Institute of Electrical Engineering and Information Technology
Paderborn University
Department of Power Electronics and Electrical Drives

Master Thesis

Model-based Exploration for Reinforcement Learning in Power Electronic Systems

by

Rushi Nikunjkumar Dave

Student ID: 6854683

Supervisor: Dr.-Ing. Oliver Wallscheid, Daniel Weber and Maximilian Schenke

Thesis Nr.: MA 130

Filing Date: August 14, 2023

Declaration of Authorship

I declare that I have authored this thesis independently, that I have not used other than the declared sources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources. This paper was not previously presented to another examination board and has not been published.

Paderborn, 14/08/2023 

Date Signature

Abstract

Reinforcement learning (RL) controllers relying on exploration techniques during training to sufficiently excite the system to learn an optimal policy. Deep deterministic policy gradient, a well-known RL algorithm, uses ornstein–uhlenbeck process superimposed on the action given by its deterministic policy. This method of exploration is neither guided nor temporally correlated, leading to a need for longer training and is data inefficient. Furthermore, the random nature of this exploration is a threat to safety-critical systems, such as electrical power grids, and does not provide a framework to ensure well-balanced state space coverage.

To address this issue, model-based predictive control (MPC) technique is used to generate targeted-exploration trajectories to obtain information-rich samples. A novel histogram-based coverage algorithm is introduced, which ensures that samples from all regions of state space are collected using MPC. This coverage algorithm is capable to operate in dynamic setting, providing a real-time identification of regions of low density. Using this improved method of exploration, an upgraded RL agent is designed and then deployed on grid-forming 3-phase voltage source inverter with unknown resistive load. The upgraded RL agent is able to learn an optimal policy with half of training time compared to its counter-part. Additionally, it proves to be more robust to changes at the load-side in islanded operation mode.

Contents

1 Introduction **1**

2 Microgrids **3**
2.1 System Model 4
2.2 Simulation Environment 6

3 Reinforcement Learning **8**
3.1 Introduction 8
3.2 Reinforcement Learning Problem 9
3.3 Deep Reinforcement Learning Algorithms 12
3.3.1 Deep Q-learning 12
3.3.2 Deep Deterministic Policy Gradient 13
3.4 Task Specific Implementation of Reinforcement Learning Controller 15

4 Model Predictive Control **17**

5 Histogram-based Count Maximization Algorithm **22**
5.1 Grid World Example 34

6 Upgraded Reinforcement Learning Agent **38**

7 Evaluation and Conclusion **43**
7.1 Performance of basic RL agent 45
7.2 Performance of upgraded RL agent 47
7.3 Conclusion and Outlook 53

Appendix **54**
A.1 Iteration of Exploration in Grid World 54
A.2 Safeguard 58

Lists **60**
List of Tables 60
List of Figures 60
Acronyms 62
Glossary 63
Nomenclature 63

References 63

1 Introduction

Data-driven control approaches such as reinforcement learning (RL) allow a self-adaptive and model-free controller design with minimal human effort. This makes them interesting for the control of continuously evolving systems such as the electrical power grid. The reinforcement learning-based controller, also known as an RL agent, learns by interacting with the system. [1] shows that RL agents, trained using deep deterministic policy gradient (DDPG) algorithm, are suitable for control applications of three-phase voltage source inverters.

DDPG agent learns a deterministic policy and uses additional action noise for exploration. In [2] ornstein–uhlenbeck (OU) noise, a stochastic process, is used to generate this action noise. The goal of exploration is to excite the system and learn from its response. However, the stochastic behavior of action noise, which is essential to provide optimal system excitation, does not take the safety of the system into account. Furthermore, this method of exploration does not ensure uniform coverage of state-space. An insufficiently explored state-space could lead to sub-optimal policy as the agent has never seen data from underrepresented regions during the training phase. Therefore, model-free property of RL comes with its challenges. On the other hand, optimal control methods such as model predictive control (MPC) facilitate the integration of operational constraints based on a prior model [3]. In this context, the system's behavior and constraints are inherently factored into the optimization process to determine the optimal control input, rendering this approach model-based which complicates adaptability. This makes the fusion of model-based algorithms for safety and model-free algorithms for flexibility highly advantageous.

To avoid the need of an accurate system model which may or may not be available, system identification techniques can be used. [4] identifies a linear model of the system online, using recursive least squares (RLS) algorithm, to validate safety during and after the training process for a DDPG agent. The developed safeguard not only guarantees safety but also improvement in the learning process. As this model has to be identified to ensure safety, it can also be used for exploration during the learning process. Therefore, the focus of this thesis is to use the available or identified information as efficiently as possible in order to improve the learning behaviour.

Based on the assumption that a linear state-space model is available, MPC is used to generate targeted-exploration trajectories during the training phase. These trajectories are later used by the RL agent to learn an optimal policy. The motivation behind these trajectories is to ensure that all regions of state-space are visited and uniformly represented in the replay buffer of DDPG. A histogram-based count maximization algorithm (HCXA) is developed to choose the targets for MPC in a dynamic setting. This algorithm provides a framework to ensure enough information-rich samples are collected from all parts of state-space during the learning process. The upgraded RL agent is then deployed in microgrid setting where it controls the modulation indices given to a three-phase voltage source inverter connected to resistive load.

Section 2 outlines the hierarchical structure of microgrids. An overview on reinforcement learning is provided in section 3. Introduction to receding horizon control and MPC is given section 4. The novel algorithm used for uniform state-space coverage is explained in Section 5. Section 6 offers a comprehensive presentation of the main algorithm followed by evaluation and results in section 7.

2 Microgrids

Centralized electric power generation, such as coal, gas, and nuclear power plants, are located far from the areas where they are used. To transport this generated electricity, high-voltage transmission lines and distribution systems are used. On the contrary, microgrids (MG) involve small-scale generation on-site. These are small electrical distribution systems that connect multiple consumers to multiple distributed sources of generation and storage. The idea of microgrids provides an appropriate solution to integrate more and more renewable energy sources (RES) in the existing distribution networks. A microgrid is capable of operating in both islanded and grid-connected modes, which increases the reliability of power grids. However, it also has many significant challenges from a stability and control point of view[5]. Low inertia, uncertainties, dynamic modeling, and bidirectional power flow are known to be the most relevant challenges in MGs control and protection[5].

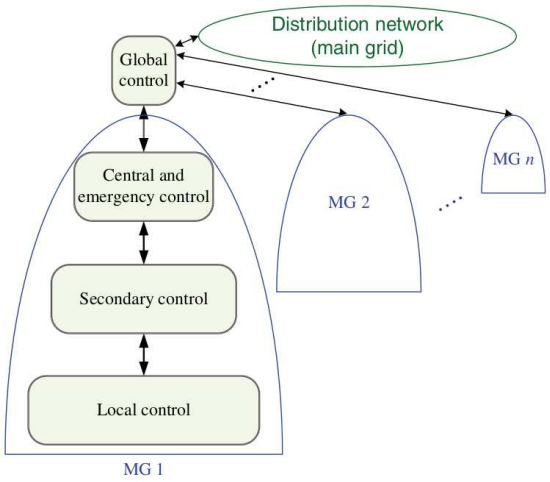


Fig. 2.1: Hierarchical control levels in microgrids[5]

Many control loops are used to improve the microgrid's stability and performance. The hierarchical control structure of MG is responsible for providing proper load sharing

and distributed generators (DG) coordination, voltage/frequency regulation in both the grid-connected mode and the islanded mode[5]. The hierarchical control strategy consists of four levels, local (primary), secondary, central / emergency, and global controls as shown in Fig. 2.1[5]. In this work, the primary control level is chosen for testing the upgraded RL agent. It includes fundamental control hardware, comprising DG's internal voltage and current control loops. Modern microgrids are largely driven by power electronic converters due to their high efficiency and flexibility. Many RESs such as fuel cells and photovoltaic (PV) plants cannot be directly connected to AC grid. Due to this, the ratio of the inverter-based generation system to the inertia-based generation system is increasing, placing greater responsibility on the performance of both grid-supporting and grid-forming inverter control systems, especially in islanded operation mode[5]. In islanded mode, the grid-forming inverter is responsible for generating the reference voltage that will be followed by other grid-following inverters. In this work, a three-phase voltage source inverter is the chosen topology to show a grid-forming inverter whose schematic diagram is shown in Fig. 2.2. The internal resistances of inductor L_1 and capacitor C_1 are not depicted in the diagram for simplicity.

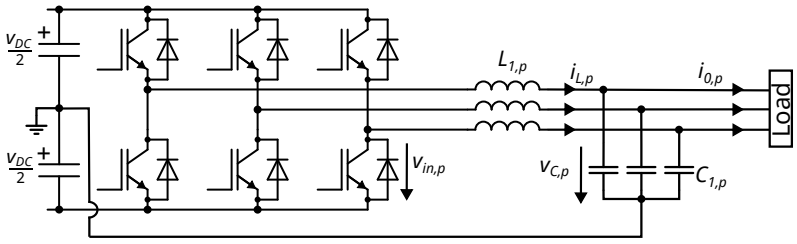


Fig. 2.2: Three-phase four-wire voltage source inverter connected to a load via LC filter.

2.1 System Model

In Fig. 2.2, a three-phase full-bridge inverter is connected to a symmetric, purely resistive load (R_{load}) via the LC filter. For an arbitrary phase $p \in \{a, b, c\}$, the values of the inductor and capacitor are given by $L_{1,p}$ and $C_{1,p}$, respectively. A DC link voltage v_{DC} is given as input to the inverter and represents the output of some DC source. The LC filter configuration also consists of internal resistance, which are not shown for simplicity. Using Kirchhoff's voltage and current rule, the dynamics of the above grid-forming inverter can be described using inductor current $i_{L,p}$ and capacitor voltage $v_{C,p}$:

$$i_{L,p} = C_{1,p}\frac{\mathrm{d}v_{C,p}}{\mathrm{d}t} + i_{0,p} \tag{2.1}$$

with switch voltage $v_{\text{in,p}}$

$$v_{in,p} = v_{C,p} + L_{1,p}\frac{\mathrm{d}i_{L,p}}{\mathrm{d}t} \tag{2.2}$$

The load current $i_{0,p}$ can be measured but is not used in this work. From Eq. (2.1) and (2.2), a state-space representation of the system can be formed.

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} + \boldsymbol{E}i_0 \\ \boldsymbol{y} &= \boldsymbol{C}\boldsymbol{x} \end{aligned} \tag{2.3}$$

with

$$\begin{aligned} \boldsymbol{x} &= \begin{bmatrix} i_{\mathrm{L,a}} \; i_{\mathrm{L,b}} \; i_{\mathrm{L,c}} \; v_{\mathrm{C,a}} \; v_{\mathrm{C,b}} \; v_{\mathrm{C,c}} \end{bmatrix}^{\mathrm{T}}, \\ \boldsymbol{u} &= \begin{bmatrix} v_{\mathrm{in,a}} \; v_{\mathrm{in,a}} \; v_{\mathrm{in,a}} \end{bmatrix}^{\mathrm{T}}. \end{aligned} \tag{2.4}$$

The state-space matrix are defined by

$$\begin{aligned} \mathbf{A} &= \begin{bmatrix} 0 & 0 & 0 & -\frac{1}{L_{1,a}} & 0 & 0 \\ 0 & 0 & 0 & 0 & -\frac{1}{L_{1,b}} & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{1}{L_{1,a}} \\ \frac{1}{C_{1,a}} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{C_{1,b}} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{C_{1,c}} & 0 & 0 & 0 \end{bmatrix}, \\ \mathbf{B} &= \begin{bmatrix} \frac{1}{L_{1,a}} & 0 & 0 \\ 0 & \frac{1}{L_{1,b}} & 0 \\ 0 & 0 & \frac{1}{L_{1,c}} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \mathbf{E} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{C_{1,a}} & 0 & 0 \\ 0 & \frac{1}{C_{1,b}} & 0 \\ 0 & 0 & \frac{1}{C_{1,c}} \end{bmatrix} \end{aligned} \tag{2.5}$$

The output matrix $\mathbf{C} = \boldsymbol{I}_6$ is an identity matrix, and therefore full observability is given. To simplify the control problem, the relevant currents and voltages are transformed from a fixed abc reference frame to a rotating reference frame as shown in Fig. 2.3. When the angular speed of rotating frame is set equal to the grid frequency, sinusoidal grid voltages and currents become DC-variables. The Park-Clarke transformation can be used to map the system variables to a rotating coordinate system.

$$\begin{bmatrix} x_d \\ x_q \\ x_0 \end{bmatrix} = \frac{2}{3} \begin{bmatrix} \cos(\theta) & \cos(\theta - \frac{2\pi}{3}) & \cos(\theta - \frac{4\pi}{3}) \\ -\sin(\theta) & -\sin(\theta - \frac{2\pi}{3}) & -\sin(\theta - \frac{4\pi}{3}) \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} x_a \\ x_b \\ x_c \end{bmatrix} \tag{2.6}$$

