



Final Year Thesis

In order to obtain the Master's degree

Field of Study: Electrotechnical engineering

Subfield: Energy and Environment

Presented by : Tehami Younes and Ksantini Lydia

Topic

**PWM rectifier AI control for wind
energy conversion system based on a dual star
synchronous generator**

Publicly defended on 21/06/2026 ,Before the examination committee composed of

M.MEBROUKI M.	Doctor	ESSA. Tlemcen	Chair
M.KARBOUA A.	Doctor	ESSA. Tlemcen	Supervisor
M.BOUKli HACEN	F.Professor	ESSA. Tlemcen	Thesis supervisor
M.TAHOUR A.	Professor	ESSA. Tlemcen	Examiner 1
M.BENKHORIS M.F.	Professor	Polytech.NANTES	Examiner 2

Academic year :2025/2026

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

وَأَن لَّيْسَ لِلْإِنسَانِ

أَلَّا مَّا سَعَىٰ

[النجم]

Dedication

I dedicate this thesis to my younger self for believing in herself and choosing to follow the major that she likes, although the hard work and efforts it took and how she was able to go through all of that.

To my parents Amel and Zine El abidine Ksantini who were always there for me every step of the way. Their financial and emotional support ensured that I studied in the best conditions. I wouldn't have arrived here without their support.

I also dedicate this thesis to my aunt Linda, who always puts our needs, mine and my sister's, before hers. Her support means a lot to me.

To my sister Liza, who always made me laugh with her silly jokes and made me realize that whatever happens in life, the most powerful move you can do is to smile.

I also dedicate this thesis to my grandparents and the rest of my family members who supported me in every way through this journey.

I also dedicate this thesis to my colleague and dear friend Younes Tehami, who was always there for me at my lowest point and supported me and helped me to keep going when I had no strength left in my limbs.

Finally, I dedicate this thesis to my professor, Mr. Mebrouki Mohammed, who helped and inspired a lot of students, including me. The professor who truly cared about his students and whose only and most important concern was that we keep learning and improving. His concern was never about getting the highest grade or being the top student. He truly cared about helping students to improve and fall in love with the fascinating world of science and learning.

Lydia

Dedication

I dedicate this thesis to my father, TEHAMI Mohammed, whose financial support made this journey possible. His sacrifice and dedication ensured that I had everything I needed to focus on my studies and reach this milestone.

I also dedicate this work with deep love and gratitude to my mother, MEHTARI Naima, who passed away before having the chance to see me graduate and witness this achievement. She was the person who supported me the most throughout my journey, always encouraging me with her unconditional love, kindness, and belief in me. Her presence gave me strength during difficult times, and her memory continues to inspire me every day. Although she is no longer here to celebrate this moment with me, her influence is present in every step that led me here. This accomplishment is as much hers as it is mine.

To my brother, Mokhtar Tehami, who has always been by my side. His presence and encouragement meant more than words can express.

I also dedicate this work to my English teacher, Mrs. BENHABIB SOULIMANE Nouzha, who made the process of learning English truly enjoyable. She gave us countless opportunities to present and express ourselves in this language, and through that, she built in us a confidence we carry to this day.

Finally, I dedicate this thesis to Mr. TAHOUR Ahmed, our Power Electronics teacher, whose lessons made the choice of this graduation project feel natural and familiar. He sparked in us a genuine interest in the concept of advanced inverters, and without his teaching, this topic would have felt far more distant and intimidating.

Younes

Acknowledgments

Above all, we thank Allah, the Almighty and Merciful, whose support and blessing have enabled us to complete this work.

We wish to express our deep gratitude to our supervisor, Mr. KERBOUA Abdelfettah, for accepting to supervise this project. He believed in us and gave us enough freedom and flexibility to work comfortably and to support our opinions and ideas. He motivated us to fully explore our potential in this work. His wise advice and rigorous supervision greatly contributed to the success of this work.

We would also like to extend our sincere gratitude to Mr. BOUKLI HACENE Fouad, our co-supervisor, for his guidance throughout this work, his constant support, and the encouragement that motivated us to see it through to completion.

We sincerely thank Mr.MEBROUKI Mohammed for the honor of accepting to chair our thesis defense committee.. We also sincerely thank Mr.TAHOUR Ahmed and Mr.Mohamed Fouad BENKHOUS for accepting to evaluate our work as examiners.

Finally, we express our heartfelt gratitude to all the professors for their constant support, the quality of their supervision, and their commitment throughout our academic journey.

Table of Contents

General Introduction	1
Chapter 1: Reinforcement Learning: From Theory to Control System Applications	3
1.1 Introduction	4
1.2 Fundamentals of Reinforcement Learning	4
1.2.1 The Agent–Environment Interface	4
1.2.2 Goals and Rewards	5
1.2.3 Returns	5
1.3 Markov Property and Markov Decision Process	6
1.4 Value Functions	6
1.5 Actor-Critic Methods	7
1.6 Reinforcement Learning for Control Systems Application	8
1.7 Deep Reinforcement Learning Algorithms for Continuous Control	10
1.7.1 Deep Deterministic Policy Gradient (DDPG)	10
1.7.2 Twin Delayed Deep Deterministic Policy Gradient (TD3)	15
1.8 Conclusion	16
Chapter 2: Direct Replacement of the PI Controller: A DDPG-Based Approach	18
2.1 Introduction	19
2.2 System Overview: Dual-Star Synchronous Generator with PWM Rectifier	19
2.2.1 The Dual-Star Synchronous Generator (DSSG)	20
2.2.2 The PWM Rectifier	20
2.2.3 Cascade Control Structure	20
2.3 Why the DSSG-PWM Rectifier Conversion Chain is Suitable for RL	21
Implementation	
2.4 Simulink Model of the DSSG-PWM Rectifier Conversion Chain	21
2.5 Reinforcement Learning Agent Integration in Simulink	22
2.5.1 RL Agent Block in Simulink	23
2.5.2 Block Configuration and Functionality	24
2.5.3 Agent Object Configuration	24
2.6 Real-Time Execution	25
2.7 Observation Space, Action Space, and Reward Design	26
2.7.1 Observation Space	26
2.7.2 Action Space	27
2.7.3 Reward Function Design	28
2.8 RL Agent Configuration (DDPG) in MATLAB	30
2.8.1 Block 1: Observation and Action Specifications	30
2.8.2 Block 2: Critic Network Architecture	34

2.8.3 Block 3: Actor Network Architecture	36
2.8.4 Block 4: DDPG Agent Configuration and Initialization	38
2.8.5 Block 5: Training Configuration and Execution	40
2.9 RL Controller Implementation and Initial Training Challenges	41
2.10 Training Results and Discussion	42
2.10.1 Current Tracking Performance and Learning Evolution	42
2.10.2 Control Signals and Outer Loop Behavior	44
2.10.3 Reward Evolution During Training	47
2.10.4 PI and DDPG Control: A Comparative Perspective	48
2.11 Conclusion	50
Chapter 3: Grid Integration and RL-Based PI Tuning for the Energy Conversion Chain	51
3.1 Introduction	52
3.2 Grid Integration and Control Strategy	52
3.2.1 Choice of the Universal Bridge Model	52
3.2.2 Grid and Inverter Modeling	53
3.2.3 Three-Phase Quantities and Reference Frame Transformation	53
3.2.4 Clarke Transformation	54
3.2.5 Park Transformation	54
3.2.6 Phase Locked Loop (PLL) Synchronization	55
3.2.7 Current Control in the dq Reference Frame	55
3.2.8 Voltage Equations in the Synchronous Reference Frame	56
3.2.9 Active and Reactive Power Equations	56
3.2.10 Inverse Transformation and PWM Generation	56
3.2.11 Implementation of the Grid-Connected Inverter in MATLAB/Simulink	57
3.2.12 Simulation Results and Grid Performance	58
3.2.13 Transition from Conventional PI Control to Reinforcement Learning-Based Tuning	60
3.3 TD3-Based Adaptive PI Control	60
3.4 RL Agent Configuration (TD3) in MATLAB	63
3.4.1 Block 1: Observation and Action Specifications	63
3.4.2 Block 2: Actor Network	64
3.4.3 Block 3: Critic Networks	64
3.4.4 Block 4: TD3 Agent Options	66
3.4.5 Block 5: Training Procedure	66
3.4.6 Block 6: Simulation and Gain Extraction	67
3.4.7 Training Results and Performance Analysis	68
3.5 TD3-Based Adaptive Control of the Dual-Star Generator	71
3.5.1 Motivation and Control Objective	71
3.5.2 Observation and Reward Design	72
3.5.3 Agent Architecture and Training	73
3.5.4 Training Results	73
3.5.5 Observations and Interpretation	74
3.5.6 Comparison with Conventional PI Control	76
3.6 Conclusion	76
General Conclusion	78

List of Figures

Figure 1.1: The agent–environment interaction in reinforcement learning	5
Figure 1.2: Correspondence between Reinforcement Learning and Control Systems	9
Figure 1.3: DDPG algorithm as presented in [2]	13
Figure 1.4: DDPG architecture and training flow [3]	14
Figure 1.5: TD3 algorithm as presented in [5]	15
Figure 2.1: Complete Simulink model of the DSSG-PWM rectifier system	21
Figure 2.2: Cascade Control structure of the PWM rectifier	21
Figure 2.3: Conventional PI controller for i_q current control	23
Figure 2.4: RL Agent block icon	23
Figure 2.5: RL Agent block parameters configuration showing agent object and environment structure	24
Figure 2.6: RL agent block integration with observation, reward, and isDone subsystems	25
Figure 2.7: Observation subsystem in Simulink showing the five signals	27
Figure 2.8: Simulink model showing affine scaling and saturation block for action signal processing	28
Figure 2.9: Simulink base reward subsystem	29
Figure 2.10: Actor Network Architecture	37
Figure 2.11: System response during Episode 10	42
Figure 2.12: System response during Episode 30	44
Figure 2.13: DC bus voltage tracking behavior during training (early and intermediate episodes)	46
Figure 2.14: DC bus voltage tracking behavior after convergence	46
Figure 2.15: Episode reward evolution over 250 training episodes	47
Figure 2.16: Episode reward details (first and last 10 training episodes)	47
Figure 2.17: Comparison between the conventional PI controller and the DDPG-based control approach	48
Figure 2.18: Comparison between the conventional PI controller and the DDPG-based control approach	48
Figure 2.19: V_q Scaled Signal	45
Figure 2.20: Reference Signals for PWM Carrier Comparison	45
Figure 2.21: DC bus voltage tracking behavior during training (early and intermediate episodes)	46
Figure 2.22: DC bus voltage tracking behavior after convergence	46

Figure 2.23: Episode reward evolution over 250 training episodes	47
Figure 2.24: Episode reward details (first and last 10 training episodes)	47
Figure 2.25: i_q current tracking (hybrid PI-DDPG vs conventional PI)	49
Figure 2.26: DC bus voltage regulation (hybrid PI-DDPG vs conventional PI)	49
Figure 2.27: Reference current i_q^* (hybrid PI-DDPG vs conventional PI)	50
Figure 3.1: Control structure of the grid-connected inverter	53
Figure 3.2: MATLAB implementation of the grid-connected inverter with LCL filter	57
Figure 3.3: Control structure of the grid-connected inverter	58
Figure 3.4: Overall system architecture used for grid integration simulation	58
Figure 3.5: Active power response under step changes in reference power	58
Figure 3.6: Three-phase grid voltages (top) and currents (bottom) under varying power references	58
Figure 3.7: Modified control structure using the TD3 agent	60
Figure 3.8: Observation generation subsystem	64
Figure 3.9: Custom PI controller with gains provided by the TD3 agent	65
Figure 3.10: Reward calculation subsystem	65
Figure 3.11: Complete TD3-based adaptive PI control system	66
Figure 3.12: System response during Episode 10	69
Figure 3.13: System response during Episode 30	70
Figure 3.14: System response during Episode 30	70
Figure 3.15: System response during Episode 98	71
Figure 3.16: Generator-side control structure: conventional PI for I_d and TD3 agent for I_q reference generation	72
Figure 3.17: Observation subsystem for the generator-side TD3 agent	73
Figure 3.18: Reward calculation subsystem for the generator-side agent	73
Figure 3.19: Complete TD3-based adaptive control system for the dual-star generator	74
Figure 3.20: Rectifier output voltage across selected training episodes, illustrating the learning progression of the TD3 agent	74
Figure 3.21: I_q Ref And I_q Results	75
Figure 3.22: Comparison of rectifier output voltage: fixed PI controller versus TD3-tuned adaptive controller	77

Abbreviations list

Abriviation	Signification
RL	Reinforcement learning
DDPG	Deep deterministic policy gradient
TD3	Twin Delayed Deep Deterministic policy gradient
DNN	Deep Neural Network
DRL	Deep Reinforcement Learning
MDP	Markov Decision Process
ReLU	Rectified Linear Unit
tanh	Hyperbolic Tangent
PI	Proportional-Integral
PLL	Phase-Locked Loop
DSSG	Dual-Star Synchronous Generator
PWM	Pulse Width Modulation
THD	total Harmonic Distortion
MLI	Multilevel Inverter
FFT	Fast Fourier Transform

General Introduction

Electrical energy generation from renewable energy sources has become increasingly important in response to the growing global demand for electricity and the urgent need to reduce dependence on fossil fuels. The integration of renewable energy systems into the electrical grid plays a central role in modern energy infrastructure, as it allows clean and distributed power sources to contribute directly to the utility network. However, the intermittent and variable nature of renewable energy makes direct grid injection technically challenging, requiring power electronics converters as an interface between the energy source and the grid.

Among the machine configurations suitable for renewable energy conversion, the Dual-Star Synchronous Generator (DSSG) has attracted considerable research interest due to its advantages over conventional three-phase machines. By distributing the stator windings into two spatially shifted three-phase sets, the DSSG offers improved power density, enhanced fault tolerance, reduced torque ripple, and better harmonic performance. These characteristics make it particularly well suited for high-power energy conversion applications where reliability and efficiency are critical requirements.

To fully exploit the power delivered by the DSSG, a controlled rectification stage is required. The PWM rectifier provides regulated DC voltage with instantaneous control of the current waveform, enabling near-unity power factor operation and superior power quality compared to conventional diode-based rectifiers. The control of such a system is typically organized as a cascade structure, with an outer voltage regulation loop and an inner current control loop, both relying on proportional-integral controllers whose performance depends heavily on the quality of the selected gains. On the output side, a grid-connected inverter is responsible for transferring the regulated DC power back into the utility grid as balanced three-phase AC voltage, synchronized through appropriate reference frame transformations and a Phase Locked Loop.

Conventional PI controllers remain widely adopted in industrial applications due to their simplicity and robustness. However, their performance is inherently limited to operating regions where the system behaves in a nearly linear fashion, and their gains must be carefully tuned by hand based on prior knowledge of the plant dynamics. This manual tuning process is time-consuming, requires expert knowledge, and may not produce optimal results under varying operating conditions. With the rapid development of artificial intelligence and machine learning, reinforcement learning has emerged as a compelling alternative for control system design, offering the ability to learn optimal control strategies directly from interaction with the system, without requiring an explicit mathematical model of the plant.

This work investigates the integration of reinforcement learning into the control architecture of a DSSG-PWM rectifier energy conversion chain connected to the electrical grid. Two complementary approaches are studied. The first approach consists of directly replacing the conventional PI controller in the inner current loop with a reinforcement learning agent based on the Deep Deterministic Policy Gradient algorithm, which learns to generate the control voltage directly from system observations. The second approach preserves the PI controller structure while employing a Twin Delayed Deep Deterministic Policy Gradient agent as a supervisory layer that continuously and automatically tunes the proportional and integral gains according to the current operating conditions of the system.

This adaptive tuning approach is applied to both the PWM rectifier current controller and the grid-side inverter current controllers, allowing a comprehensive evaluation of reinforcement learning as a practical tool for improving the dynamic

Chapter 1

Reinforcement Learning: From Theory to Control System Applications

1.1 Introduction

The idea that we learn by interacting with our environment is probably the first to occur to us when we think about the nature of learning. For example, when an infant plays or looks about, it has no explicit teacher, but it does have a direct sensorimotor connection to its environment. Exercising this connection produces a wealth of information about the consequences of actions and about what to do in order to achieve goals. Learning by interaction is a fundamental idea underlying nearly all theories of learning and intelligence. A computational approach to learning by interaction, rather than theorizing how creatures learn, is reinforcement learning, which is more focused on goal-directed learning from interaction than other approaches of machine learning. According to [1]:

Reinforcement learning problems involve learning what to do ,how to map situations to actions so as to maximize a numerical reward signal. In an essential way they are closed-loop problems because the learning system's actions influence its later inputs. Moreover, the learner is not told which actions to take, as in many forms of machine learning, but instead must discover which actions yield the most reward by trying them out. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These three characteristics : being closed-loop in an essential way, not having direct instructions as to what actions to take, and where the consequences of actions, including reward signals, play out over extended time periods are the three most important distinguishing features of reinforcement learning problems.

1.2 Fundamentals of Reinforcement Learning

1.2.1 The Agent–Environment Interface

The RL learning problem is meant to be a straightforward framing of the problem of learning from interaction to achieve a goal. The term "agent" refers to the learner and decision-maker. The "environment" is the item it interacts with, which includes everything outside the agent. These constantly interact the agent selecting actions and the environment responding to those action and presenting new situations to the agent gives rise to rewards, special numerical values that the agent tries to maximize over time

More specifically, the agent and environment interact at each of a sequence of discrete time steps, $t = 0, 1, 2, 3, \dots$. At each time step t , the agent receives some representation of the environment's state, $S_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of possible states, and on that basis selects an action, $A_t \in \mathcal{A}(S_t)$, where $\mathcal{A}(S_t)$ is the set of actions available in state S_t . One time step later, mainly due to its action, the agent receives a numerical reward, $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$, and finds itself in a new state, S_{t+1} .

At each time step, the agent implements a mapping from states to probabilities of selecting each possible action. This mapping is called the agent's policy and is denoted π_t , where $\pi_t(a|s)$ is the probability that $A_t = a$ if $S_t = s$. Reinforcement learning methods specify how the agent changes its policy in response to its experience. In general, the agent's objective is to maximize the total reward it receives over time.

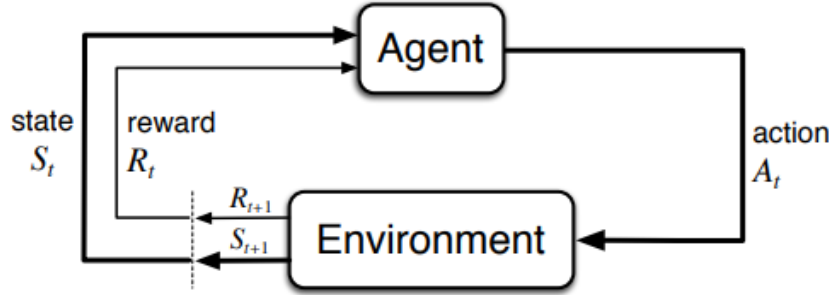


Figure 1.1: The agent–environment interaction in reinforcement learning. [1]

1.2.2 Goals and Rewards

In reinforcement learning, the purpose of the agent is formalized in terms of a special reward signal passing from the environment to the agent at each time step. The reward is a simple number, $R_t \in \mathbb{R}$. The agent’s goal is to maximize the total amount of reward it receives. This means maximizing the cumulative reward in the long run. If we want it to do something for us, we must provide rewards to it in such a way that in maximizing them the agent will also achieve our goals. It is thus critical that the rewards we set up truly indicate what we want accomplished.

1.2.3 Returns

In reinforcement learning, at every time step t the agent looks at the current state S_t , picks an action A_t based on its policy π , and receives a reward R_{t+1} while moving to a new state S_{t+1} . The agent’s goal is to maximize the cumulative return:

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T \quad (1.1)$$

which is simply the sum of all future rewards. If the task has a natural ending point, like a chess game or a maze, it is called an **episodic task**, where each run from start to finish is one episode. If the task runs forever with no natural ending, like a robot working continuously, it is called a **continuing task**. The problem with continuing tasks is that summing rewards up to $T = \infty$ makes the return infinite and meaningless:

$$G_t = R_{t+1} + R_{t+2} + \cdots = \infty \quad (1.2)$$

which is why a smarter formulation called **discounting** is needed.

To address this, we introduce a **discount factor** γ to reduce the value of rewards that are far in the future, rather than summing all future rewards equally. The agent therefore seeks to maximize the **discounted return**:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (1.3)$$

where $0 \leq \gamma \leq 1$ is called the **discount rate**. A reward received k steps in the future is worth only γ^{k-1} times its immediate value, the further away it is, the less it counts. When $\gamma = 0$, the agent is completely **myopic**, caring only about the immediate reward R_{t+1} and

ignoring everything after. When γ approaches 1, the agent becomes more **farsighted**, treating future rewards almost as valuable as immediate ones. In practice, a value like $\gamma = 0.9$ or $\gamma = 0.99$ is chosen to balance short and long term thinking, and as long as the reward sequence is bounded and $\gamma < 1$, the infinite sum always converges to a finite value, solving the $G_t = \infty$ problem we had with continuing tasks.

1.3 Markov Property and Markov Decision Process

A reinforcement learning environment is often described as satisfying the **Markov property**, when the current state contains all relevant information from the past needed for decision-making, meaning that the system’s future evolution depends only on the current state and action, and not on the full history of previous states, actions or rewards. Formally this is expressed as:

$$P(S_{t+1}, R_{t+1} \mid S_t, A_t) = P(S_{t+1}, R_{t+1} \mid S_0, A_0, R_1, \dots, S_t, A_t) \quad (1.4)$$

which states that once the current state S_t is known, all earlier information becomes unnecessary for predicting the next state and reward. Based on this property, a reinforcement learning problem is modeled as a **Markov Decision Process**, usually defined by the tuple:

$$(S, A, P, R, \gamma) \quad (1.5)$$

where S is the set of states, A the set of actions, $P(S_{t+1} \mid S_t, A_t)$ the transition probability, R the reward function, and γ the discount factor that determines the importance of future rewards. In this framework, the agent interacts with the environment by observing a state S_t , selecting an action A_t , receiving a reward R_{t+1} , and transitioning to a new state S_{t+1} , with the objective of maximizing the cumulative discounted return:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (1.6)$$

This formulation is fundamental in reinforcement learning and underlies the majority of algorithms, including those used in continuous control like **Deep Deterministic Policy Gradient (DDPG)** and **Twin Delayed Deep Deterministic Policy Gradient (TD3)**, which rely on the assumption of Markov states to enable the agent to make the best choices based only on the current observation.

1.4 Value Functions

In reinforcement learning, most algorithms rely on the concept of a **value function**, which is used to evaluate how good it is for an agent to be in a given state, or to take a given action in that state. This notion of “how good” is not based on immediate reward, but on the expected cumulative future reward, also called the **return**. Since future rewards depend on the actions taken by the agent, value functions are defined with respect to a policy π , which specifies the agent’s behavior.

The **state-value function** $v^\pi(s)$ gives the expected return when starting from a state s and following a policy π thereafter, and is defined as:

$$v^\pi(s) = \mathbb{E}_\pi [G_t \mid S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right] \quad (1.7)$$

Similarly, the **action-value function** $q^\pi(s, a)$ represents the expected return when the agent starts in state s , takes action a , and then follows policy π :

$$q^\pi(s, a) = \mathbb{E}_\pi [G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right] \quad (1.8)$$

These value functions can be estimated from experience by averaging observed returns, and in large-scale problems they are often approximated using parameterized models such as neural networks.

A key property of value functions is that they satisfy a recursive relationship, known as the **Bellman equation**. This expresses the value of a state in terms of the immediate reward and the value of its successor states:

$$v^\pi(s) = \sum_a \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a) [r + \gamma v^\pi(s')] \quad (1.9)$$

This recursive structure is fundamental in reinforcement learning because it allows value functions to be learned incrementally by propagating information from future states back to earlier ones. In practice, this property plays a central role in many modern reinforcement learning approaches used in continuous control problems. Building on this idea, the next step is to introduce learning architectures that explicitly exploit value function estimation for decision-making in a more structured way.

1.5 Actor-Critic Methods

Not all reinforcement learning methods select actions directly from estimated action-value functions. A separate class of methods instead represents the policy explicitly, learning it independently from the value function. Here, the value function is not used to choose actions directly, rather it acts as a feedback signal that guides policy improvement. This is the core idea behind **actor-critic methods**.

In this setup, two components work together: the **actor**, which acts as the **decision-maker** by representing the policy and selecting actions, and the **critic**, which evaluates the **goodness of each action** by estimating the value of the resulting states. In practice, the critic learns a state-value function $V(s)$, and uses it to compute a **temporal-difference (TD) error**:

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \quad (1.10)$$

This quantity captures how much better or worse the observed outcome was compared to what the critic expected. When $\delta_t > 0$, the result exceeded expectations and the action

should be encouraged. When $\delta_t < 0$, the result fell short and the tendency to take that action should be weakened. In this way, the critic does not control behavior directly, it simply tells the actor whether things went better or worse than anticipated.

It is worth noting that the policy in actor-critic methods is not restricted to a single form. It can be stochastic, sampling actions from a probability distribution, or deterministic, mapping states directly to actions. In many standard reinforcement learning settings, stochastic policies are preferred. However, in continuous control problems, where the action space is high-dimensional and exhaustive search is computationally infeasible, deterministic policies tend to be more practical. This observation motivates algorithms like **Deep Deterministic Policy Gradient (DDPG)** and **Twin Delayed Deep Deterministic Policy Gradient (TD3)**, both of which implement the actor and critic as neural networks and will be examined in detail in the following sections.

Taken together, actor-critic methods offer a structured and scalable approach to policy learning. By separating the roles of action selection and value estimation, they handle the complexity of real-world environments more effectively than methods that rely on either component alone. This becomes especially important when reinforcement learning is applied to engineering control problems, where the goal extends beyond reward maximization to include stability, robustness, and precise tracking performance.

1.6 Reinforcement learning for control systems application

To better understand the role of reinforcement learning in engineering applications, it is useful to establish a correspondence between the RL framework and classical control system architecture. Despite coming from different theoretical origins, these two fields share a common objective: using observed system behavior to generate control actions that optimize system performance over time. In both cases, decisions are made according to measured information, applied to a physical system, and continuously assessed using a performance criterion. This similarity makes RL particularly relevant for control applications, especially in nonlinear and complex dynamical systems where conventional control strategies may face limitations. Figure 1.2 illustrates how the key elements of a classical feedback control system correspond to those of reinforcement learning.

