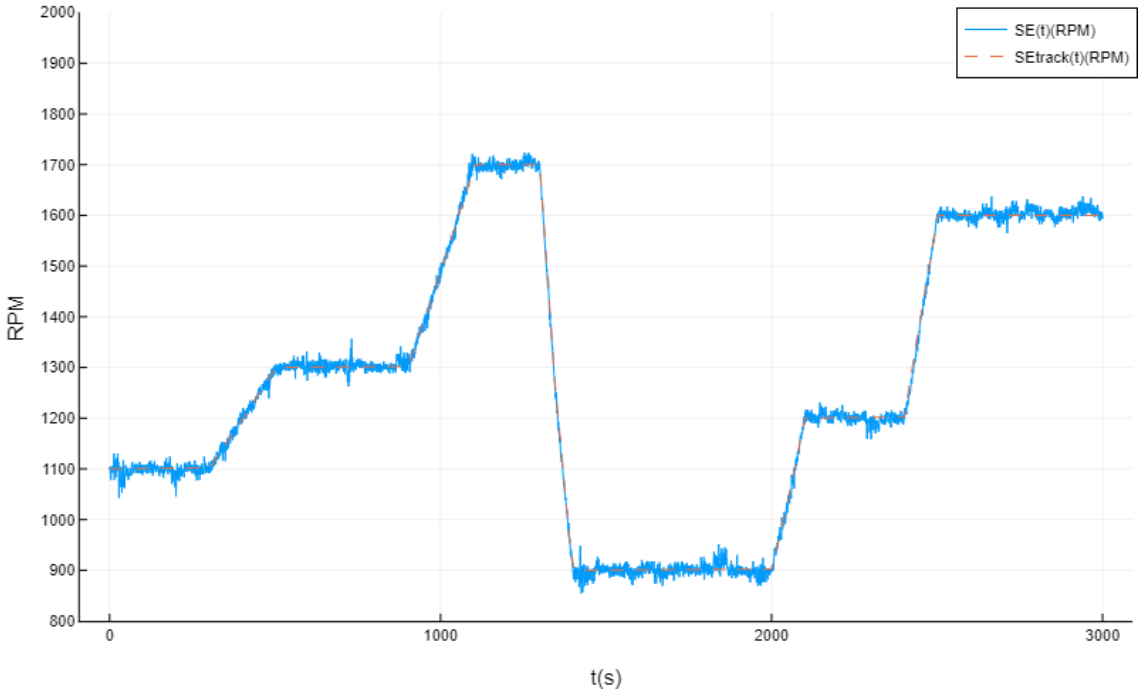

Figure 6.4: Double action control simulation SE-SEtrack last episode DQN.

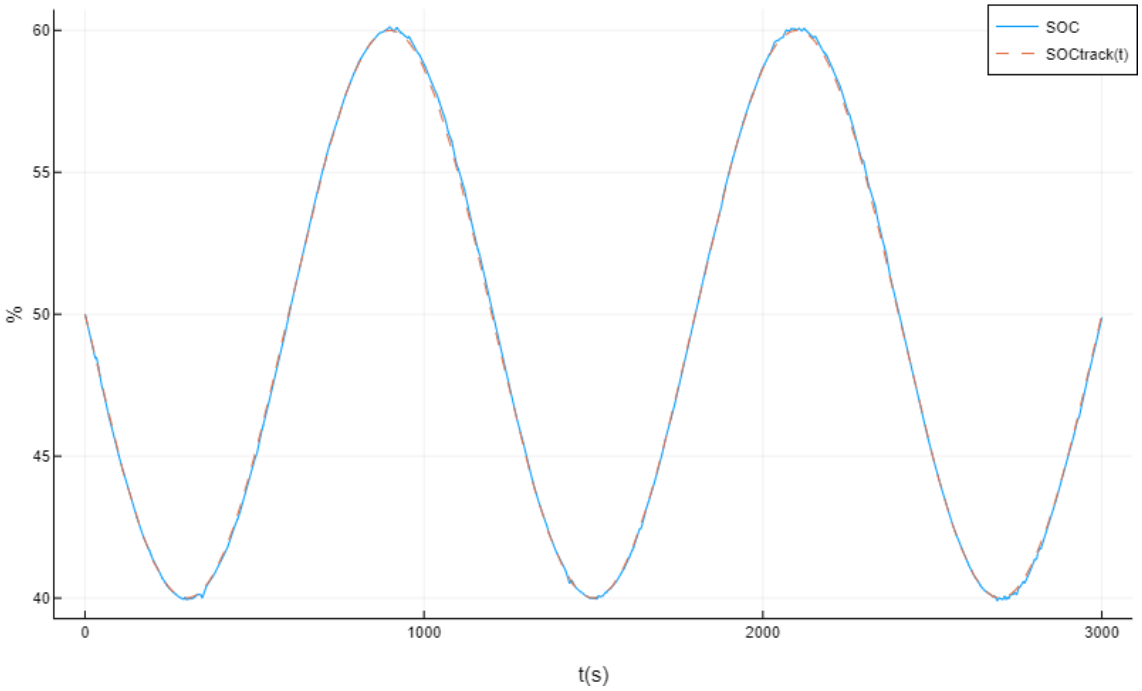

Figure 6.5: Double action control simulation SOC-SOCtrack last episode DQN.

trained to combat in the training process, but its shown exceptional results facing it. Since the training is considered complete and we are satisfied with the performance of the agent we can extract the trained agent and use it as a controller in scenarios relevant to the training ones, and also test the performance in potential real life applications.