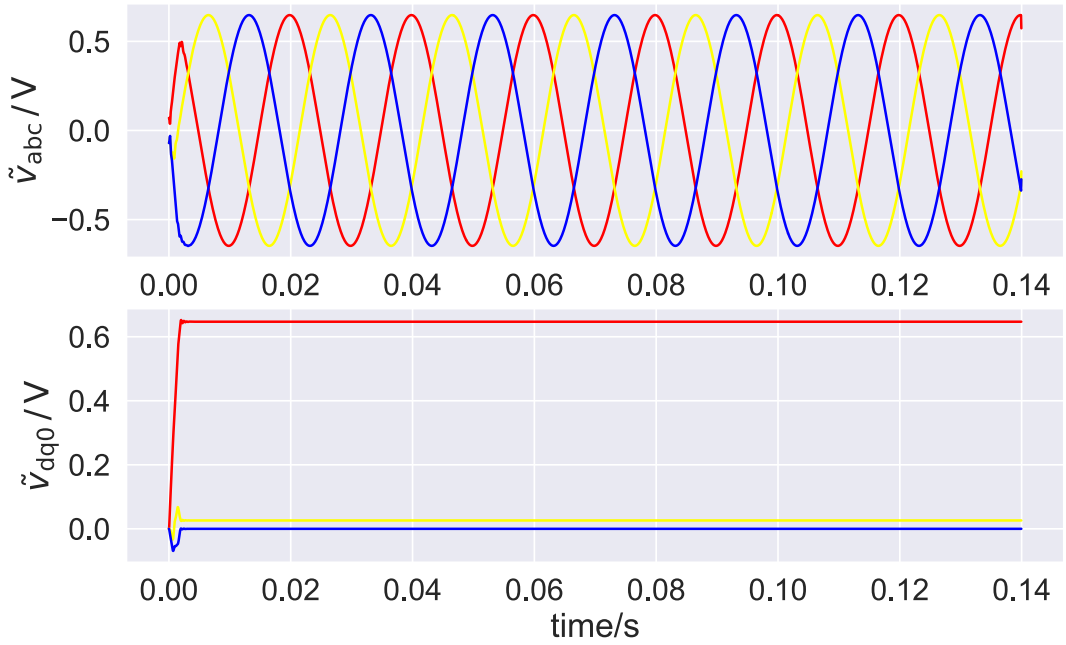


Fig. 2.3: The Park-Clarke Transformation.

2.2 Simulation Environment

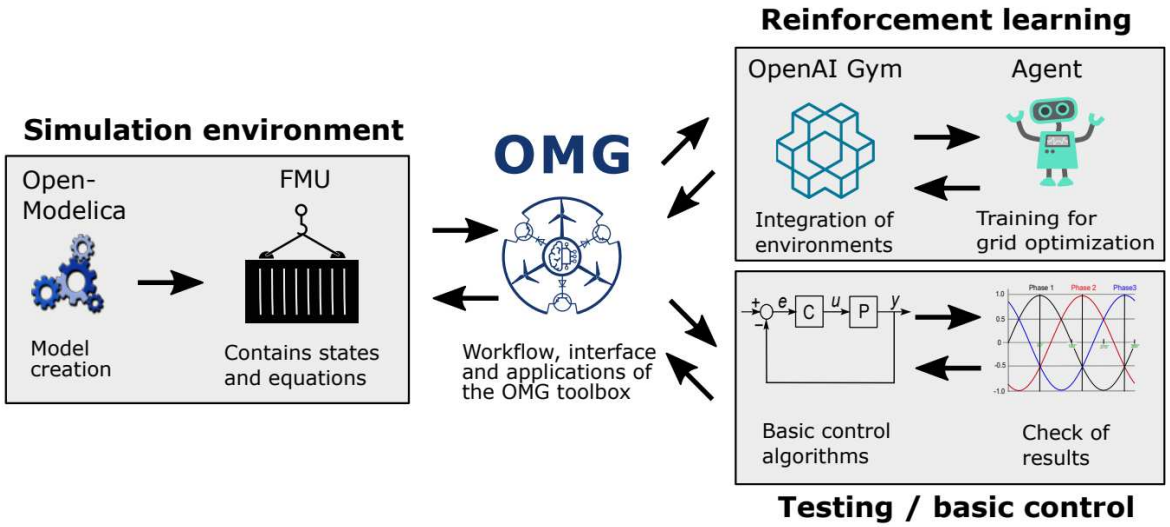


Fig. 2.4: Overview of the interconnections between the different parts of the OMG toolbox. The OpenModelica and OpenAI Gym logos are the property of their respective owners.

OpenModelica Microgrid Gym (OMG), an open-source software toolbox for the simulation and control optimization of microgrid is used in the thesis as a simulation environment[1][6]. It is capable of modeling and simulating arbitary microgrid topologies and offers Python-based testing for plug & play controllers. A special feature of OMG is that it uses the standardized OpenAI Gym interface, allowing for easy integration of RL-based controller. An overview of the toolbox is shown in Fig. 2.4. In Fig. 2.5 is the setup that is used, in OpenModelica to create a functional mock-up unit(FMU), to generate differential equations (2.1)(2.2) [6].

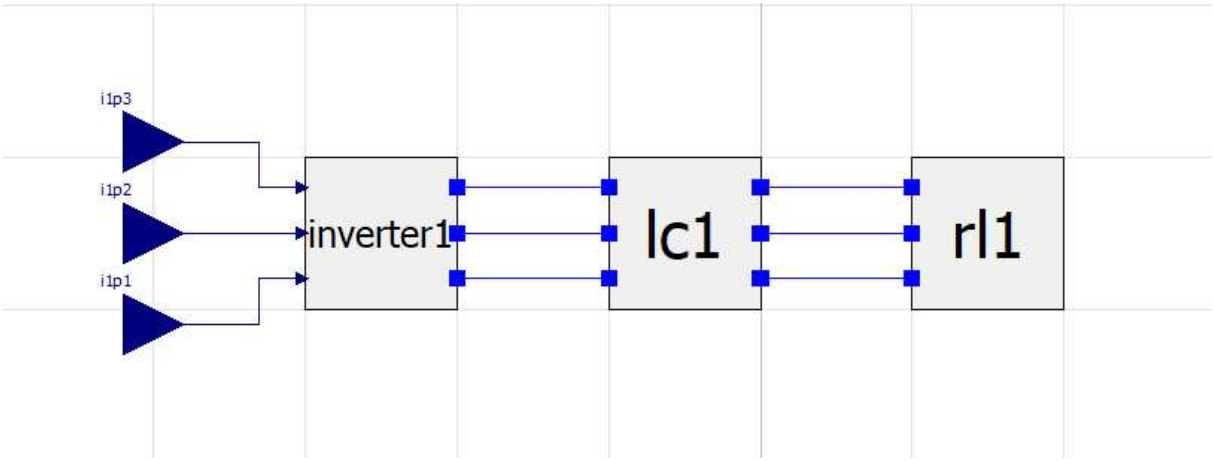


Fig. 2.5: Schematic view of example grid in OpenModelica.

[1]Github: https://github.com/upb-lea/openmodelica-microgrid-gym

3 Reinforcement Learning

Model-free controllers have been employed in microgrid control because of their independence from the physical model of the microgrid components. Fuzzy logic controllers and adaptive controllers, for instance, can modify their output based on pre-defined membership functions and adaption laws, respectively[7][8]. However, they are difficult to expand and cannot handle the emerging uncertainties in microgrids. Reinforcement learning provides a more intelligent way to learn by repeatedly interacting with the environment. In [1], a standard deep deterministic policy gradient (DDPG) algorithm is extended with an integral action to minimize steady-state error. No prior knowledge was required for the enhancement, which makes it a model-free technique. In this section, a brief introduction to reinforcement learning is given.

3.1 Introduction

Reinforcement learning is a field of machine learning (ML) in which the ML model(the agent) learns — how to map situations to actions — by interacting with the system(the environment). The goal of RL agent is to maximize the reward[9]. At every interaction with the environment, the agent receives the state and the reward signal. The state at any timestep is given by $\boldsymbol{x}_k$ and represents the current system dynamics. It can contain information about position, velocity, orientation, or any other relevant factors that describe the system at timestep k. In the case of a grid-forming inverter, measured voltage and current are used as states along few additional features which are discussed in section 3.4. The reward signal r_k is a numerical value that represents how good or bad the past action $\boldsymbol{u}_k$ was given the past state $\boldsymbol{x}_k$ in the context of the task. Similar to other ML fields, the agent also has a training and testing phase. During the training process, the agent uses the state $\boldsymbol{x}_k$ to make decisions and select an action $\boldsymbol{u}_k$, which it applies to interact with the environment. As a result, the agent obtains the next state $\boldsymbol{x}_{k+1}$ and the corresponding reward signal r_{k+1}. However, when the agent is deployed to a test environment, the exploration is stopped and agent has to solely exploit what it has learned during training.

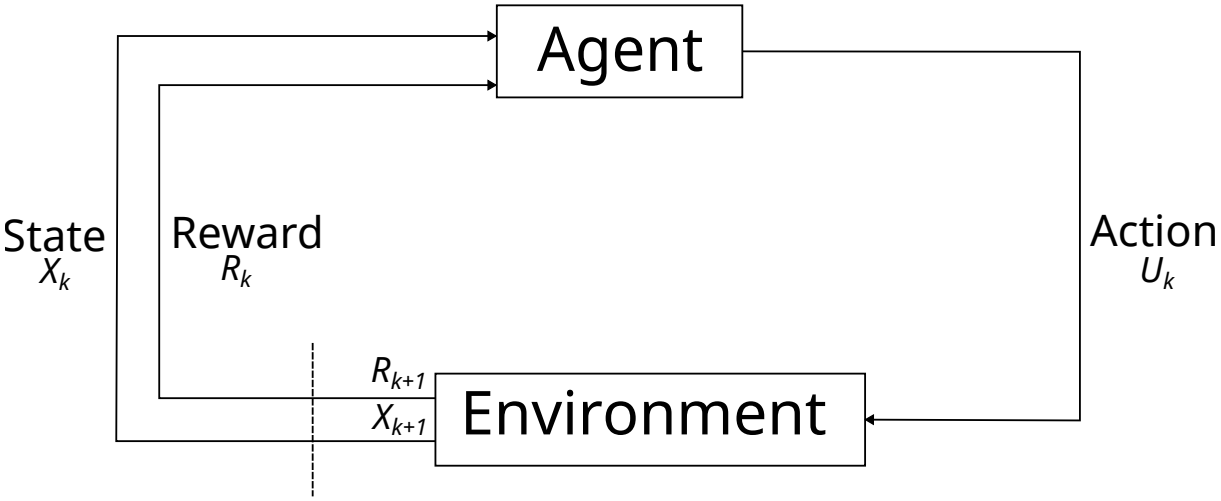


Fig. 3.1: The agent-environment interaction in a Markov decision process

3.2 Reinforcement Learning Problem

Markov decision process (MDP) is a mathematical framework for deterministic and stochastic processes used to model sequential decision-making problems. To solve any optimal control problem using reinforcement learning, the problem must be described using MDP framework. It is an extension of the Markov chain (MC), a stochastic model that follows the Markov property. The Markov property, also known as memory-less property, states that the future state of the process/system/plant depends solely on the current state and is independent of the sequence of events that led to the current state. Markov property is mathematically described below. [1]

$$\mathbb{P}(\boldsymbol{X}_k = \boldsymbol{x}_k | \boldsymbol{X}_{k-1} = \boldsymbol{x}_{k-1}, ..., \boldsymbol{X}_0 = \boldsymbol{x}_0) = \mathbb{P}(\boldsymbol{X}_k = \boldsymbol{x}_k | \boldsymbol{X}_{k-1} = \boldsymbol{x}_{k-1}) \tag{3.1}$$

An MDP is a tuple $\langle \mathcal{X}, \mathcal{U}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ with:

- $\mathcal{X} \rightarrow$ set of all possible situations or states that the environment can be in. $\boldsymbol{X}_k \in \mathcal{X}$.
- $\mathcal{U} \rightarrow$ set of all possible control actions that the agent can take $\boldsymbol{U}_k \in \mathcal{U}$.
- $\mathcal{P} \rightarrow$ state transition probability $\mathcal{P} = \mathcal{P}^{\boldsymbol{u}}_{\boldsymbol{x}\boldsymbol{x}'} = \mathbb{P}[\boldsymbol{X}_{k+1} = \boldsymbol{x}' | \boldsymbol{X}_k = \boldsymbol{x}_k, \boldsymbol{u}_k = \boldsymbol{u}_k]$. These probabilities define the dynamics of the environment.
- $\mathcal{R} \rightarrow$ reward function, $\mathcal{R} = \mathcal{R}^{\boldsymbol{u}}_{\boldsymbol{x}} = \mathbb{E}[R_{k+1} | \boldsymbol{X}_k = \boldsymbol{x}_k, \boldsymbol{U}_k = \boldsymbol{u}_k]$. It maps the state-action pairs to immediate rewards.
- $\gamma \rightarrow$ discount factor, $\gamma \in \{\mathbb{R} | 0 \leq \gamma \leq 1\}$. It determines how much importance the agent places on immediate rewards vs. rewards obtained in the future.