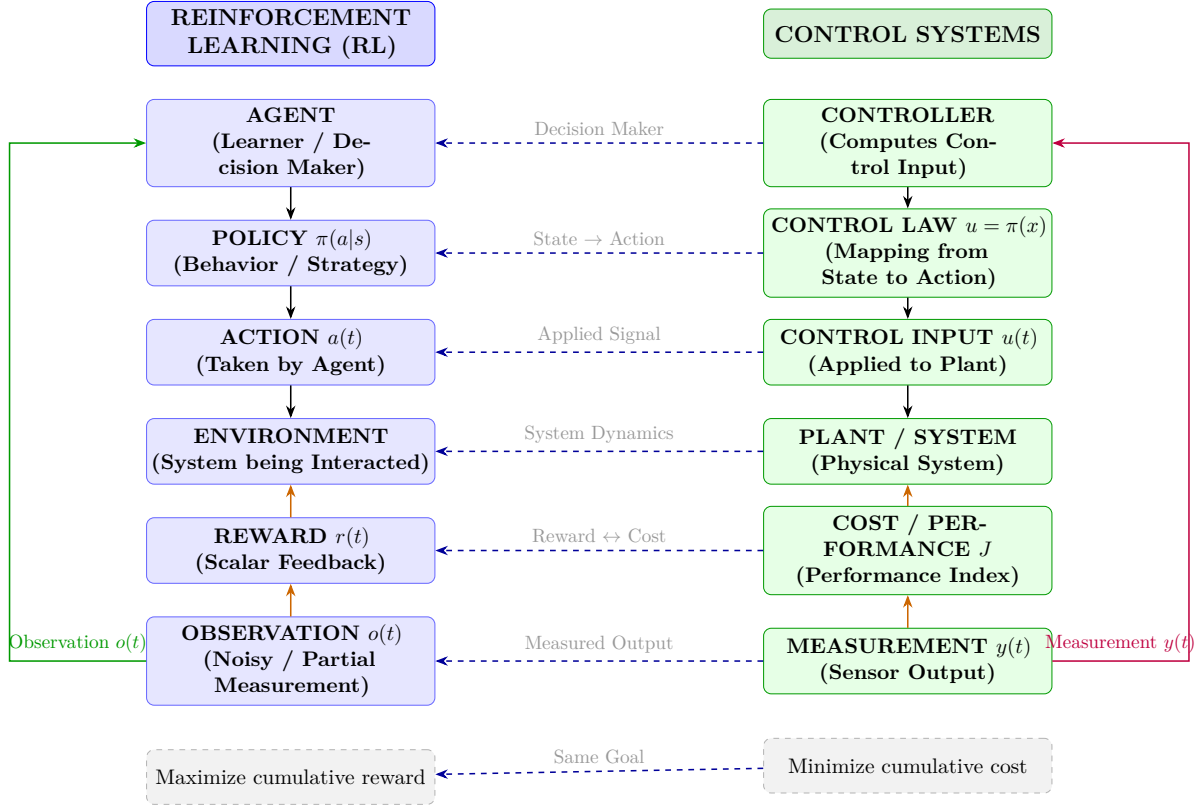


Figure 1.2: Correspondence between Reinforcement Learning and Control Systems

As shown in Figure 1.2, the **RL agent** plays the role of the **controller**, while the learned **policy** represents the **control law** that maps system observations to control actions. The **environment** represents the controlled system, including the physical plant, its operating conditions, external disturbances, and measured feedback. The **reward function** acts as a **performance criterion**, similarly to the **cost function** used in conventional control design. Similarly, the **observations** received by the agent correspond to measured system variables obtained from sensors. This correspondence highlights how RL can be naturally integrated into control applications, where the goal is to learn an **optimal control strategy** through continuous interaction with the system while minimizing **tracking error** and improving **dynamic performance**.

However, applying reinforcement learning in control systems requires selecting algorithms that are compatible with the characteristics of real engineering applications. Unlike many classical reinforcement learning problems involving **discrete action spaces**, control systems such as **electrical drives** and **energy conversion chains** require **continuous-valued control inputs**. In addition, these systems often exhibit **nonlinear dynamics**, **parameter uncertainties**, and **strong coupling between variables**, which demand stable learning behavior and efficient convergence. Although conventional linear controllers such as **PI regulators** are widely used in industrial applications due to their simplicity and ease of implementation, their performance is often limited to operating regions where the system can be accurately approximated by a **linear model**. Outside these conditions, tracking performance may degrade due to **nonlinearities**, **parameter variations**, or **external disturbances**. Furthermore, determining the **optimal controller gains** often requires extensive manual tuning and expert knowledge, and achieving the best compromise between **dynamic response**, **stability**, and **steady-state accuracy** can be challenging.

in complex systems.

In this context, reinforcement learning provides an attractive alternative. It can be used either to learn a **nonlinear control policy** directly from system interactions, or to **automatically optimize the parameters** of conventional controllers such as **PI regulators**. Under these conditions, **actor–critic** methods based on **deterministic policy learning** have emerged as particularly effective solutions. Among these approaches, **Deep Deterministic Policy Gradient (DDPG)** and its improved variant **Twin Delayed Deep Deterministic Policy Gradient (TD3)** have demonstrated strong performance in **continuous control tasks**. For this reason, these algorithms are adopted in this work to develop and optimize the **control strategies** of the studied system.

1.7 Deep Reinforcement Learning Algorithms for Continuous Control

Deep Reinforcement Learning (DRL) is simply the combination of reinforcement learning (RL) with **deep neural networks (DNN)**. The idea is straightforward: instead of using simple lookup tables or linear models to represent the policy and value functions, we use neural networks, which are capable of approximating complex **nonlinear** relationships. This allows the agent to handle **high-dimensional** states and **continuous actions** that are difficult for traditional RL methods. In energy conversion systems for example, control signals such as voltage and current are continuous by nature, which makes DRL particularly suitable for such applications. In this section, two DRL algorithms are presented: the **Deep Deterministic Policy Gradient (DDPG)** and the **Twin Delayed Deep Deterministic Policy Gradient (TD3)**, both of which are based on the **actor–critic** architecture introduced in the previous section.

1.7.1 Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) is a model-free RL algorithm designed for environments with **continuous action spaces**. In RL, **model-free** means that the agent does not require an explicit mathematical model of the environment, such as differential equations or transfer functions. Instead, the agent learns directly through interactions with the environment by observing the system state, applying control actions, and receiving feedback in the form of rewards.

Unlike value-based methods such as **Q-learning**, which estimate the value of different actions before selecting one, DDPG belongs to the family of **policy gradient methods**. In reinforcement learning, a policy is the decision-making rule that maps the current system state to a control action. Policy gradient methods learn this decision rule directly by adjusting its parameters in the direction that increases the expected cumulative reward. In simpler terms, instead of searching among multiple possible actions, the agent gradually learns how to generate better actions through optimization. This makes policy gradient methods particularly suitable for engineering applications involving continuous control signals such as current, voltage, or torque references. DDPG combines policy gradient learning with the **actor–critic architecture**.

Actor (Policy Network)

The actor tells the agent which action to take given the state it is in. The network parameters are represented by θ^μ :

$$a_t = \mu(s_t \mid \theta^\mu) \quad (1.11)$$

where:

- s_t is the observed system state,
- a_t is the control action,
- $\mu(\cdot)$ is the actor policy,
- θ^μ represents the actor network parameters.

The term **deterministic** means that for a given state, the actor always produces the same control action. To address this challenge, **exploration noise** is added to actions during training to encourage the agent to explore as much of the action space as possible:

$$a_t = \mu(s_t \mid \theta^\mu) + \mathcal{N}(t) \quad (1.12)$$

The noise $\mathcal{N}$ can be chosen to suit the environment. In the paper that first introduced DDPG [2], an **Ornstein–Uhlenbeck (OU) process** was used to generate temporally correlated exploration noise. Unlike purely random noise, the OU process produces smooth, correlated noise sequences that decay toward zero over time. This is particularly useful in physical control problems with inertia, such as robotic arms or electric drives, where actions need to change gradually rather than randomly jumping between values.

Critic (Q-value Network)

The critic evaluates how good the action taken by the actor is, by estimating the **Q-value** of the state-action pair:

$$Q(s_t, a_t \mid \theta^Q) \quad (1.13)$$

where $Q(s, a)$ estimates the expected cumulative reward obtained by applying action a in state s , and θ^Q represents the critic network parameters.

To train the critic, DDPG computes a target value using the **Bellman equation**:

$$y_t = r_t + \gamma Q'(s_{t+1}, \mu'(s_{t+1})) \quad (1.14)$$

where:

- r_t is the immediate reward,
- γ is the discount factor,

- Q' and μ' are the target networks.

The critic parameters are then updated by minimizing the following **loss function**:

$$L(\theta^Q) = \frac{1}{N} \sum_{i=1}^N \left(y_i - Q(s_i, a_i \mid \theta^Q) \right)^2 \quad (1.15)$$

where N is the mini-batch size, representing the number of randomly selected experiences used in one training update. Rather than learning from a single interaction at a time, the network learns from a small group of experiences simultaneously, which produces smoother updates and improves numerical stability. Once the critic evaluates the action quality, the actor is updated using the **deterministic policy gradient**:

$$\nabla_{\theta^\mu} J \approx \nabla_a Q(s, a \mid \theta^Q) \nabla_{\theta^\mu} \mu(s \mid \theta^\mu) \quad (1.16)$$

This update allows the actor to gradually adjust its parameters in the direction that improves long-term control performance.

Experience Replay

One of the main challenges in reinforcement learning is that consecutive interactions with the environment are highly correlated, especially in dynamical systems where states evolve continuously over time. Training directly on such correlated data may cause unstable learning or slow convergence.

To overcome this issue, DDPG uses an **experience replay buffer**, which acts as a memory that stores past interactions in the form:

$$(s_t, a_t, r_t, s_{t+1}) \quad (1.17)$$

where each transition contains the current state, the applied action, the received reward, and the next state. During training, the algorithm does not use only the latest experience. Instead, it randomly samples **mini-batches** from the replay buffer. This improves learning stability for two main reasons:

1. It breaks the temporal correlation between consecutive samples, making the training data more statistically independent.
2. It allows previous experiences to be reused multiple times, improving learning efficiency and reducing the amount of interaction required with the system.

Target Networks

Another source of instability comes from the fact that the critic network learns from target values that are themselves changing during training. If these target values change too rapidly, the learning process may become unstable or even diverge.

To address this problem, DDPG uses slowly updated copies of both the actor and critic networks, called **target networks**. These are updated using the **soft update rule**:

$$\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'} \quad (1.18)$$

$$\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'} \quad (1.19)$$

where:

- θ^Q and θ^μ are the parameters of the main networks,
- $\theta^{Q'}$ and $\theta^{\mu'}$ are the parameters of the target networks,
- $\tau \ll 1$ is a small update coefficient.

This means that the target networks evolve gradually instead of changing abruptly. As a result:

- the critic learns from smoother and more stable target values,
- large oscillations in parameter updates are reduced,
- convergence becomes more stable and consistent.

Algorithm 1.3 summarizes the complete DDPG training procedure as originally presented in [2], bringing together the actor-critic architecture, the experience replay buffer, the exploration noise, and the soft update of target networks into a single unified training loop.

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ .
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
 Initialize a random process $\mathcal{N}$ for action exploration
 Receive initial observation state s_1
 for $t = 1, T$ **do**
 1 Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
 2 Execute action a_t and observe reward r_t and observe new state s_{t+1}
 3 Store transition (s_t, a_t, r_t, s_{t+1}) in R
 4 Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
 5 Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
 6 Update critic by minimizing the loss: $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
 7 Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

8 Update the target networks:

$$\begin{aligned} \theta^{Q'} &\leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'} \\ \theta^{\mu'} &\leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'} \end{aligned}$$

end for
end for

Figure 1.3: DDPG algorithm as presented in [2]

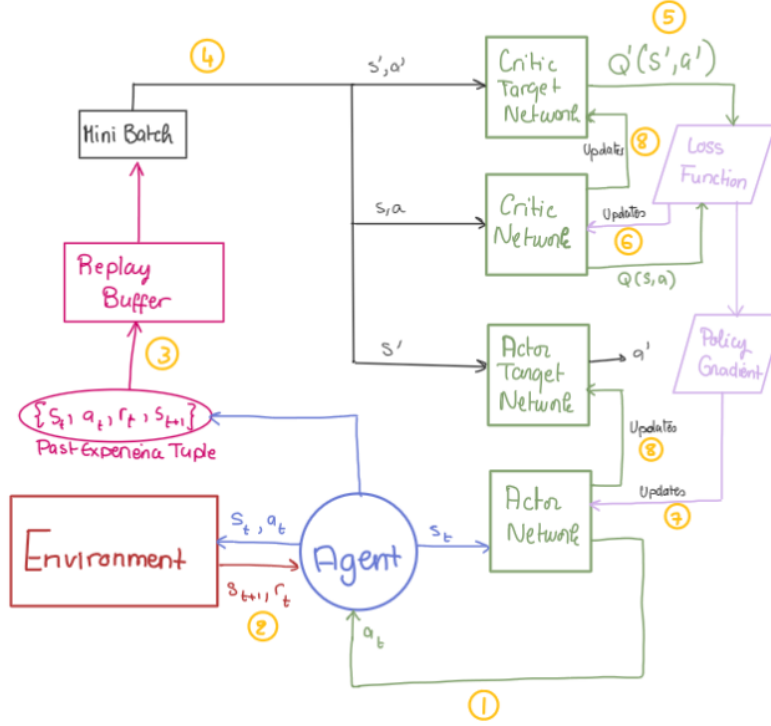


Figure 1.4: DDPG algorithm: actor-critic structure with experience replay and target network updates. [3]

Figure 1.4 above provides a visual representation of the DDPG architecture, illustrating how the different components interact during training. The numbered steps in the diagram correspond directly to the numbered lines in Algorithm 1.3, allowing the reader to follow the sequential flow of information through the system, from the agent's interaction with the environment, through the experience replay mechanism, to the network updates that gradually improve the control policy.

By combining actor-critic learning, deterministic policy optimization, experience replay, and target networks, DDPG is able to learn complex nonlinear control laws in continuous action spaces. This makes it particularly suitable for engineering applications where conventional linear controllers may show limited performance under nonlinear operating conditions. In this work, DDPG is employed as the core algorithm enabling the reinforcement learning agent to replace the conventional PI regulator used in the control of the studied energy conversion chain, learning an optimal control policy directly through interaction with the system.

However, despite its effectiveness, DDPG is known to suffer from several limitations that can affect its reliability in practice. The first is **overestimation bias**, where the critic network tends to overestimate Q-values, meaning it assigns higher expected returns to state-action pairs than what is actually achievable. This leads to suboptimal policy updates, as the actor is guided by an overly optimistic critic. The second limitation is **sensitivity to hyperparameters**: the performance of DDPG is highly dependent on the choice of learning rates, network architecture, replay buffer size, and exploration noise parameters, making it difficult to tune reliably across different systems. Finally, DDPG can exhibit **training instability**, where the interaction between the actor and critic networks causes oscillations or divergence during learning, particularly when the target

values change too rapidly or when the exploration noise introduces highly inconsistent experiences. These limitations become especially problematic in control applications where consistent and reliable convergence is essential.

To address these limitations, an improved variant of DDPG was proposed, known as **Twin Delayed Deep Deterministic Policy Gradient (TD3)**, which introduces several key modifications to improve stability and robustness. The TD3 algorithm is presented in the following section.

1.7.2 Twin Delayed Deep Deterministic Policy Gradient (TD3)

Twin Delayed Deep Deterministic Policy Gradient (TD3) is an improved variant of DDPG, designed to address its main limitations while preserving its core strengths [4]. Like DDPG, TD3 is a **model-free, off-policy** algorithm built for **continuous action spaces**, but it introduces three key modifications that significantly improve stability and reliability.

The first modification is the use of **twin critics**. Instead of relying on a single critic network, TD3 maintains two independent critic networks and uses the **minimum Q-value** between them during updates. This directly addresses the **overestimation bias** that affected DDPG, where the critic would assign unrealistically high values to certain actions. By taking the more conservative of the two estimates, the actor is guided by more accurate feedback, leading to better decision-making.

The second modification is **delayed policy updates**. In TD3, the actor is updated less frequently than the critics, typically once every two critic updates. This prevents the policy from being adjusted based on noisy or unreliable Q-value estimates, which was one of the main sources of **training instability** in DDPG. By allowing the critics to first converge to more stable estimates, the actor updates become more informed and consistent.

The third modification is **target policy smoothing**. TD3 adds small clipped noise to the target actions used during the critic update. This prevents the Q-function from **overfitting**, meaning it avoids becoming too specialized to specific state-action pairs seen during training and losing its ability to generalize well to new or slightly different situations. The result is a smoother and more robust policy that performs reliably across a wider range of operating conditions. Together, these three innovations make TD3 a more stable and reliable algorithm than DDPG, particularly in control applications where consistent convergence is essential.

The TD3 training procedure follows a similar structure to DDPG but with the three modifications discussed above clearly embedded in the loop. At each timestep, the agent observes the current state and selects an action using the actor network with added exploration noise. This experience is then stored in the replay buffer. During the update phase, a random mini-batch of past experiences is sampled and used to update **both critic networks** by minimizing the difference between their predictions and the target Q-value, which is computed using the **minimum** of the two target critics. This is where the overestimation bias is directly addressed. The actor and target networks are then updated only every d steps, where d is the policy delay parameter, ensuring that the policy is only adjusted once the critics have reached sufficiently stable estimates. Finally, all target networks are updated using the same **soft update rule** introduced in DDPG.

Algorithm 1.5 presents the complete procedure as originally proposed in [5].

Algorithm 1 TD3

Initialize critic networks $Q_{\theta_1}, Q_{\theta_2}$, and actor network π_ϕ
 with random parameters θ_1, θ_2, ϕ
 Initialize target networks $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2, \phi' \leftarrow \phi$
 Initialize replay buffer $\mathcal{B}$
for $t = 1$ **to** T **do**
 Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$,
 $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
 Store transition tuple (s, a, r, s') in $\mathcal{B}$

 Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
 $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon$, $\epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
 $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
 Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
 if $t \bmod d$ **then**
 Update ϕ by the deterministic policy gradient:
 $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
 Update target networks:
 $\theta'_i \leftarrow \tau \theta_i + (1 - \tau) \theta'_i$
 $\phi' \leftarrow \tau \phi + (1 - \tau) \phi'$
 end if
end for

Figure 1.5: TD3 algorithm as presented in [5]

In this work, TD3 is employed not to replace the conventional PI controller, but rather to **automatically tune its gains**. Instead of manually selecting the proportional and integral gains K_p and K_i , the TD3 agent learns through interaction with the system to find the optimal gain values that minimize the tracking error and improve the dynamic response. In this way, the PI controller retains its simple and well-understood structure, while TD3 provides an intelligent and adaptive mechanism for optimizing its parameters without requiring explicit knowledge of the system model.

1.8 Conclusion

This chapter introduced the theoretical foundations of reinforcement learning and established its connection to control systems. Starting from the basic agent-environment interaction and the Markov Decision Process framework, the key concepts of value functions and actor-critic methods were progressively developed. The correspondence between RL concepts and classical control systems terminology was then established, highlighting how reinforcement learning can be naturally applied to engineering control problems. Building on these foundations, two deep reinforcement learning algorithms designed for continuous action spaces were presented in detail: **DDPG**, which forms the basis of the RL controller

used in this work, and **TD3**, which extends DDPG with improved stability and is employed here for the automatic tuning of PI controller gains. The following chapter presents the simulation environment developed in **MATLAB/Simulink**, including the tools used for reinforcement learning implementation, as well as the two control architectures studied in this work: the replacement of the conventional PI controller using DDPG, and its automatic gain tuning using TD3.

Chapter 2

Direct Replacement of the PI Controller: A DDPG-Based Approach

2.1 Introduction

With the rapid evolution of AI and its different branches being implemented across engineering fields, learning about AI, especially machine learning and reinforcement learning, has become crucial for engineering students to keep up with technology. However, many conventional control systems like PID controllers are still outperforming AI methods in several domains. The real research opportunity is to investigate how conventional systems work, explore where AI can improve them, and figure out how to combine both approaches. This means understanding when to use conventional control, when to use AI, and how to create hybrid systems that merge the reliability of traditional methods with the adaptability of AI.

In this work, we investigate two approaches for transitioning toward reinforcement learning in PWM rectifier control and grid integration. The first approach is direct replacement, where an RL agent using the DDPG algorithm fully replaces the PI controller in the inner control loop for i_q current control. The second approach is adaptive gain tuning, where an RL agent acts as a supervisor that continuously adjusts the K_p and K_i parameters of the underlying PI controller based on the current system state. This adaptive approach combines RL intelligence with the stability of classical feedback control and is applied to both the i_q current controller and the grid integration PI controller using the TD3 algorithm.

Our goal is not to prove that RL outperforms conventional PI control. In fact, some results may show that PI controllers perform better. Instead, we aim to investigate how conventional and AI-based control systems can coexist and complement each other. By comparing these approaches, we develop a comprehensive understanding that enables engineers to make informed decisions about which tools to apply to specific problems and, ultimately, to design hybrid systems that leverage the reliability of classical controllers with the adaptability of AI.

This chapter focuses on the first approach: the direct replacement of the PI controller with an RL agent using the DDPG algorithm. The adaptive gain tuning approach will be explored in subsequent chapters.

2.2 System Overview: Dual-Star Synchronous Generator with PWM Rectifier

Before discussing how a reinforcement learning agent using the DDPG algorithm replaces the PI controller in the inner control loop for i_q current control, we first introduce the conversion chain consisting of the DSSG and PWM rectifier. These are two key components found in grid-connected energy conversion chains for renewable energy systems. In this section, we present the system in a simplified manner to provide a clear understanding and introduce the key concepts. We do not provide detailed mathematical modeling or equations, as these are not the main focus of our study. Instead, we focus on the functional behavior and control objectives of the system. For detailed mathematical modeling of the DSSG and PWM rectifier, the reader is referred to [1]

2.2.1 The Dual-Star Synchronous Generator (DSSG)

The Dual-Star Synchronous Generator consists of a stator with two sets of balanced three-phase windings, (a, b, c) and (a', b', c') , spatially shifted by an electrical angle γ . The rotor contains permanent magnets rotating at constant speed. Six-phase synchronous machines offer advantages in efficiency, stability, high power handling, fault tolerance, and power density compared to conventional three-phase machines.

2.2.2 The PWM Rectifier

The PWM rectifier converts AC voltage from the DSSG into regulated DC voltage with instantaneous control of the current waveform. Unlike diode-based rectifiers, PWM rectifiers maintain the absorbed current nearly in phase with the AC source voltage, achieving power factors close to unity. They also enable reactive power control and superior power quality management.

2.2.3 Cascade Control Structure

The control system employs a cascade structure with two nested control loops:

Outer Loop (Voltage Control): Maintains the DC voltage at the rectifier output at a desired reference level despite changes in load or source conditions. The voltage controller outputs a power reference, which is converted to an i_q current reference signal for the inner loop.

Inner Loop (Current Control): Tracks the i_q current reference generated by the outer loop. The i_q current directly controls the shape of the current waveform drawn from the DSSG source, ensuring it remains quasi-sinusoidal and synchronized with the source voltage. This inner loop must respond much faster than the outer loop because current dynamics are inherently faster than voltage dynamics, and fast inner loop response ensures stable cascade operation.

2.3 Why the DSSG-PWM Rectifier Conversion Chain is Suitable for RL Implementation

The DSSG-PWM rectifier conversion chain presents an ideal testbed for investigating reinforcement learning control for several reasons:

- 1. Well-Defined Control Objective** The inner current loop has a clear, measurable goal: track a reference i_q current with minimal error and fast response. This creates a well-defined reward signal that an RL agent can learn from.

- 2. Cascade Structure Enables Hybrid Approaches** The cascade structure naturally supports hybrid control: the outer loop remains a conventional PI controller, while the inner loop is replaced by an RL agent. This allows us to investigate how AI and classical control can coexist.

- 3. Rich Observation Space** The system provides multiple measurable states (current, reference current, errors) that give the RL agent sufficient information to learn

effective control policies.

4. Existing Baseline for Comparison The conventional PI controller provides a well-tuned baseline for comparison. We can evaluate whether the RL agent learns meaningful control strategies and where it offers advantages over classical methods.

5. Real-World Relevance This is a practical system used in renewable energy applications. Understanding how RL can improve or complement conventional PI control here has direct industrial relevance.

2.4 Simulink Model of the DSSG-PWM Rectifier Conversion Chain

Figure 2.1 below shows the complete Simulink model of the DSSG-PWM rectifier system used for RL integration. The model consists of the DSSG and PWM rectifier with their cascade control structure as discussed in Section 1. Additionally, the model includes several coordinate transformation blocks (temporal, Concordia plane, Park plane, and $\alpha\beta z_1 z_2 z_3 z_4$) to convert between different reference frames. These transformations enable simplified equations that facilitate the model implementation. For further details on the mathematical foundations and coordinate transformations used in this model, the reader is referred to [1].

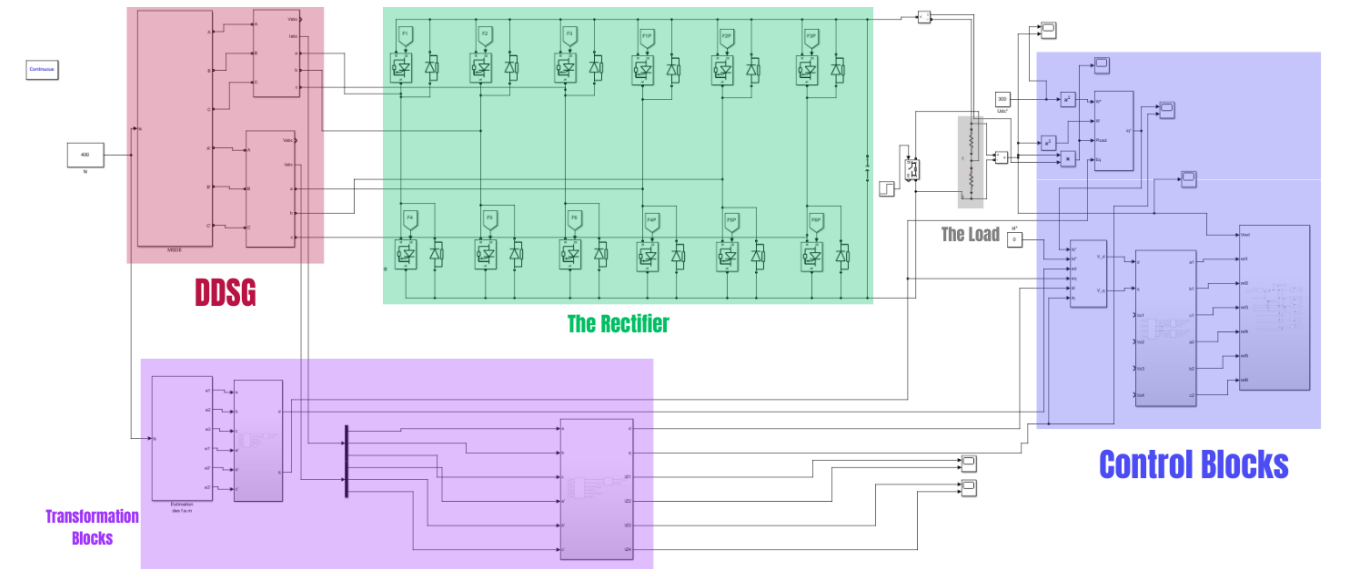


Figure 2.1: Complete Simulink model of the DSSG-PWM rectifier system

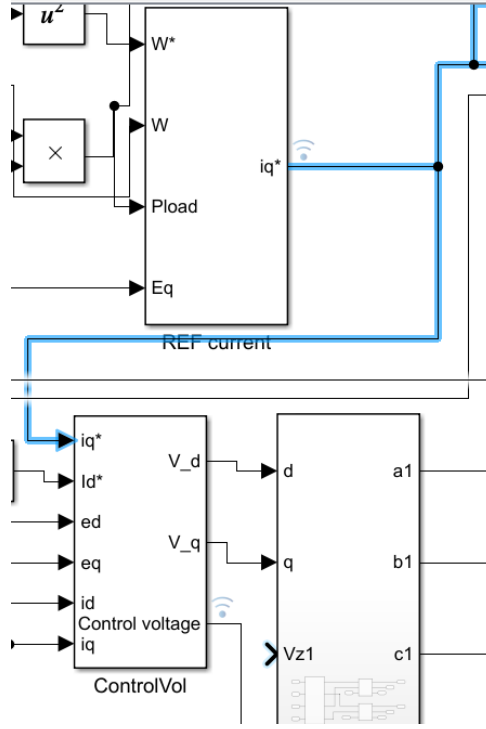


Figure 2.2: Cascade Control structure of the PWM rectifier

Figure 2.2 illustrates the cascade control structure of the PWM rectifier, which operates as follows. The outer voltage loop regulates the DC output voltage to track a reference voltage regardless of load disturbances and changes. This outer loop generates a reference power, which is then converted to i_q current reference using mathematical equations. In a Dual-Star Synchronous Generator, the transformation of EMFs from the stationary reference frame to the rotating reference frame shows that the direct component is null. Therefore, the i_d current reference is set to zero. The i_q current reference is fed into the inner loop, where a PI controller regulates the actual i_q current of the DSSG to track this reference. Similarly, an i_d current reference (which is zero) is processed through another PI controller with identical parameters to generate the corresponding control signal. By controlling both i_q and i_d currents, the system achieves a quasi-sinusoidal waveform for the phase current drawn from the DSSG. The inner loop PI controllers output regulation voltage signals denoted V_{qr} and V_{dr} . These signals are then combined with coupling signals V_{qc} and V_{dc} respectively, using a feedforward decoupling control strategy [1] to produce the final control signals V_q and V_d . These signals are subsequently transformed back into the temporal domain to generate reference signals for controlling the semiconductor switches of the PWM rectifier. In this work, the i_q current PI controller is the component that will be replaced by the RL agent in the following sections, while the i_d current controller remains conventional.