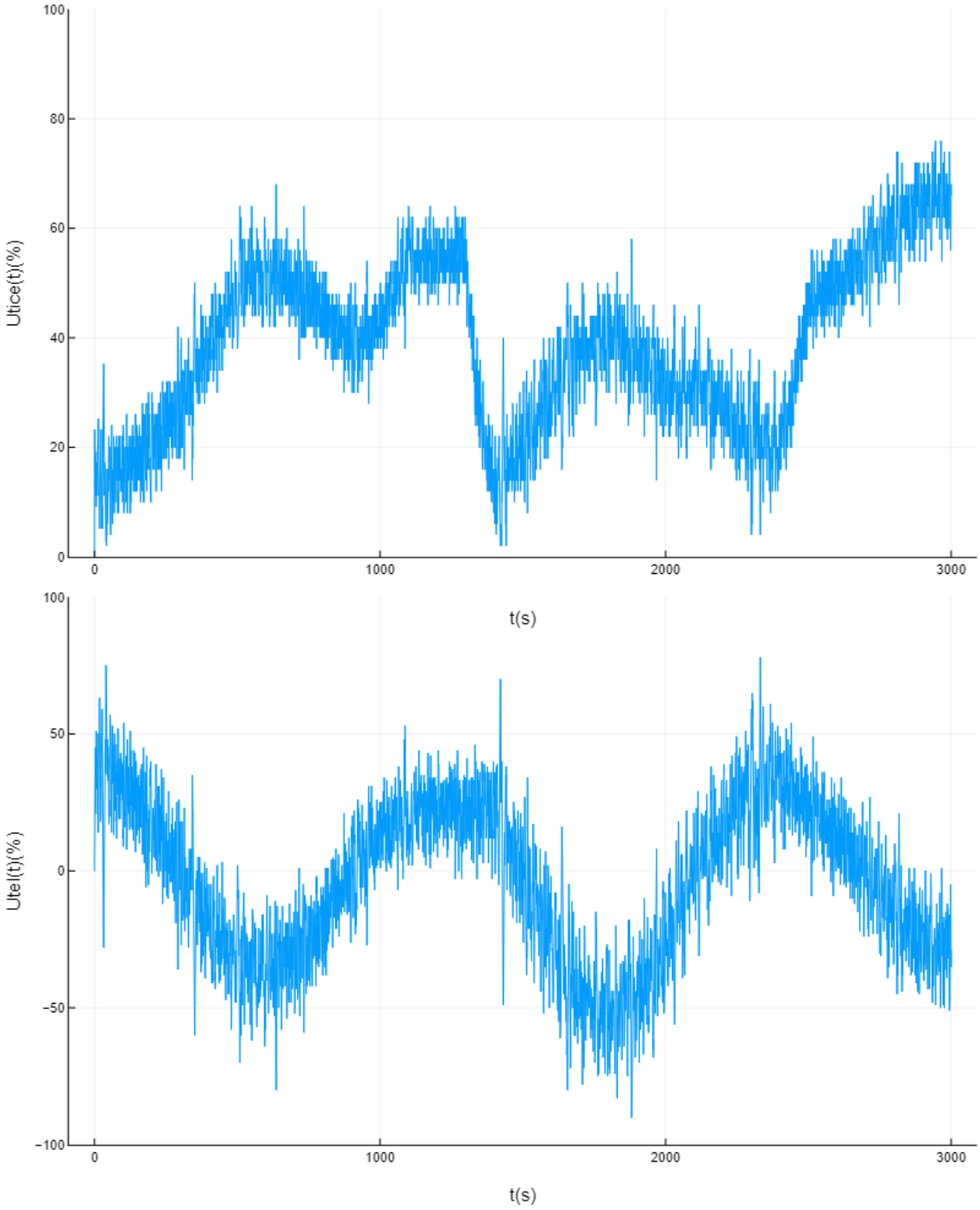

Figure 6.6: Double action control simulation u_Tice-u_Tel last episode DQN.

6.1.2 PPO Simulation Case 1

The training scenarios were developed over a specified time of 60 sec episodes, in an approach to cover multi possible states and transitions over the SE and SOC. the most appropriate way of testing it is to simulate a cycle of changes of the states over an amount of time capable to demonstrate resilience and generality, alongside robustness over randomized disturbances. The previous cycle was used for this purpose for comparison reasons.

Again, in respect to the randomized disturbance a change in 3.2.4 constant c_{res} that

corresponds to the potential resistance torque on the shaft, in relation to the Speed. 6.1

Holding the simulation process for 5 episodes, with the specified reward function and termination condition from the training we have the following results. First, we can evaluate the achievements in completing each episode in full steps, alongside the cost and reward function accumulation.6.76.8

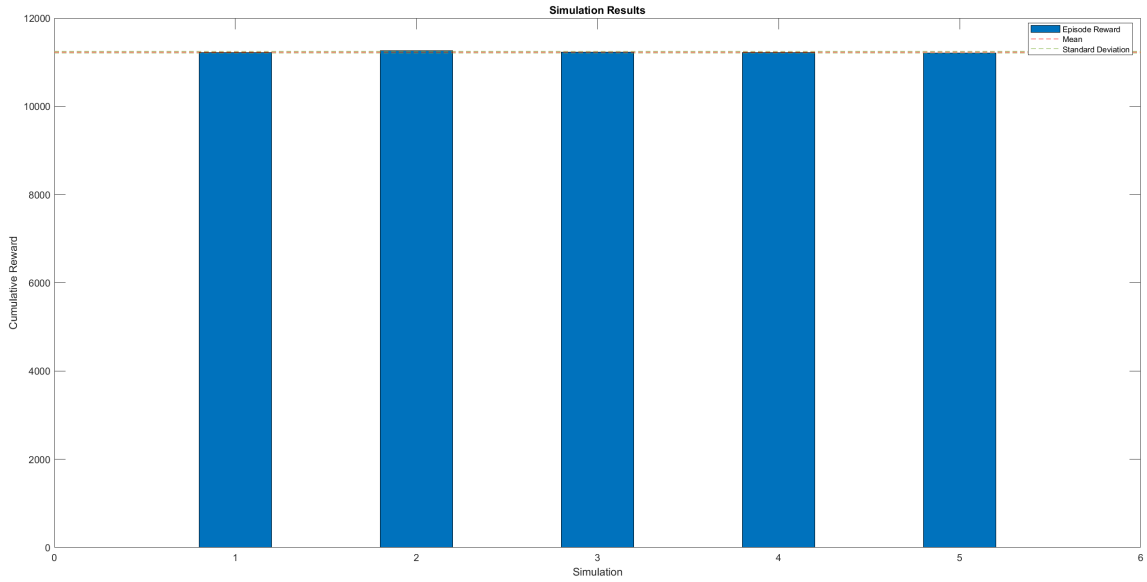

Figure 6.7: Rewards in Double action PPO Simulation for 5 episodes.

What we can observe here is that the simulation process achieves a full step state and indicates very good repeatability over the 5 episodes, with the relative cost function and reward following a converging path to a min/max value and the convergence appears to be close to optimum.

Moving further, we can get a clearer view over the performance of the agent on 5 episodes.6.9 As we can observe here, the simulation results are almost identical to each other. The error difference SEtrack-SE and SOCtratck-SOC mainly contributes in the cost and reward function the bold line corresponds to the last episode. To have a better view on the scale of the error from the tracking value the last episode states to the reference are presented.6.10 & 6.11 Here, we can observe a very good performance over the Speed tracking and a small but minor deviation from the high points of SOC tracking value. Last the variables u_{Tice} u_{Tel} that derive from the control variables of the agent action $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice}$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel}$ are presented below. 6.12

It is important to highlight here that although the training process was almost identical between the two agents DQN and PPO, the DQN agent is showing some better results on the SOC tracking control error against the PPO. This could indicate that the DQN gives better accuracy due to algorithmic simplicity, although PPO covers a greater range of solutions due to the continuous action space it provides. Here again we could say the agent has shown exceptional results facing the disturbances although it was not trained to do so. Since the training is considered complete and we are satisfied with the performance of the agent we can extract the trained agent and use it as a controller in scenarios relevant to the training ones, and also test the performance in potential real life applications.