[1] Capital letters represents a random variables, while lower case letters are used to represent realization of those random variables.

A crucial component of MDP is the policy (π). It is the strategy or the rule that guides the agent's decision-making process. It maps states to actions, indicating which action the agent should take in each state to maximize cumulative reward. Mathematically, it is defined as a distribution of actions over states (3.2).

$$\pi(\boldsymbol{u}_k|\boldsymbol{x}_k) = \mathbb{P}[\boldsymbol{U}_k = \boldsymbol{u}_k|\boldsymbol{X}_k = \boldsymbol{x}_k] \tag{3.2}$$

By learning a policy that maximizes the cumulative reward, the agent solves the control task. The sum of all the rewards accumulated up to the horizon is known as *return*. Based on horizon of the task, it can be classified into two categories: episodic task and continuing task. A task is classified as episodic if it has a finite time horizon, meaning it has a well-defined beginning and end. The task is divided into distinct episodes, each with a clear start and finish. On the other hand, a task is considered a continuing task if it has an infinite time horizon, meaning there is no predefined end point. In the case of a continuing task, the expected return can potentially be infinite, which poses a challenge. To address this, the concept of a discount factor (γ) is introduced. The revised formulation of the expected return is expressed as the sum of discounted rewards (3.3). The value of γ, which ranges from 0 to 1, determines how rewards are weighted. A smaller γ leads to a more short-sighted agent, where immediate rewards are favored, while a value closer to or equal to 1 treats future rewards with equal importance as immediate rewards, providing a more farsighted expected return. This allows for an approach that is capable of handling rewards in both episodic and continuing tasks.

$$\begin{aligned} G_k &= R_{k+1} + \gamma R_{k+2} + \gamma^2 R_{k+3} + ... = \sum_{i=0}^{\infty} \gamma^i R_{k+i+1} \\ &= R_{k+1} + \gamma R_k t + 2 + \gamma^2 R_{k+3} + \gamma^3 R_{k+4} + ... \\ &= R_{k+1} + \gamma(R_{k+2} + \gamma^1 R_{k+3} + \gamma^2 R_{k+4} + ...) \\ &= R_{k+1} + \gamma G_{k+1} \end{aligned} \tag{3.3}$$

Depending on the state in which the agent is at time-step k ($\boldsymbol{x}_k$) and the policy (π) it follows, the expected return (G_k) can vary. This is known as the state-value function (v_π) and is defined as:

$$\begin{aligned} v_\pi(x_k) &= \mathbb{E}_\pi[G_k|\boldsymbol{X}_k = \boldsymbol{x}_k] \\ &= \mathbb{E}_\pi[\sum_{i=0}^{\infty} \gamma^i R_{k+i+1}|\boldsymbol{X}_k = \boldsymbol{x}_k] \end{aligned} \tag{3.4}$$

Similarly, the action-value function, denoted as $q_\pi(\boldsymbol{x}_k, \boldsymbol{u}_k)$, is the expected return starting from the state ($\boldsymbol{x}_k$), taking the action ($\boldsymbol{u}_k$), and then following policy (π):

$$\begin{aligned} q_\pi(\boldsymbol{x}_k, \boldsymbol{u}_k) &= \mathbb{E}_\pi[G_k | \boldsymbol{X}_k = \boldsymbol{x}_k, \boldsymbol{U}_k = \boldsymbol{u}_k] \\ &= \mathbb{E}_\pi[\sum_{i=0}^{\infty} \gamma^i R_{k+i+1} | \boldsymbol{X}_k = \boldsymbol{x}_k, \boldsymbol{U}_k = \boldsymbol{u}_k] \end{aligned} \tag{3.5}$$

Using the idea of recursive relationships from dynamic programming (DP), state-value function $v_\pi(\boldsymbol{x}_k)$ can be expressed as the sum of its immediate reward (r_k) and the discounted state-value function of next state $v_\pi(\boldsymbol{x}')$. This recursive formulation enables one to efficiently compute the value function for each state by breaking down the problem into smaller sub-problems. By considering the rewards obtained in the current state and the future rewards achieved in the next state, the optimal policy is found by iteratively evaluating the value function in the MDP setting.

$$\begin{aligned} v_\pi(x_k) &= \mathbb{E}_\pi \left[G_k | \boldsymbol{X}_k = \boldsymbol{x}_k \right] \\ &= \mathbb{E}_\pi \left[R_{k+1} + \gamma G_{k+1} | \boldsymbol{X}_k = \boldsymbol{x}_k \right] \\ &= \sum_{\boldsymbol{u}} \pi(\boldsymbol{u}|\boldsymbol{x}) \sum_{\boldsymbol{x}'} \sum_{r} p(\boldsymbol{x}', r | \boldsymbol{x}, \boldsymbol{u}) \left[r + \gamma \mathbb{E}_\pi \left[G_{k+1} | \boldsymbol{x}_{k+1} = \boldsymbol{x}' \right] \right] \\ &= \sum_{\boldsymbol{u}} \pi(\boldsymbol{u}|\boldsymbol{x}) \sum_{\boldsymbol{x}', r} p(\boldsymbol{x}', r | \boldsymbol{x}, \boldsymbol{u}) \left[r + \gamma v_\pi(\boldsymbol{x}') \right] \end{aligned} \tag{3.6}$$

The above equation is known as the Bellman expectation equation. The policy that maximizes the state-value function would be the optimal policy. A policy (π) is said to be optimal if its expected return is greater than or equal to all other policies for all states (3.7). According to Bellman's principle of optimality, "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from first decision"[10].

$$\begin{aligned} v_*(\boldsymbol{x}_k) &= \max_\pi v_\pi(\boldsymbol{x}_k) \\ &= \max_{\boldsymbol{u}} q_{\pi^*}(\boldsymbol{x}_k, \boldsymbol{u}) \end{aligned} \tag{3.7}$$

Bellman's Optimality Equations (3.8) are system of equations, one for each state. If the dynamics of environment are known in form of state-transition probabilities and it follows Markov property then optimal values (v_*) and (q_*) can be found by solving (3.8).

$$\begin{aligned} v_*(\boldsymbol{x}_k) &= \max_{\boldsymbol{u} \in \mathcal{U}} q_{\pi_*}(\boldsymbol{x}_k, \boldsymbol{u}) \\ &= \max_{\boldsymbol{u}} \sum p(\boldsymbol{x}', r | \boldsymbol{x}, \boldsymbol{u}) \left[r + \gamma v_*(\boldsymbol{x}') \right] \\ q_*(\boldsymbol{x}_k, \boldsymbol{u}_k) &= \max_\pi q_\pi(\boldsymbol{x}_k, \boldsymbol{u}_k) \\ &= \mathbb{E} \left[R_{k+1} + \gamma \max_{\boldsymbol{u}_{k+1}} q^*(\boldsymbol{X}_{k+1}, \boldsymbol{U}_{k+1}) \right] \\ &= \sum p(\boldsymbol{x}', r | \boldsymbol{x}, \boldsymbol{u}) \left[r + \gamma \max_{\boldsymbol{u}'} q_*(\boldsymbol{x}', \boldsymbol{u}') \right] \end{aligned} \tag{3.8}$$

Solving Bellman's equation exactly is often infeasible due to its computational complexity. When dealing with large state and action spaces, as encountered in DP, the exhaustive approach becomes exponentially demanding. To handle such extensive scenarios, alternative techniques such as Monte Carlo methods as well as temporal difference (TD) learning, come into play. Unlike traditional dynamic programming, these methods utilize sampling-based approximation approaches. Notably, they work without requiring full knowledge of the environment's dynamics.

3.3 Deep Reinforcement Learning Algorithms

Deep reinforcement learning (DRL) combines reinforcement learning with deep learning techniques that utilize artificial neural network (ANN) to approximate state and action-value functions [11]. In more recent algorithms such as DDPG [2] and twin-delayed deep deterministic policy gradient (TD3) [12], agent's policy is also approximated using ANN. These methods are classified as actor-critic (AC) methods. This enables DRL to handle high-dimensional continuous state and action spaces.

3.3.1 Deep Q-learning

Deep Q-networks (DQN) algorithm uses neural network as function approximator for Q-value functions. Networks are trained using a variant of Q-learning (3.9) and weights are updated using stochastic gradient descent.

$$\hat{q}(\boldsymbol{x}, \boldsymbol{u}) \leftarrow \hat{q}(\boldsymbol{x}, \boldsymbol{u}) + \alpha \left[r + \gamma \max_{\boldsymbol{u}'} \hat{q}(\boldsymbol{x}', \boldsymbol{u}') - \hat{q}(\boldsymbol{x}, \boldsymbol{u}) \right] \tag{3.9}$$

'labeled data' refers to instances where each input is associated with a corresponding correct output or label, enabling the model to learn patterns and make accurate predictions. Traditional machine learning techniques learn from these pre-existing labeled data pairs(input-output pairs), however, reinforcement learning operates differently. RL agent creates its own labeled data via interaction with the environment. These data also need to be independent in order to achieve stable learning. However, the sequences encountered during implementation of RL task are, normally, highly correlated[11]. Another major challenge in order to achieve stable learning is the data distribution, which changes with underlying policy. A good policy leads to information-rich data and iteratively improves the learning and approximation of true Q-values. But the inverse is also true and could lead to oscillations in the behaviors of RL agent[11].

To counter these issues, in [11] few things were added:

- Experience Replay Buffer $\mathcal{D}$: A memory structure used to store and manage a collection of experience tuple $(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}', r, d)$. By storing the experiences in buffer and sampling randomly during gradient step helps break the temporal correlation. This promotes stability during the training process and increases sample efficiency. The

experience tuple consists of state $\boldsymbol{x}$, action $\boldsymbol{u}$, the next state $\boldsymbol{x}'$, the reward r and terminal flag 'done' d.

- Parameterized Q-value: Using an ANN as non-linear function approximator to provide Q-value for state ($\boldsymbol{x}$) for all actions in discrete action space parameterized on the weights θ of the Q-network. In practice, training with a single example at a time can be inefficient and lead to unstable learning. Instead, a mini-batch $\mathcal{B} \subset \mathcal{D}$ is used. Then the loss function $\mathcal{L}_i(\theta_i)$ is computed based on the samples from this mini-batch.
$$\mathcal{L}_\theta = \frac{1}{\mathcal{B}} \sum_{(\boldsymbol{x},\boldsymbol{u},\boldsymbol{x}',d)\in\mathcal{B}} \left[(y_i - q(\boldsymbol{x},\boldsymbol{u};\theta_i))^2\right] \tag{3.10}$$
The y_i is known as a target and is given as $y_i = \mathbb{E}_{\boldsymbol{x}'}\left[r + \gamma \max_{\boldsymbol{u}'} q(\boldsymbol{x}',\boldsymbol{u}';\theta_{i-1})|\boldsymbol{x},\boldsymbol{u}\right]$.
- ϵ-greedy policy: To ensure that the agent explores a large part of the state and action space during training, the ϵ -greedy policy was introduced. This means that instead of using only greedy actions, a random action was drawn from the action space with probability ϵ.

In order to extend the idea of using DRL to continuous action-space, policy-based algorithms were introduced in [13]. These algorithms follow actor-critic structure and learn from a parameterized version of the policy. The AC framework has two ANN models, namely actor-critic. The actor updates the policy parameters, and the critic updates the parameters for Q-value functions. Depending on the nature of the actor, the algorithm can have a stochastic or deterministic policy. A deterministic policy $\boldsymbol{u} = \pi_\phi(\boldsymbol{x})$ is considered a special case of stochastic policy $\boldsymbol{u} \sim \pi_\phi(\boldsymbol{u}|\boldsymbol{x})$. This occurs when the variance of the policy tends to zero [2].

3.3.2 Deep Deterministic Policy Gradient

DDPG is considered an extension of DQN (3.3.1) to the continuous action-space. DQN is limited to discrete action-space whereas DDPG is capable of handling continuous state-space. Similar to DQN, DDPG is model-free algorithm. All the algorithms discussed till now, including DDPG, are off-policy algorithm. This means that policy used to collect the data is not same as the policy that is been improved[14]. Off-policy algorithms allows RL to collect the data from various methods, e.g., using MPC and utilize experience replay $\mathbb{D}$ to learn optimal policy.