2.5 Reinforcement Learning Agent Integration in Simulink

The RL agent replaces the conventional PI controller in the inner control loop for i_q current control. This section describes how the RL agent is integrated into the Simulink model and how it communicates with the rest of the system. Figure 2.3 shows the conventional

PI controller for i_q current control. The PI controller has the following parameters: $K_i = 2722.22$ and $K_p = 14$. These parameters were chosen taking into account the system dynamics and the internal impedance of the machine, which inherently behaves as an RL filter.

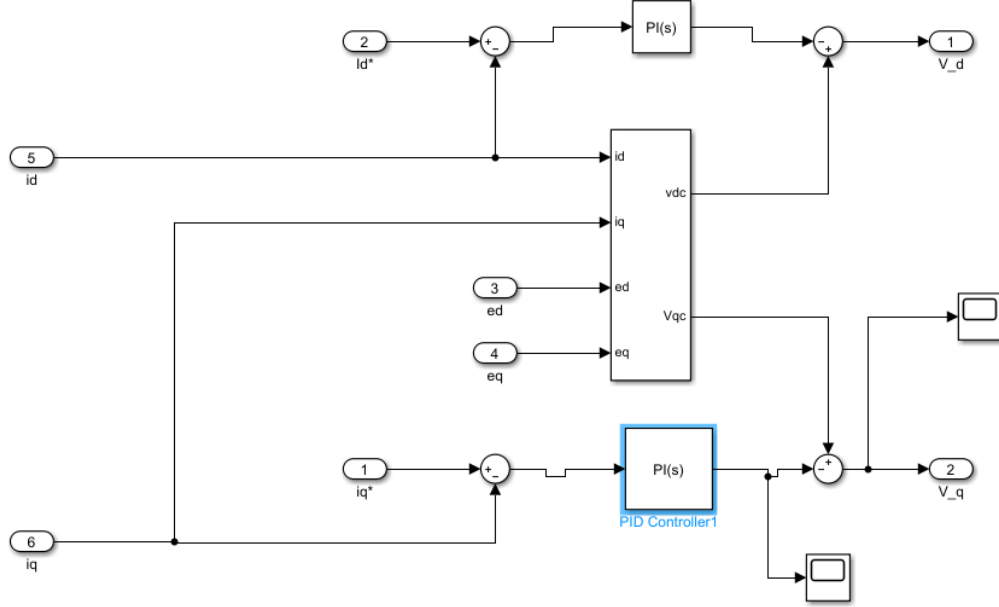


Figure 2.3: Conventional PI controller for i_q current control

2.5.1 RL Agent Block in Simulink

The RL agent is implemented using Simulink's Reinforcement Learning Toolbox, which provides a dedicated RL Agent block that can be integrated directly into the control model. This subsection describes the configuration and functionality of this block. Figure 2.4 shows the RL Agent block icon used in this work.

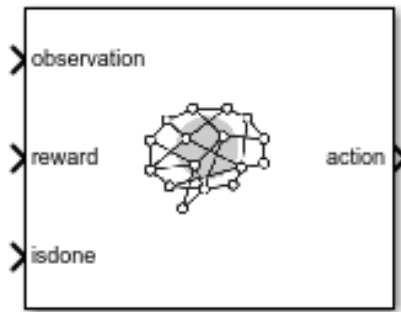


Figure 2.4: RL Agent block icon[2]

2.5.2 Block Configuration and Functionality

The RL Agent block is configured to simulate and train a reinforcement learning agent using the Simulink model as both a training and simulation environment. Figure 2.5 shows the block parameters dialog. The block has three main inputs / output ports:

- **Observation:** State information received from the system
- **Reward:** Scalar reward signal computed based on control performance
- **isDone:** Binary signal indicating episode termination
- **Action:** Control output from the agent

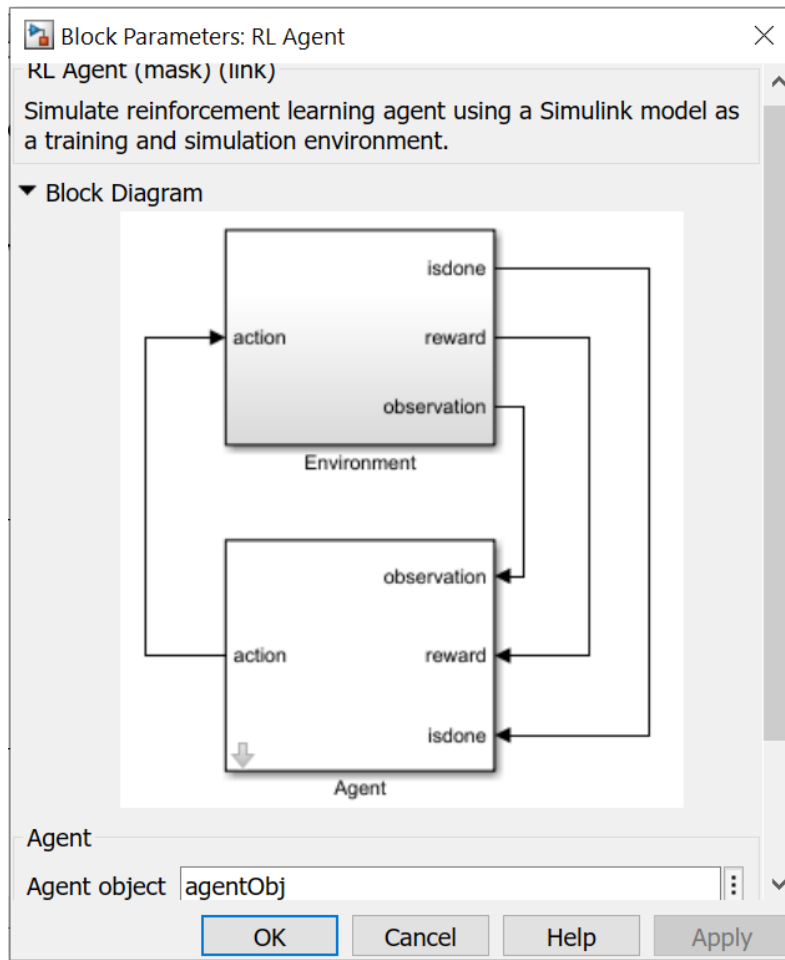


Figure 2.5: RL Agent block parameters configuration showing agent object and environment structure

2.5.3 Agent Object Configuration

The agent object (agentObj) is configured with the DDPG algorithm parameters and is passed to the RL Agent block. The block uses this agent to:

- Receive observations from the Simulink environment at each time step
- Compute actions based on the learned policy (or exploration during training)
- Receive reward signals that guide the learning process
- Update network weights during training phases

2.6 Real-Time Execution

The RL Agent block executes synchronously with the Simulink model at the specified sampling time, ensuring that the agent’s decisions are updated consistently with the inner control loop dynamics. During training, the block collects experience (observations, actions, rewards) and updates the DDPG networks. During testing, the block operates in control mode, using only the learned policy without updating the networks.

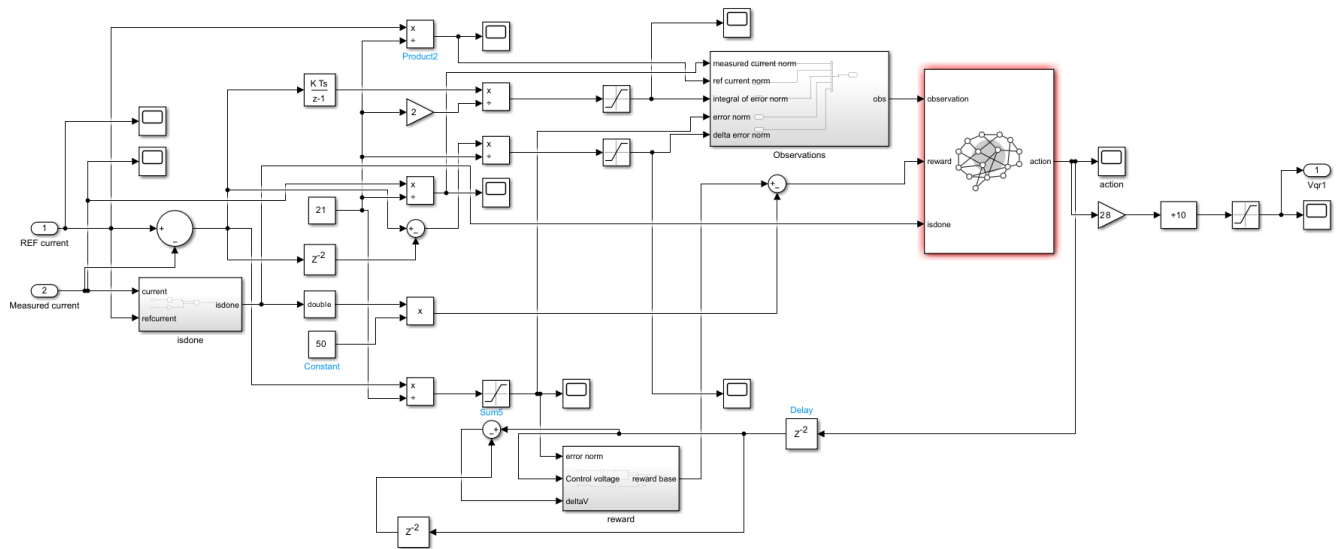


Figure 2.6: RL agent block integration with observation, reward, and isDone subsystems

Figure 2.6 illustrates the complete Simulink system architecture for replacing the conventional PI controller with the RL agent block. The diagram shows the different subsystems that work together: the observation subsystem that computes state information, the reward subsystem that calculates performance feedback, and the isDone subsystem that determines episode termination conditions. In the following sections, we will provide detailed descriptions of the design and structure of each subsystem, explaining the choices made for observation selection, reward function design, and episode termination criteria.

2.7 Observation Space, Action Space, and Reward Design

This section details the design of the three critical components that define how the RL agent learns and interacts with the DSSG-PWM conversion chain: the observation space (what the agent observes), the action space (what the agent can control), and the reward function (how the agent is evaluated).

2.7.1 Observation Space

The observation space defines the state information available to the RL agent at each time step. The observations enable the agent to understand the current system condition and make informed control decisions. The choice of observations is really important because the type and quality of the signals that you feed the RL agent block will determine how well the RL agent has clear and enough information to understand the system current conditions. For our study case, we designed the observation subsystem to contain five signals that we judged enough and gave us stable and good learning. The five signals are: measured current i_q , reference current $i_{q,ref}$, error e_q , integral of error $\int e_q$, and delta error Δe_q (rate of change of error).

Signal Selection Justification

- **Measured current i_q :** Provides direct feedback of the actual system state
- **Reference current $i_{q,ref}$:** Tells the agent what target to track
- **Error $e_q = i_{q,ref} - i_q$:** Indicates tracking performance and control urgency
- **Integral of error $\int e_q$:** Helps eliminate steady-state error, similar to the integral term in PI controller.
- **Delta error Δe_q :** Shows error rate, enabling predictive control and derivative action

Signal Normalization

All these signals are normalized using a maximum value of the reference current that was estimated based on the system operating conditions. The normalization keeps approximately all the signals fed into the observation subsystem in the range $[-1, 1]$. This is important in neural networks because it ensures all inputs have similar scales, which improves training stability, accelerates convergence, and prevents large input values from saturating the activation functions. Additionally, without normalization, inputs with

larger magnitudes can dominate the learning process and introduce bias in the network weights. When some inputs are much larger than others, the network may give them disproportionate importance during training, causing the agent to overweight certain observations and underweight others. Normalization ensures that each signal contributes fairly to the decision-making process, allowing the agent to learn balanced and robust control policies.

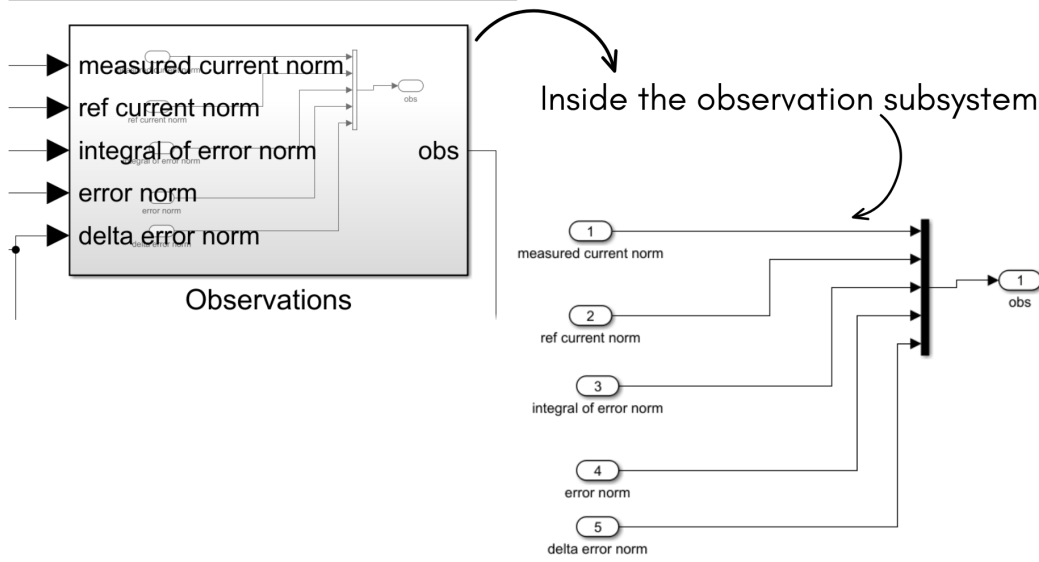


Figure 2.7: Observation subsystem in Simulink showing the five signals

Figure 2.7 shows the observation subsystem implemented in Simulink. The subsystem contains all five signals discussed above, each properly computed and normalized before being fed to the RL Agent block. The chosen value for normalization was 21, and the signals were all divided by this value except for the error integral, which was normalized by $21 \times 2 = 42$. The error integral is normalized by a larger factor because it accumulates error over time and can grow much larger than the instantaneous error signal. By dividing by 42 instead of 21, we ensure that the integral term remains in the same normalized range $[-1, 1]$ as the other signals, preventing it from dominating the observation space due to its accumulated nature.

2.7.2 Action Space

The action space defines what the RL agent can control. In this work, the agent outputs a continuous control signal denoted V_{qr} . This signal is then combined with a coupling signal V_{qc} . These two signals (V_{qc} , V_{qr}) are decoupled using a feedforward decoupling control strategy [1], then recombined to produce the final control signal V_q . This signal, along with another control signal V_d that is produced using a similar process with a conventional PI controller, is then transformed to the temporal domain to generate reference signals used to control the semiconductor switches of the PWM rectifier.

Action Scaling

The action signal that the RL agent outputs is in the range of $[-1, 1]$. That is why proper and optimal scaling is needed in order to scale the network output range from

$[-1, 1]$ to the physical voltage range required by the system. For action scaling, we used an affine scaling method. Affine scaling is a linear transformation that maps the normalized action range to the physical voltage range, ensuring that the agent's output can be directly applied to control the PWM rectifier switches. This method is optimal because it preserves the proportionality between the normalized action and the physical control signal, allowing the agent to have uniform control authority across the entire range of possible voltages. The affine scaling is performed using the equation $V_q^{physical} = V_q^{agent} \times \text{scale} + \text{offset}$, where the scale factor and offset convert the $[-1, 1]$ range to the required voltage range. The values for the affine scaling were chosen after several tests and estimations of the DSSG-PWM rectifier conversion chain operating conditions.

After affine scaling, the scaled voltage signal is then passed through a saturation block. The saturation block is useful because it enforces hard physical limits on the control voltage, preventing the signal from exceeding the maximum and minimum voltage values that the PWM rectifier switches can handle. This protection is especially important for RL agents because the neural network outputs can be affected by numerical errors, computational uncertainties, or unexpected system conditions that cause the scaled voltage to exceed the physically realizable range. By applying saturation, we ensure that the control signal is clipped to the valid voltage bounds, protecting the system from damage and maintaining system stability even when the RL agent's output is outside the expected range.

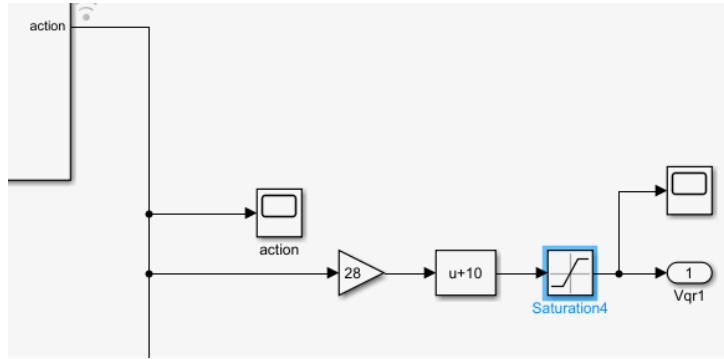


Figure 2.8: Simulink model showing affine scaling and saturation block for action signal processing

Figure 2.7 shows the Simulink implementation of the affine scaling and saturation block used to process the RL agent's action signal.

2.7.3 Reward Function Design

The reward function is the key signal that guides the agent's learning. It must incentivize the desired behavior: fast and accurate tracking of the i_q current reference.

Reward Formulation

The agent's goal is to maximize the total amount of reward it receives. This means maximizing the cumulative reward in the long run. If we want the agent to do something for us, we must provide rewards to it in such a way that in maximizing them, the agent will also achieve our goals. That is why it is important that the reward we set up truly indicates what we want to accomplish. For our study, the reward function at each time step was chosen based on [3]. The reward signal is computed as:

$$r_t = -(Q_1 \cdot e_{iq}^2 + Q_2 \cdot V_{qr}^2 + Q_3 \cdot \Delta V_{qr}^2) - 50 \cdot d \quad (2.1)$$

where Q_1 , Q_2 , and Q_3 are weighting factors, e_{iq} is the iq current tracking error, V_{qr} is the control voltage (action), ΔV_{qr} is the rate of change of voltage, and d is the episode termination penalty.

The negative sign here gives us what is called the cost function. Maximizing the reward function is the same as minimizing the cost function. Since we used the cost function here, the agent's goal would be to minimize it, which drives the agent to reduce tracking error, limit control effort, and maintain smooth voltage changes, ultimately achieving our control objectives.

Reward Justification

We can divide the chosen reward structure that we used into two main parts. The first part, which consists of $-(Q_1 \cdot e_{iq}^2 + Q_2 \cdot V_{qr}^2 + Q_3 \cdot \Delta V_{qr}^2)$, we call the reward base. The weighting factors are $Q_1 = 15$, $Q_2 = 0.1$, and $Q_3 = 0.15$. These parameters were chosen after several tests and were shown to give accurate and stable results. Choosing these values (also called hyperparameters) and tuning them is not the easiest thing to do, as hyperparameter tuning is one of the main challenges in reinforcement learning. Especially for the DDPG algorithm, which is really sensitive to hyperparameters, proper tuning is critical. Small changes in Q_1 , Q_2 , or Q_3 can significantly affect training convergence, learning speed, and final control performance. Finding optimal hyperparameter values often requires extensive experimentation and testing, which is a well-known challenge in deep reinforcement learning research.

The second part is the episode termination penalty $-50 \cdot d$, where d is basically the isDone signal, which is a binary signal converted to double for proper use and then multiplied by -50 . The isDone signal is just a binary signal that indicates when an episode should terminate. The termination condition must be set up based on some criteria; for example, when the maximum episode length is reached or when the system violates safety constraints. In our study, we chose $\text{isDone} = 1$ when $i_q \geq 2 \cdot i_{q,ref}$. Figures 2.9 and 2.10 below show the Simulink base reward subsystem and the final reward fed into the RL agent block.

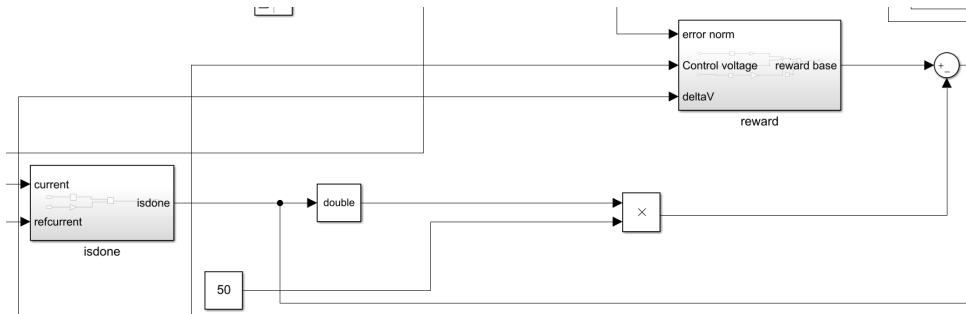


Figure 2.9: Final reward signal fed into the RL agent block

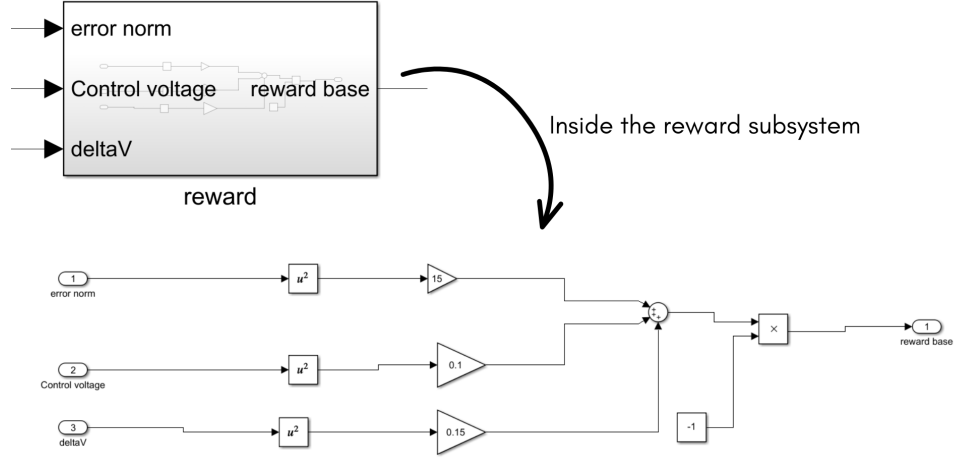


Figure 2.10: Simulink base reward subsystem

2.8 RL Agent Configuration (DDPG) in MATLAB

This section details the implementation of the Deep Deterministic Policy Gradient (DDPG) algorithm in MATLAB for controlling the iq current in the DSSG-PWM rectifier system. The implementation consists of seven main components: (1) observation and action specifications that define the agent-environment interface, (2) the critic network that evaluates state-action pairs, (3) the actor network that generates control actions, (4) the DDPG agent configuration with algorithm parameters, (5) the training configuration specifying episode and batch parameters, (6) the training execution loop, and (7) visualization of training results.

Each component is implemented as a distinct MATLAB code block with a specific purpose. Together, these blocks create a complete reinforcement learning system that learns to output a control voltage that is properly scaled and combined with other signal, then transformed into the temporal frame to be used as a reference signal for controlling the switching time of the PWM switches.

2.8.1 Block 1: Observation and Action Specifications

This block represents the RL agent-environment interface. It is what enables these two to communicate. These specifications define exactly what information the agent receives about the environment (observations) and what it can control (actions).

- **Defining the Observation Space**

The observation space specifies what information the RL agent receives from the system at each time step. It is defined using the following MATLAB code:

```
obsInfo = rlNumericSpec([5 1], ...
    'LowerLimit', [-inf -inf -inf -inf -inf]', ...
    'UpperLimit', [ inf   inf   inf   inf   inf]');

obsInfo.Name = 'observations';
numObservations = obsInfo.Dimension(1);
```

The agent receives five observations organized as a column vector $[5 \times 1]$: iq current reference, measured iq current, tracking error, integral of error, and rate of change of error (as discussed in Section 2.7). The bounds are set to unbounded ($\pm\infty$) because theoretically these signals could become very large. However, in practice they are normalized to $[-1, 1]$ in the Simulink subsystem before reaching the agent. The variable `numObservations = obsInfo . Dimension (1) = 5` is extracted and stored for later use when building the neural network input layer.

- **Defining the Action Space**

The next part creates the action specification:

```
actInfo = rlNumericSpec([1 1], ...
    'LowerLimit', -1, ...
    'UpperLimit', 1);

actInfo.Name = 'control_voltage';
numActions = actInfo.Dimension(1);
```

The agent outputs a single continuous action: the control voltage V_{qr} constrained to the range $[-1, 1]$. This range is standard for neural networks using tanh activation function, which naturally outputs values between -1 and 1 . This prevents unbounded outputs that could damage hardware. The normalized action is later scaled to the actual physical voltage range required by the PWM rectifier through affine scaling. The variable `numActions = actInfo . Dimension (1) = 1` is extracted for use when building the neural network output layer.