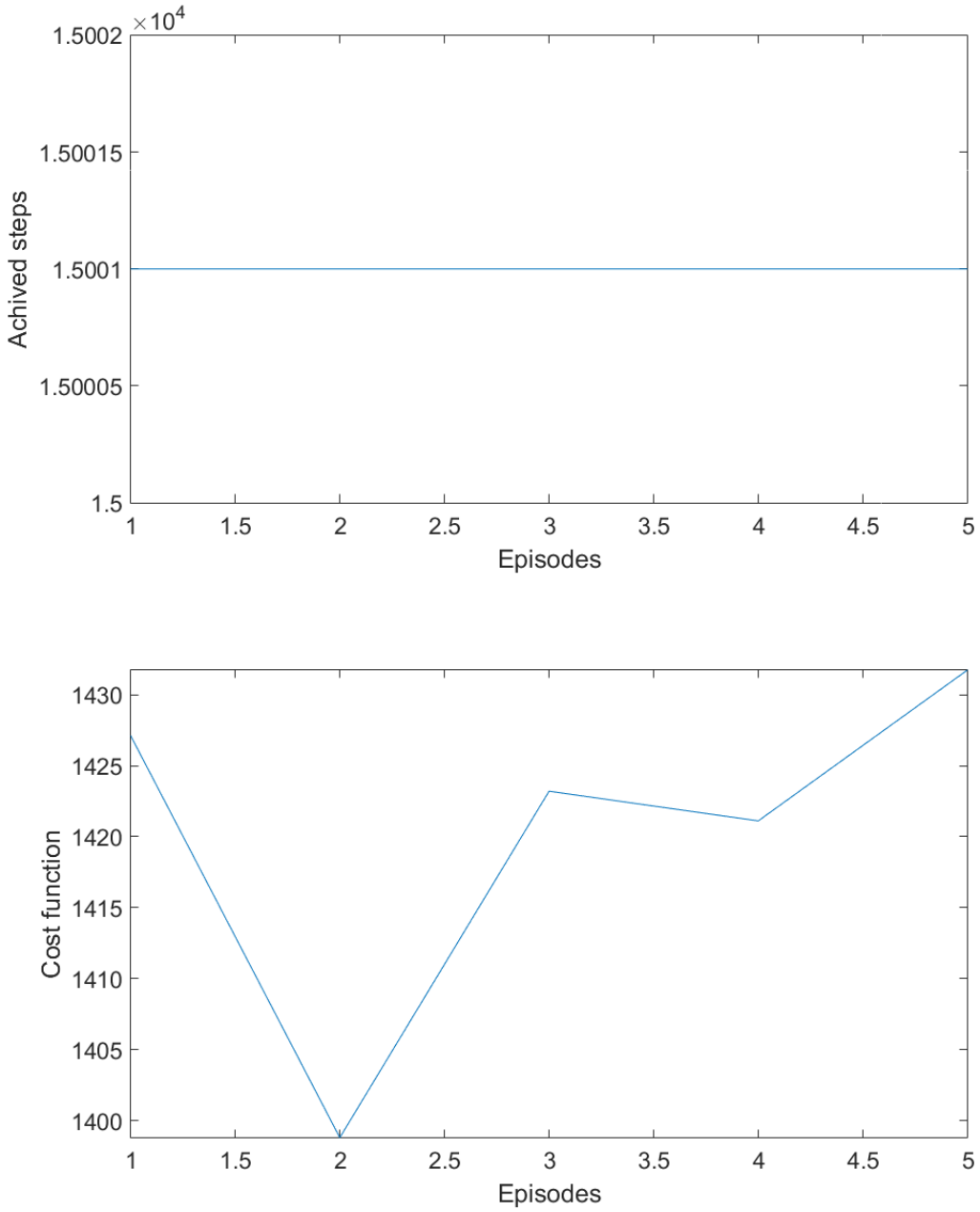

Figure 6.8: Episode achievements cost function in Double action PPO Simulation.

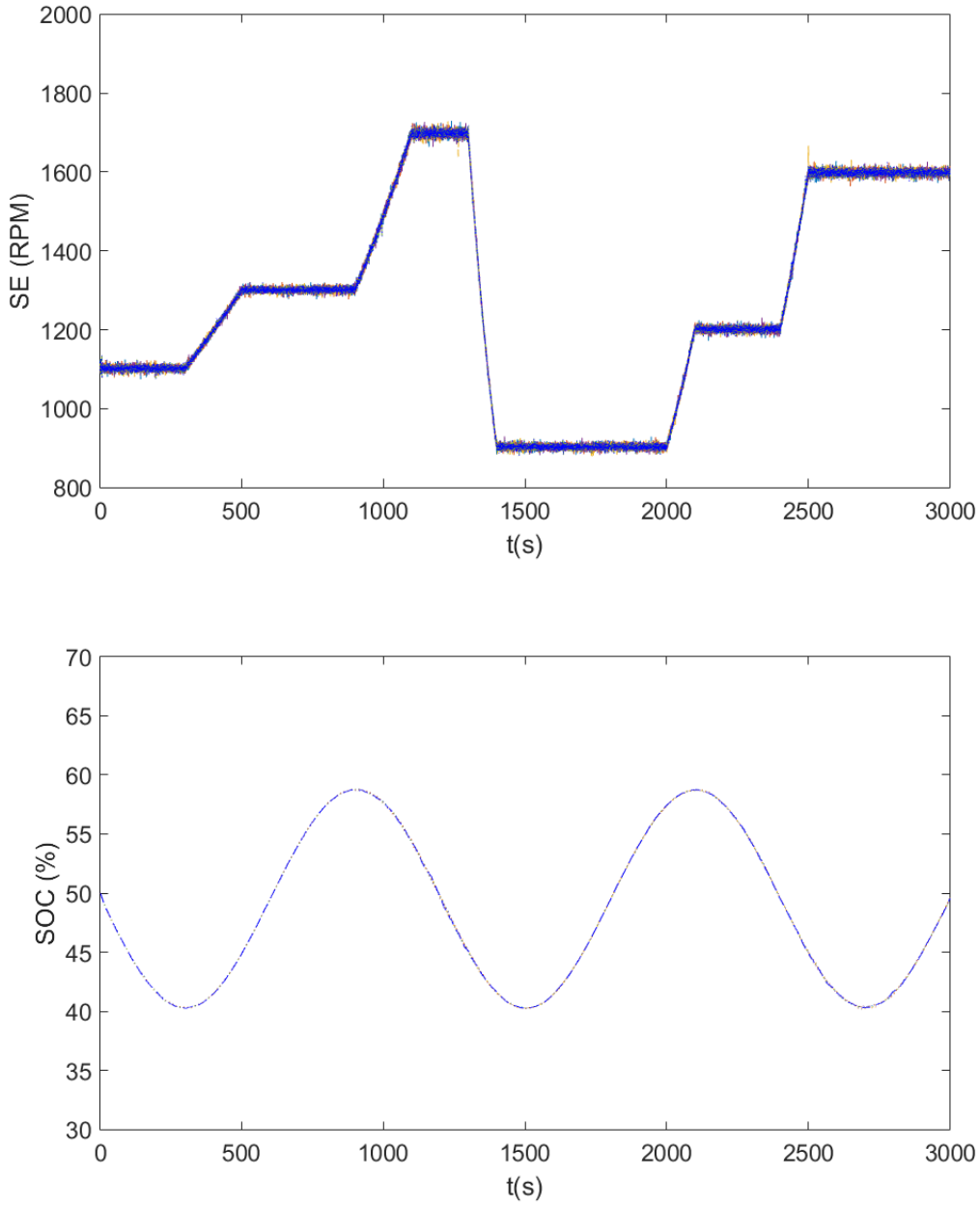

Figure 6.9: Double action control simulation SE-SOC 5 last episodes PPO.