Improvement compared to DQN:

- Target Networks: the targets(y) in (3.10) are computed using parameters which are similar to (θ, ϕ) but not same. Instead of using the same ANN model to generate the target values to calculate the loss, another ANN model is used which is called Target Networks. The parameters for target networks are updated with some time delay and are very close to the actual critic network parameters. This lag provides additional stability in training by providing robustness during the gradient descent step. The target networks are also used in DQN, however, only for Q-values.

$$\nabla_\theta \mathcal{L}_q = \nabla_\theta \frac{1}{|\mathcal{B}|} \sum_{(\boldsymbol{x},\boldsymbol{u},\boldsymbol{x}',d)\in\mathcal{B}} (Q_\theta(\boldsymbol{x},\boldsymbol{u}) - y(r,\boldsymbol{x}',d)^2) \tag{3.11}$$

with target given by:

$$y(r,\boldsymbol{x}',d) = r + \gamma(1-d)Q_{\theta_{\text{target}}}(\boldsymbol{x}', \pi_{\phi_{\text{target}}}(\boldsymbol{x}')) \tag{3.12}$$

where $(\phi_{target}, \theta_{target})$ are parameters for target networks. d in (3.12) is set to 1 when $\boldsymbol{x}'$ is terminal state and 0 otherwise. The target parameters are update in low-pass filter fashion as in (3.13)

$$\begin{aligned} \theta_{\text{target}} &\leftarrow (1-\tau)\theta_{\text{target}} + \tau\theta \\ \phi_{\text{target}} &\leftarrow (1-\tau)\phi_{\text{target}} + \tau\phi \end{aligned} \tag{3.13}$$

- Exploration noise: DDPG is deterministic by nature and therefore needs additional excitation superimposed on output of the policy during training particularly in exploration phase. In this thesis, discrete-time Ornstein-Uhlenbeck (OU) noise with zero mean is used.

$$v_k = v_{k-1} + \lambda(\eta - v_{k-1})\frac{\Delta_t}{1\text{s}} + \sigma\sqrt{\frac{\Delta_t}{1\text{s}}}D_n(0,1) \tag{3.14}$$

  Herein, Δ_t defines the sampling time, while λ, σ and η are the stiffness, diffusion, and mean, respectively. $D_n(0,1)$ denotes a realization of the standard Gaussian random distribution.

- Parameterized policy: Unlike DQN, DDPG uses ANN as non-linear function approximator (known as actor)to provide an action given state $\boldsymbol{x}$. The policy π is parameterized by weights ϕ, which are updated in similar manner as the Q-networks by using $\mathcal{B}$. However, instead of taking gradient descent, gradient ascent is calculated for Eq. 3.15.

$$\nabla_\phi \frac{1}{\mathcal{B}} \sum_{x\in\mathcal{B}} Q_\phi(x, \pi_\phi(x)) \tag{3.15}$$

Algorithm 1 DDPG pseudocode [1][2]

Input: initial policy parameters ϕ, Q-function parameters θ, replay buffer $\mathcal{D}$.
Set target parameters equal to main parameters $\phi_{target} \leftarrow \phi$, $\theta_{target} \leftarrow \theta$.
repeat
  observe x_k.
  apply $u_k = \pi_\theta(x_k) + v_k$
  observe r_{k+1}, x_{k+1} and d_{k+1}
  store experience $\langle x_k, u_k, r_{k+1}, d_{k+1} \rangle$ to $\mathcal{D}$
  if x_{k+1} is terminal, reset environment
  if time to update **then**
    sample mini-batch $\mathcal{B} \subset \mathcal{D}$
    update θ by performing one-step gradient descent using (3.11)
    update ϕ by performing one-step gradient ascent using (3.15)
    update weights of target networks based on (3.13)
  end if
  k ← k + 1
until convergence condition is met

3.4 Task Specific Implementation of Reinforcement Learning Controller

As mentioned in section (3.3.2), DDPG is a model-free control algorithm. However, to improve and aid learning, feature engineering and prior knowledge-based reward design are incorporated. The control algorithm is implemented using the rotating dq0 reference frame. Assuming balanced sinusoidal grid voltages and current, the input vector to the agent becomes constant. Furthermore, the reference signal is also constant.

The output consists of measurement signals ($\boldsymbol{v}_{\mathrm{dq0}}$, $\boldsymbol{i}_{\mathrm{dq0}}$) and a constant reference signal $\boldsymbol{v}^*_{\mathrm{dq0}}$ representing the nominal operation of the grid with respect to the voltage amplitude and frequency.

Additional features include:

- the error $\rightarrow \Delta\boldsymbol{v}_{\mathrm{dq0}} = \frac{1}{2}\left(\boldsymbol{v}^*_{\mathrm{dq0}} - \boldsymbol{v}_{\mathrm{dq0}}\right)$.
- the past action $\rightarrow \boldsymbol{u}_{k-1} = \boldsymbol{m}_{\mathrm{dq0},k-1}$.
- the ξ past voltage measurements $\rightarrow \boldsymbol{v}_{\mathrm{dq0},\{k-\xi,..,k-1\}}$.

The latter provides additional history information to the agent. These features are additionally normalized to a range of $[-1, +1]$ through division by the corresponding limit value. Action signals are interpreted as modulation indices for the inverter, that is, $\boldsymbol{u}_k \in [-1, 1]^3$.

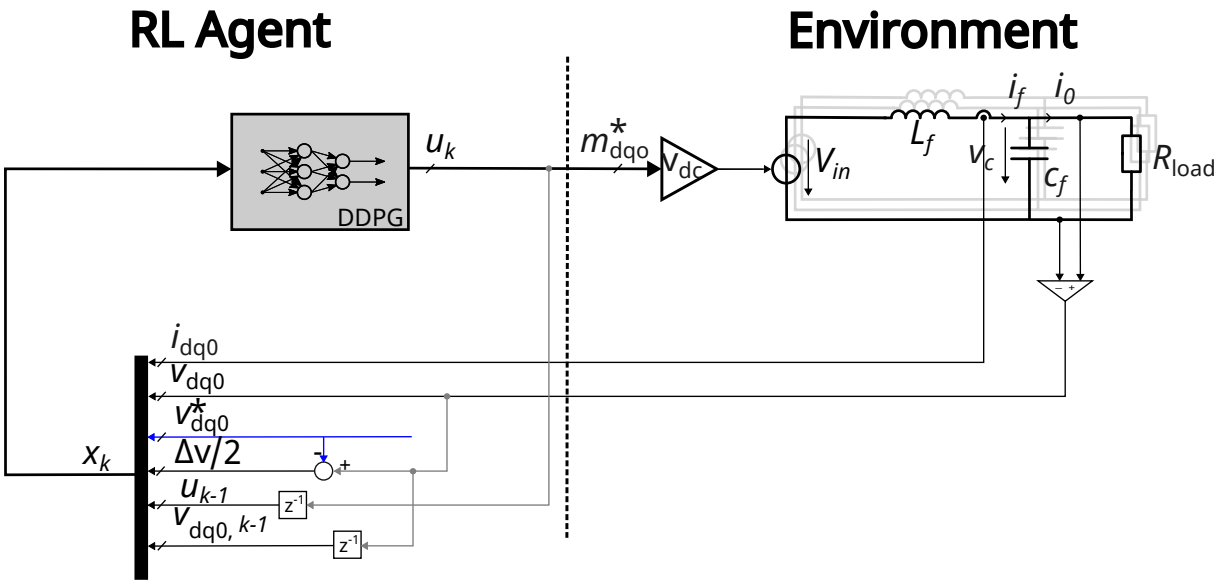


Fig. 3.2: Simplified control diagram integrating DDPG with 3-phase Voltage Source Inverter

$$\boldsymbol{x}_k = \left[\frac{\boldsymbol{i}^{\mathrm{T}}_{\mathrm{dq0},k}}{i_{\mathrm{lim}}}, \frac{\boldsymbol{v}^{\mathrm{T}}_{\mathrm{dq0},k}}{v_{\mathrm{lim}}}, \frac{(\boldsymbol{v}^*_{\mathrm{dq0}})^{\mathrm{T}}}{v_{\mathrm{lim}}}, \frac{\Delta \boldsymbol{v}^{\mathrm{T}}_{\mathrm{dq0}}}{v_{\mathrm{lim}}}, \frac{\boldsymbol{u}^{\mathrm{T}}_{\mathrm{dq0},k-1}}{\frac{v_{\mathrm{DC}}}{2}}, \frac{\boldsymbol{v}^{\mathrm{T}}_{\mathrm{dq0},\{k-\xi,..,k-1\}}}{v_{\mathrm{lim}}} \right]^{\mathrm{T}} \tag{3.16}$$

The reward is computed using a piecewise continuous function that is evaluated differently based on voltage and current measurements. For value of $\gamma = 0.94$, the maximum achievable reward $r_{\max}$ is 0.06. Since the control agent seeks to maximize the reward through interaction, the reward needs to be designed cautiously. Eq. (3.17) shows how the reward for this work is formulated. In case of system state and/or input violations, the reward is set to the minimum value possible $r = -1$. Once the instantaneous values of currents and voltages are within their respective limit, mean-root-error (MRE) between normalized measured voltages and the normalized reference values is used to calculate weighted reward. Compared to the classical mean-squared-error(MSE)metric, MRE is better at penalizing smaller deviations around zero much more strongly which helps reducing steady-state control errors[6].

$$r = \begin{cases} \frac{1-\gamma}{3} \sum_{p\in\{d,q,0\}} \left(1 - \sqrt{\frac{|v^*_p - v_p|}{2v_{\max}}}\right), & v_{\mathrm{abc}} \le v_{\mathrm{lim}} \wedge i_{\mathrm{abc}} \le i_{\mathrm{nom}}. \\ (1-\gamma)\left(\frac{1}{3}\sum_{p\in\{d,q,0\}}\left(1 - \sqrt{\frac{|v^*_p - v_p|}{2v_{\max}}}\right) - \frac{1}{4}\sum_{p\in\{a,b,c\}}\left(\frac{i_p - i_{\mathrm{nom}}}{i_{\mathrm{lim}} - i_{\mathrm{nom}}}\right)\right), & v_{\mathrm{abc}} \le v_{\mathrm{lim}} \wedge i_{\mathrm{nom}} \le i_{\mathrm{abc}} \le i_{\mathrm{lim}}. \\ -1, & \text{otherwise.} \end{cases} \tag{3.17}$$

4 Model Predictive Control

Model predictive control is an advanced control strategy that effectively deals with complex systems that have constraints, and uncertainties. MPC is based on the receding-horizon principle. The basic idea of MPC is to predict the future behavior of the controlled system over a finite time horizon and compute an optimal control input that, while ensuring satisfaction of the given state and input constraints, minimizes the cost function. In this work, a linear model predictive control (MPC) approach is implemented to steer the agent towards various areas within the state space. This is achieved by designing a cost function that penalizes deviations from a reference trajectory. Fig. 4.1 depicts the integration of MPC with the system in a closed-loop configuration. Here, $\boldsymbol{x}^*$ is the reference state.

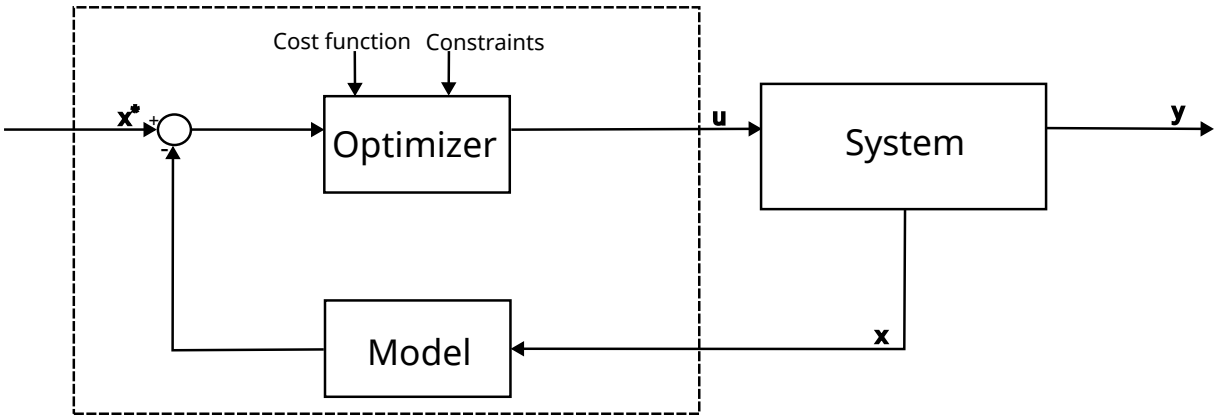


Fig. 4.1: State feedback model predictive controller.

An infinite horizon sub-optimal controller can be designed by repeatedly solving finite time optimal control problems in a receding horizon fashion [15]. At each time-step, starting with the current state, a finite-horizon open-loop optimal control problem is solved (Fig. 4.2). The optimal manipulated input signal that is computed is applied to the process only during the next sampling interval $[k, k+1]$. At time-step $k+1$, a new optimal control problem is solved based on new measurements of the state over a shifted horizon. This

type of receding horizon controller, where the finite-time optimal control law is computed by solving an optimization problem on-line, is known as model predictive control.