- **Creating the Simulink Connection**

The specifications are then connected to the Simulink model:

```
env = rlSimulinkEnv( ...
    'RLexp', ...
    'RLexp/ControlVol/RLsubsyst/RLAgent', ...
    obsInfo, ...
    actInfo);
```

This creates the communication bridge between MATLAB and Simulink. The first argument 'RLexp' is the Simulink model file name. The second argument 'RLexp/ControlVol/RLsubsyst/RL Agent' is the exact hierarchical path to the RL Agent block inside Simulink model. The `obsInfo` and `actInfo` define the interface. Once created, this environment enables closed-loop interaction: the agent sends actions to Simulink, Simulink simulates the system response, and returns observations and rewards back to the agent.

- **Setting Timing Parameters**

```

Ts = 1e-3;      % Sampling time = 0.001 seconds
Tf = 0.5;      % Episode duration = 0.5 seconds

rng(0)         % Random seed for reproducibility

```

- **Sampling Time Selection**

The sampling time $T_s = 1 \times 10^{-3}$ seconds (1 millisecond) corresponds to a control frequency of 1 kHz. The choice of this sampling time is justified by analyzing the DSSG internal dynamics.

When the PWM rectifier is connected to an AC source with negligible internal impedance, it is generally necessary to integrate an R-L filter between the AC source and the PWM rectifier to ensure harmonic filtration. However, in the case of the DSSG, this requirement is relaxed since the internal impedance of the machine inherently behaves as an R-L filter, thus eliminating the need for an external filtering stage.

The equivalent RL filter is formed by:

- R : Per-phase resistance of the DSSG stator winding
- L_f : Leakage inductance of the DSSG
- M_{ss} : Mutual inductance between the two stator winding sets at zero angular displacement

The time constant of this RL filter is given by:

$$\tau = \frac{L}{R} \approx 5.14 \times 10^{-3} \text{ seconds} = 5.14 \text{ ms} \quad (2.2)$$

where $R = 1.4 \Omega$ and $L = L_f + 3M_{ss}$. For more detailed explanations about the DSSG modeling and equations, the reader is referred to [1].

For effective control of a system with this time constant, the sampling time should be significantly smaller than the system time constant. The chosen sampling time $T_s = 1$ ms is approximately 5.14 times smaller than the system time constant $\tau = 5.14$ ms. Additionally, the 5% settling time (time for the system to settle within 5% of its final value) is:

$$t_s(5\%) \approx 3 \times \tau = 3 \times 5.14 \text{ ms} = 15.42 \text{ ms} \quad (2.3)$$

With the selected sampling time, approximately 15 samples are obtained during the settling period:

$$N_{\text{samples}} = \frac{15.42 \text{ ms}}{1 \text{ ms}} = 15 \text{ samples} \quad (2.4)$$

This combination of a fast sampling rate relative to both the system time constant and settling time provides several advantages:

1. **Adequate temporal resolution:** The sampling period is sufficiently short to capture the transient dynamics of the R-L filter. Approximately 15 measurements are collected during the complete settling process, providing detailed information about the system response for learning and control purposes.
2. **Control effectiveness:** Since the sampling interval is significantly shorter than the dominant system time constant, the reinforcement learning agent can observe the system and update its control actions multiple times during the transient response, improving its ability to regulate the system effectively.
3. **Accurate representation of system dynamics:** The selected sampling frequency of 1 kHz is sufficiently high relative to the dominant dynamics of the R-L filter, allowing accurate representation of the transient response while reducing discretization errors and supporting stable control performance.

Finally, the sampling time of the reinforcement learning agent must be consistent with the discrete-time integrator used for calculating e_{iq} in the Simulink model. Any mismatch between these sampling rates may result in inaccurate state observations, delayed control actions, and degraded training or control performance.

- **Episode Duration**

The episode duration $T_f = 0.5$ seconds means each training episode lasts 0.5 seconds. This corresponds to:

$$\text{Number of steps per episode} = \left\lceil \frac{T_f}{T_s} \right\rceil = \left\lceil \frac{0.5}{0.001} \right\rceil = 500 \text{ steps} \quad (2.5)$$

This duration is long enough for the system to respond to control actions and approach steady-state, allowing the agent to observe the consequences of its decisions. At the same time, 0.5 seconds per episode keeps training time reasonable: $250 \text{ episodes} \times 0.5 \text{ seconds} = 125 \text{ seconds}$ total training time, enabling practical experimentation and iteration.

- **Random Seed**

```
rng(0)
```

This command sets the random number generator seed to 0. The effect is:

- Neural network weights are initialized with the same random values every time the code runs
- All stochastic processes follow the same random sequence
- Running the code multiple times produces identical results
- Enables reproducibility for debugging and result verification

For robustness testing, the code can be run with different seeds (`rng(1)`, `rng(2)`, etc.) to verify that results are not dependent on a specific random initialization.

2.8.2 Block 2: Critic Network Architecture

This block constructs the critic network, which is a neural network that evaluates the quality of state-action pairs. The critic learns to estimate the Q-value, representing how good a particular action is in a given state.

- **Critic Network Structure**

The critic network has three distinct pathways that process observations and actions separately before combining them:

```
statePath = [
    imageInputLayer([5 1 1], ...
        'Name','netObsIn', ...
        'Normalization','none')
    fullyConnectedLayer(64, ...
        'Name','CriticStateFC1')
    reluLayer('Name','CriticStateRelu1')
    fullyConnectedLayer(64, ...
        'Name','CriticStateFC2')
];

actionPath = [
    imageInputLayer([1 1 1], ...
        'Name','netActIn', ...
        'Normalization','none')
    fullyConnectedLayer(64, ...
        'Name','CriticActionFC1')
];

commonPath = [
    additionLayer(2, 'Name','add')
    reluLayer('Name','CriticCommonRelu')
    fullyConnectedLayer(1, ...
        'Name','CriticOutput')
];
```

The critic network consists of three pathways:

- **State Path:** Processes the five observations through two fully connected layers of 64 neurons each, with ReLU activation between them. ReLU (Rectified Linear Unit) is an activation function that simply passes positive values unchanged and converts negative values to zero. This introduces non-linearity to the network, allowing it to learn complex relationships between observations that simple linear calculations cannot capture. This pathway extracts features from the current system state.
- **Action Path:** Processes the single action through one fully connected layer of 64 neurons. This pathway extracts features from the agent's action.
- **Common Path:** Combines the outputs from both pathways using an addition layer, passes through ReLU activation, and outputs a single value representing the Q-value (the estimated quality of the state-action pair).

• Network Assembly and Configuration

The three pathways are assembled into a single network graph and connected:

```
criticNetwork = layerGraph();
criticNetwork = addLayers(criticNetwork, statePath);
criticNetwork = addLayers(criticNetwork, actionPath);
criticNetwork = addLayers(criticNetwork, commonPath);
criticNetwork = connectLayers(criticNetwork, ...
    "CriticStateFC2", ...
    "add/in1");
criticNetwork = connectLayers(criticNetwork, ...
    "CriticActionFC1", ...
    "add/in2");
criticNetwork = dlnetwork(criticNetwork);
```

The state pathway output is connected to the first input of the addition layer, and the action pathway output is connected to the second input. The addition layer combines both feature representations, allowing the network to learn how state and action interact to determine Q-value.

• Critic Representation and Options

The critic is then configured with learning parameters:

```
criticOpts = rlRepresentationOptions( ...
    'LearnRate', 5e-4, ...
    'GradientThreshold', 1);

critic = rlRepresentation( ...
    criticNetwork, ...
    obsInfo, ...
    actInfo, ...
    'Observation', {'netObsIn'}, ...
    'Action', {'netActIn'}, ...
    criticOpts);
```

The learning rate is set to 5×10^{-4} (lower than typical to ensure stable learning of value estimates). The gradient threshold of 1 prevents gradient explosion by clipping large gradients. These parameters help the critic network converge smoothly during training.

• Network Visualization

The network structure can be displayed and plotted for verification:

```
disp(criticNetwork.Layers)
disp(criticNetwork.Connections)
disp(criticNetwork)
criticLG = layerGraph(criticNetwork);
plot(criticLG)
```

These commands allow visualization of the network layers, their connections, and the overall architecture, ensuring the network is constructed correctly before training.

2.8.3 Block 3: Actor Network Architecture

This block constructs the actor network, which learns the control policy. The actor takes observations as input and directly outputs control actions. Unlike the critic network with separate pathways, the actor uses a sequential architecture.

- **Actor Network Structure**

```
actorNetwork = [
    imageInputLayer([numObservations 1 1], ...
        'Normalization','none', ...
        'Name','State')
    fullyConnectedLayer(64, ...
        'Name','actorFC1')
    reluLayer('Name','actorRelu1')
    fullyConnectedLayer(64, ...
        'Name','actorFC2')
    reluLayer('Name','actorRelu2')
    fullyConnectedLayer(numActions, ...
        'Name','ActionFC')
    tanhLayer('Name','ActionTanh')
];
```

The actor network is a straightforward sequential network with the following layers:

1. **Input Layer:** Accepts the five observations from the system state.
2. **First Fully Connected Layer:** 64 neurons that process the observations.
3. **First ReLU Activation:** Introduces non-linearity by keeping positive values and zeroing negative values. This allows the network to learn complex relationships between observations that simple linear transformations cannot capture.
4. **Second Fully Connected Layer:** Another 64 neurons that further refine the feature representation.
5. **Second ReLU Activation:** Further non-linear processing for deeper feature extraction.
6. **Output Fully Connected Layer:** Maps the 64 hidden features to the single action (control voltage V_{qr}).
7. **Tanh Activation:** The output layer applies a hyperbolic tangent function, which naturally constrains the action output to the range $[-1, 1]$. This ensures the network always produces valid control commands within the specified bounds.

- **Actor Configuration and Learning Rate**

```
actorOptions = rlRepresentationOptions( ...
    'LearnRate', 1e-4, ...
    'GradientThreshold', 1);

actor = rlDeterministicActorRepresentation( ...
    actorNetwork, ...
    obsInfo, ...
    actInfo, ...
    'Observation', {'State'}, ...
    'Action', {'ActionTanh'}, ...
    actorOptions);
```

The actor is configured with a learning rate of 1×10^{-4} , which is significantly lower than the critic's learning rate of 5×10^{-4} .

Why Different Learning Rates? The learning rate difference between actor and critic is critical for DDPG stability:

- **Critic (Learning Rate = 5×10^{-4}):** The critic estimates values, which are abstract numbers. Even if value estimates are temporarily inaccurate, the system behavior remains stable because the critic's estimates do not directly affect the physical system. Therefore, the critic can learn faster.
- **Actor (Learning Rate = 1×10^{-4}):** The actor directly outputs control actions that will be scaled properly and used in the PWM rectifier. Aggressive policy updates can cause erratic control behavior, instability, or even safety issues. Therefore, the actor learns at 1/5 the rate of the critic to ensure smooth, gradual policy improvements.
- **Network Visualization**

The actor network structure can be displayed and visualized:

```
disp(actorNetwork)
actorLG = layerGraph(actorNetwork);
plot(actorLG)
```

Figure 2.10 below shows the complete actor network structure. The network follows a linear sequential flow where observations pass through increasingly complex feature extraction layers before producing the final control action.

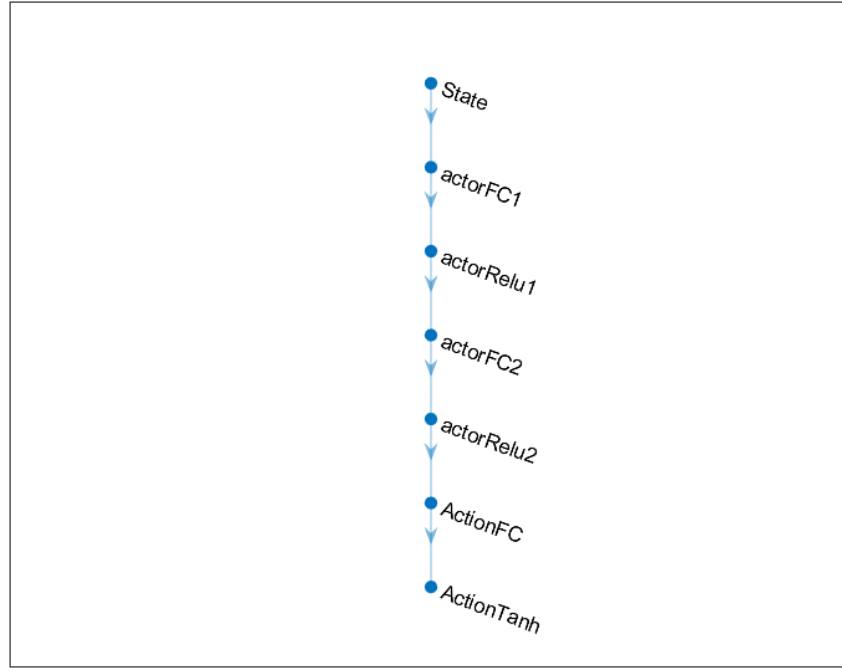


Figure 2.10: Actor Network Architecture

2.8.4 Block 4: DDPG Agent Configuration and Initialization

This block configures the DDPG algorithm parameters and creates (or loads) the agent. The agent combines the actor and critic networks with hyperparameters that control learning behavior. ‘

- **DDPG Agent Options**

```

agentOpts = rlDDPGAgentOptions;
agentOpts.SampleTime = Ts;
agentOpts.TargetSmoothFactor = 1e-3;
agentOpts.DiscountFactor = 0.99;
agentOpts.MiniBatchSize = 64;
agentOpts.ExperienceBufferLength = 1e6;
agentOpts.NoiseOptions.Variance = 0.03;
agentOpts.NoiseOptions.VarianceDecayRate = 5e-5;
    
```

The DDPG agent options control how the algorithm learns:

- **SampleTime** ($T_s = 1 \times 10^{-3}$ seconds): The time interval at which the agent receives observations and takes actions. This must match the Simulink model sample time for synchronization.
- **TargetSmoothFactor** (1×10^{-3}): Controls how quickly the target networks update. Target networks are copies of the actor and critic that update slowly to stabilize learning. A small factor (0.001) means target networks update slowly, providing stable targets for learning.

- **DiscountFactor (0.99)**: A value between 0 and 1 that weighs immediate rewards versus future rewards. A value of 0.99 means the agent values future rewards nearly as much as immediate rewards, encouraging long-term planning.
 - **MiniBatchSize (64)**: The number of experience samples used in each training update. Larger batches provide more stable gradient estimates but require more memory.
 - **ExperienceBufferLength (1×10^6)**: The maximum number of experiences stored in memory. The agent learns from random samples of past experiences, breaking correlations between consecutive samples.
 - **NoiseVariance (0.03)**: The standard deviation of exploration noise added to actions. The agent explores by adding random noise to its actions, preventing convergence to suboptimal policies. A variance of 0.03 provides moderate exploration.
 - **VarianceDecayRate (5×10^{-5})**: The noise variance decreases over time at this rate. As training progresses, exploration noise gradually decreases, allowing the agent to exploit learned knowledge.
- **Agent Creation and Loading Logic**

```
if exist('FinalAgent.mat','file')
    load('FinalAgent.mat','agentObj');
    disp(" Loaded existing agent (continuing training)");
else
    agentObj = rlDDPGAgent(actor,critic,agentOpts);
    disp(" Created new agent");
end
```

This conditional logic provides two modes of operation:

1. **Continuing Training**: If a saved agent file `FinalAgent.mat` exists in the current directory, it is loaded. This allows resuming training from a previous state, preserving all learned knowledge. This is useful when training is interrupted or when we want to refine an already-trained agent.
2. **Fresh Training**: If no saved agent exists, a new agent is created from the actor and critic networks with the specified options. This starts training from random initialization.

When to use each mode:

- **Create new agent**: First training run or when starting fresh experiment
- **Load existing agent**: Resume interrupted training, fine-tune hyperparameters, or compare different training approaches

2.8.5 Block 5: Training Configuration and Execution

This block configures the training process and executes the learning loop.

- **Training Parameters**

```
scoreWindow = 20;
maxsteps = ceil(Tf/Ts);
maxepisodes = 250;

trainOpts = rlTrainingOptions( ...
    'MaxEpisodes', maxepisodes, ...
    'MaxStepsPerEpisode', maxsteps, ...
    'Verbose', true, ...
    'Plots','training-progress', ...
    'StopTrainingCriteria','EpisodeCount', ...
    'StopTrainingValue', maxepisodes);
```

The training configuration specifies 250 episodes of training, with 500 steps per episode (based on $T_f = 0.5$ seconds and $T_s = 0.001$ seconds). The score window averages rewards over 20 episodes to smooth the training curve. Training progress is displayed in real-time plots, and training stops after reaching the maximum episode count.

- **Training Execution and Agent Management**

```
doTraining = true;

if doTraining
    disp(" Training started for 250 episodes...");
    trainingStats = train(agentObj, env, trainOpts);
    disp(" Training finished for 250 episodes...");
    save('FinalAgent.mat','agentObj');
    disp(" Final agent saved");
else
    load('FinalAgent.mat','agentObj');
    disp(" Loaded pretrained agent");
end
```

The training loop can be toggled with the `doTraining` flag. When true, the agent trains for 250 episodes and the trained agent is saved to `FinalAgent.mat`. When false, a previously trained agent is loaded from the saved file, allowing rapid evaluation without retraining.

- **Training Progress Visualization**

```
figure;
plot(trainingStats.AverageReward,'LineWidth',1.5);
grid on;
xlabel('Episode');
ylabel('Average_Reward_for_250_ep');
title('Training_Progress_(Reward_for_250_ep)');
```

After training completes, the average reward over training episodes is plotted. A rising curve indicates improving agent performance, while a plateau suggests convergence to a learned policy.

2.9 RL controller Implementation and Initial Training Challenges

In the proposed implementation, the conventional PI controller was removed and replaced by a complete RL-based control subsystem. Upon running the simulation for the very first time, a critical issue was immediately observed: the reference current i_q^* exhibited a continuous and unbounded decrease, dropping linearly from zero to approximately (-400A) over 0.5 seconds, as shown in figure(2.11) below. At the very start of training, the RL agent has no learned policy and therefore outputs random V_{qr} values that go through a complete processing chain (scaled, combined with decoupling terms), and finally used in the PWM control of the rectifier. Since these signals are large and uncontrolled at the beginning, they cause the rectifier to operate incorrectly, driving the outer PI error in the negative direction and causing its integrator to accumulate negatively and continuously. The resulting negative output of the outer PI, once converted into i_q^* , produces the unbounded decreasing values observed in figure(2.11) a problem commonly known as integrator windup, which is particularly critical in cascaded control architectures where inner loop instability directly propagates to the outer loop.

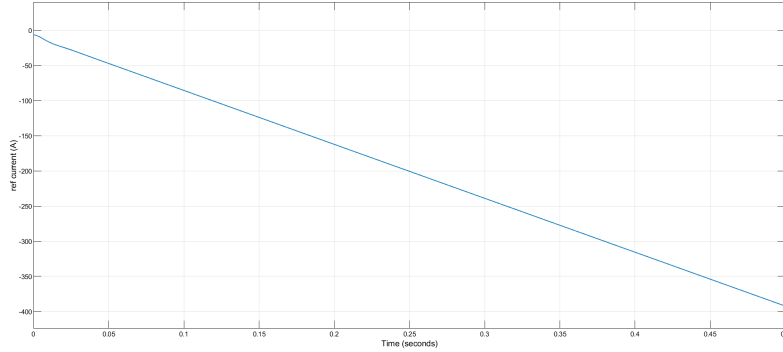


Figure 2.11: i_q reference divergence

To address this, an anti-windup back-calculation mechanism was introduced into the outer PI controller. This is fully justifiable given that at the start of training, the RL agent acts entirely randomly with no prior knowledge of the system, and without any mechanism to limit the accumulation of the integral term, the outer PI integrator keeps accumulating error indefinitely. Anti-windup is a well-established solution in control engineering, specifically designed to prevent the integrator from accumulating beyond the actuator saturation limits. By clamping the integral term when the output exceeds the defined bounds, the anti-windup mechanism stabilizes the training environment, ensuring that the RL agent receives meaningful and bounded i_q^* reference signals rather than interacting with a diverging and uncontrollable reference.

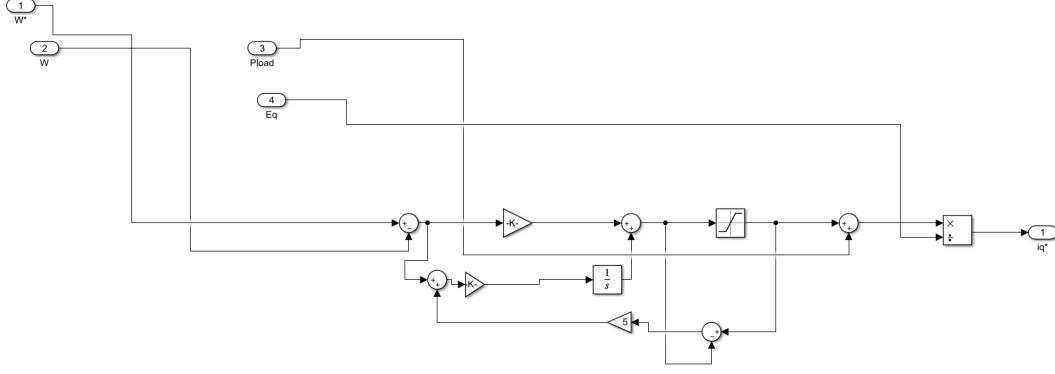


Figure 2.12: iq reference divergence

This behavior highlights an important and often underestimated challenge: replacing a conventional PI controller with an RL-based controller is not a straightforward substitution. Unlike a PI controller which operates deterministically from the very first instant, an RL agent begins with no knowledge of the system and must learn through interaction. This fundamental difference means that the surrounding control architecture must be carefully adapted to tolerate and accommodate the exploratory and initially random behavior of the agent, making proper environment design as critical as the agent design itself.

2.10 Training Results and discussion

This section presents the training results of the DDPG agent designed to directly replace the PI controller in the inner control loop. The agent was trained for a sufficient number of episodes until convergence, and the obtained results are analyzed and discussed in terms of learning stability, current tracking performance, and overall behavior compared to the classical PI controller.

2.10.1 Current Tracking Performance and Learning Evolution

The training conditions were set such that the resistive load within the DSSG-PWM rectifier conversion chain was initially fixed at $40\ \Omega$, then reduced by 25 per cent to $30\ \Omega$ at $t = 0.25s$, introducing a sudden load variation. The agent was trained under this exact scenario in order to expose it to dynamic operating conditions rather than a fixed steady-state environment, ensuring that the learned control policy is capable of handling load disturbances and adapting its response accordingly. The three figures below illustrate the evolution of the agent throughout the training process. As shown in Figure (2.13), the agent was still exposed to a new environment with no prior knowledge, which explains the chaotic behavior of the tracked i_q , a characteristic typically observed during the early episodes of training. It should be noted that despite the agent having no prior knowledge of the system, the reference current i_q^* did not exhibit an unbounded decrease during these early episodes, which is attributed to the anti-windup mechanism integrated in the outer voltage control loop that prevents the integrator from saturating and keeps i_q^* within acceptable bounds. After several episodes, as shown in Figure (2.14), the agent began to track the reference current i_q^* but with considerable oscillations. As training progressed, these oscillations gradually diminished and the agent started to converge toward the

reference signal. By the final episodes, as shown in Figure (2.15), the agent demonstrates a significantly improved tracking behavior, successfully following the reference current even at the load disturbance introduced at $t = 0.25\text{s}$. Furthermore, as the agent gradually gained knowledge about the overall system dynamics, the shape of i_q^* remained approximately stable and closely resembled the reference generated by the outer loop when the inner loop was still governed by the conventional PI controller, which further validates the consistency of the training environment and confirms that the agent has converged to a stable and sufficiently learned control policy.

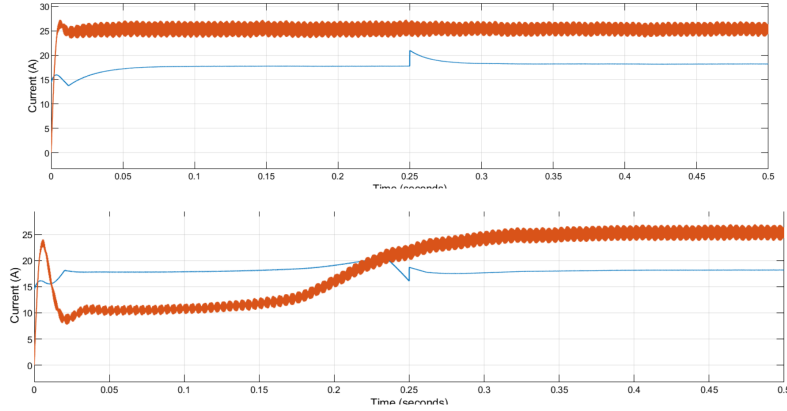


Figure 2.13: Current tracking performance (early episodes)

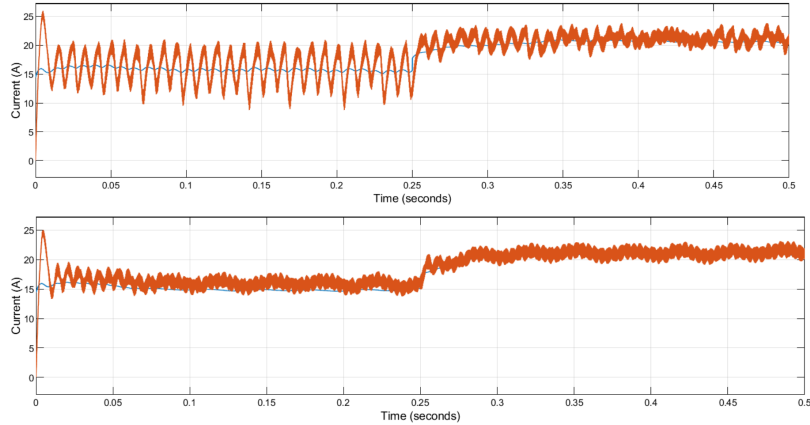


Figure 2.14: Current tracking performance (intermediate episodes)

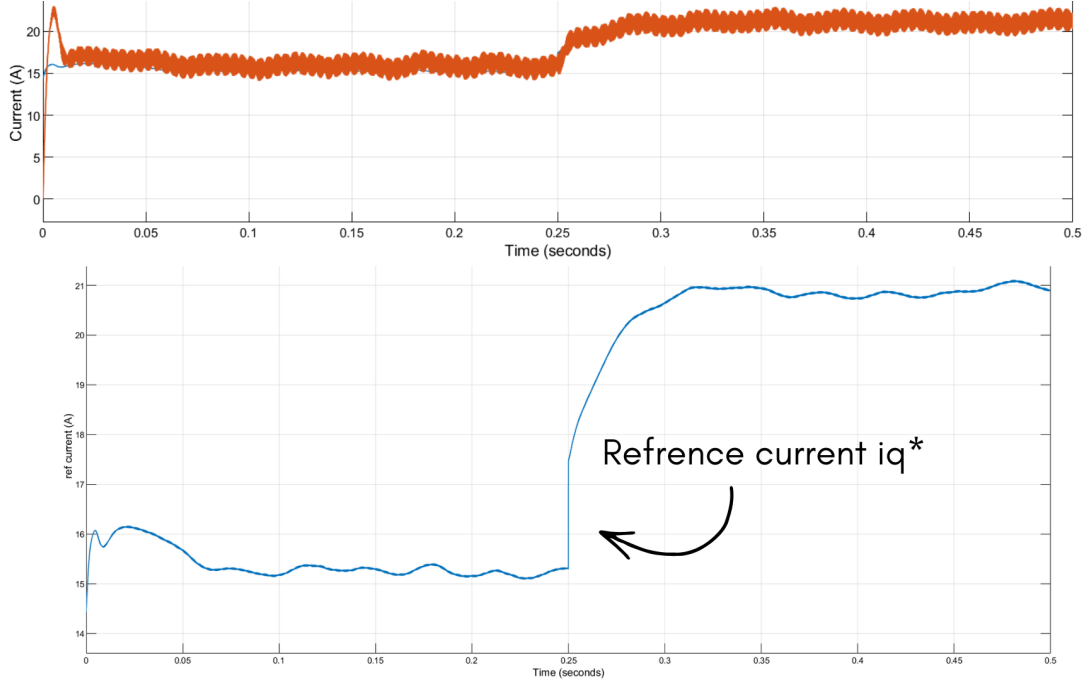


Figure 2.15: Current tracking performance (final episodes)

Figure (2.16) shows a zoomed view of two regions of the tracking performance of the agent after convergence. The left zoom focuses on the transition region at $t = 0.25\text{s}$ where the load step change occurs, and it can be seen that the agent responds to the disturbance and successfully follows the new reference value. The right zoom shows the steady state behavior after the transition, where the oscillations are further reduced and the tracked current remains close to i_q^* . It should be noted that the residual ripple observed in both regions is inherent to the PWM switching and does not reflect a weakness in the agent's tracking capability.