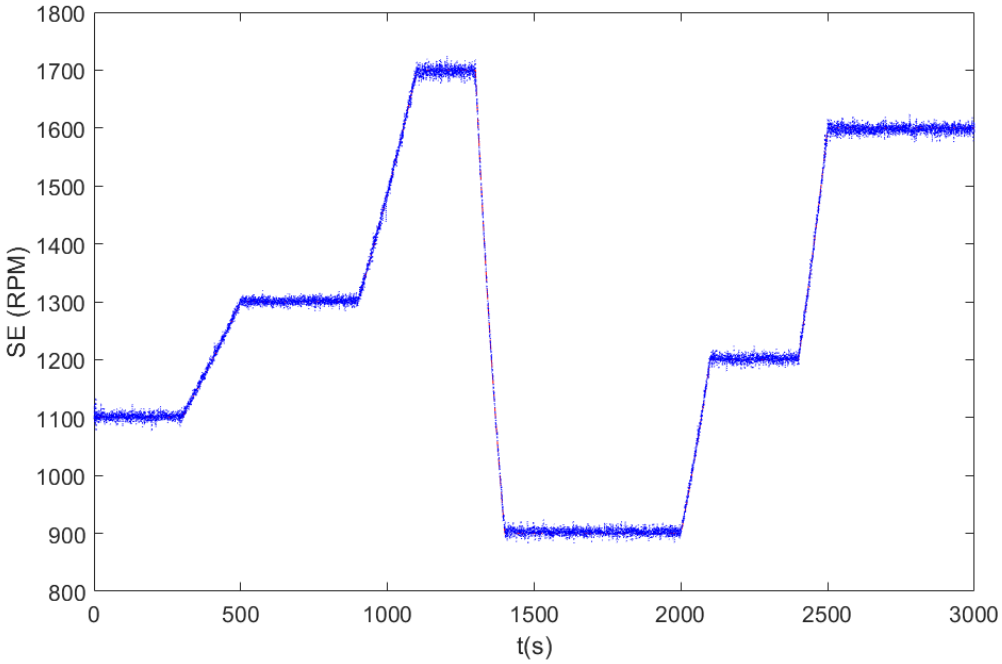

Figure 6.10: Double action control simulation SE-SEtrack last episode PPO.

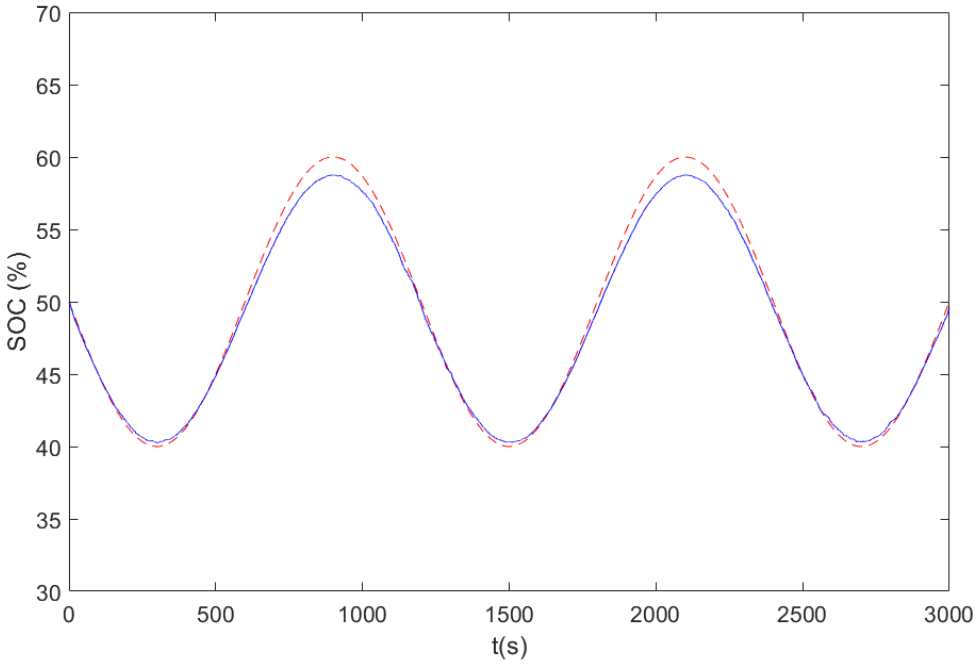

Figure 6.11: Double action control simulation SOC-SOCtrack last episode PPO.

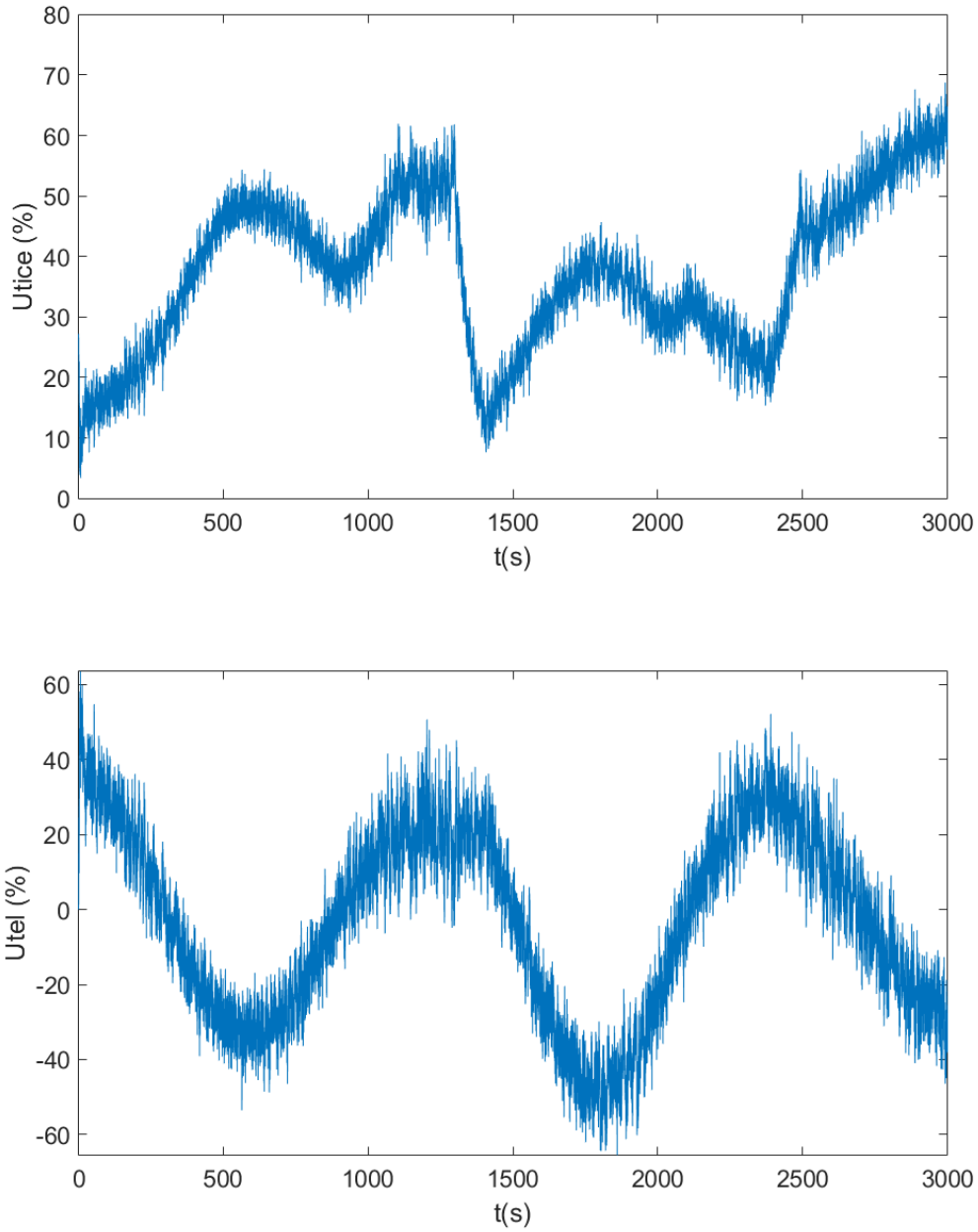

Figure 6.12: Double action control simulation u_Tice-u_Tel last episode PPO.