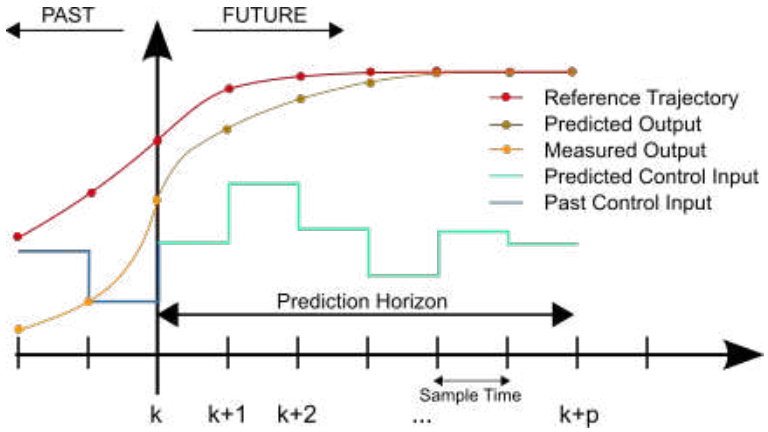


Fig. 4.2: Receding horizon idea (Derivative of the work from Martin Behrendt licensed under CC BY-SA 3.0)

A discrete-time state-space for a linear time invariant (LTI) system can be defined as:

$$\boldsymbol{x}[k+1] = \boldsymbol{A}_{\mathrm{d}}\boldsymbol{x}[k] + \boldsymbol{B}_{\mathrm{d}}\boldsymbol{u}[k+1] \tag{4.1}$$

where states $\boldsymbol{x}_k \in \mathbb{R}^n$, control input $\boldsymbol{u}_k \in \mathbb{R}^m$, and matrices $\boldsymbol{A}_{\mathrm{d}} \in \mathbb{R}^{nxn}$, $\boldsymbol{B}_{\mathrm{d}} \in \mathbb{R}^{nxm}$. The continuous state-space model of three-phase voltage source inverter described in (2.5) can be represented in discrete-time using (4.2). This is essential for problem formulation for MPC.

$$\boldsymbol{A}_{\mathrm{d}} = e^{\boldsymbol{A}T_s}, \quad \boldsymbol{B}_{\mathrm{d}} = \int_0^{T_S} e^{\boldsymbol{A}\tau}\boldsymbol{B}\mathrm{d}\tau \tag{4.2}$$

The constrained finite-time optimal control problem CFTOC can be given as:

$$J^*_{0\to k_f}(x_0) = \min_{u_0,u_1,..,u_{k_f}} \left\{ \mathcal{T}(\boldsymbol{x}(k_f)) + \sum_{k=0}^{k_f-1} \mathcal{V}(\boldsymbol{x}[k], \boldsymbol{u}[k]) \right\} \tag{4.3a}$$

$$\text{subj. to} \quad \boldsymbol{x}[k+1] = f(\boldsymbol{x}_k, \boldsymbol{u}_k), \quad \boldsymbol{x}[0] = \boldsymbol{x}_0, \tag{4.3b}$$

$$h(\boldsymbol{x}_k, \boldsymbol{u}_k) \leq 0, \quad \forall k \in [0, k_f - 1]. \tag{4.3c}$$

where $J^*(x_0)$ represent the optimal cost function associated with trajectory starting from $\boldsymbol{x}_0$ and reaching the terminal state $\boldsymbol{x}_{k_f}$ by executing optimal input sequence $\boldsymbol{u}^*$. $\mathcal{T}(.)$ is the terminal cost and $\mathcal{V}(.)$ is the running cost. These evaluate the deviation of closed-loop system behaviour from target behaviour (in this work, reference tracking). The system

dynamics are defined by $f(\boldsymbol{x}_k, \boldsymbol{u}_k)$ and the constraints are represented by $h(\boldsymbol{x}_k, \boldsymbol{u}_k)$[15]. Like many real-world control tasks, control of three-phase voltage source inverter is also a infinite horizon task. These problems can be challenging to solve directly due to their infinite nature. Hence, we use receding horizon control(RHC).

By using Bellman's optimality principle:

$$J^*(x_k) = \min_{\boldsymbol{u}[k]} \Big\{ \mathcal{V}(\boldsymbol{x}[k], \boldsymbol{u}[k]) + J^*(\boldsymbol{x}[k+1]) \Big\} \tag{4.4}$$

the infinite control task is transformed into a sequence of finite horizon optimization problem. $J^*(x_k)$ represents the optimal cost-to-go from time step k onward, given initial state x_k. This values stand for minimum expected cumulative cost (or maximum expected cumulative reward in RL setting) that can be achieved by following the optimal control strategy from time step k. $\mathcal{V}(\boldsymbol{x}[k], \boldsymbol{u}[k])$ represents the immediate cost with the current state and the control input. This term captures the immediate impact of the control input on the system. $J^*(\boldsymbol{x}[k+1])$ is the optimal cost-to-go from next time step $k+1$, given the state $\boldsymbol{x}[k+1]$ resulting from applying the control input $\boldsymbol{u}[k]$ at time step k. According to Bellman's principle, if $\boldsymbol{u}^*[k] = \boldsymbol{u}[k]$ is applied at time step k and $J^*(\boldsymbol{x}[k+1])$ is the optimal cost-to-go at time step $k+1$ then we have an optimal input trajectory. Thus, rather than optimizing over an infinite time horizon, RHC optimizes within a finite window of future time steps(N) and estimates the optimal cost-to-go $J^*(\boldsymbol{x}[k+N])$ from time step $k+N$. The extended version of (4.4) over finite prediction horizon N:

$$J^*(\boldsymbol{x}[k]) = \min_{\boldsymbol{U}_k} \quad J(\boldsymbol{x}[k], \boldsymbol{U}_k) = \min_{u_k, u_{k+1}, .. u_{k+N}} \left\{ \sum_{i=k}^{k+N-1} \mathcal{V}(\boldsymbol{x}[i], \boldsymbol{u}[i]) + \mathcal{T}(\boldsymbol{x}[k+N]) \right\} \tag{4.5}$$

In this work, a linear time-invariant (LTI) system is controlled using MPC with quadratic cost as shown in Eq. (4.6).

$$J(\boldsymbol{x}[k], \Delta\boldsymbol{u}[k]) = (\boldsymbol{x}^*[k+N] - \boldsymbol{x}[k+N])^T \boldsymbol{S} (\boldsymbol{x}^*[k+N] - \boldsymbol{x}[k+N]) \tag{4.6a}$$

$$+ \sum_{i=k}^{k+N-1} (\boldsymbol{x}^*[i] - \boldsymbol{x}[i])^T \boldsymbol{Q} (\boldsymbol{x}^*[i] - \boldsymbol{x}[i]) + \Delta\boldsymbol{u}[i]^{\mathrm{T}} \boldsymbol{R} \Delta\boldsymbol{u}[i]. \tag{4.6b}$$

Above $\boldsymbol{Q} \in \mathbb{R}^{nxn}$ and $\boldsymbol{S} \in \mathbb{R}^{mxm}$ are symmetric, positive semi-definite weighting matrices and $\boldsymbol{R} \in \mathbb{R}^{mxm}$ is a symmetric, positive definite weighting matrix[16]. The input and state constraints are defined by:

$$\boldsymbol{W}_x \boldsymbol{x}[k] \leq \boldsymbol{w}_x, \quad \boldsymbol{W}_{x_f} \boldsymbol{x}[k_f] \leq \boldsymbol{w}_{x_{k_f}}, \quad \boldsymbol{W}_u \boldsymbol{u}[k] \leq \boldsymbol{w}_u. \tag{4.7}$$

The problem defined in (4.6)(4.7) can be reformulated as quadratic programming (QP) and solved using tools such as CasADi[17] or IPOPT[18]. In this work, the QP is solved

in python using do-mpc library[19]. A control horizon $N = 3$ is chosen to save the computational time. Fig. 4.3 shows the performance of the implemented MPC controller in OMG. The reference is changed every few timesteps as the goal of using MPC is to generate target-exploration trajectories as will be explained in section 3. At every change in the reference value, the currents $\tilde{\boldsymbol{i}}_{\mathrm{abc}}$ have some ripple. This is the transient response of the system to change in the reference voltage $\boldsymbol{v}^*_{\mathrm{dq0}}$ and the phenomenon is known as current overshoot. When the $\boldsymbol{v}^*_{\mathrm{dq0}}$ is changed, the modulation indices $\boldsymbol{m}_{\mathrm{dq0}}$ generated by MPC also change to match the reference, leading to different voltage and current waveforms of inverter. The primary goal of LC-filter is to reduce the harmonics and provide a clean output waveform. As the MPC tries to track the reference as quickly as possible, there is an abrupt change in the the current through inductor $\boldsymbol{i}_{\mathrm{L}}$ and it resists these changes due to its inherent property of storing energy leading to overshoot.

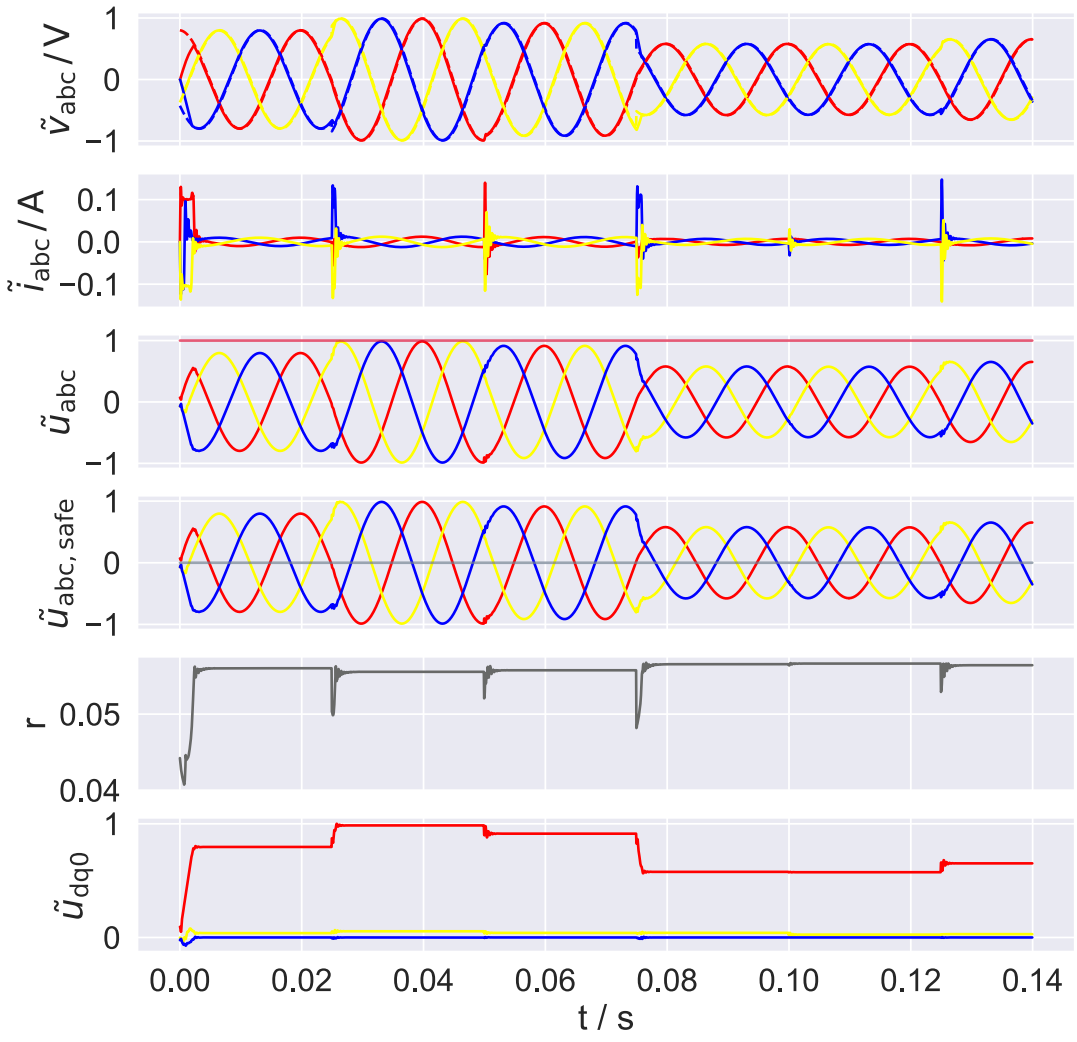


Fig. 4.3: Model predictive controller combined with HCXA to perform reference tracking in OMG

5 Histogram-based Count Maximization Algorithm

A reinforcement learning agent learns via interacting with the environment in a closed loop. The rate at which the agent is able to learn an optimal policy depends on the quality of interaction, specifically the samples that are collected during the training phase. Depending on environment's dynamics and constraints, a random exploration could lead to dense sample accumulation in few regions of state space, e.g., in the proximity of attractive equilibria. Due to the nature of data-driven controllers, the policy derived using these samples would be biased and hence could lead to sub-optimal performance.