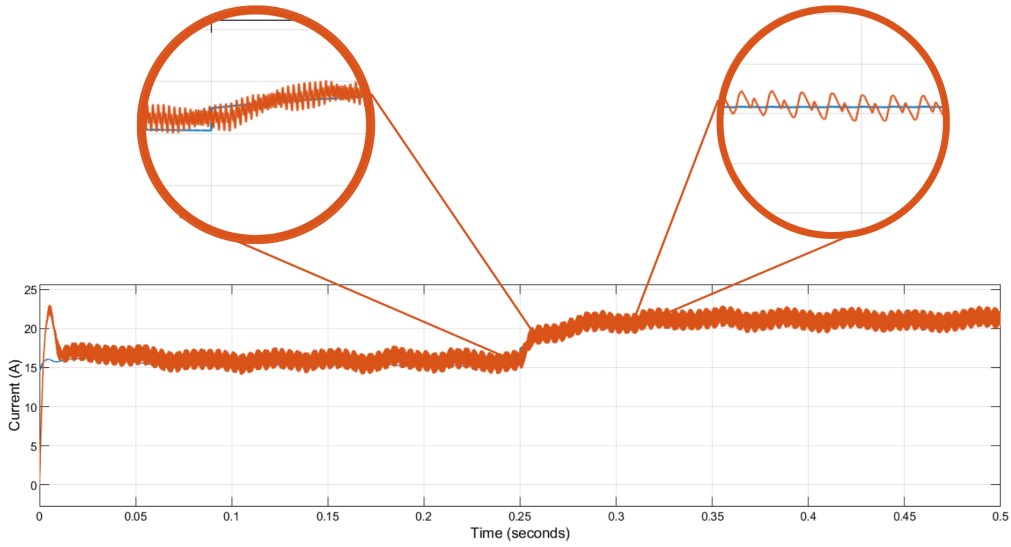


Figure 2.16: Current tracking performance after convergence (zoomed view)

2.10.2 Control Signals and Outer Loop Behavior

This subsection presents the control signals generated by the DDPG agent, including the action signal v_{qr} and its temporal representation used for PWM carrier comparison, as well as the outer voltage control loop tracking behavior.

Action Signal: Normalized and Scaled Forms

Figures (2.17) and (2.18) show the action signal generated by the DDPG agent in its normalized and scaled forms respectively. Both signals show a stable behavior during the first load condition and a clear increase following the load step change at $t = 0.25\text{s}$, which confirms that the agent correctly adjusts its output in response to the disturbance. The rapid variations observed in both signals are due to the noisy nature of the system states caused by the PWM switching, to which the actor network reacts at every time step by adjusting its output accordingly.

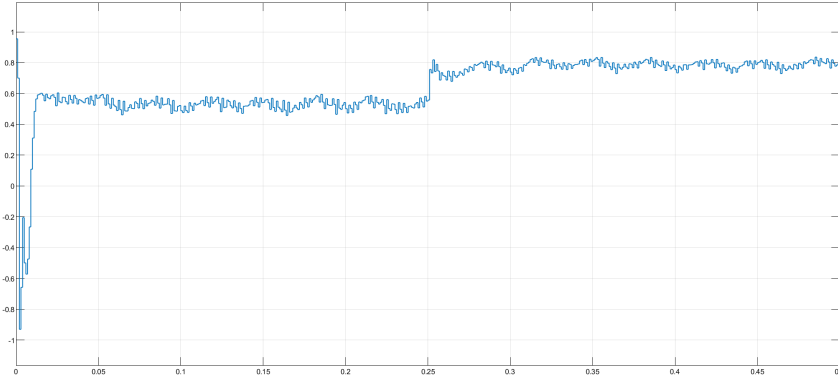


Figure 2.17: Normalized action signal generated by the DDPG agent

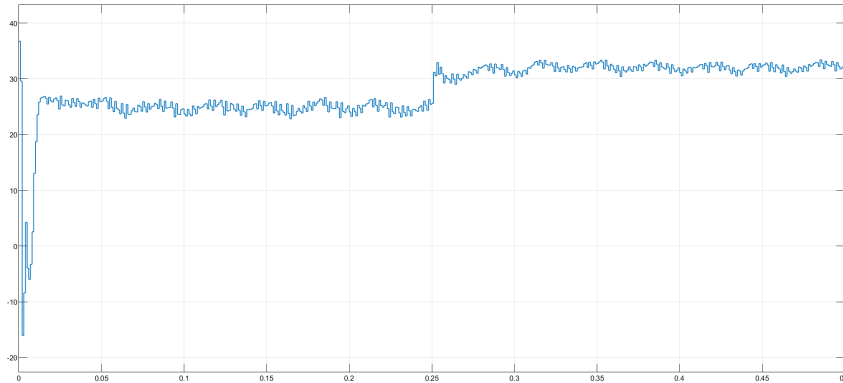
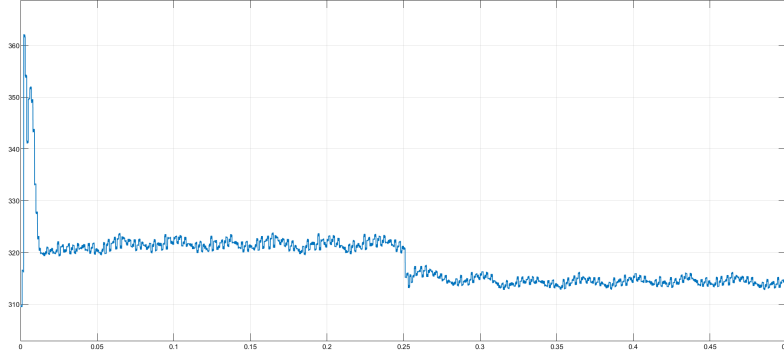


Figure 2.18: Scaled action signal

voltage command in the dq reference frame

Figure (2.19) shows the v_q signal obtained by scaling the normalized action output of the DDPG(v_{qr}) agent and combining it with the v_{qc} component, which together represent the voltage commands in the rotating dq reference frame. The signal exhibits an initial transient at startup before settling into a stable steady state during the first load condition. Following the load step change at $t = 0.25\text{s}$, the signal drops slightly to a new steady state level, reflecting the agent's adaptation to the new operating condition.


 Figure 2.19: Scaled vq Signal

Reference Signals for PWM Carrier Comparison

Figure (2.20) shows the first three reference signals in the temporal domain, which will be used for comparison with the triangular carriers to generate the PWM switching signals. The three signals are sinusoidal, balanced and mutually shifted by 120° , confirming that the transformation was correctly applied. The slight irregularities visible on the signals are a direct consequence of the rapid variations present in the vq signal discussed previously, which are inherent to the nature of the DDPG agent that continuously reacts to the noisy system states at every time step.

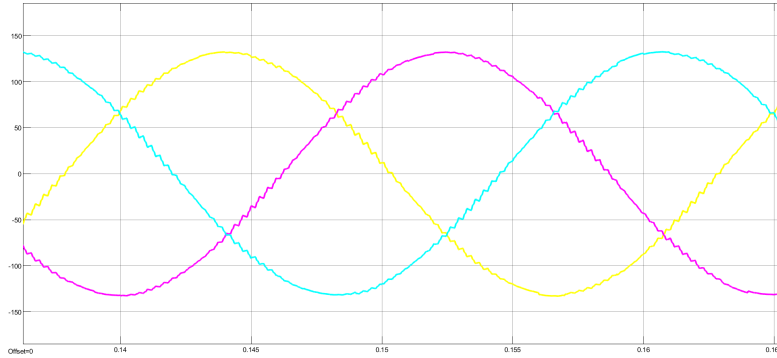


Figure 2.20: Reference Signals for PWM Carrier Comparison

Outer Voltage Loop Behavior and DC Bus Voltage Regulation

Figures (2.21) and (2.22) show the DC bus voltage tracking behavior of the outer voltage control loop throughout the training process. It is worth recalling that the control architecture adopted in this study follows a hybrid approach, where the outer voltage loop relies on a conventional PI controller to regulate the DC bus voltage and generate the reference current iq^* , while the inner current loop is handled by the DDPG agent. As shown in Figure (2.21), during the early training episodes the outer PI controller fails to bring the DC bus voltage to the 300V reference, which is directly linked to the poor iq tracking performance of the agent at this stage. As training progresses, the agent successfully learns to track the reference during the first load condition, however it still struggles to recover after the load step change at $t = 0.25s$, revealing that the agent has not yet learned how to handle disturbances. After several more episodes, as shown in Figure (2.22), the agent demonstrates a clear improvement where it not only tracks the 300V reference during the first load condition but also successfully recovers after the

disturbance, confirming that the agent has progressively learned to support the outer PI controller in maintaining voltage regulation under both operating conditions.

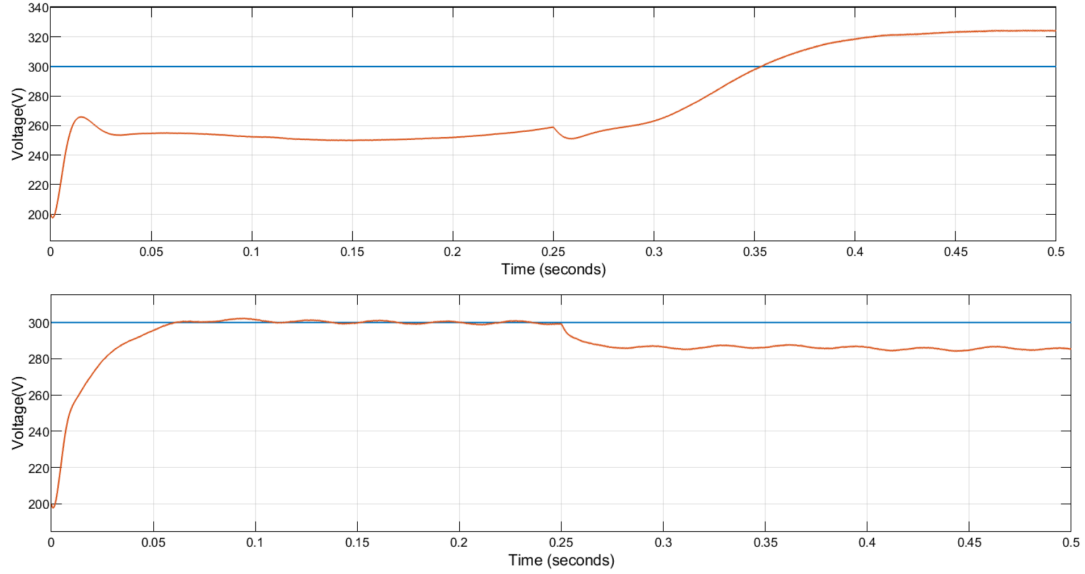


Figure 2.21: DC bus voltage tracking behavior during training (early and intermediate episodes)

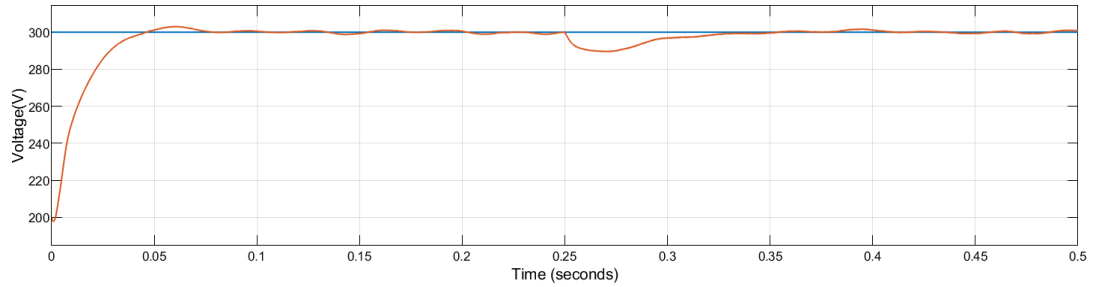


Figure 2.22: DC bus voltage tracking behavior after convergence

2.10.3 Reward Evolution During Training

Figure (2.23) shows the episode reward evolution over the 250 training episodes. As clearly observed, the reward starts at very large negative values during the first episodes, reflecting the agent's lack of knowledge about the system. As training progresses, the reward increases rapidly and converges to a stable value around episode 50, after which it remains consistent until the end of training. This behavior confirms that the agent has successfully learned a stable control policy within a relatively small number of episodes. This is further supported by Figure (2.24), which presents the first and last 10 episodes in detail. During the first 10 episodes, the episode reward ranges from approximately -1021 to -112, while by the last 10 episodes it stabilizes consistently around -38, representing a significant improvement and confirming the convergence of the agent's policy.

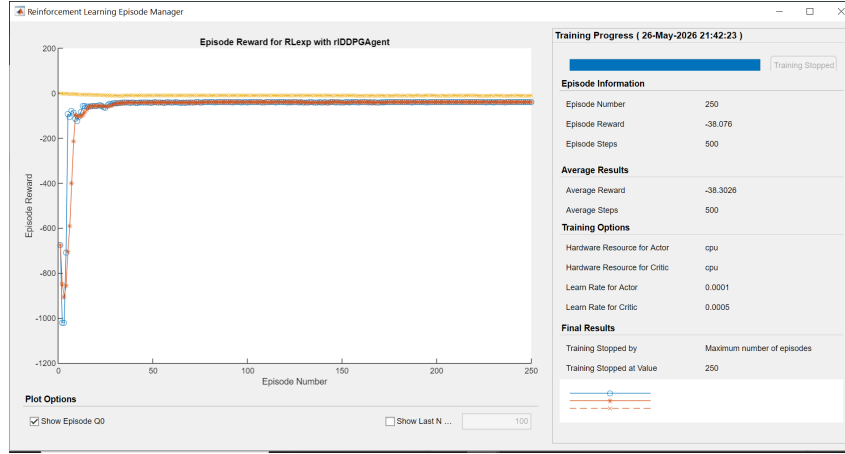


Figure 2.23: Episode reward evolution over 250 training episodes

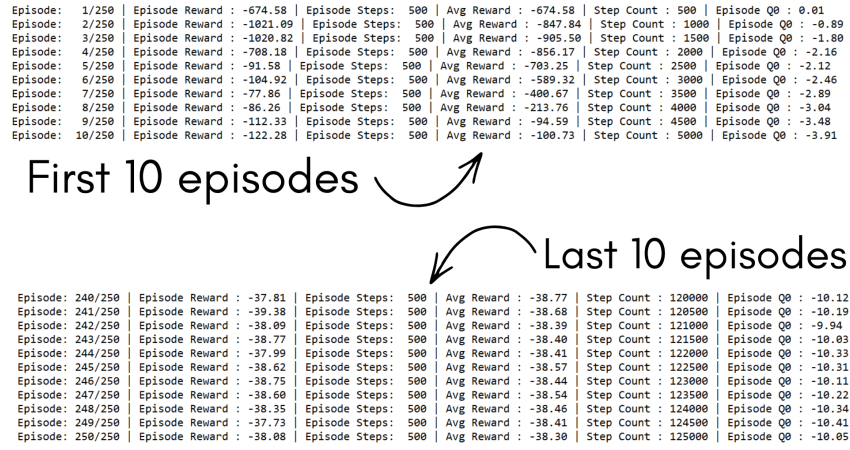


Figure 2.24: Episode reward details (first and last 10 training episodes)

2.10.4 PI and DDPG Control: A Comparative Perspective

This subsection presents a comparison between the conventional PI controller and the DDPG-based control approach. It should be noted that the goal of this comparison is not to prove that one approach is better than the other, but rather to understand how classical and AI-based control can coexist and complement each other, which ultimately helps in designing hybrid systems that combine the reliability of classical controllers with the adaptability of AI.

Figures (2.26), (2.27) and (2.28) present a comparison between the conventional PI controller used for both control loops and the hybrid PI-DDPG control approach in terms of i_q tracking, reference current i_q^* generation and DC bus voltage regulation respectively. Regarding the i_q tracking shown in Figure (2.26), both approaches follow the reference current with acceptable performance, however the conventional PI exhibits a smoother response with less ripple. In terms of the reference current i_q^* shown in Figure (2.27), the conventional PI generates a significantly cleaner and smoother signal compared to the hybrid approach, where the noisy nature of the DDPG agent introduces visible irregularities. As for the DC bus voltage regulation shown in Figure (2.28), both approaches successfully

track the 300V reference, however the conventional PI demonstrates a faster and cleaner recovery following the load step change at $t = 0.25\text{s}$, while the hybrid approach shows a more pronounced undershoot before recovering. It should be emphasized that the goal of this comparison is not to prove that one approach outperforms the other, but rather to demonstrate that an AI-based controller can achieve comparable performance to a well tuned classical PI controller, which highlights the potential of reinforcement learning as a viable and complementary tool in the design of advanced control systems for power electronics applications.

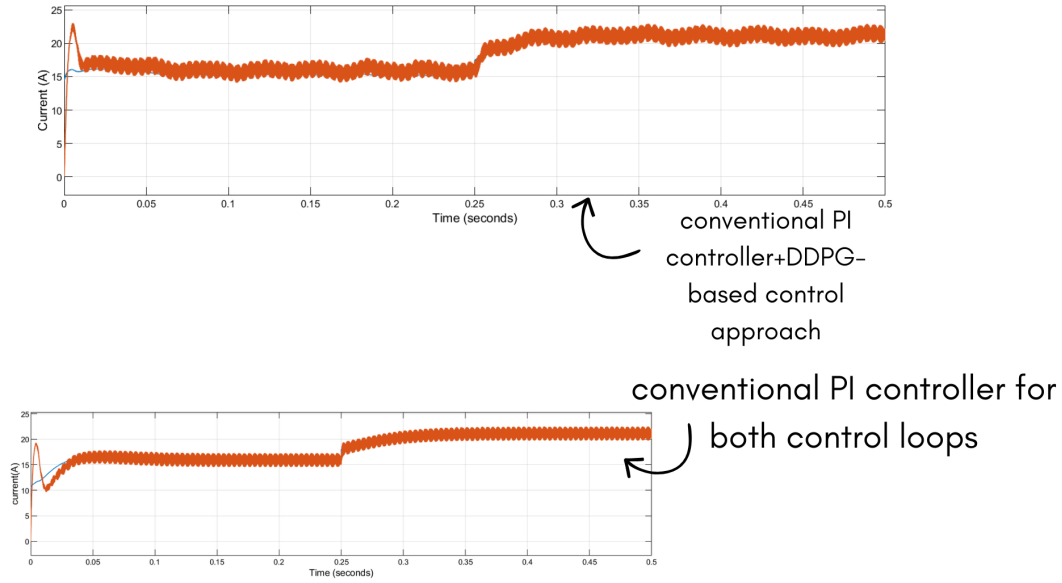


Figure 2.25: i_q current tracking (hybrid PI-DDPG vs conventional PI)

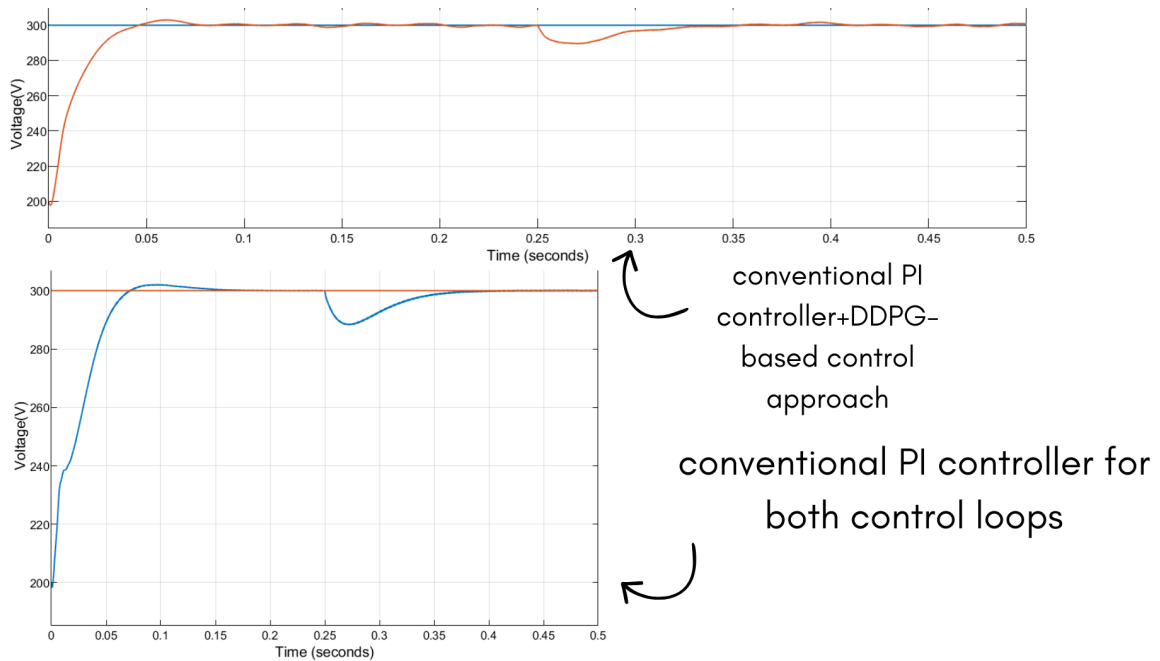


Figure 2.26: DC bus voltage regulation (hybrid PI-DDPG vs conventional PI)

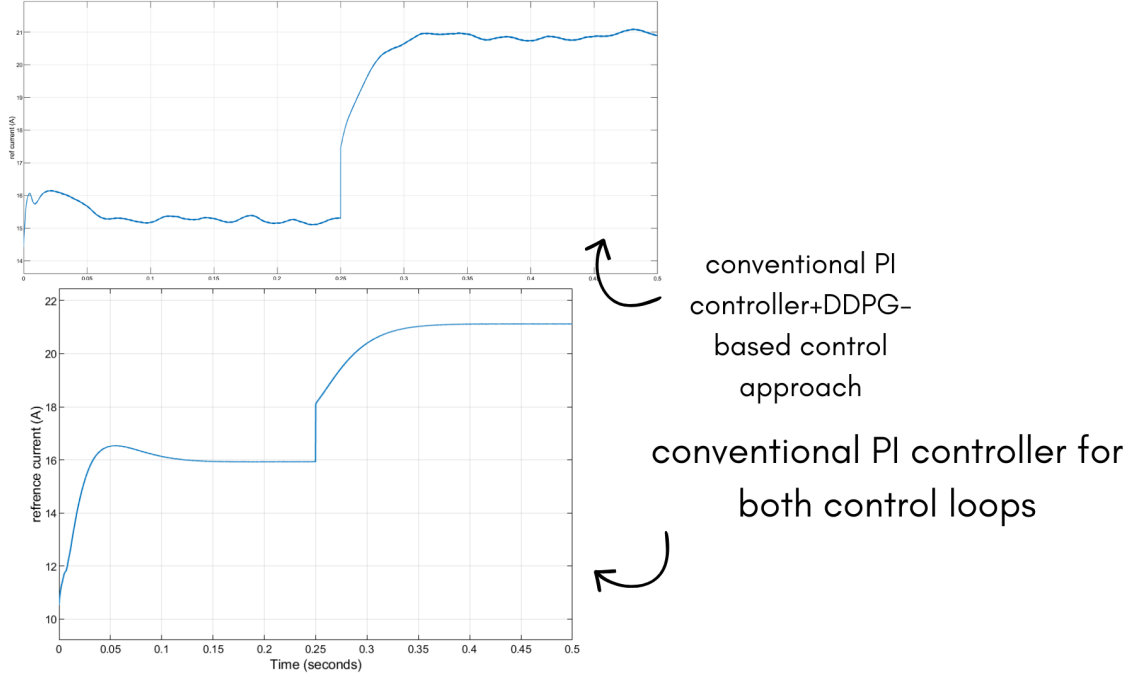


Figure 2.27: Reference current i_q^* (hybrid PI-DDPG vs conventional PI)

2.11 Conclusion

This chapter presented the training and evaluation of a DDPG agent designed to directly replace the PI controller in the inner current control loop of a PWM rectifier. The agent was trained under dynamic operating conditions and successfully learned a stable control policy capable of tracking the reference current i_q^* and supporting the outer voltage regulation loop. The comparative analysis showed that while the conventional PI exhibits a smoother response, the DDPG agent achieves comparable overall performance, which confirms the feasibility of integrating reinforcement learning into classical control architectures. These results lay the foundation for the adaptive gain tuning approach that will be investigated in the following chapter.

Chapter 3: Grid Integration and Reinforcement Learning-Based PI Tuning for the Energy Conversion Chain

3.1 Introduction

This chapter presents the complete control strategy developed for the energy conversion chain, covering two complementary aspects. The first part addresses the grid integration stage, describing the interface between the generator-side system and the electrical grid, the synchronization method using a Phase Locked Loop, the reference frame transformations, and the conventional PI-based current control in the synchronous dq frame. The second part introduces a reinforcement learning approach to replace manual PI gain tuning. A Twin Delayed Deep Deterministic Policy Gradient (TD3) agent is applied in two distinct contexts: adaptive tuning of the grid-side inverter current controllers, and adaptive tuning of the I_q PI controller responsible for regulating the DC output voltage of the PWM rectifier connected to the dual-star synchronous generator. The objective of both applications is to improve tracking performance and dynamic response compared to fixed-gain conventional PI control.

3.2 Grid Integration and Control Strategy

3.2.1 Choice of the Universal Bridge Model

The inverter is implemented using the MATLAB/Simulink *Universal Bridge* block due to its simplicity and suitability for grid-connected power electronics studies. It provides a ready-made three-phase switching model that can represent different semiconductor devices while avoiding the complexity of building a detailed converter from individual components. This makes it efficient in terms of simulation time and implementation effort.