6.2 PPO Matlab Simulation Case 2

The training scenarios were developed over a specified time of 60 sec episodes, in an approach to cover multi possible states and transitions over the SE, SOC and NOx emissions. The most appropriate way of testing the concept of NOx reduction and the value of the trained agent, is to simulate a cycle of transition of the states over an amount of time capable to demonstrate resilience and generality, alongside robustness over randomized disturbances. The same 50 minute cycle as previous was used for this purpose, with the addition of the mnoxtrack that was developed in the training scenarios in 5.

In respect to the randomized disturbance, as previous a change in 3.2.4 constant c_{res} that corresponds to the potential resistance torque on the shaft, in relation to the Speed.6.1

Holding the simulation process for 5 episodes, with the specified reward function and termination condition from the training we have the following results.

First, we can evaluate the achievements in completing each episode in full steps, alongside the cost and reward function accumulation. 6.136.14

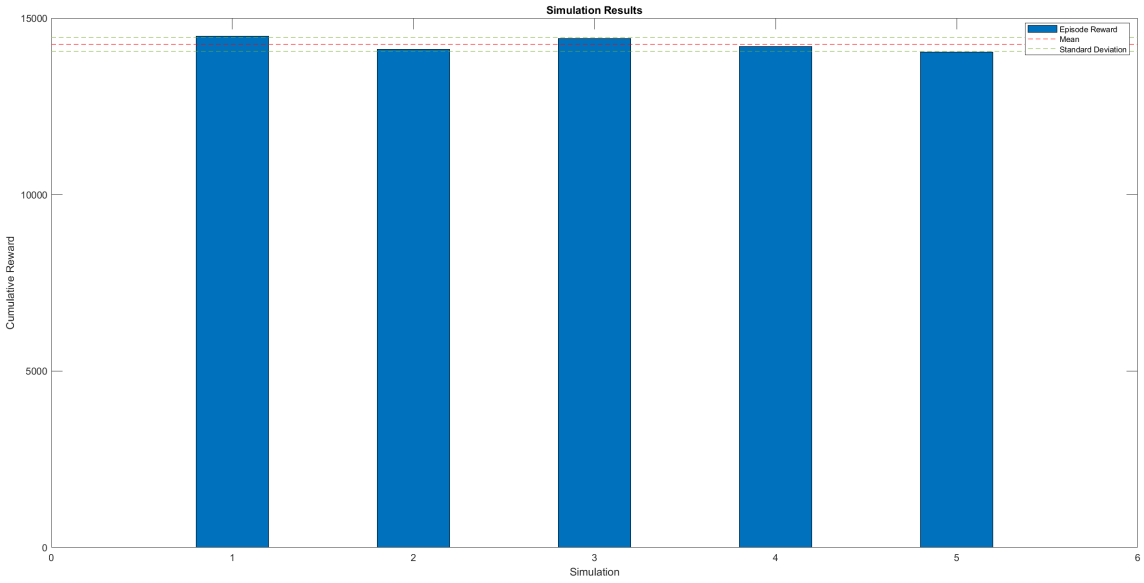

Figure 6.13: Rewards in Double action NOx control PPO Simulation for 5 episodes.

What we can observe here is that the simulation process achieves a full step state and indicates very good repeatability over the 5 episodes, with the relative cost function and reward following a converging path to a min/max value and the convergence appears to be close to optimum.

Moving further, we can get a clearer view over the performance of the agent on 5 episodes.6.15

As we can observe here, the simulation results are almost identical to each other. The error difference SEtrack-SE and SOCtratck-SOC mainly contributes in the cost and reward function. The bold line corresponds to the last episode. To have a better view on the scale of the error from the tracking value the last episode states to the reference are presented.6.16 & 6.17

Here, we can observe a very good performance over the Speed tracking and a small but minor deviation from the high points of SOC tracking value. Also, the variables u_{Tice} u_{Tel} that derive from the control variables of the agent action $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tice}$ $\frac{\mathrm{d}}{\mathrm{d}t}u_{Tel}$ are presented below. 6.18

Last, by approaching the goal that was set in case 2, the over all NOx emission pro-

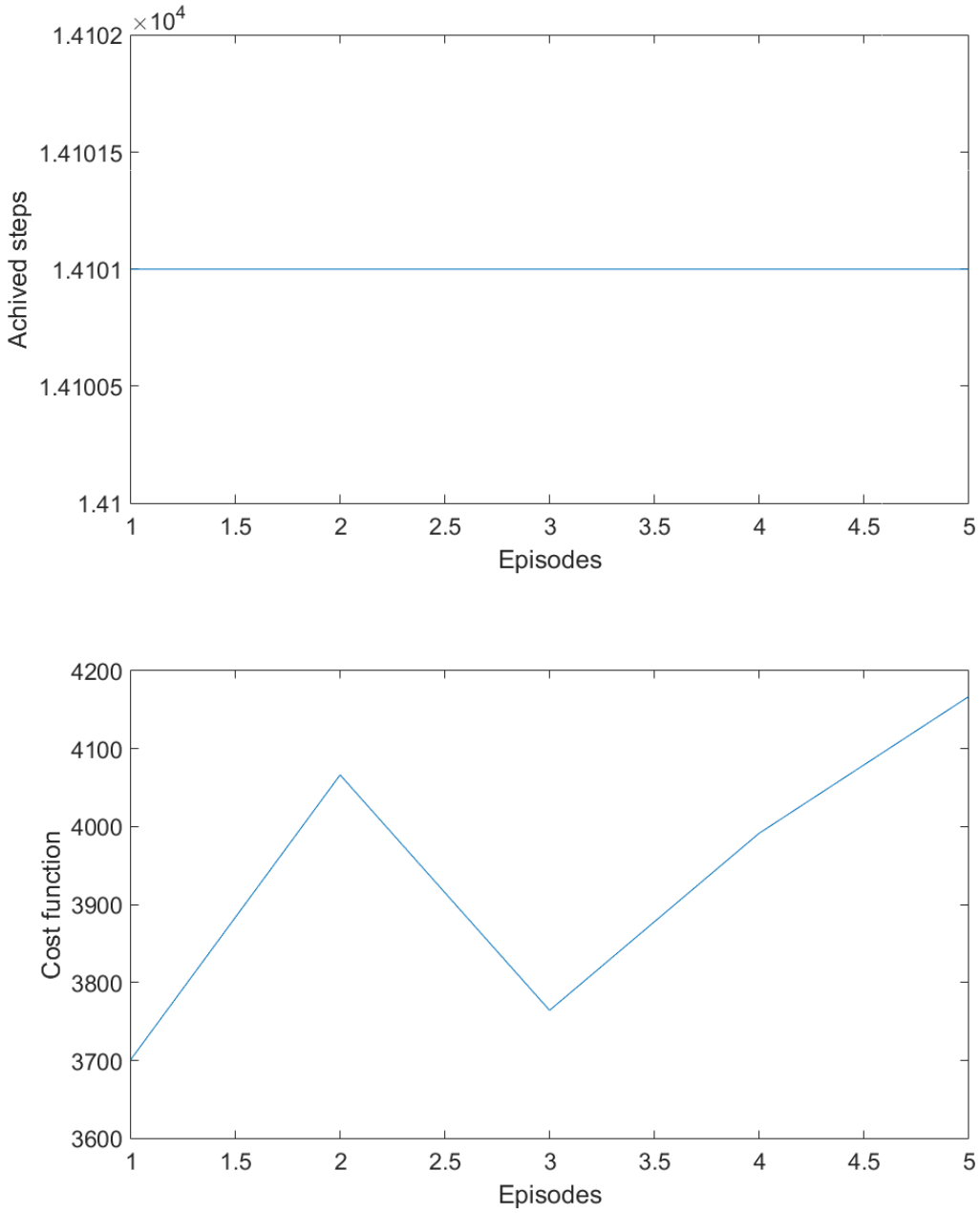

Figure 6.14: Simulation episode achievements and cost function Double action NOx control.

duction reduction, we have the following figure. 6.19

Here we notice a clear reduction of a 32.3 grams of NOx in the last cycle, in comparison with a single Diesel engine operation for the same 50 minute cycle.