A naive solution is to extend the exploration phase in order to increase the chances of visiting entire state space. However, this leads to data-inefficiency and increases training time. Furthermore, as it provides no framework to ensure uniform state space coverage, the RL agent is more likely to be biased after the training. Another very intuitive approach for discrete state-space is count-based algorithm [20], where a table of counters is maintained to assess the states visited least amount of times. In [21], initial states are randomly selected to gather information in different sub-spaces. This is referred as exploring starts (ES). A relatively similar approach was extended to continuous state-space in [22]. Using kernel density estimator (KDE), an informative initial state was chosen. This meant prioritizing infrequently visited states for a more desirable and balanced coverage of state space during training. This method is referred as density estimation-based state space coverage acceleration DESSCA [22]. Although DESSCA is a good way to improve exploration, it has its limitation. In many real-world control scenarios, it may not be feasible or even possible to start from all possible states. Additionally, computing a probability density function using KDE for each sample in state space is computationally inefficient. This means that DESSCA would get slower and slower as more data points are collected by interacting with the environment. As the name suggest, ES can only be used at the beginning of the episode and is a static exploration method.

Hence, an algorithm that supports online exploration and is computationally efficient is needed. This means that the developed algorithm must assist the RL agent in exploration not just at the beginning, but consistently throughout the entire episode. One way to achieve this is to utilize the states gathered during interactions with the environment to

determine the next optimal state to visit such that a balanced distribution of visited states is maintained.

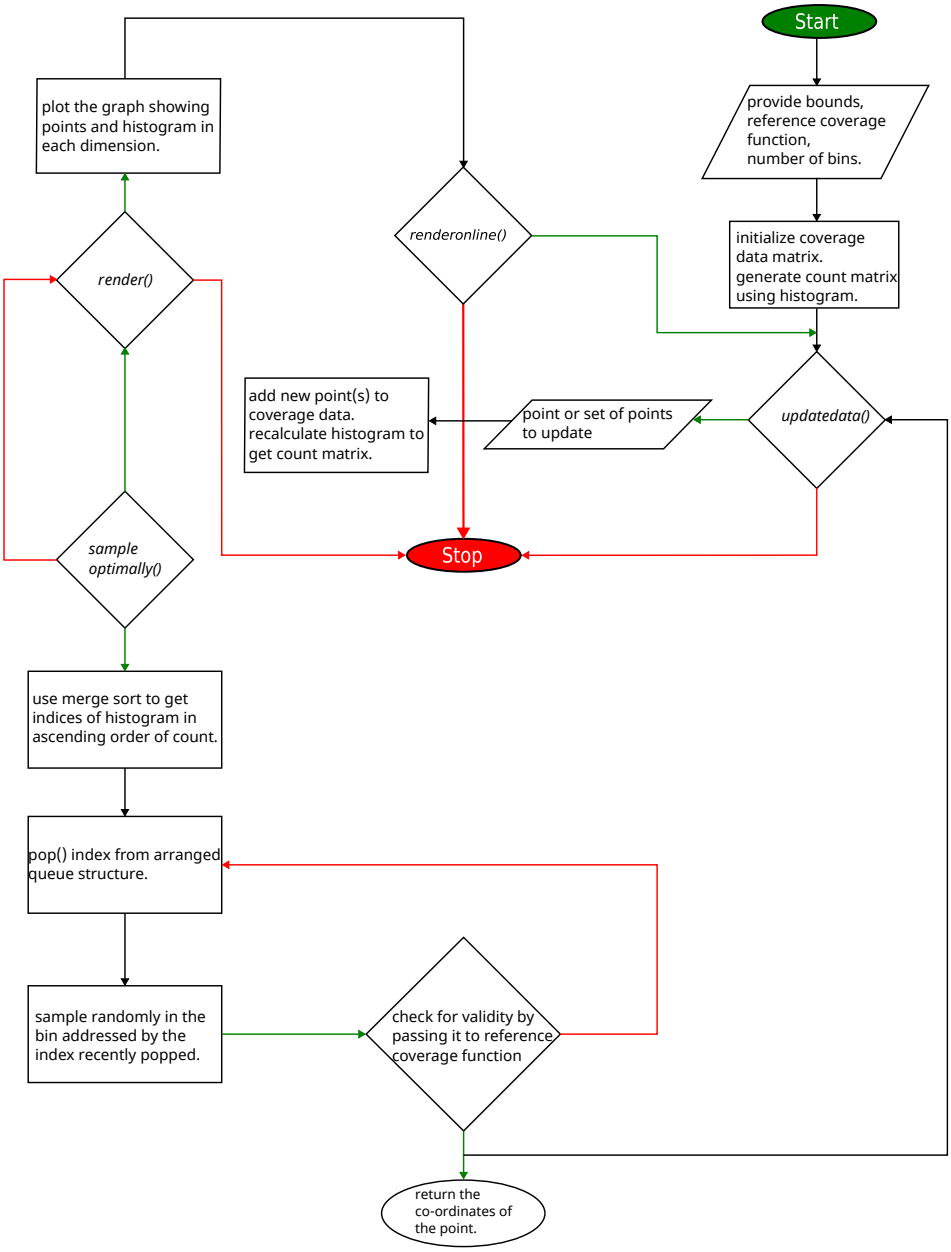


Fig. 5.1: Flow Chart of histogram-based count maximization algorithm. The red and green line from decision block represents *True* and *False* respectively

Histogram-based count maximization algorithm (HCXA)is a online exploration tool that allows for choosing new reference state while taking previously visited states into account. Instead of using KDE to estimate the current probability density function over the continuous state space, a rather simple solution is proposed. HCXA discretizes the continuous state space and uses histograms as N-dimensional counter tables shown in Fig. 5.2. Histograms offer a visual depiction of data distribution, inherently encoding information about data counts within their graphical representation.

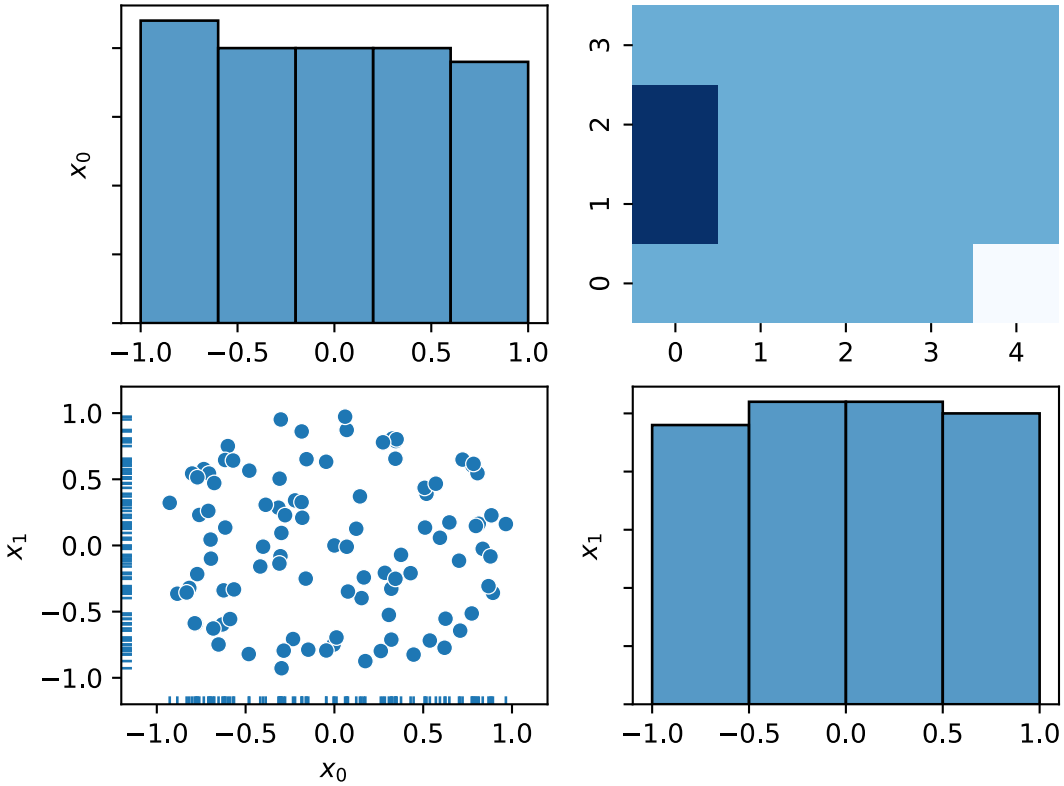


Fig. 5.2: An example case of HCXA discretizing a 2D state space using 5 bins and 4 bins in x_0, x_1 axis respectively. The subplots on diagonal axis are 1D representation of state-space along its respective axis. On the bottom left corner is the scatter plot of points sampled in the state space and on top right corner is discretized state space shown using 2D histogram.

As the continuous state space is approximated by its discrete counter part, the accuracy of this representation depends on resolution which in turn depends on the number of bins used in histogram to depict each dimension. Higher number of bins results in better resolution and finer approximation of the continuous space. In Fig. 5.2, (5, 4) bins are used to represent x_0 and x_1 respectively. By utilizing the *Merge Sort* algorithm, HCXA effectively explores the state-space, prioritizing regions that are underrepresented in the

current sample set. This approach ensures a more balanced and accurate representation of the reference coverage function. The sorted list is then traversed in a first-in-first-out (FIFO) fashion and a validity check is performed. An index is considered valid when a point $p = (x_{0,p}, x_{1,p})$ sampled randomly from the corresponding bin and is within the reference coverage space given by $c^*(p)$. A standard form for equation of circle is used as the reference coverage function in Fig. 5.2.

$$\text{Validity check} = \begin{cases} \text{True}, & x_0^2 + x_1^2 \leq 1 \\ \text{False}, & \text{otherwise} \end{cases} \tag{5.1}$$

If p is within the valid space, it is returned to the user(or agent); otherwise, next index in the list is tried. HCXA keeps traversing the list until a valid point is found. Generally, the real-world control task are not constrained to 2 dimensions and the need for high resolution in each dimension may or may not be important.

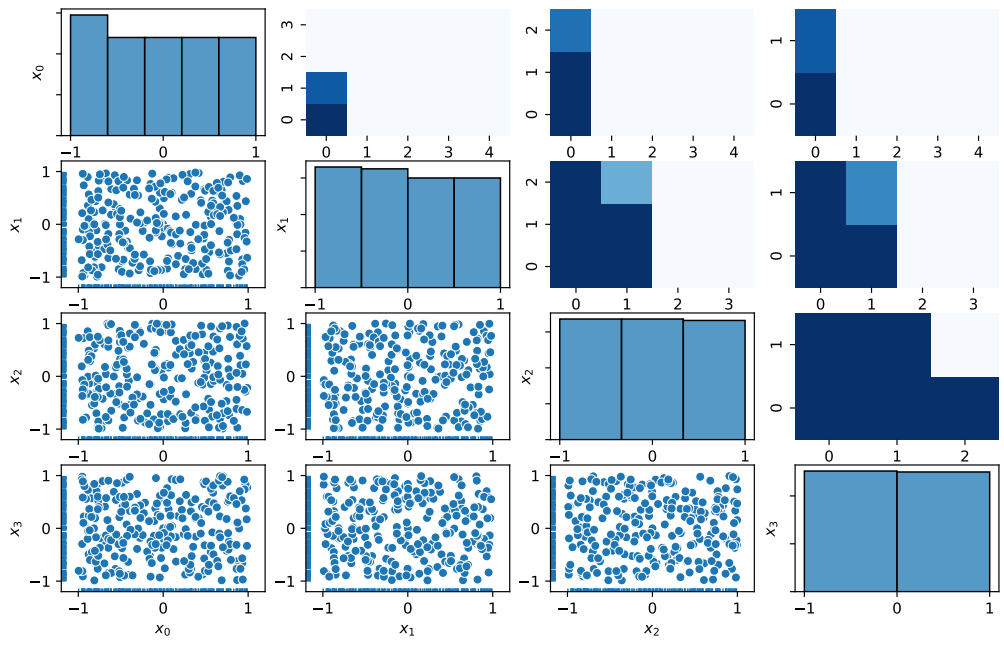


Fig. 5.3: An exemplary case where state space is 4D. Similar to 2D case, bottom triangle of 4x4 matrix is a scatter plot shown in 2D and upper triangle is its 2D histogram representation. The diagonal subplots represent the count when seen in 1D

Fig. 5.3 shows how HCXA achieves uniform distribution in a 4 dimensional normalized state space. Bin sizes of $(5, 4, 3, 2)$ are used along (x_0, x_1, x_2, x_3) respectively. A higher

resolution is captured along x_0 dimension by selecting higher number of bins as opposed to x_3 where only 2 bins are used to discretize the state space.