The main goal of this chapter is to focus on the control of grid integration, particularly the regulation of active and reactive power using established control methods. The Universal Bridge therefore serves as a practical power stage model that allows the control strategy to be evaluated under realistic operating conditions without unnecessary modeling complexity.

and

$$I_a, I_b, I_c$$

These signals are sinusoidal, time-varying, and shifted by 120° from one another.

Directly controlling these AC quantities using conventional PI controllers is difficult because PI controllers are designed to regulate DC or constant signals. The controller continuously compares a measured value with a reference value in order to minimize the error between them. However, when the signals are sinusoidal and continuously oscillating, the error also oscillates, making accurate regulation more complicated.

For this reason, the three-phase quantities are transformed from the abc reference frame into the rotating synchronous reference frame known as the dq frame. This transformation converts sinusoidal AC quantities into approximately constant DC quantities under steady-state conditions, which significantly simplifies the control process.

This transformation is achieved using two mathematical transformations:

- Clarke Transformation
- Park Transformation

3.2.4 Clarke Transformation

The Clarke transformation converts the three-phase quantities from the abc frame into the stationary $\alpha\beta$ frame.

The transformation is expressed as:

$$\begin{bmatrix} V_\alpha \\ V_\beta \end{bmatrix} = \frac{2}{3} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix} \quad (3.1)$$

The same transformation can also be applied to the current components.

3.2.5 Park Transformation

The Park transformation converts the stationary $\alpha\beta$ components into the rotating synchronous dq frame using the electrical angle θ obtained from the PLL.

The Park transformation is expressed as:

$$\begin{bmatrix} V_d \\ V_q \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} V_\alpha \\ V_\beta \end{bmatrix} \quad (3.2)$$

The same operation is applied to the current components to obtain:

$$I_d, I_q$$

The transformation into the synchronous frame simplifies the regulation process because the AC variables become nearly constant in steady-state operation.

3.2.6 Phase Locked Loop (PLL) Synchronization

In order to connect the inverter to the electrical grid, synchronization is essential. The inverter output must have:

- The same frequency as the grid
- The same phase angle
- The same phase sequence

If synchronization is not achieved correctly, large circulating currents and severe disturbances may occur during grid connection.

To ensure synchronization, a three-phase Phase Locked Loop (PLL) block is used. The PLL continuously tracks the grid voltage and generates the electrical angle θ required for the Park transformation.

The PLL detects the phase position and zero-crossing points of the grid voltage in order to maintain accurate synchronization between the inverter and the utility grid.

3.2.7 Current Control in the dq Reference Frame

After transformation into the synchronous reference frame, the current components I_d and I_q can be controlled independently.

The direct-axis current I_d is mainly associated with active power control, while the quadrature-axis current I_q is associated with reactive power control.

Therefore:

- I_d controls active power
- I_q controls reactive power

PI controllers are used to regulate both current components according to their desired reference values.

The controller receives:

- The measured current components
- The reference current components

The error between the reference and measured values is processed by the PI controllers to generate the control voltages:

$$V_d, V_q$$

These voltages are then used to generate the switching signals of the inverter.

3.2.8 Voltage Equations in the Synchronous Reference Frame

The dynamic behavior of the inverter in the synchronous reference frame can be described by the following equations:

$$V_d = RI_d + L \frac{dI_d}{dt} - \omega LI_q \quad (3.3)$$

$$V_q = RI_q + L \frac{dI_q}{dt} + \omega LI_d \quad (3.4)$$

where:

- R is the filter resistance
- L is the filter inductance
- ω is the angular frequency

These equations show the coupling between the d and q current components.

3.2.9 Active and Reactive Power Equations

The active and reactive powers delivered by the inverter to the grid can be expressed in the synchronous reference frame as:

$$P = \frac{3}{2}(V_d I_d + V_q I_q) \quad (3.5)$$

$$Q = \frac{3}{2}(V_q I_d - V_d I_q) \quad (3.6)$$

These equations clearly illustrate the relationship between the current components and the exchanged power.

Typically, when only active power injection is required, the inverter is controlled such that:

$$V_q = 0$$

In this case, the inverter injects only real power into the grid.

Reactive power control can also be achieved by assigning a non-zero reference value to I_q .

3.2.10 Inverse Transformation and PWM Generation

Once the control voltages V_d and V_q are generated by the PI controllers, they must be transformed back into the three-phase reference frame.

This is achieved using the inverse Park transformation and inverse Clarke transformation to reconstruct the three-phase voltage references:

$$V_a^*, V_b^*, V_c^*$$

These reference voltages are then applied to a PWM generator block in order to generate the switching pulses required for the inverter semiconductor devices.

The PWM generator compares the voltage references with carrier signals to produce the gating signals that regulate the inverter output voltage and current.

3.2.11 Implementation of the Grid-Connected Inverter in MATLAB/Simulink

The complete grid-connected inverter system was implemented and constructed in MATLAB/Simulink in order to validate the proposed control strategy and analyze the behavior of the inverter during grid integration. The system includes the grid model, the inverter, the filtering stage, synchronization blocks, reference frame transformations, PI current controllers, and PWM generation blocks.

The electrical grid was modeled using a three-phase voltage source configured in grounded star connection (Y_g) with a frequency of 50 Hz . In order to reduce harmonic distortion and smooth the inverter output currents before grid injection, an LCL filter was used between the inverter and the electrical grid.

The filter parameters were selected as follows:

$$L = 500 \times 10^{-6} \text{ H} \quad (3.7)$$

$$C = 100 \times 10^{-6} \text{ F} \quad (3.8)$$

The complete MATLAB/Simulink implementation of the grid-connected inverter and filter stage is presented in Figure 3.2.

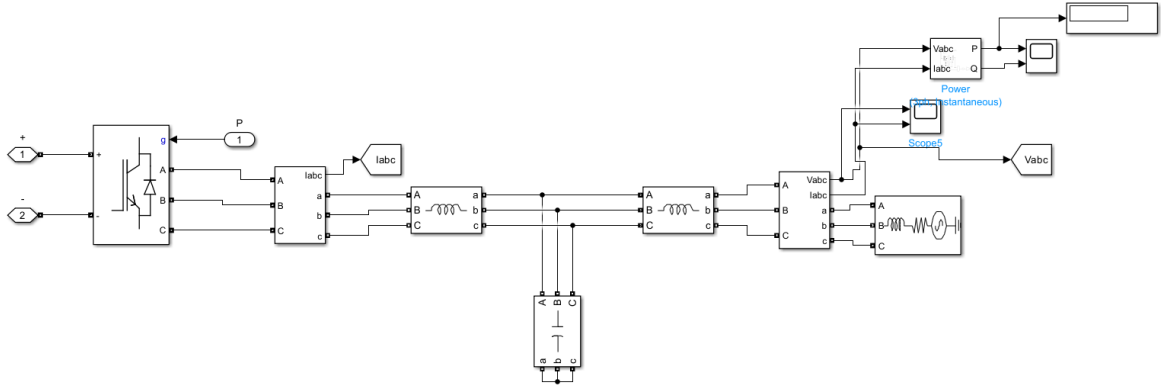


Figure 3.2: MATLAB implementation of the grid-connected inverter with LCL filter

The control structure was implemented using the synchronous dq reference frame control method. The measured three-phase voltages and currents were transformed from the abc frame into the dq frame using Clarke and Park transformations. A three-phase PLL block was used to continuously track the grid voltage angle and ensure synchronization between the inverter and the utility grid.

PI controllers were implemented to regulate the I_d and I_q current components independently. The direct-axis current component I_d was used to control the active power injected into the grid, while the quadrature-axis current component I_q was used to regulate the reactive power exchanged with the grid.

After generating the control voltages V_d and V_q , inverse $dq0$ transformations were applied to reconstruct the three-phase voltage references. These references were then

used by the PWM generator block to produce the switching pulses required for inverter operation.

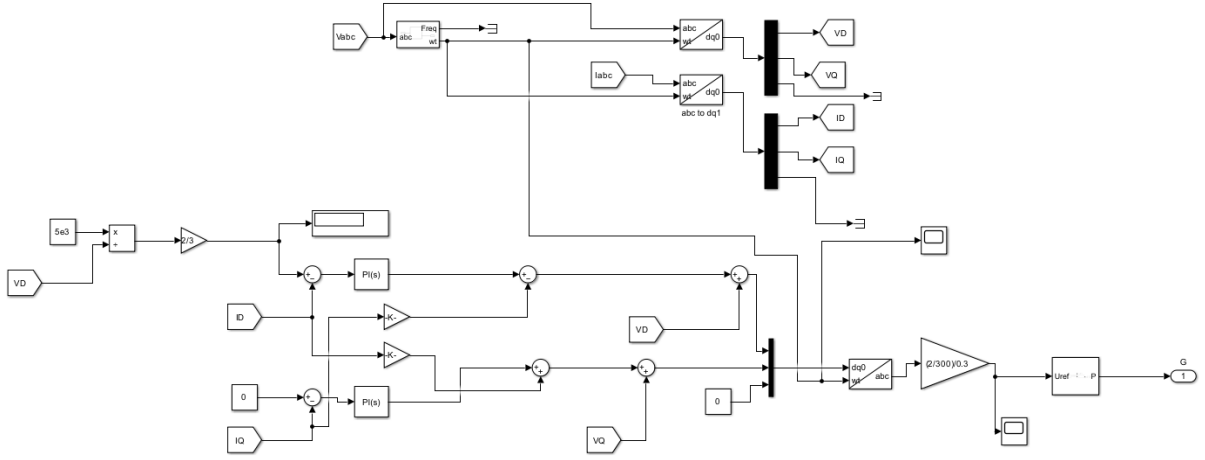


Figure 3.3: Control structure of the grid-connected inverter

The implemented control strategy allows stable synchronization with the grid, accurate active and reactive power regulation, and proper control of the inverter output currents during grid-connected operation. All of these components are merged into a single system alongside the generator system to simulate and evaluate the grid integration efficiency.

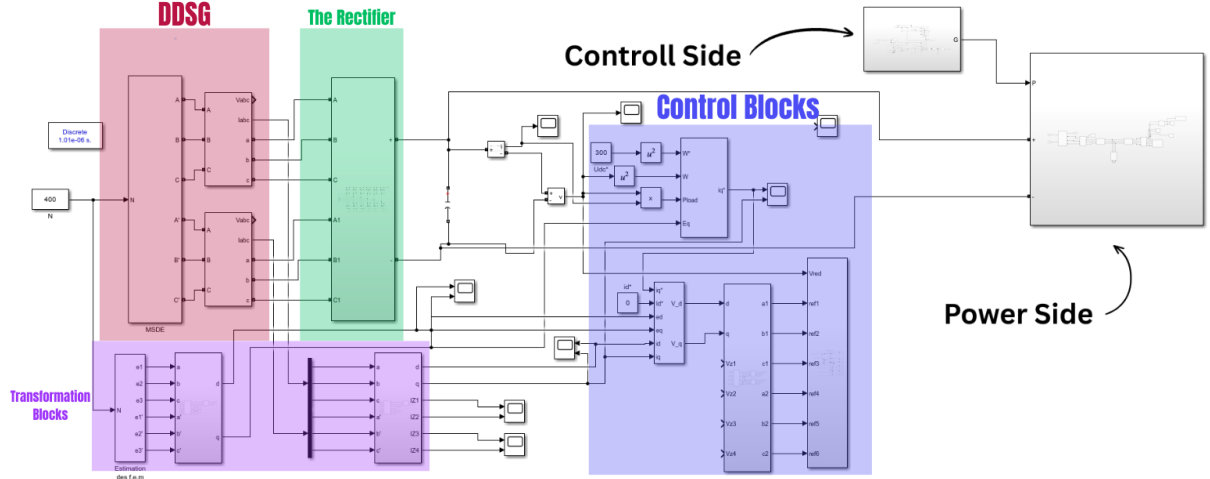


Figure 3.4: Overall system architecture used for grid integration simulation

3.2.12 Simulation Results and Grid Performance

In the second simulation, three reference active power levels were imposed on a total simulation time of 1s. The system was first ordered to deliver 2 kW during the interval $0 \leq t < 0.3$ s, then increased to 4 kW for $0.3 \leq t < 0.6$ s, and finally reduced to 1 kW for $0.6 \leq t \leq 1$ s. The results obtained, illustrated in Fig. 3.5, show that the active power closely tracks the reference values with a fast dynamic response and only small oscillations around the setpoints, which is expected due to the switching nature of the converter and control dynamics.

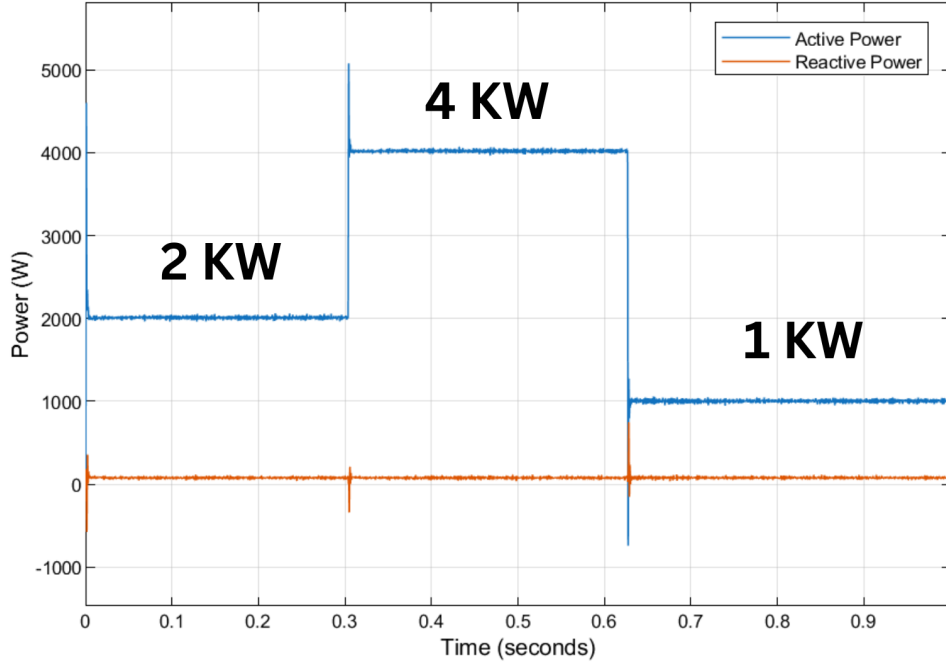


Figure 3.5: Active power response under step changes in reference power.

Furthermore, the corresponding three-phase grid voltages are presented in Fig. 3.6. The results confirm that the grid voltage remains balanced and sinusoidal despite the connection of the generator-side system. This indicates that the grid is not significantly affected by converter operation, which is consistent with the low total harmonic distortion (THD) achieved in the system. The lower part of Fig. 3.6 shows the three-phase grid currents.

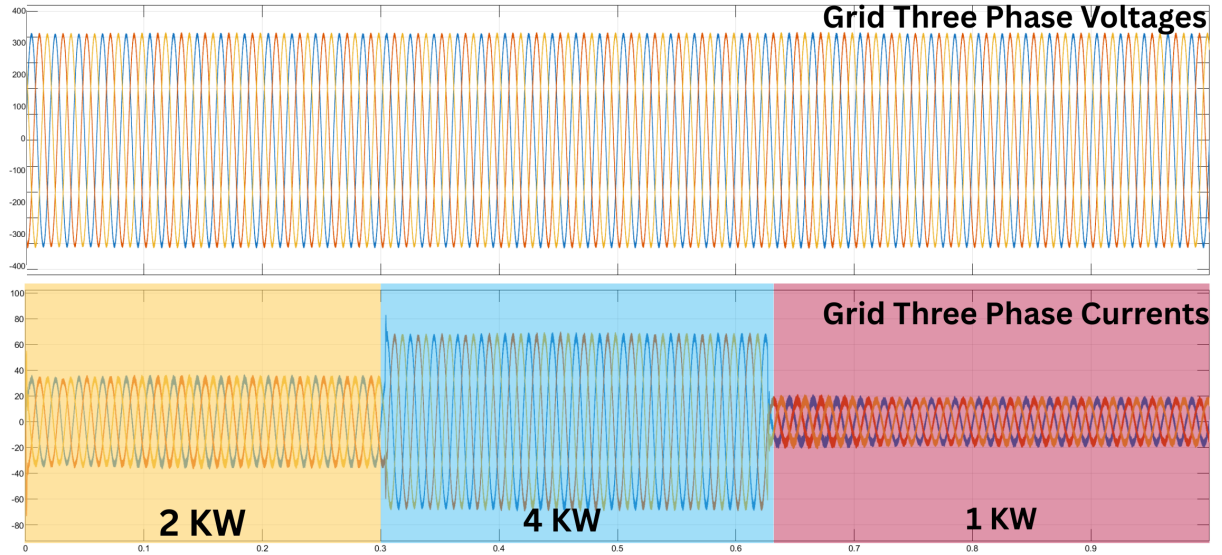


Figure 3.6: Three-phase grid voltages (top) and currents (bottom) under varying power references.

It can be clearly observed that the current amplitudes vary between the three operating intervals (0–0.3 s, 0.3–0.6 s, and 0.6–1 s), following the imposed changes of reference for active power. As expected, an increase in active power demand results in an increase in current magnitude, while a decrease in power leads to a reduction in current. These

transitions occur smoothly, with limited fluctuations, which confirms the effectiveness of the control strategy and the stability of the grid integration.

3.2.13 Transition from Conventional PI Control to Reinforcement Learning-Based Tuning

After implementing the conventional PI control strategy with fixed gains, an adaptive approach was introduced using reinforcement learning. Instead of replacing the PI controller, the existing control structure was preserved while a TD3 agent was employed to automatically tune the proportional and integral gains according to the system operating conditions. In this configuration, the reinforcement learning agent acts as a supervisory layer that continuously adjusts the controller parameters to improve tracking performance and dynamic response. This approach combines the simplicity and robustness of PI control with the adaptability of reinforcement learning, allowing a direct comparison between fixed-gain control and self-tuning control within the same grid-connected system.

3.3 TD3-Based Adaptive PI Control

The conventional control strategy presented previously relies on two fixed PI controllers to regulate the direct-axis current (I_d) and quadrature-axis current (I_q). Although this approach provides satisfactory performance under nominal operating conditions, its effectiveness depends heavily on the selected controller gains. To improve adaptability and eliminate manual tuning, the fixed PI controllers were replaced by a reinforcement learning-based controller using the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm [12].

The overall modification of the control structure is illustrated in Figure 3.7. Instead of sending the current errors directly to two conventional PI controllers, the errors are processed by a reinforcement learning agent that continuously adjusts the controller parameters according to the current operating conditions of the system.

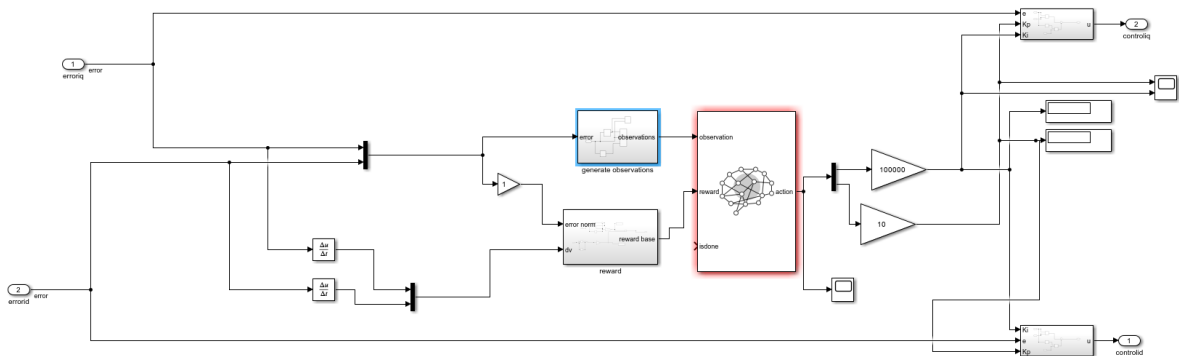


Figure 3.7: Modified control structure using the TD3 agent.

The inputs of the proposed controller are obtained from the current regulation loops. The errors between the reference and measured current components are first calculated as:

$$e_d = I_{d,ref} - I_d \quad (3.9)$$

$$e_q = I_{q,ref} - I_q \quad (3.10)$$

These signals are then used to construct the observation vector supplied to the TD3 agent. To provide sufficient information regarding the dynamic behavior of the system, three quantities are extracted from each error signal:

- The instantaneous error.
- The integral of the error.
- The derivative of the error.

Consequently, the observation vector contains six elements:

$$Obs = [e_d, \int e_d dt, \dot{e}_d, e_q, \int e_q dt, \dot{e}_q] \quad (3.11)$$

The implementation of the observation subsystem is presented in Figure 3.8.

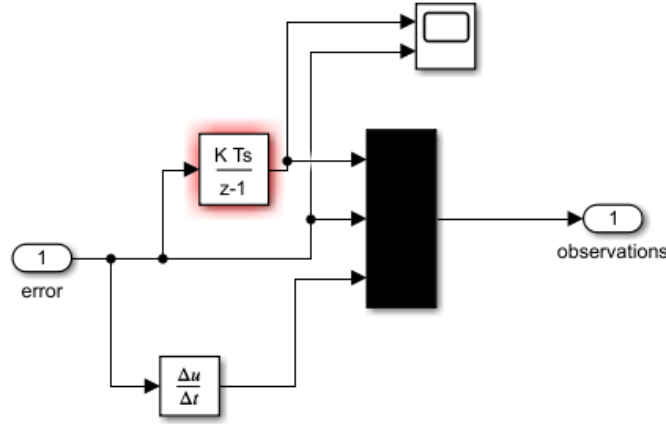


Figure 3.8: Observation generation subsystem.

The TD3 agent receives these observations and generates two actions corresponding to the controller gains:

$$A = [K_p, K_i] \quad (3.12)$$

These gains are updated continuously during training and are used to construct a custom PI controller implemented directly in Simulink.

Since the standard PI controller block does not allow online modification of its parameters during reinforcement learning training, a custom PI controller was developed.[10] The controller output is generated according to the classical PI equation:

$$u(t) = K_p e(t) + K_i \int e(t) dt \quad (3.13)$$

where the proportional and integral gains are provided directly by the TD3 agent. The implementation of the custom PI controller is shown in Figure 3.9.

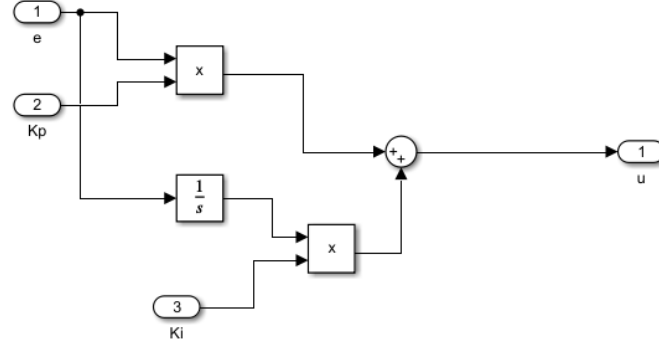


Figure 3.9: Custom PI controller with gains provided by the TD3 agent.

To guide the learning process, a reward function was designed using the control errors and their variations. The objective of the reward is to minimize both the tracking error and undesired oscillations in the system response. Large errors result in a reduction of the reward value, while excessive signal variations are also penalized. As a result, the agent is encouraged to achieve accurate reference tracking while maintaining smooth dynamic behavior.

The reward subsystem used during training is illustrated in Figure 3.10.

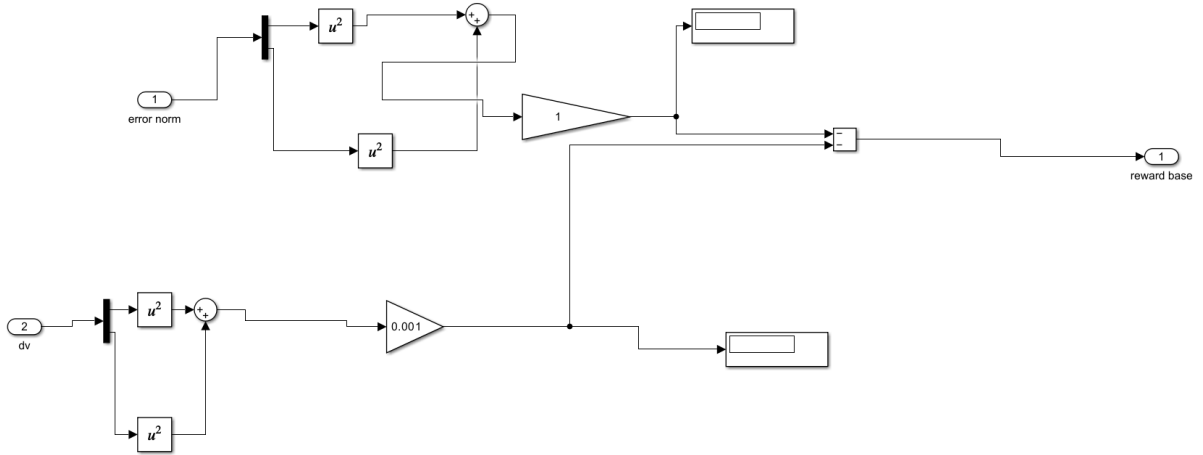


Figure 3.10: Reward calculation subsystem.

The complete reinforcement learning architecture consists of the environment, observation subsystem, reward subsystem, custom PI controller, and TD3 agent. During each training episode, the agent observes the system state, generates controller gains, receives a reward according to the system performance, and gradually improves its policy through repeated interactions with the environment.

The complete implementation of the reinforcement learning controller in MATLAB/Simulink is shown in Figure 3.11.

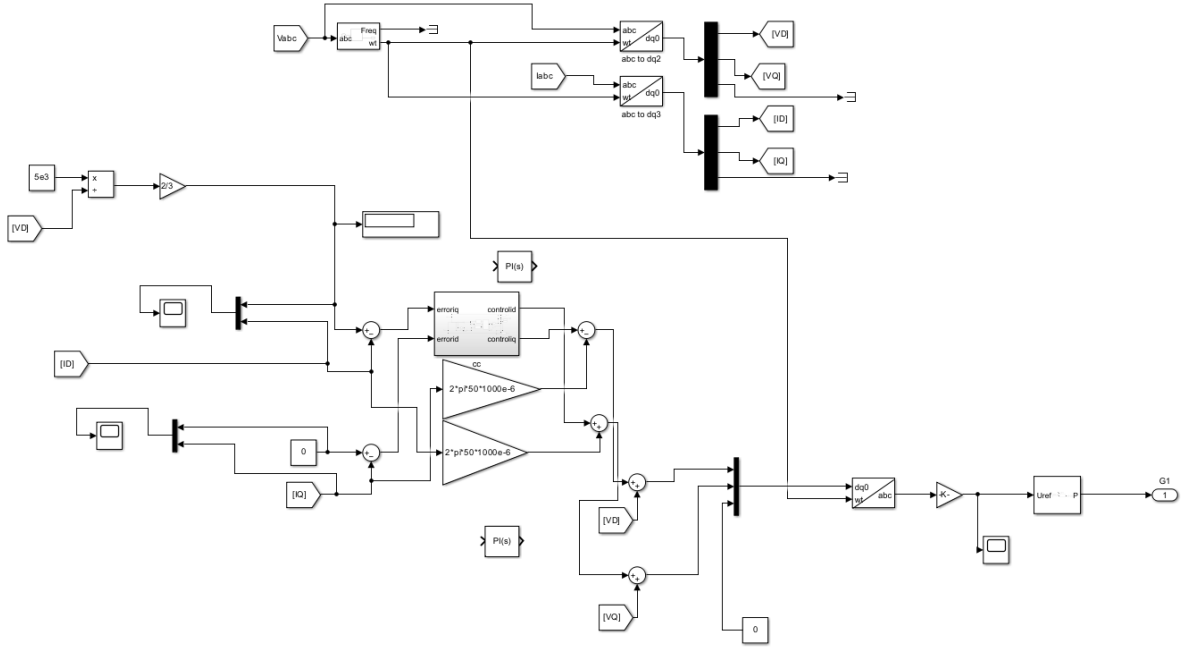


Figure 3.11: Complete TD3-based adaptive PI control system.