The most important thing to mention here is that although the agent in the training process was trained to follow a tracking control in SE and NOx production, plus a "loose" control over a SOC reference, it was un avoidable to reach some physical battery charging limits. These were indicated to [25,75] and hard coded to be avoided when reached

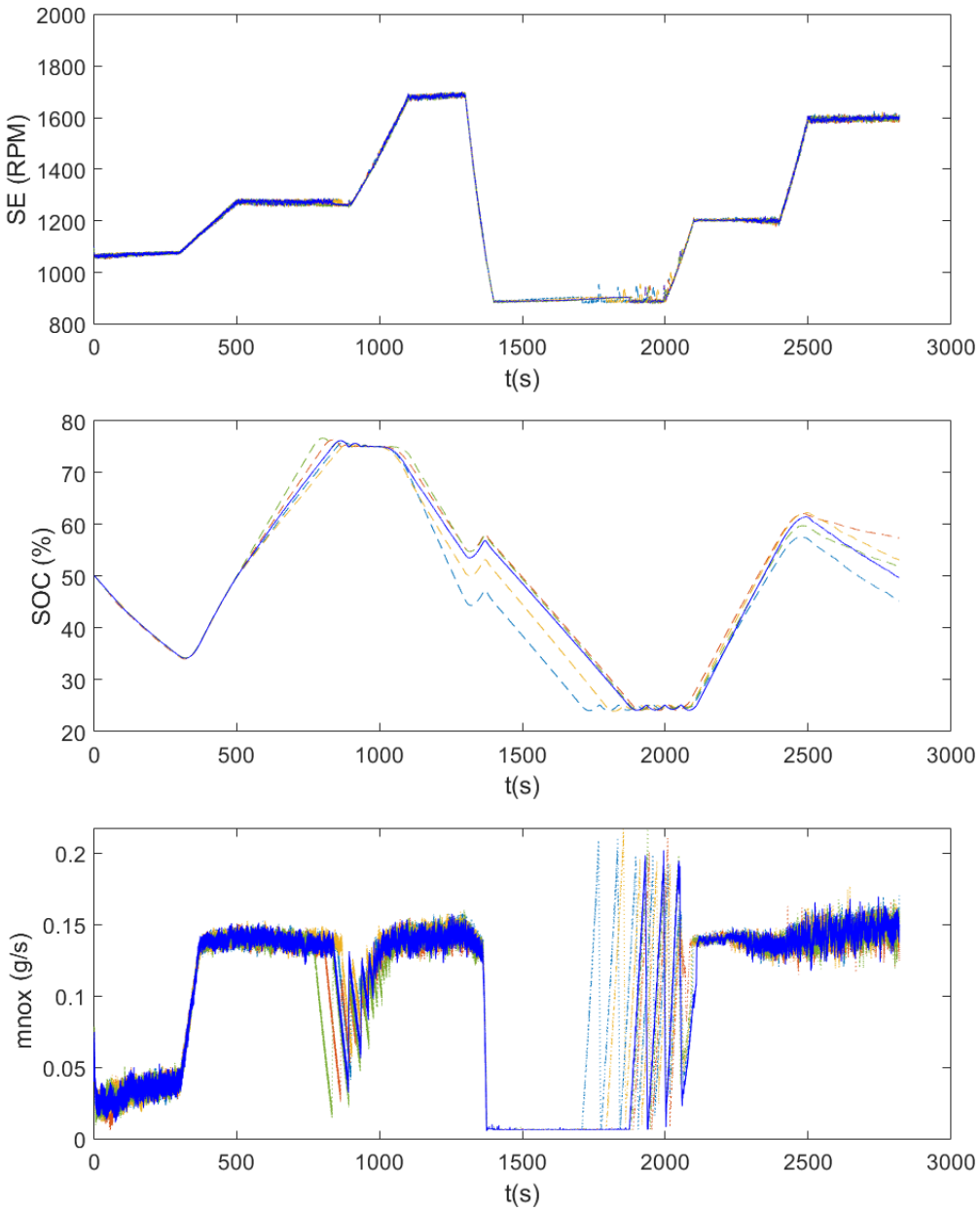

Figure 6.15: Double action NOx control simulation SE-SOC 5 last episodes PPO.

without "messing" the logic of the agent over them. Ofcourse a larger battery would be less prone to reaching those limits, but it is not totally avoidable in real life scenarios. In order for the agent to learn to avoid and act differently in an area reaching the limits a different training set up is required which was not covered for the purposes of this study. To conclude, here again we could say the agent has shown exceptional results facing the disturbances although it was not trained to do so. Since the training is considered complete and we are satisfied with the performance of the agent we can extract the trained agent and use it as a controller in scenarios relevant to the training ones, and also test the

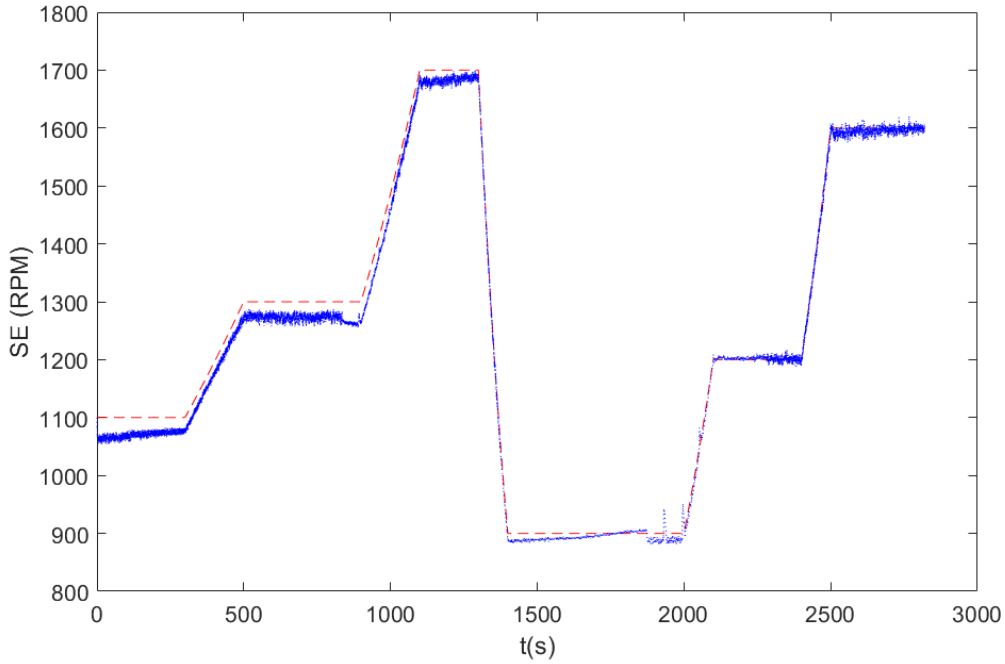

Figure 6.16: Double action NOx control simulation SE-SEtrack last episode PPO.