Using histogram as a counter table makes HCXA a highly computationally efficient algorithm. The merge sort algorithm is $\mathcal{O}(n \log n)$ in worst case, where n is the length of the list to be sorted. Therefore, there is trade-off between the number of bins used vs. computational time needed. Nevertheless, HCXA takes significantly less time to compute compared to other algorithms such DESSCA [1].

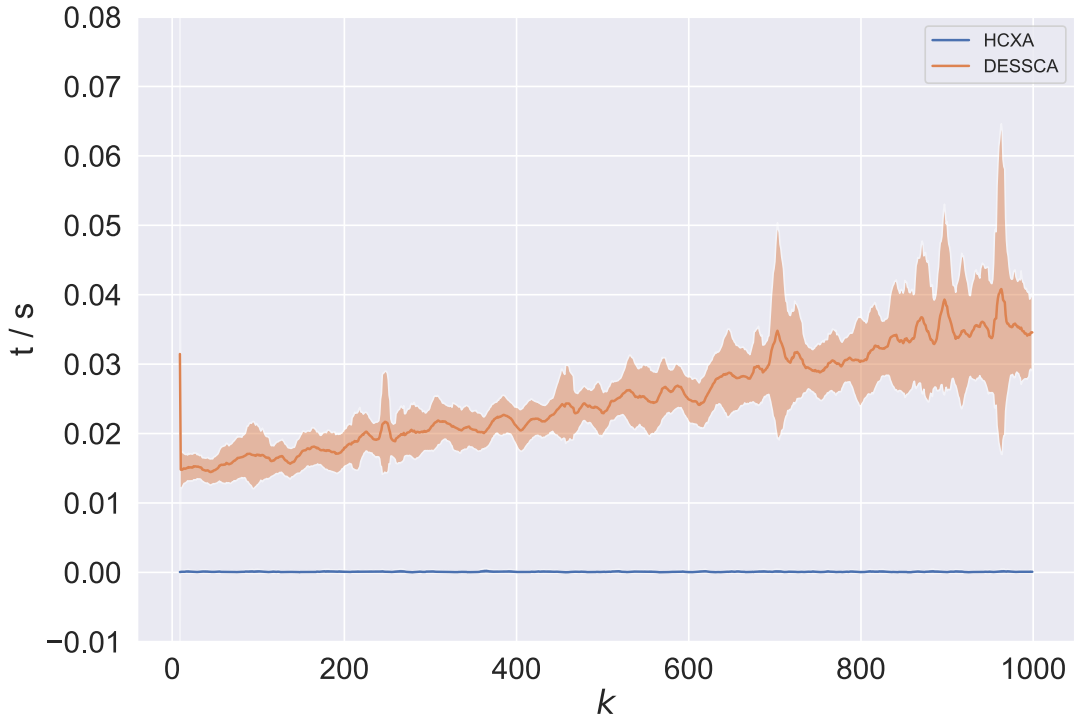


Fig. 5.4: Computational time to receive next optimal point in state space as a function of size of coverage data over 10 iterations.

Figure 5.4 demonstrates that the time it takes to receive a point from DESSCA increases in proportion to size of already accumulated data points. In contrast, HCXA has a constant time which depends on the resolution selected and not on the number of data points covered previously. The mean time taken by HCXA is in the range of 3ms which is significantly less than DESSCA's gradually increasing mean time.

Apart from the computational time, the uniformity of the sampled points is another important factor defining the performance of an exploration algorithm. After all, the goal is to

[1]The points are generated by the code provide by the author https://github.com/max-schenke/DESSCA.git.

get a well-balanced distribution in the state space. To measure this, discrepancy(D) is used. It is a metric that quantifies the uniformity of data points in a specified space[23].

$$D_{\mathrm{N}}(P) = \sup_{B \in J} \left| \frac{A(B;P)}{N} - \lambda_s(B) \right| \tag{5.2}$$

The discrepancy of a set of points P is given by Eq. (5.2) where N is the number of points in set P[24]. $\lambda_s(B)$ is s-Lebesque measure and $A(B;P)$ is the ratio number of points in B to number of points in P. J is the set of s-dimensional boxes of the form $\prod_{i=1}^{s}[a_i, b_i) = \{\mathbf{x} \in \mathbb{R}^s : a_i \leq x_i < b_i\}$ where $0 \leq a_i < b_i \leq 1$[24]. In this work, L2-star discrepancy from scipy is used[25]. A lower discrepancy value corresponds to a more uniform distribution of points.

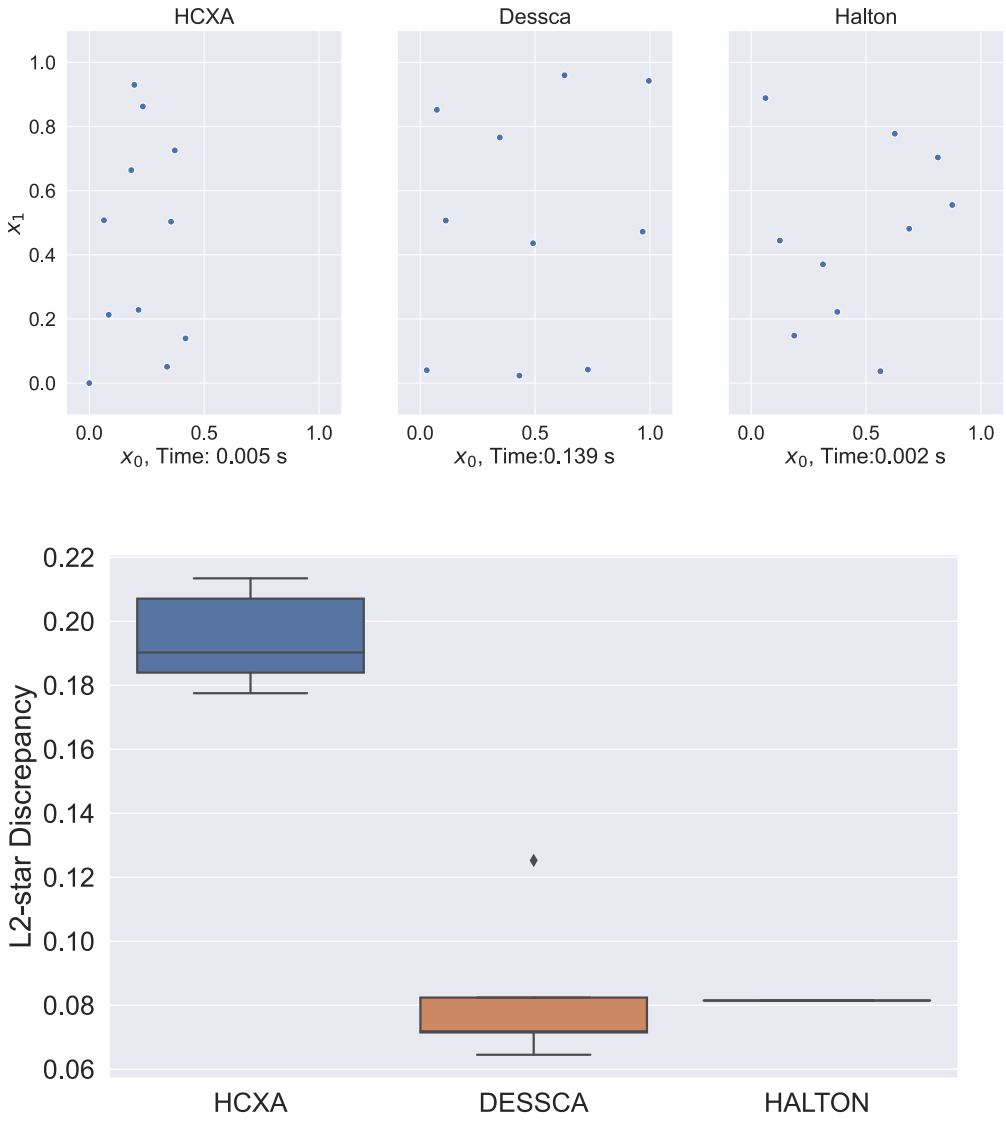


Fig. 5.5: 10 points sampled from 3 different algorithms. The top group of scatter plots shows the co-ordinates of points in 2D space along with total computation time. The box plot at the bottom shows the uniformity measure of 5 trials using HCXA, DESSCA and Halton sequence

To provide a better inference of uniformity, the sampled points are juxtaposed with Halton sequence[23]. Halton sequence are an example of a quasi-random number sequence and have low-discrepancy property making it an ideal comparison.These are constructed according to a deterministic method that uses coprime numbers as its bases[26].

Fig. 5.5 shows the limitation of HCXA. When the size of sampled points is very small, HCXA is unable to provide a uniform distribution. This is due to the merge sort algorithm

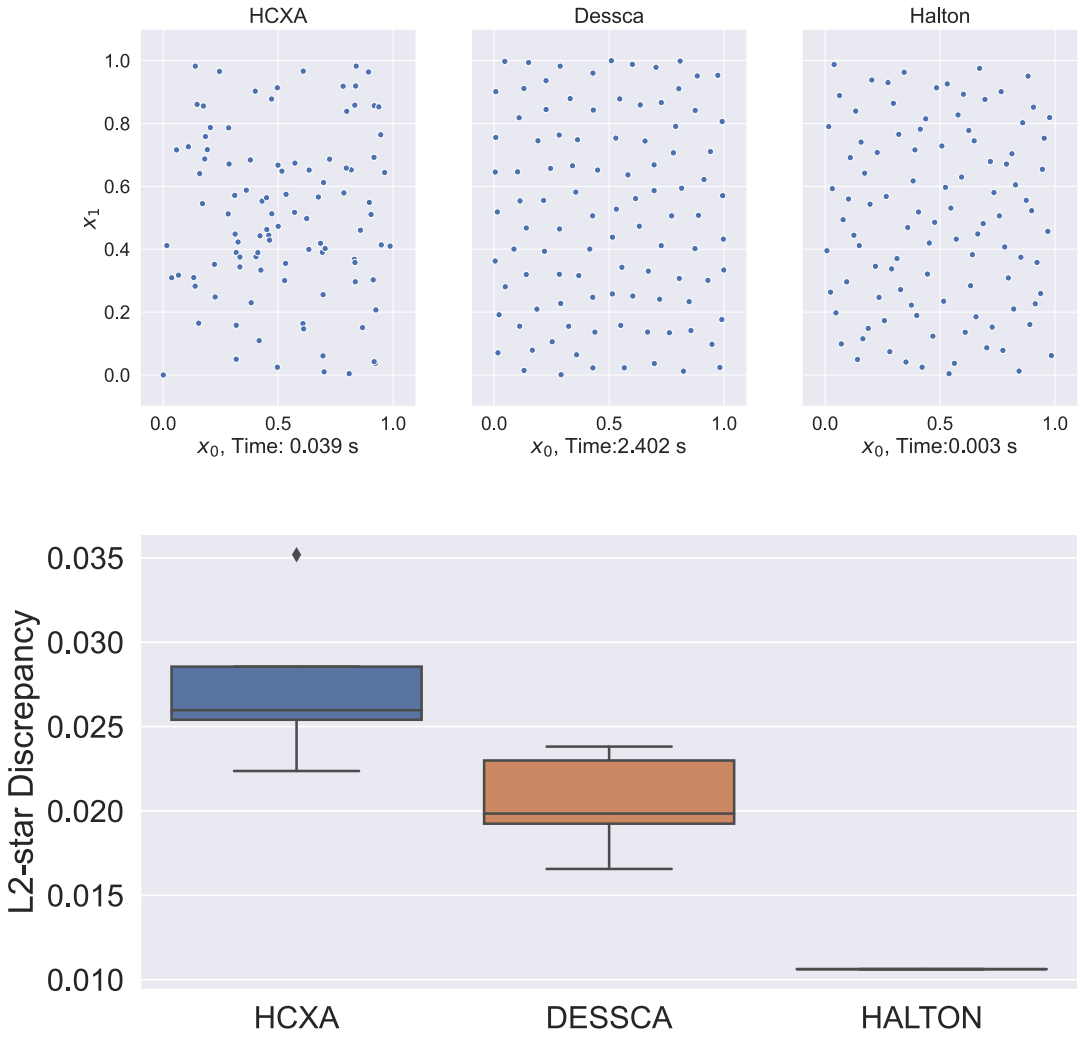


Fig. 5.6: 100 points sampled using HCXA, DESSCA and Halton sequence

which aranges the indicies according to the count in each bin and does not consider the state space as a whole. This improves the computation time but provides biased distribution initially. However, this is not a very big limitation as for real-world control application, a large amount of data is collected. Similarly, Fig. 5.6 and Fig. 5.7 shows the how discrepancy decreases with increase in sample size. The box plots represent the mean and variance of discrepancy along these 3 different methods for 5 trails. Fig. 5.7 shows that Halton sequence is faster and provided a more uniform distribution than DESSCA and HCXA. However, due to the deterministic nature of the sequence, it is a static method of exploration. These points are generated independent of points already visited and without taking the actual state space distribution into account. Halton sequences can used in a manner similar to the utilization of ES. In contrast to the conventional practice of randomly selecting an initial state, Halton sequences offer a systematic approach by

using points from the sequence. This introduces an element of determinism in the initial state selection process, potentially influencing the exploration process in a more controlled manner. However, the goal of this work is dynamic exploration.