3.4 RL Agent Configuration (TD3) in MATLAB

This section details the implementation of the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm in MATLAB for adapting the PI controller gains in the grid-connected system. The implementation is divided into several Code blocks, each having a specific role in the reinforcement learning workflow. These blocks define the observation and action spaces, build the actor and critic networks, configure the TD3 agent, organize the training procedure, and finally extract the learned controller gains[10][12].

3.4.1 Block 1: Observation and Action Specifications

This block represents the interface between the Simulink model and the RL agent. It defines the information received by the agent and the values that the agent is allowed to generate.

```
mdl = 'aigridintegrationuniversalbridge';
Ts = 0.1;
Tf = 1;

numObs = 6;
obsInfo = rlNumericSpec([numObs 1], ...
    'LowerLimit', -Inf(numObs,1), ...
    'UpperLimit', Inf(numObs,1));
obsInfo.Name = 'observations';
obsInfo.Description = 'e, integral_e, derivative_e for both loops';

numAct = 2;
actInfo = rlNumericSpec([numAct 1], ...
    'LowerLimit', 1e-5 * ones(numAct,1), ...
```

```

    'UpperLimit',          ones(numAct,1));
actInfo.Name = 'PID gains';

env = rlSimulinkEnv mdl, [mdl '/s/cc/RL Agent'], obsInfo, actInfo);
rng(0, 'twister');

```

The observation vector contains six values corresponding to the error, its integral, and its derivative for both control loops. The action vector contains two values, which represent the proportional and integral gains generated by the TD3 agent.

3.4.2 Block 2: Actor Network

The actor network receives the observation vector and produces the controller gains. It is built as a deep feedforward neural network composed of feature input layers, fully connected layers, and activation layers.

```

actorNet = [
    featureInputLayer(numObs, 'Name','obsIn')
    fullyConnectedLayer(128, 'Name','fc1')
    reluLayer('Name','relu1')
    fullyConnectedLayer(128, 'Name','fc2')
    reluLayer('Name','relu2')
    fullyConnectedLayer(64, 'Name','fc3')
    reluLayer('Name','relu3')
    fullyConnectedLayer(numAct,'Name','fcOut')
    sigmoidLayer('Name','sigmoid')
];
actorDLNet = dlnetwork(actorNet);
actor = rlContinuousDeterministicActor(actorDLNet, obsInfo, actInfo, ...
    'ObservationInputNames','obsIn');

```

The final sigmoid layer constrains the output in a normalized range, which is later mapped to the required gain values through the action specification.

3.4.3 Block 3: Critic Networks

Two critic networks are used in TD3. Each critic receives the observation and action inputs separately, processes them through independent paths, and then combines them before producing a scalar Q-value.

```

statePath1 = [
    featureInputLayer(numObs,'Name','stateInLyr1')
    fullyConnectedLayer(128, 'Name','fc1_1')
    reluLayer('Name','relu_s1')
];

actionPath1 = [
    featureInputLayer(numAct,'Name','actionInLyr1')

```

```

        fullyConnectedLayer(128, 'Name','fc2_1')
        reluLayer(
            'Name','relu_a1')
];

commonPath1 = [
    concatenationLayer(1,2, 'Name','concat1')
    fullyConnectedLayer(128, 'Name','fc3_1')
    reluLayer(
        'Name','relu_c1')
    fullyConnectedLayer(1, 'Name','qvalOutLyr1')
];

criticLG1 = layerGraph();
criticLG1 = addLayers(criticLG1, statePath1);
criticLG1 = addLayers(criticLG1, actionPath1);
criticLG1 = addLayers(criticLG1, commonPath1);
criticLG1 = connectLayers(criticLG1,'relu_s1','concat1/in1');
criticLG1 = connectLayers(criticLG1,'relu_a1','concat1/in2');
criticNet1 = dlnetwork(criticLG1);

critic1 = rlQValueFunction(criticNet1, obsInfo, actInfo, ...
    'ObservationInputNames','stateInLyr1', ...
    'ActionInputNames', 'actionInLyr1');

```

The second critic is built in the same way. The use of two critics improves training stability and reduces estimation errors.

```

statePath2 = [
    featureInputLayer(numObs,'Name','stateInLyr2')
    fullyConnectedLayer(128, 'Name','fc1_2')
    reluLayer(
        'Name','relu_s2')
];

actionPath2 = [
    featureInputLayer(numAct,'Name','actionInLyr2')
    fullyConnectedLayer(128, 'Name','fc2_2')
    reluLayer(
        'Name','relu_a2')
];

commonPath2 = [
    concatenationLayer(1,2, 'Name','concat2')
    fullyConnectedLayer(128, 'Name','fc3_2')
    reluLayer(
        'Name','relu_c2')
    fullyConnectedLayer(1, 'Name','qvalOutLyr2')
];

criticLG2 = layerGraph();
criticLG2 = addLayers(criticLG2, statePath2);
criticLG2 = addLayers(criticLG2, actionPath2);
criticLG2 = addLayers(criticLG2, commonPath2);

```

```

criticLG2 = connectLayers(criticLG2,'relu_s2','concat2/in1');
criticLG2 = connectLayers(criticLG2,'relu_a2','concat2/in2');
criticNet2 = dlnetwork(criticLG2);

critic2 = rlQValueFunction(criticNet2, obsInfo, actInfo, ...
    'ObservationInputNames','stateInLyr2', ...
    'ActionInputNames',    'actionInLyr2');

critic = [critic1 critic2];

```

3.4.4 Block 4: TD3 Agent Options

This block defines the learning parameters used during training, including the optimizer settings, replay buffer size, target smoothing noise, exploration noise, and policy update frequency.

```

actorOpts = rlOptimizerOptions( ...
    'LearnRate',          1e-4, ...
    'GradientThreshold', 1);

criticOpts = rlOptimizerOptions( ...
    'LearnRate',          1e-4, ...
    'GradientThreshold', 1);

agentOpts = rlTD3AgentOptions( ...
    'SampleTime',          Ts, ...
    'MiniBatchSize',       256, ...
    'ExperienceBufferLength', 1e6, ...
    'DiscountFactor',      0.99, ...
    'ActorOptimizerOptions', actorOpts, ...
    'CriticOptimizerOptions', criticOpts);

agentOpts.TargetPolicySmoothModel.StandardDeviation = 0.1;
agentOpts.TargetPolicySmoothModel.UpperLimit        = 0.2;
agentOpts.TargetPolicySmoothModel.LowerLimit        = -0.2;
agentOpts.TargetSmoothFactor                        = 5e-3;

agentOpts.ExplorationModel = rl.option.OrnsteinUhlenbeckActionNoise;
agentOpts.ExplorationModel.StandardDeviation        = 0.3;
agentOpts.ExplorationModel.MeanAttractionConstant   = 1;

agentOpts.PolicyUpdateFrequency = 2;

agent = rlTD3Agent(actor, critic, agentOpts);

```

3.4.5 Block 5: Training Procedure

The training process is organized in batches in order to monitor the evolution of the agent and save the best-performing policy. After each batch, the episode rewards are evaluated

and the best agent is stored for later use.

```

evaluator = rlEvaluator( ...
    'NumEpisodes',      5, ...
    'EvaluationFrequency',50, ...
    'RandomSeeds',      1:5);

if ~exist('savedAgents', 'dir'), mkdir('savedAgents'); end
if ~exist('savedFigures','dir'), mkdir('savedFigures'); end

bestReward      = -Inf;
bestAgentFile    = '';
maxEpisodes      = 1000;
episodeBatch     = 10;
maxSteps         = ceil(Tf / Ts);

for batch = 1 : (maxEpisodes / episodeBatch)

    trainOpts = rlTrainingOptions( ...
        'MaxEpisodes',      episodeBatch, ...
        'MaxStepsPerEpisode', maxSteps, ...
        'ScoreAveragingWindowLength', episodeBatch, ...
        'Verbose',          false, ...
        'Plots',             'training-progress', ...
        'StopTrainingCriteria', 'none');

    stats = train(agent, env, trainOpts, 'Evaluator', evaluator);

    batchBest = max(stats.EpisodeReward);

    if batchBest > bestReward
        bestReward      = batchBest;
        bestAgentFile    = sprintf('savedAgents/best_reward_%.0f_batch_%03d.mat', ...
                                   batch, batchBest);
        save(bestAgentFile, 'agent');
        fprintf('[Batch %03d] NEW RECORD: %.4f -> saved: %s\n', ...
                batch, bestReward, bestAgentFile);
    else
        fprintf('[Batch %03d] No improvement. Record still: %.4f\n', ...
                batch, bestReward);
    end
end
end

```

3.4.6 Block 6: Simulation and Gain Extraction

After training, the best saved agent is reloaded and used in simulation. The learned gains are then extracted from the actor network by passing a dummy observation vector to the policy.

```

if isfile(bestAgentFile)
    data = load(bestAgentFile);
    agent = data.agent;
    fprintf('\nLoaded best agent from: %s\n', bestAgentFile);
else
    fprintf('No saved agent found using last trained agent.\n');
end

simOpts = rlSimulationOptions('MaxSteps', maxSteps);
experiences = sim(env, agent, simOpts);

learnedActor = getActor(agent);
dummyObs = {zeros(numObs, 1)};
gainVec = getAction(learnedActor, dummyObs);
gainVec = gainVec{1};

Kp1 = gainVec(1);
Ki1 = gainVec(2);

fprintf('\n=== Learned PID Gains ===\n');
fprintf('Loop 1 Kp: %.6f | Ki: %.6f\n', Kp1, Ki1);

```

3.4.7 Training Results and Performance Analysis

To evaluate the learning capability of the proposed TD3-based adaptive PI controller, several training episodes were analyzed throughout the learning process. The objective of the agent was to determine suitable values of the proportional and integral gains that minimize the tracking error while maintaining a stable and smooth system response.

During the early stages of training, the agent had limited knowledge of the environment and therefore explored different gain combinations. Figure 3.12 presents the system response obtained during Episode 10.

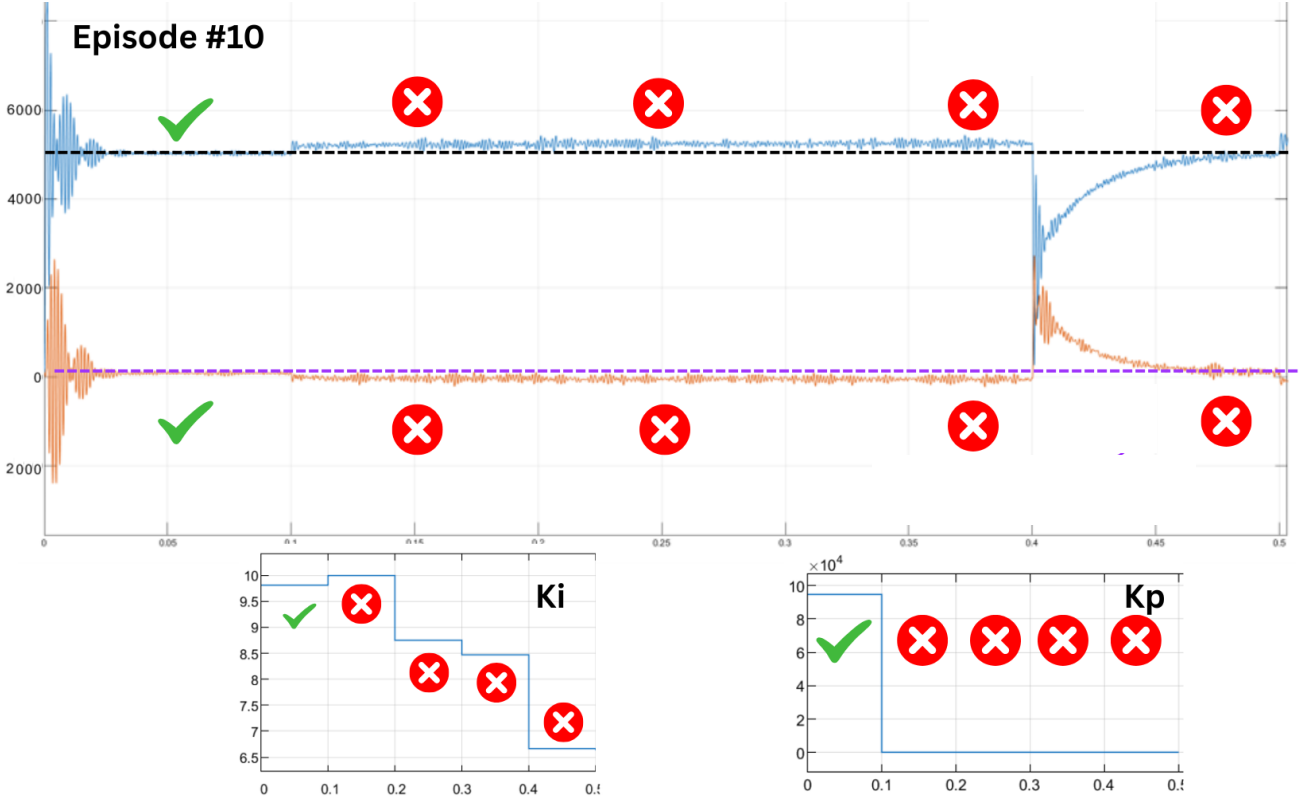


Figure 3.12: System response during Episode 10.

At this stage, the controller output varied significantly throughout the simulation. Since the gains generated by the agent were continuously updated every sampling period, different control actions were tested in order to evaluate their effect on system performance. Although acceptable tracking behavior was obtained during certain intervals, deviations from the reference value were still present. In particular, the active power remained close to the desired value of 5000 W , but noticeable steady-state errors and response variations could still be observed. These results indicate that the agent was still exploring the search space and had not yet converged toward an optimal control policy.

As training progressed, the quality of the selected controller gains improved. Figure 3.14 illustrates the response obtained during Episode 30.

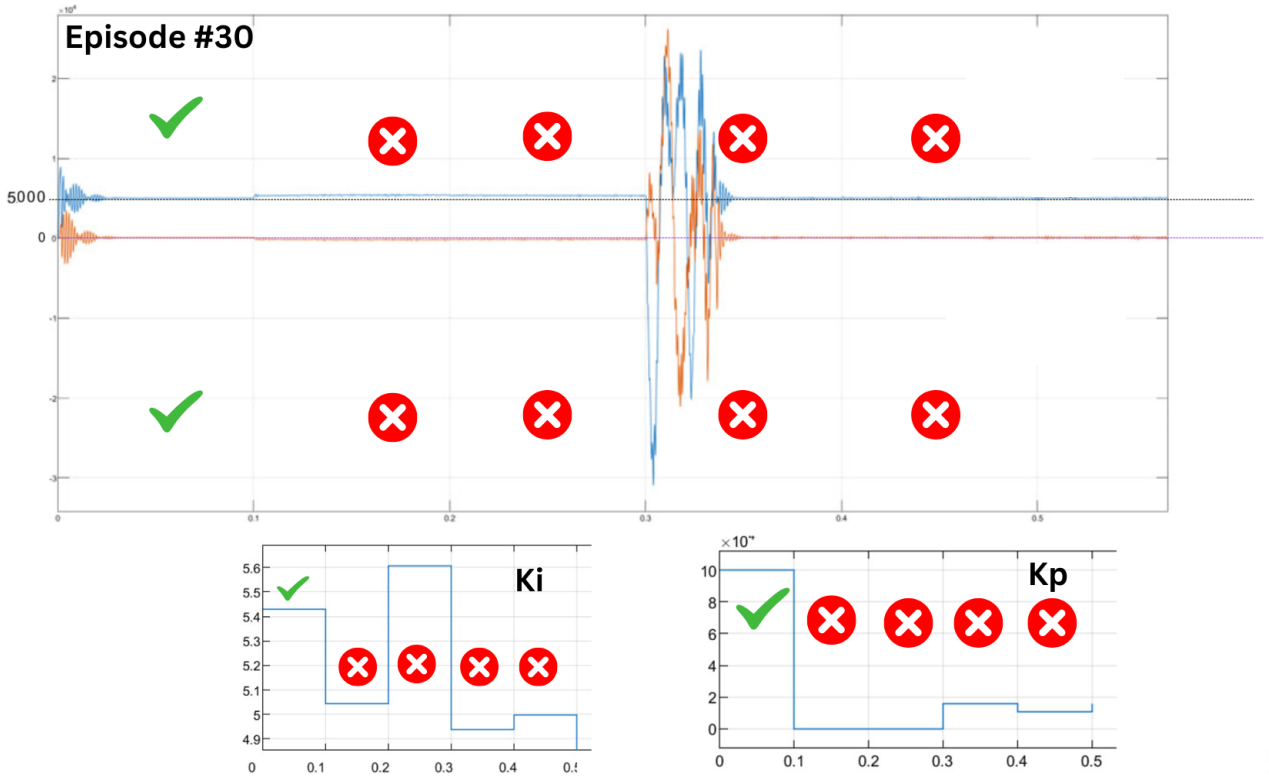


Figure 3.13: System response during Episode 30.

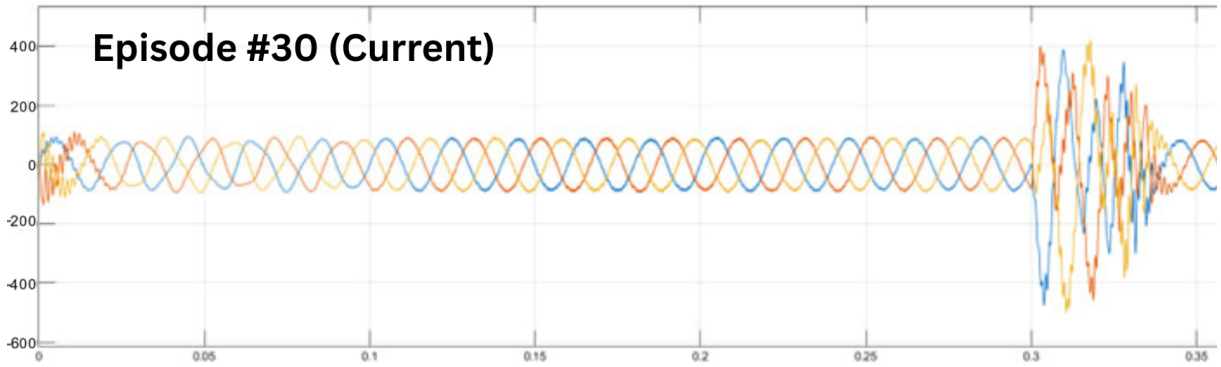


Figure 3.14: System response during Episode 30.

Compared to Episode 10, the tracking performance became more stable during a large portion of the simulation. The generated gains produced smoother responses and reduced oscillations. However, instability and undesirable fluctuations still appeared near the end of the episode, indicating that the agent had not yet completely identified the most suitable control parameters. Nevertheless, the overall behavior demonstrates that the learning process was progressing in the correct direction.

A clear improvement in the reward values can be observed as the number of training episodes increases. This behavior indicates that the TD3 agent gradually learned to avoid control actions that produced large errors or oscillatory responses and increasingly favored actions that resulted in better tracking performance and improved stability.

After sufficient training, the agent converged toward a significantly improved control policy. Figure 3.15 presents the system response obtained during Episode 98.

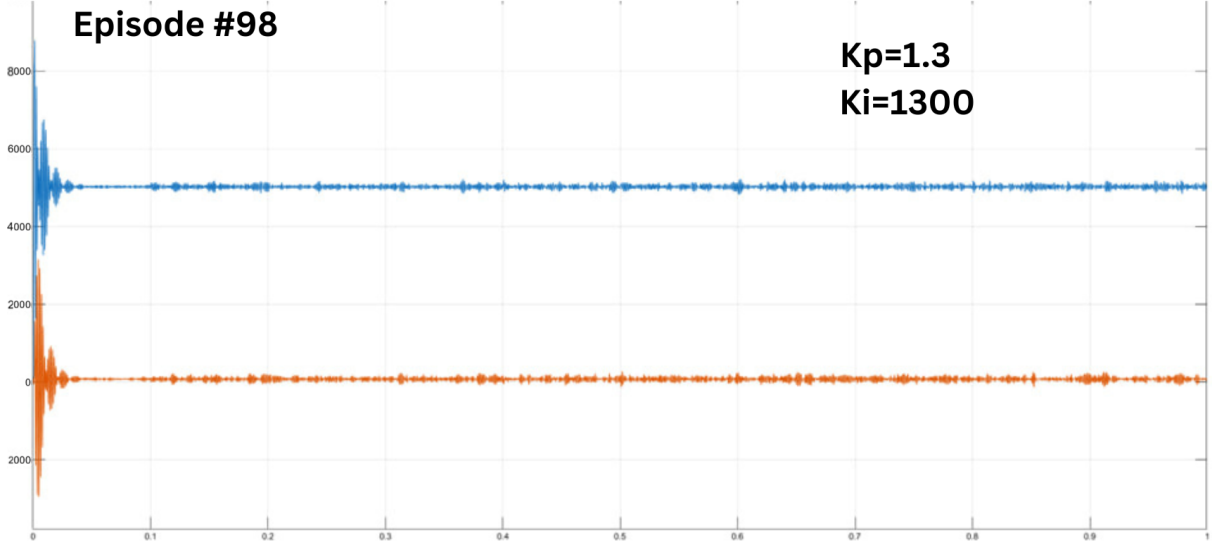


Figure 3.15: System response during Episode 98.

The obtained results show a substantial improvement compared to the previous episodes. The active power closely follows the reference value of 5000 W throughout the entire simulation interval with minimal oscillations and negligible steady-state error. The system response becomes significantly smoother, indicating that the controller gains selected by the agent provide an effective compromise between tracking accuracy and stability. The waveforms remain balanced and exhibit good sinusoidal characteristics, confirming that the adaptive controller maintains proper grid integration conditions while ensuring accurate power regulation. These results demonstrate the ability of the reinforcement learning agent to continuously improve its control policy through interaction with the environment and to identify controller gains that provide superior performance compared to the initial training stages.

The final gains obtained after convergence were approximately:

$$K_p = 1.3 \quad (3.14)$$

$$K_i = 1300 \quad (3.15)$$

These values produced the best overall performance during training and resulted in the most accurate active power tracking, reduced oscillations, and improved dynamic behavior of the grid-connected system.

3.5 TD3-Based Adaptive Control of the Dual-Star Generator

3.5.1 Motivation and Control Objective

The dual-star synchronous generator is coupled to the DC bus through an uncontrolled rectifier. The rectifier output voltage must be regulated to a target value of 300 V in order to ensure stable operation of the conversion chain. In the conventional control architecture, a PI controller is responsible for generating the quadrature-axis current reference $I_{q,ref}$, which indirectly regulates the DC voltage through the field excitation of the machine.

While the direct-axis current loop retains its standard PI structure, the I_q reference generation is replaced by a TD3-based reinforcement learning agent. The agent learns to select the proportional and integral gains of a custom PI controller acting on the I_q error, with the explicit objective of driving the rectifier output voltage to 300 V as accurately and as rapidly as possible.

The overall control structure showing both loops is presented in Figure 3.16.

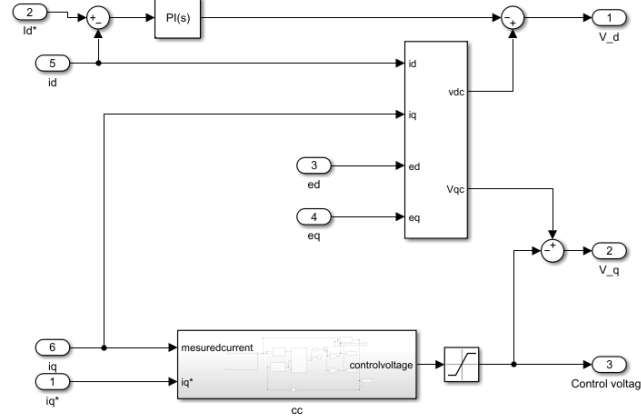


Figure 3.16: Generator-side control structure: conventional PI for I_d and TD3 agent for I_q reference generation.

3.5.2 Observation and Reward Design

Since only the I_q loop is handled by the agent, the observation vector is reduced to three signals extracted from the single tracking error:

$$e_q = I_{q,ref} - I_q \quad (3.16)$$

The three components are the instantaneous error, its integral, and its derivative:

$$Obs = \left[e_q, \int e_q dt, \dot{e}_q \right] \quad (3.17)$$

The internal structure of the observation subsystem is shown in Figure 3.17.

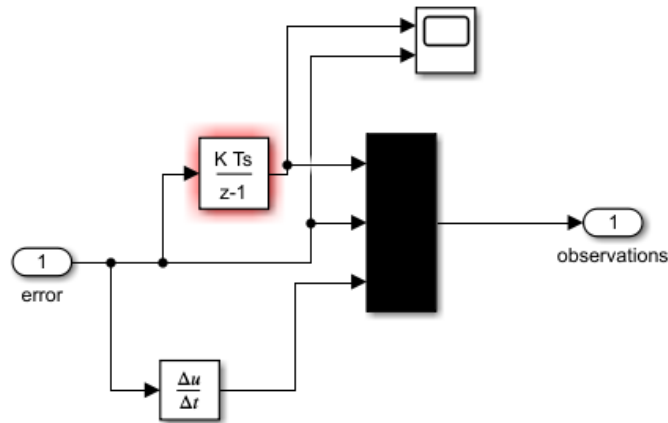


Figure 3.17: Observation subsystem for the generator-side TD3 agent.

The reward function is designed around two objectives: minimizing the deviation of the rectifier output voltage V_{dc} from the 300 V reference, and penalizing slow convergence. A time-weighted penalty is therefore applied:

$$r(t) = -|V_{dc} - 300| \cdot (1 + \lambda t) \quad (3.18)$$

where λ is a positive weighting coefficient that amplifies the penalty as time progresses. This formulation encourages the agent to reach the voltage target not only accurately but also quickly, since lingering errors accumulate increasingly large negative rewards. The complete reward subsystem is illustrated in Figure 3.18.