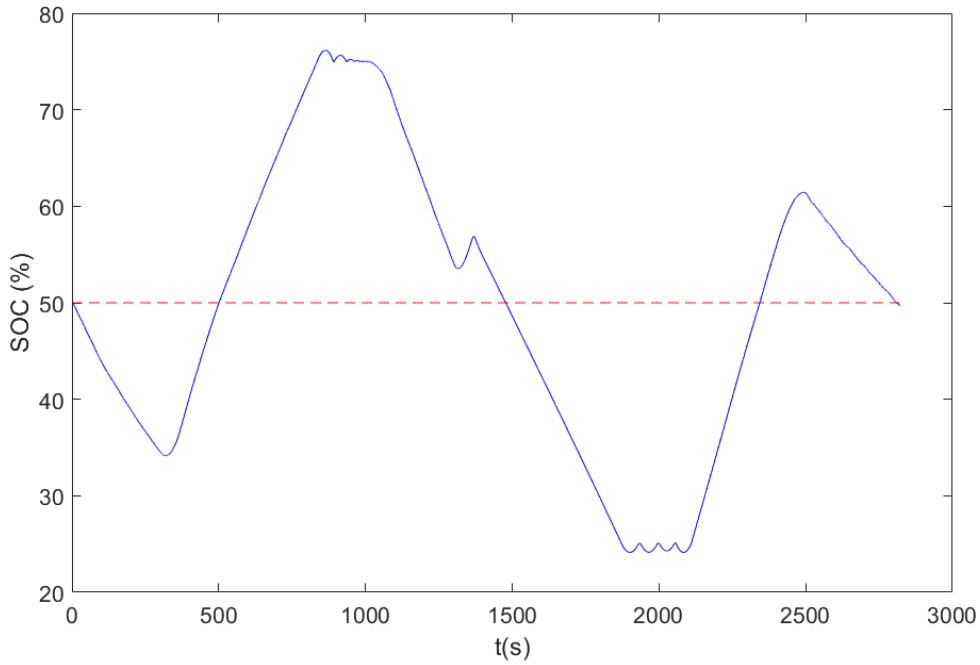

Figure 6.17: Double action NOx control simulation SOC-SOCtrack last episode PPO.

performance in potential real life applications.

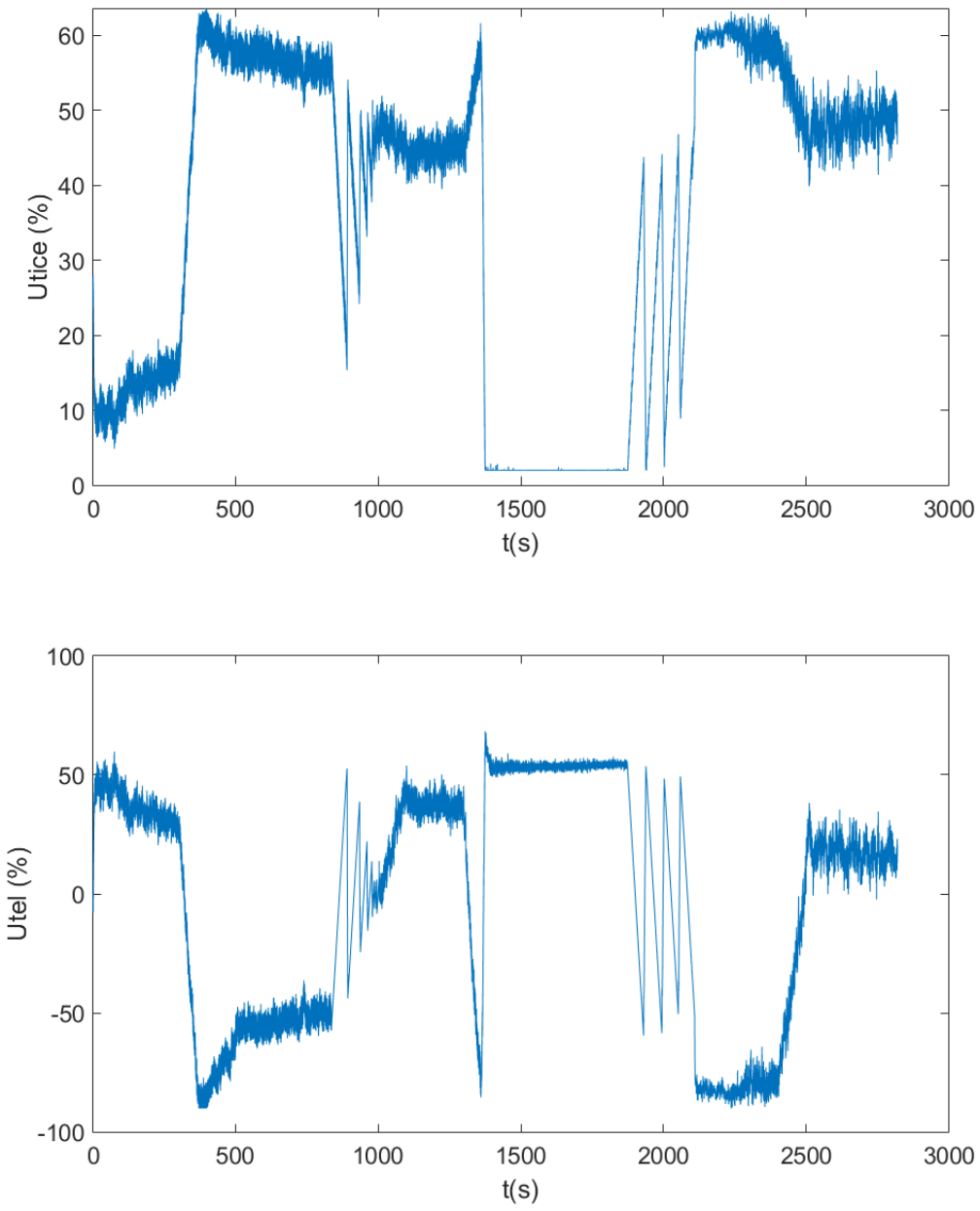

Figure 6.18: Double action NOx control simulation u_Tice-u_Tel last episode PPO.

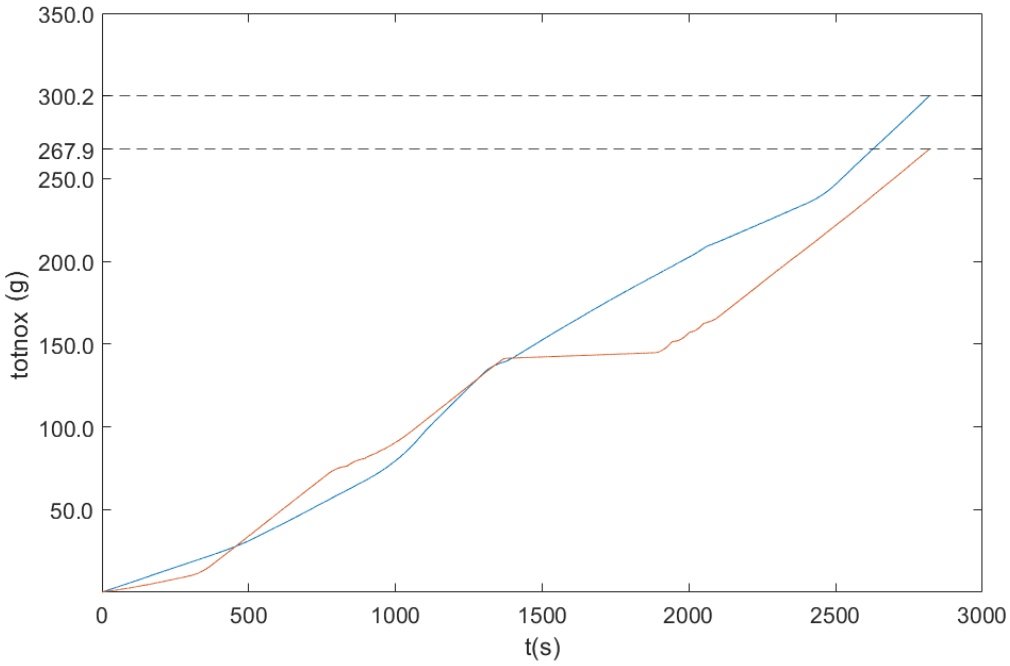

Figure 6.19: Double action control simulation NOx total cycle reduction last episode PPO.

Chapter 7

Conclusions and Future Work

This is the conclusion chapter, in which all the work is summarized. Furthermore, some suggestions over the RL design parameters, the plat model and goals, and the potential of further development is included.

7.1 Conclusions

In this work, the potential of using deep Reinforcement Learning control on nonlinear model for conducting efficient Diesel and Motor control over specified trajectories, for a hybrid diesel-electric marine power plant has been investigated.

Initially, polynomial differential and static models for the engine were used for the training process in different scenarios and validated via several simulations. Electric motor/generator was used to contribute in the parallel goals of Speed control and State of Charge of a battery. Furthermore, two kinds of agents DQN and PPO were developed and compared on their performance. As far as the RL design problem formulation and procedure concerned, two approaches were considered. The first approach, is referred as parallel engine-motor control a traditional PD controller and with a DQN agent, and its purpose was to regulate the EM torque in respect to the RPMs engine operation. The second, is referred as direct engine control, as both of the power sources of the plant are controlled directly from the double action RL agent. In the second scheme, manipulation of engine dynamics with a reference control over an emission model was also performed in order to reduce the NOx emissions.

The performance of the developed RL agent controllers were tested in simulations, using a hybrid diesel-electric simulation set-up, which was developed for this reason. The performance of controllers were evaluated in respect to the rewards, and step achievements, and the error from reference for each case. The training process in each case followed random scenarios of states transitions, with the aim to develop a "logic" on any trained agent capable to perform as a controller and show a level of generality. The simulations in each case were built over a predefined cycle scenario of state transitions with a random disturbance developed under realistic transient loads. Simulation results showed that RL is capable of controlling the power plant in good generality, in respect to the references, constraints and random disturbances which have been set. Moreover the emission scenario strategy which was implemented, led to significant reduction of NOx emissions.

Conclusively, RL's ability to handle nonlinear equations, constraints and multiple references makes it a powerful tool for efficient control of such plants.

7.2 Potential for Future Work

In this thesis, Rl control was employed with primary objective to track the speed reference and satisfy the implemented constraints. The considered objective was to manipulate the engine dynamic response via the electric motor.

It can be suggested that fuel reduction strategies could also be implemented in parallel. These strategies, are based on the load circle prediction, and optimization of power split via the equivalent fuel consumption. In this scheme, The performance of the developed agent-controllers could be tested experimentally on the diesel-electric testbed at LME, under realistic transient loads. Moreover the emission control strategy which was implemented, could be tested.

Furthermore, NOx static models could be constructed and integrated in the RL environments along with constraints regarding, the maximum allowed emissions (e.g. regulation limit). Different reward function designs are also suggested to explore a trade-off between a linear strict relation with a predefined state related cost function. This could allow a more generally defined training goal and result in greater generality independent of strictly defined error values.

Finally, the other RL agents could also be investigated further, and agents could be deployed for estimating other plant parameters which are not available or are tampered by disturbances, such as the turbocharger speed etc.

Bibliography

[1] V. Karystinos, "Nonlinear model predictive control of a hybrid diesel-electric marine propulsion plant," 2019.

[2] S. Singh, P. Norvig, and D. Cohn, "Agents and reinforcement learning," *Dr. Dobb's Journal of Software Tools for Professional Programmer*, vol. 22, no. 3, p. 3, 1997.

[3] R. R. Bush and F. Mosteller, "Stochastic models for learning.," 1955.

[4] R. Geertsma, R. Negenborn, K. Visser, and J. Hopman, "Design and control of hybrid power and propulsion systems for smart ships: A review of developments," *Applied Energy*, vol. 194, pp. 30 – 54, 2017.

[5] R. A. of Engineering, *Future Ship Powering Options Exploring alternative methods of ship propulsion.* IMO, 2013.

[6] G. Sulligoi, S. Castellan, M. Aizza, D. Bosich, L. Piva, and G. Lipardi, "Active front-end for shaft power generation and voltage control in fremm frigates integrated power system: Modeling and validation," in *International Symposium on Power Electronics Power Electronics, Electrical Drives, Automation and Motion*, pp. 452–457, June 2012.

[7] S. K. Topaloglou, G. Papalambrou, K. Bardis, and N. Kyrtatos, "Transient load share management of a diesel electric hybrid powertrain for ship propulsion," *International Journal of Powertrains*, vol. 7, p. 341, 01 2018.

[8] N. Planakis, G. Papalambrou, and N. Kyrtatos, "Predictive control for a marine hybrid diesel-electric plant during transient operation," pp. 989–994, 04 2018.

[9] J. Hofstetter, H. Bauer, W. Li, and G. Wachtmeister, "Energy and emission management of hybrid electric vehicles using reinforcement learning," *IFAC-PapersOnLine*, vol. 52, no. 29, pp. 19–24, 2019.

[10] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[11] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.

[12] S. Koenig and Y. Liu, "The interaction of representations and planning objectives for decision-theoretic planning tasks," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 14, no. 4, pp. 303–326, 2002.

[13] G. A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems*, vol. 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[14] Wikipedia contributors, "Reinforcement learning — Wikipedia, the free encyclopedia," 2021. [Online; accessed 13-July-2021].

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[16] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[17] I. C. E. Fundamentals and J. Heywood, "Mcgraw-hill series in mechanical engineering," 1988.

[18] L. Guzzella and A. Amstutz, "Control of diesel engines," *IEEE Control Systems Magazine*, vol. 18, no. 5, pp. 53–71, 1998.

[19] C. D. Rakopoulos and E. G. Giakoumis, *Diesel engine transient operation: principles of operation and simulation analysis*. Springer Science & Business Media, 2009.

[20] L. Guzzella, A. Sciarretta, *et al.*, *Vehicle propulsion systems*, vol. 1. Springer, 2007.

[21] A. Parmar and A. Patrel, "Speed control techniques for induction motor-a review," *International Journal for Scientific Research and Development*, vol. 2, 2014.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[23] A. Choudhary, "A hands-on introduction to deep q-learning using openai gym in python," *Analytics Vidhya. https://www. analyticsvidhya. com/blog/2019/04/introduction-deep-q-learningpython*, 2019.

[24] R. Liessner, C. Schroer, A. M. Dietermann, and B. Bäker, "Deep reinforcement learning for advanced energy management of hybrid electric vehicles.," in *ICAART (2)*, pp. 61–72, 2018.

[25] J. Tian and other contributors, "Reinforcementlearning.jl: A reinforcement learning package for the julia programming language," 2020.

[26] H.-K. Lim, J.-B. Kim, J.-S. Heo, and Y.-H. Han, "Federated reinforcement learning for training control policies on multiple iot devices," *Sensors*, vol. 20, no. 5, p. 1359, 2020.