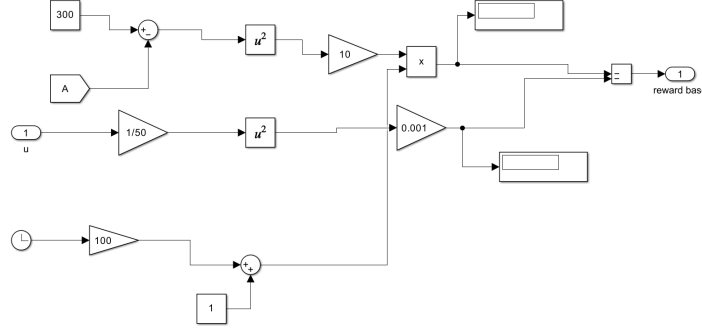


Figure 3.18: Reward calculation subsystem for the generator-side agent.

3.5.3 Agent Architecture and Training

The TD3 algorithm, actor and critic network structures, custom PI controller implementation, and batched training procedure follow the same design described in Section ?? for the grid-connected system. The only structural differences are:

- The observation space has dimension $numObs = 3$ instead of 6, reflecting the single regulated loop.
- The action space remains $numAct = 2$, corresponding to the gains $[K_p, K_i]$ of the I_q PI controller.
- The environment is built around the generator-rectifier Simulink model rather than the grid inverter model.

The complete TD3 system integrated with the generator model is shown in Figure 3.19.

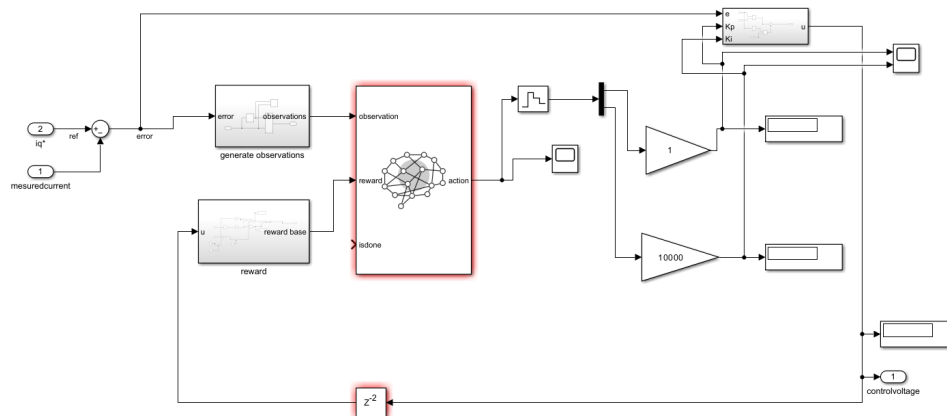


Figure 3.19: Complete TD3-based adaptive control system for the dual-star generator.

3.5.4 Training Results

The learning progression of the agent is evaluated by examining the rectifier output voltage at selected training episodes, as presented in Figure 3.20.

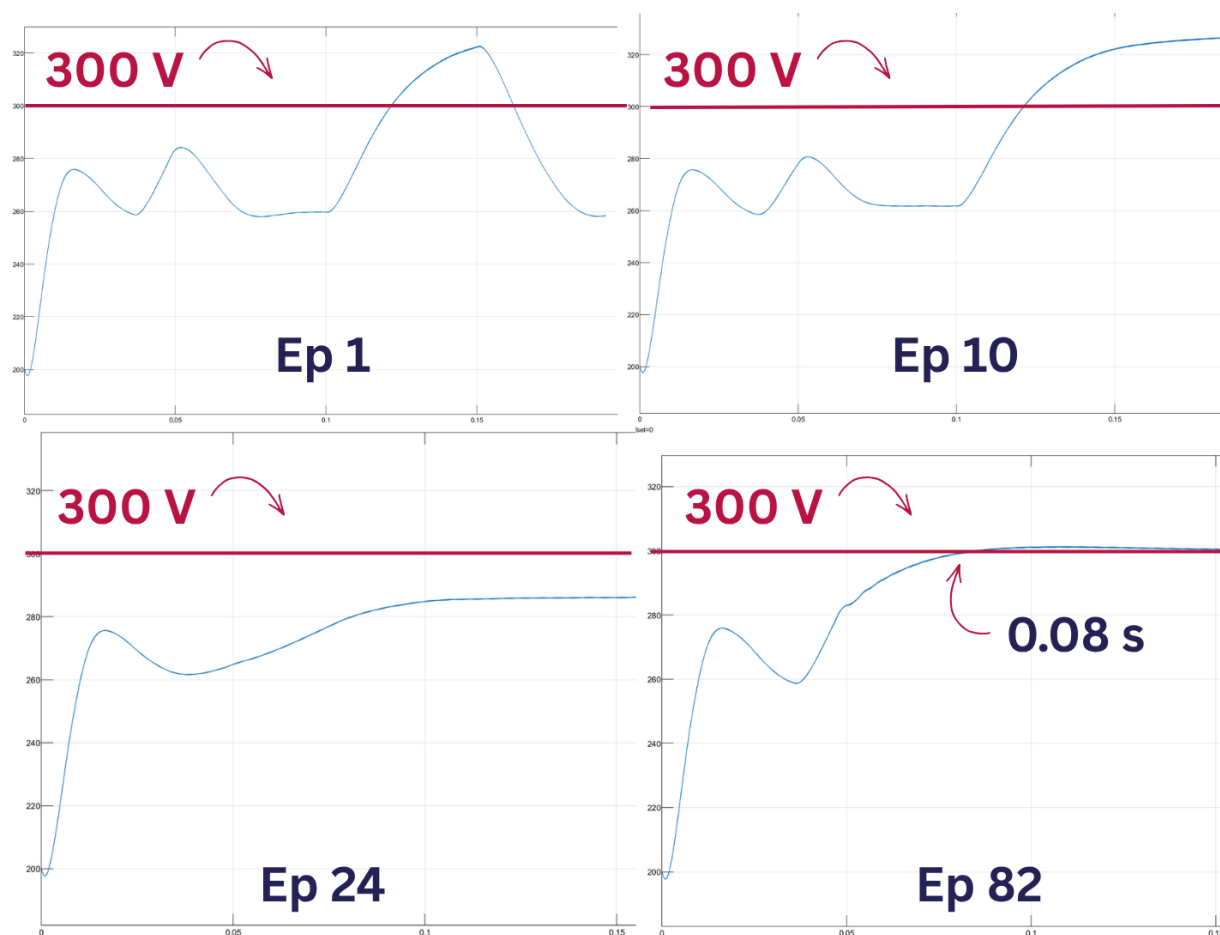


Figure 3.20: Rectifier output voltage across selected training episodes, illustrating the learning progression of the TD3 agent.

3.5.5 Observations and Interpretation

The training results demonstrate a clear improvement in controller performance as the number of episodes increases. During the initial episodes, the system was unable to reach the reference voltage of (300,V) and exhibited highly oscillatory behavior. As learning progressed, the response gradually became more stable and the voltage converged closer to the desired value.

By Episode 24, most of the instability observed during the early training phase had disappeared, although the controller still required additional time to reach the reference. After approximately 82 episodes, the agent successfully learned controller gains that allowed the system to reach (300,V) in about (0.08,s), which represents an improvement compared to the conventional control strategy.

This behavior confirms the effectiveness of the proposed reward function. Since the reward penalizes both the voltage error and the time required to eliminate that error, the agent learns not only to reach the reference value but also to achieve it as quickly as possible.

This improvement can also be observed in the next figure, which presents the measured (I_q) and the corresponding (I_q) reference. During the early training episodes, the measured current exhibits

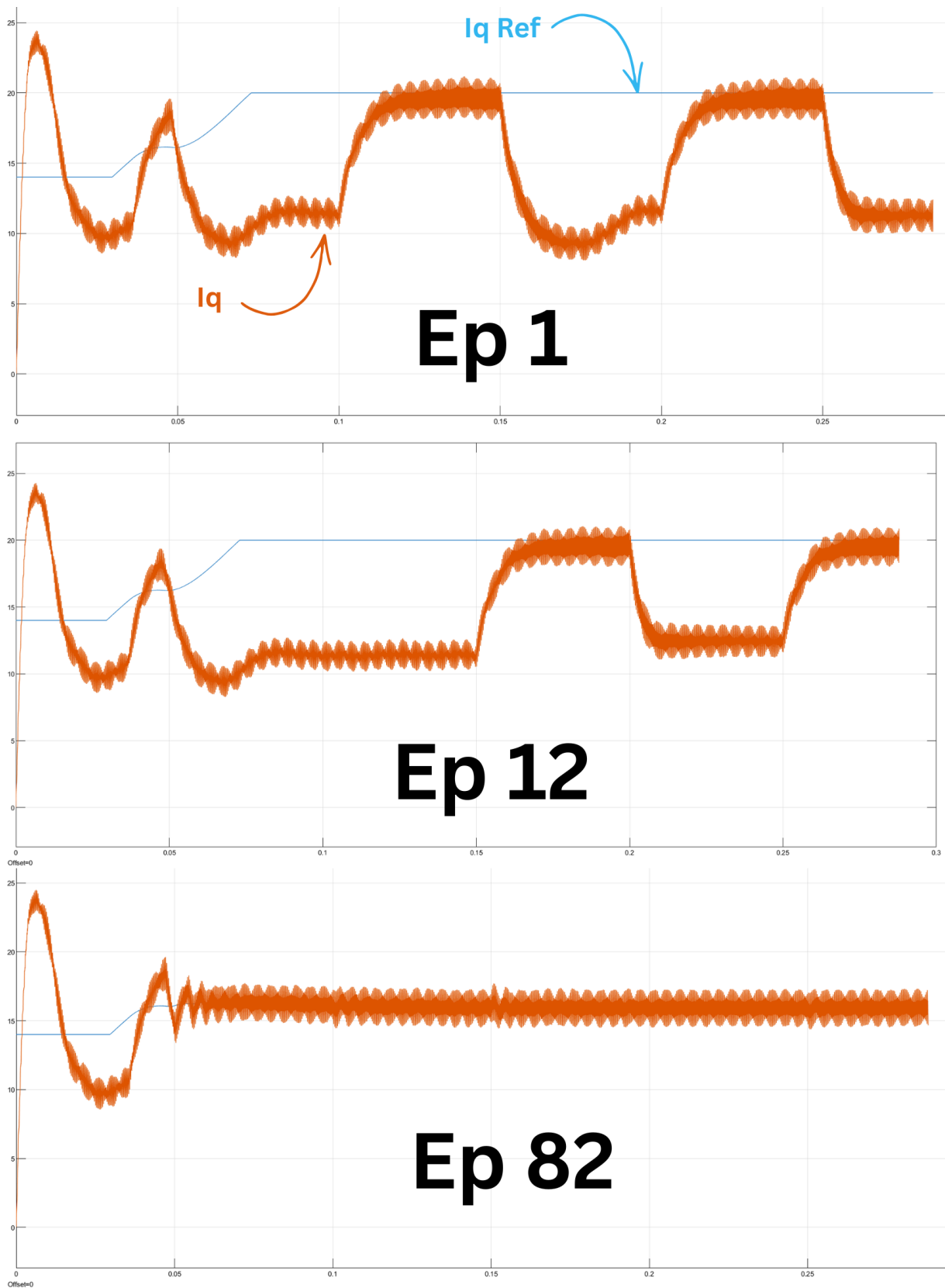


Figure 3.21: I_q Ref And I_q Results.

3.5.6 Comparison with Conventional PI Control

To assess the benefit of the learned controller, the performance of the TD3-tuned system is compared against the fixed-gain PI controller. Figure 3.22 presents both responses under identical operating conditions.

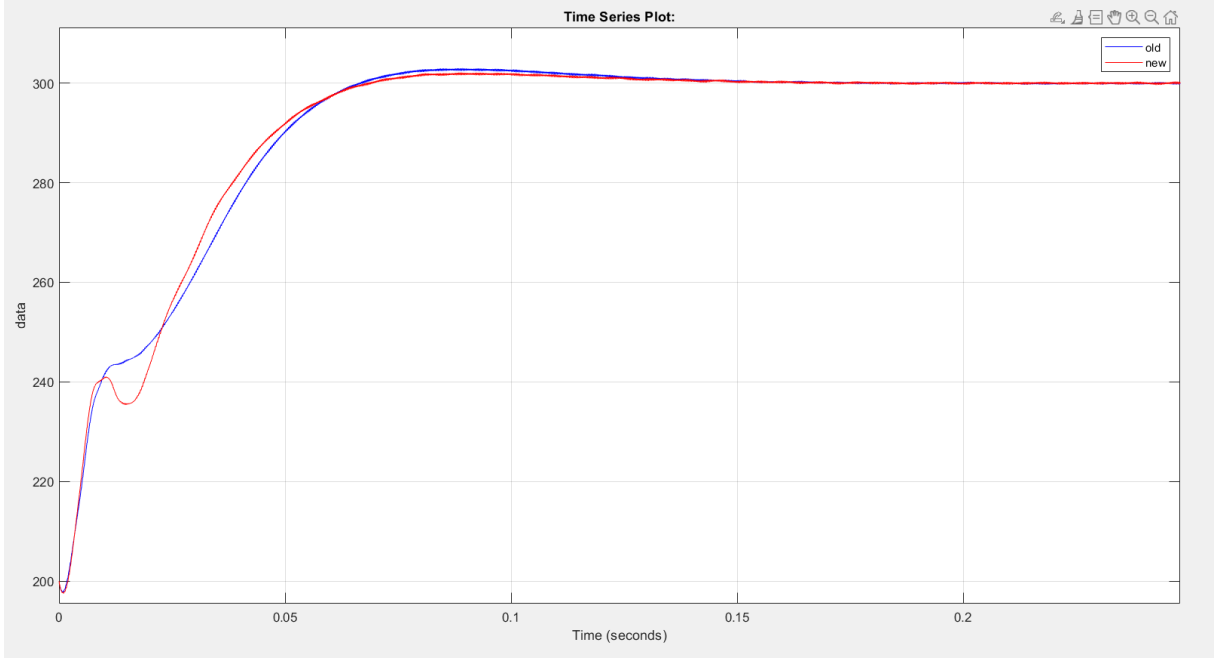


Figure 3.22: Comparison of rectifier output voltage: fixed PI controller versus TD3-tuned adaptive controller.

The results show that the adaptive controller (in red) achieves a faster rise toward 300 V, a reduced settling time, and a lower steady-state oscillation amplitude compared to the conventional approach. While the improvement margin is moderate, it demonstrates that the reinforcement learning agent successfully identified controller gains that offer better dynamic performance than manually tuned fixed parameters, without requiring any prior knowledge of the plant model.

3.6 Conclusion

This chapter presented the complete control strategy developed for the energy conversion chain, addressing both the grid integration stage and the reinforcement learning-based PI tuning applied to two distinct subsystems.

The first part established the theoretical and practical foundations of grid-connected inverter control. The Clarke and Park transformations were used to convert three-phase AC quantities into DC components in the synchronous dq reference frame, enabling the use of PI controllers for independent regulation of active and reactive power. A three-phase PLL ensured continuous synchronization between the inverter and the utility grid. The simulation results confirmed that the conventional control strategy achieves stable grid integration, with the active power accurately tracking step reference changes between 3 kW and 7 kW, and the grid voltages and currents remaining balanced and sinusoidal throughout operation.

The second part introduced a TD3-based reinforcement learning agent as a supervisory layer for automatic PI gain tuning. For the grid-side inverter, the agent successfully converged toward optimal gains, producing accurate current tracking with reduced oscillations compared to manually tuned fixed-gain controllers. The same approach was then extended to the PWM rectifier connected to the dual-star synchronous generator, where the TD3 agent was tasked with tuning the I_q PI controller to regulate the DC output voltage to 300 V as quickly and accurately as possible. The time-weighted reward function encouraged the agent to minimize both the steady-state error and the settling time simultaneously. After approximately 82 training episodes, the agent successfully learned controller gains that allowed the system to reach the reference voltage in approximately 0.08 s, outperforming the fixed-gain conventional approach in terms of rise time and settling behavior.

Taken together, these results demonstrate that the combination of conventional PI control with TD3-based adaptive tuning offers a practical and effective approach to improving the dynamic performance of grid-connected energy conversion systems, without requiring detailed analytical models of the plant.

General Conclusion

This work set out to investigate the integration of reinforcement learning into the control architecture of a Dual-Star Synchronous Generator coupled to a PWM rectifier and connected to the electrical grid. Two distinct approaches were studied and implemented from the ground up: the direct replacement of the conventional PI controller using a DDPG agent, and the adaptive tuning of PI controller gains using a TD3 agent, applied to both the PWM rectifier current loop and the grid-side inverter current controllers.

From a technical standpoint, the results obtained across the three chapters demonstrate that reinforcement learning is a viable and effective tool for control system design in power electronics applications. The DDPG agent successfully learned a stable control policy capable of tracking the reference current and supporting the outer voltage regulation loop under dynamic operating conditions, achieving performance comparable to a well-tuned conventional PI controller. The TD3-based adaptive tuning approach produced further improvements, with the agent converging toward controller gains that offered faster dynamic response, reduced settling time, and lower steady-state oscillations compared to manually selected fixed-gain parameters. The time-weighted reward function designed for the PWM rectifier application proved particularly effective, as it guided the agent to minimize not only the voltage error but also the time required to reach the reference, reaching the 300 V target in approximately 0.08 s after convergence.

Beyond the technical results, this project represented something significantly broader for us. Reinforcement learning was an entirely new field that neither of us had studied before. Everything presented in this work the theoretical foundations, the algorithm implementations, the Simulink environments, the reward function designs, the observation spaces, the training pipelines was built from scratch, learned independently, and developed through our own research and experimentation. There was no course to follow, no step-by-step procedure handed to us, and no existing template to replicate. Every design decision, from how to structure the observation vector to how to penalize the reward function, required genuine thought, creativity, and iteration.

This experience taught us that reinforcement learning is not a fixed algorithm one simply applies to a problem. It is a framework that demands careful and original engineering at every stage: designing the environment, shaping the reward, selecting and building the neural network architectures, tuning the hyperparameters, and interpreting the results. Each application scenario in this work required a custom solution designed specifically for it, and no two implementations were identical. The process was demanding, often uncertain, and at times frustrating but it was also deeply rewarding.

Through this work, we developed a genuine passion for artificial intelligence and its applications in engineering. Witnessing an agent that starts with no knowledge of a system gradually learn, adapt, and ultimately produce a functional and improved control policy is, quite simply, remarkable. The ability of these algorithms to discover solutions through interaction rather than through explicit programming represents a fundamental shift in how complex engineering problems can be approached, and we believe it opens possibilities that conventional methods alone cannot offer.

This project has given us not only technical skills in reinforcement learning, MATLAB, and Simulink, but also the confidence and curiosity to continue exploring this field independently. We leave it convinced that the combination of classical control theory and

modern reinforcement learning is one of the most promising directions in the design of advanced energy conversion systems, and we hope this work contributes, in its own modest way, to the growing body of research at the intersection of these two disciplines.

Bibliography

- [1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2nd ed. MIT Press, 2014.
- [2] Timothy P. Lillicrap et al. “Continuous Control with Deep Reinforcement Learning”. In: *International Conference on Learning Representations (ICLR)*. 2016. DOI: 10.48550/arXiv.1509.02971.
- [3] Sirine Bhouri. *Understanding DDPG: The Algorithm That Solves Continuous Action Control Challenges*. Towards Data Science. Accessed: 3 May 2026. Dec. 2024. URL: <https://towardsdatascience.com/understanding-ddpg-the-algorithm-that-solves-continuous-action-control-challenges-742c67e0783a/>.
- [4] Hey Amit. *Twin Delayed Deep Deterministic Policy Gradient (TD3)*. Medium. Accessed: 3 May 2026. Nov. 2024. URL: <https://medium.com/@heyamit10/twin-delayed-deep-deterministic-policy-gradient-td3-fc8e9950f029>.
- [5] Scott Fujimoto, Herke van Hoof, and David Meger. “Addressing Function Approximation Error in Actor-Critic Methods”. In: *International Conference on Machine Learning (ICML)*. 2018. DOI: 10.48550/arXiv.1802.09477.
- [6] MathWorks. *RL Agent*. Reinforcement Learning Toolbox Documentation. Accessed: May 24, 2026. 2026. URL: <https://www.mathworks.com/help/reinforcement-learning/ref/rlagent.html>.
- [7] MathWorks. *Train TD3 Agent for PMSM Control*. Reinforcement Learning Toolbox Documentation. Accessed: May 26, 2026. 2026. URL: <https://www.mathworks.com/help/reinforcement-learning/ug/train-td3-agent-for-pmsm-control.html>.
- [8] Abdenour Bourahla and Nour-eddine Aymen Mamenia. “Etude de l’association d’un générateur synchrone à double étoile à un redresseur hexaphasé à diodes ou MLI alimentant un bus de tension continue”. Mémoire de fin d’études, Master en Électrotechnique. Higher School of Applied Sciences, Tlemcen, 2025.
- [9] Salar Moradi et al. “Hierarchical-power-flow-based energy management for alternative/direct current hybrid microgrids”. In: *Sustainable Energy, Grids and Networks* 38 (June 2024), p. 101384. DOI: 10.1016/j.segan.2024.101384.
- [10] MathWorks. *Create Simulink Environment and Train TD3 Agent*. <https://www.mathworks.com/help/releases/R2023a/reinforcement-learning/ug/create-simulink-environment-and-train-agent.html>. Accessed June 2026. 2023.

- [11] MathWorks. *Tune PI Controller Using TD3 Agent*. <https://www.mathworks.com/help/reinforcement-learning/ug/tune-pi-controller-using-td3.html>. Accessed June 2026. 2023.
- [12] MathWorks. *Twin Delayed Deep Deterministic Policy Gradient (TD3) Agents*. <https://www.mathworks.com/help/reinforcement-learning/ug/td3-agents.html>. Accessed June 2026. 2023.

Abstract

This work investigates the integration of reinforcement learning into the control architecture of a Dual-Star Synchronous Generator coupled to a PWM rectifier and connected to the electrical grid. Two complementary approaches are studied and implemented. The first approach consists of directly replacing the conventional proportional-integral controller in the inner quadrature-axis current loop of the PWM rectifier with a Deep Deterministic Policy Gradient agent, which learns to generate the control voltage directly from system observations through interaction with a Simulink environment. The second approach preserves the PI controller structure while employing a Twin Delayed Deep Deterministic Policy Gradient agent as a supervisory layer that automatically tunes the proportional and integral gains according to the current operating conditions. This adaptive tuning strategy is applied to both the PWM rectifier current controller and the grid-side inverter current controllers. For the grid integration stage, the conventional control strategy based on Clarke and Park transformations, Phase Locked Loop synchronization, and dq-frame PI current control is first implemented and validated. The TD3 agent is then introduced to replace manual gain selection, and a time-weighted reward function is specifically designed to encourage fast and accurate regulation of the rectifier output voltage toward a 300 V reference. The results demonstrate that both reinforcement learning approaches achieve performance comparable to well-tuned conventional controllers, confirming the feasibility of integrating AI-based adaptive control into classical power electronics architectures.

This work paves the way for significant future research. Future work should focus on experimental validation through real test bench implementation, exploration of advanced control strategies using machine learning approaches, and investigating the use of these multilevel inverter topologies for grid-connected renewable energy applications, addressing the key challenges associated with grid synchronization, power quality, and compliance with grid codes.

Résumé

Ce mémoire étudie l'intégration de l'apprentissage par renforcement dans l'architecture de contrôle d'un générateur synchrone à double étoile couplé à un redresseur MLI et connecté au réseau électrique. Deux approches complémentaires sont étudiées et mises en œuvre. La première consiste à remplacer directement le régulateur proportionnel-intégral conventionnel de la boucle interne de courant en quadrature par un agent Deep Deterministic Policy Gradient, qui apprend à générer la tension de commande directement à partir des observations du système par interaction avec un environnement Simulink. La seconde approche préserve la structure du régulateur PI tout en employant un agent Twin Delayed Deep Deterministic Policy Gradient comme couche superviseuse qui ajuste automatiquement les gains proportionnel et intégral en fonction des conditions de fonctionnement actuelles du système. Cette stratégie de réglage adaptatif est appliquée à la fois au régulateur de courant du redresseur MLI et aux régulateurs de courant de l'onduleur côté réseau. Pour l'étape d'intégration au réseau, la stratégie de contrôle conventionnelle basée sur les transformations de Clarke et Park, la synchronisation par boucle à verrouillage de phase et la régulation de courant dans le repère dq est d'abord mise en œuvre et validée. L'agent TD3 est ensuite introduit pour remplacer le réglage manuel des gains, et une fonction de récompense pondérée dans le temps est spécifiquement conçue pour encourager une régulation rapide et précise de la tension de sortie du redresseur vers une référence de 300 V. Les résultats démontrent que les deux approches par apprentissage par renforcement

atteignent des performances comparables à celles des régulateurs conventionnels bien réglés, confirmant la faisabilité de l'intégration d'un contrôle adaptatif basé sur l'intelligence artificielle dans les architectures classiques d'électronique de puissance.

ملخص

يتناول هذا العمل دراسة دمج تعلّم التعزيز في بنية التحكم الخاصة بمولّد متزامن ثنائي النجمة، مقترن بمقوم تعديل عرض النبضات ومتصل بالشبكة الكهربائية. تُدرّس مقاربتان متكاملتان وتنفّذان على أرض الواقع. تتمثل المقاربة الأولى في استبدال منظّم النسبي-التكاملي التقليدي في حلقة التحكم الداخلية لتيار المحور التريبيعي بعامل تعلّم آلي يعتمد خوارزمية تدرّج السياسة الحتمية العميقة، إذ يتعلّم هذا العامل توليد جهد التحكم مباشرةً من مشاهدات النظام عبر التفاعل مع بيئة محاكاة. أما المقاربة الثانية فتحافظ على بنية المنظّم النسبي-التكاملي، مع توظيف عامل خوارزمية تدرّج السياسة الحتمية العميقة المزدوجة ذات التحديث المتأخر كطبقة إشرافية تضبط مكاسب المنظّم تلقائياً وفقاً لظروف تشغيل النظام الآنية. تُطبّق هذه الاستراتيجية التكيّفية على منظّم تيار مقوم تعديل عرض النبضات ومنظّمات تيار العاكس من جهة الشبكة على حدّ سواء. في مرحلة الربط بالشبكة، يُطبّق أولاً نظام التحكم التقليدي المستند إلى تحويلات كلارك وبارك ونظام حلقة إقفال الطور والتحكم بالتيار في الإطار الدوّار، ثم يُدرّج عامل التعلّم التكيّفي بديلاً عن الضبط اليدوي للمكاسب، حيث تُصمّم دالة مكافأة مرّجحة زمنياً لتحفيز التنظيم السريع والدقيق لجهد خرج المقوم نحو قيمة مرجعية تبلغ ثلاثمائة فولت. تُبيّن النتائج أن كلتا المقاربتين تُحقّقان أداءً مماثلاً لأداء المنظّمات التقليدية المضبوطة بعناية، مما يؤكد جدوى دمج التحكم التكيّفي المبني على الذكاء الاصطناعي في بنى الإلكترونيات الكهربائية الكلاسيكية.