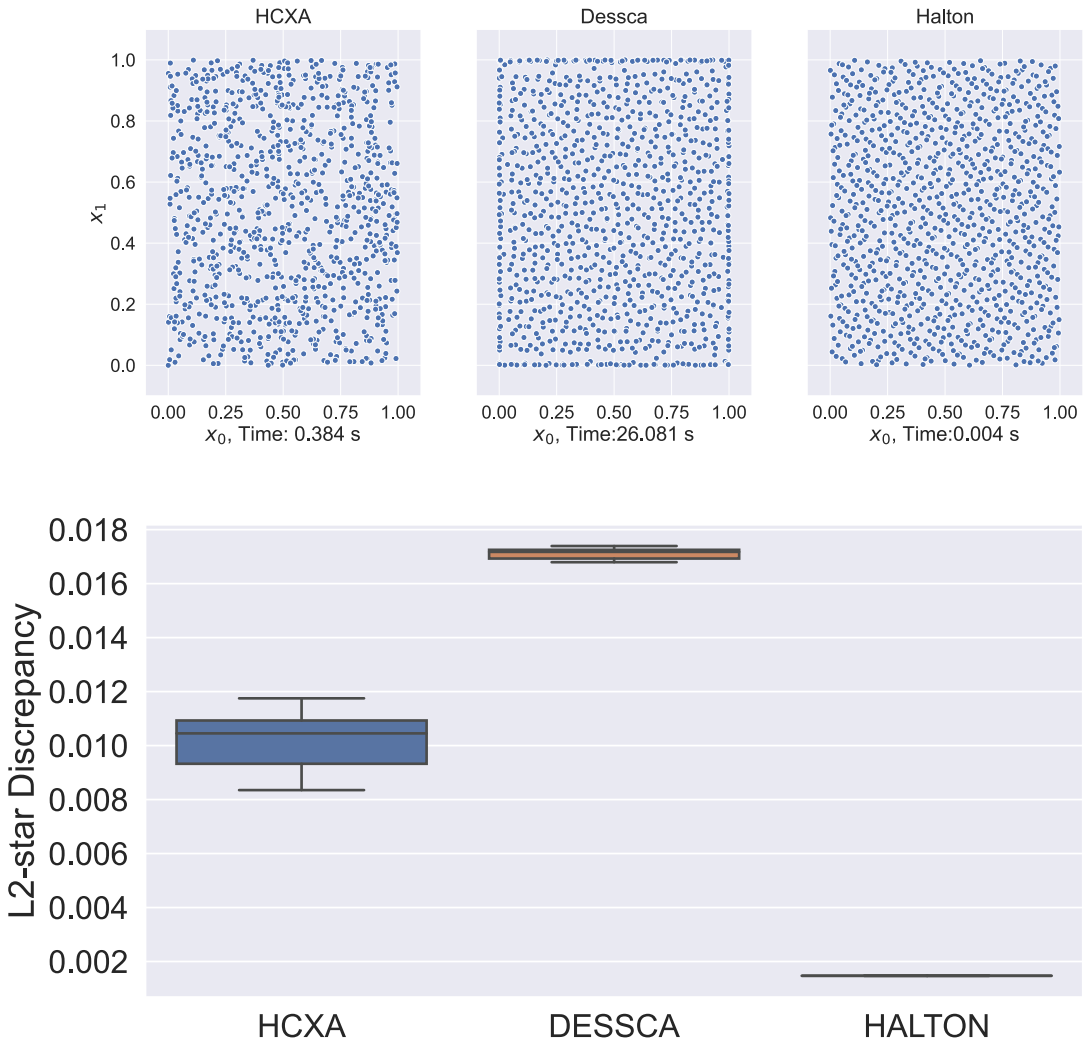


Fig. 5.7: 1000 points sampled using HCXA, DESSCA and Halton sequence

As mentioned earlier, the resolution of the discretize space determines how fine or coarse is the approximation compared to continuous state space. To illustrate this, a comparison between different sizes of bins is made. Based on the L2-star discrepancy in Fig.5.8 - 5.10, a clear trend in the granularity vs. number of bins is seen. In Fig. 5.8, $HCXA_2$[2] has lower discrepancy as it able to cover the state space much faster compared to $HCXA_{10}$ and

[2]The subscript at the bottom represents the number of bins used to discretized the continuous state space, e.g., $HCXA_2$ means 2 bins.

$HCXA_{20}$. However, as the size of data set increases, the discrepancy of $HCXA_{20}$ decreases as seen in Fig. 5.10.

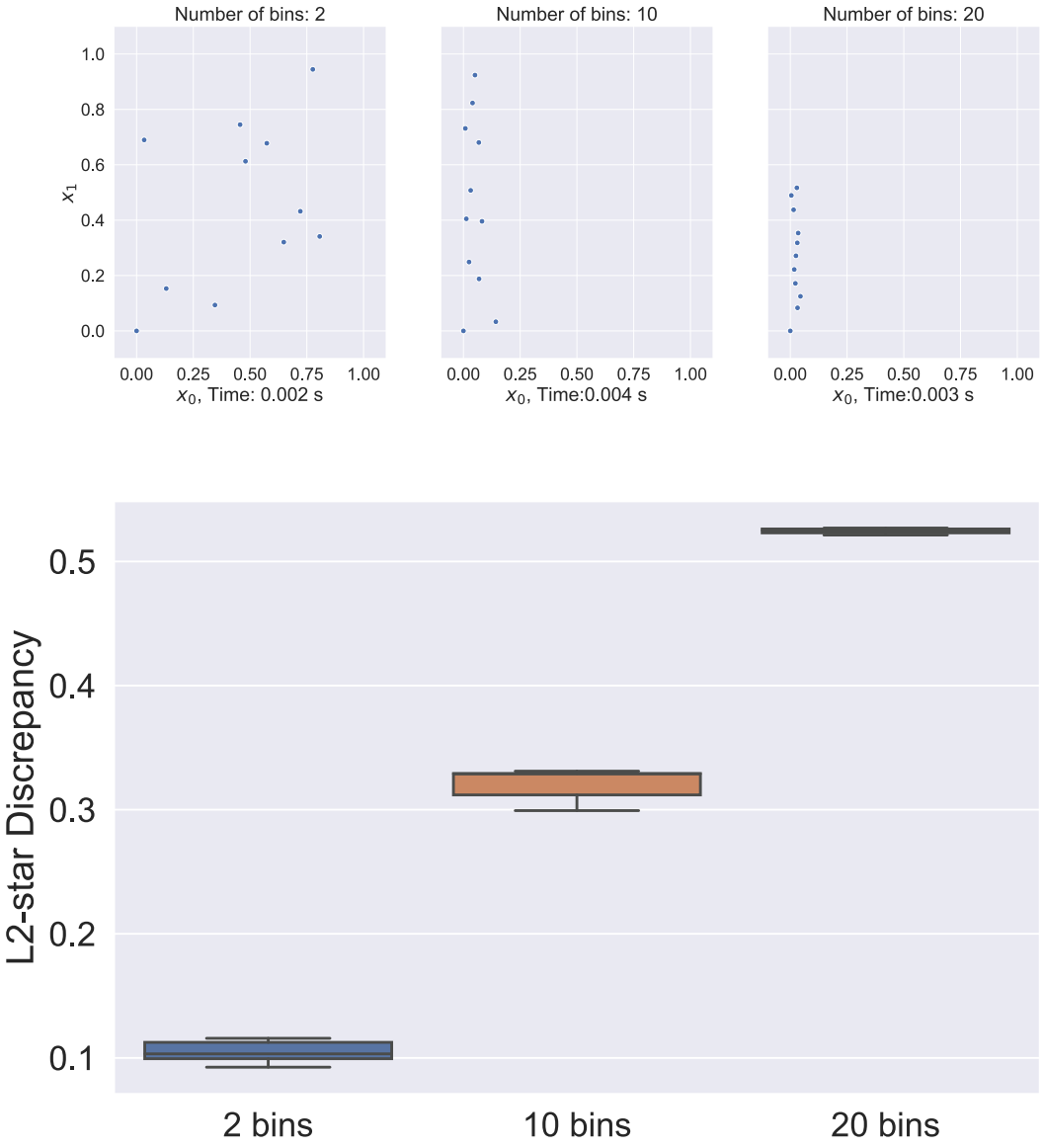


Fig. 5.8: 10 points sampled using HCXA in 2D state space with varying number of bins

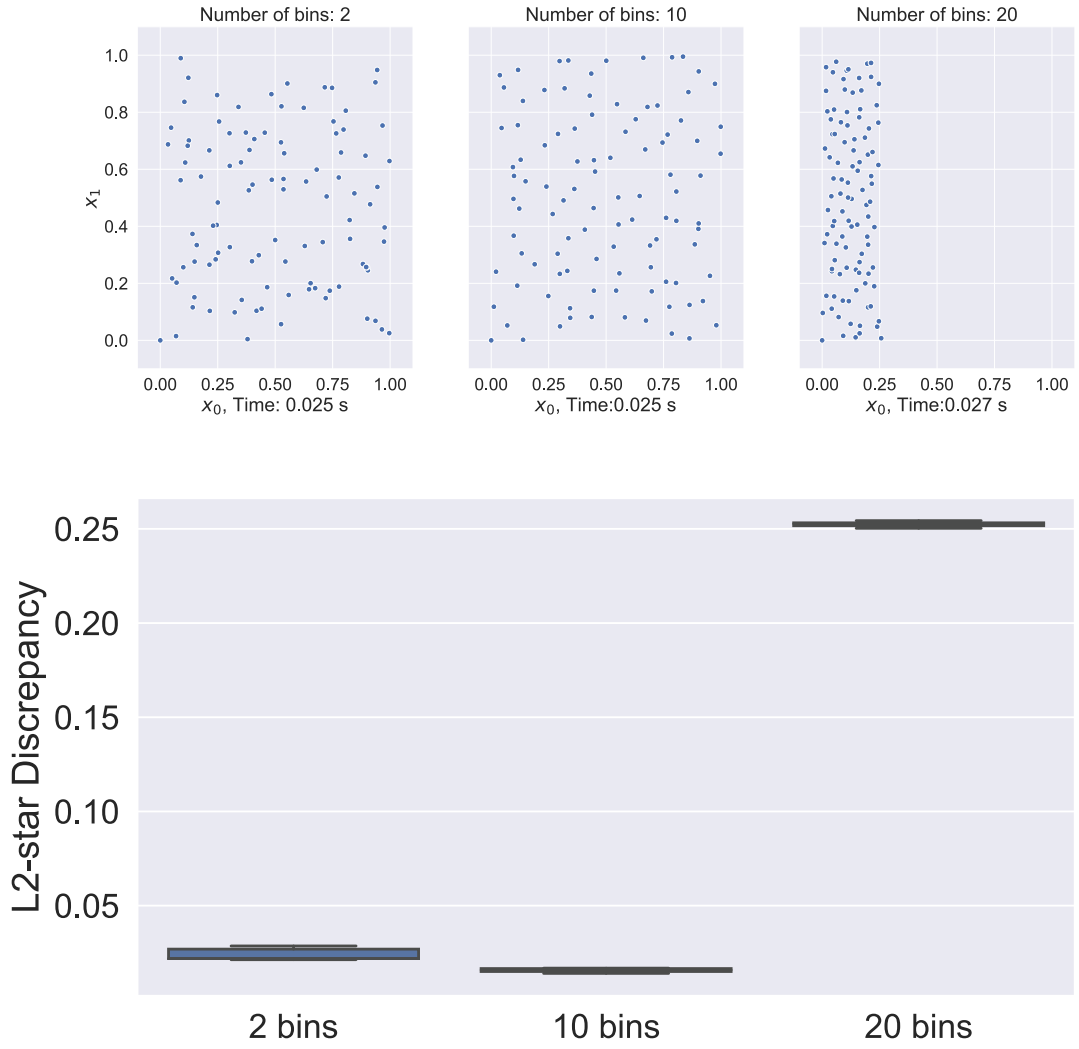


Fig. 5.9: 100 points sampled using HCXA in 2D state space with varying number of bins

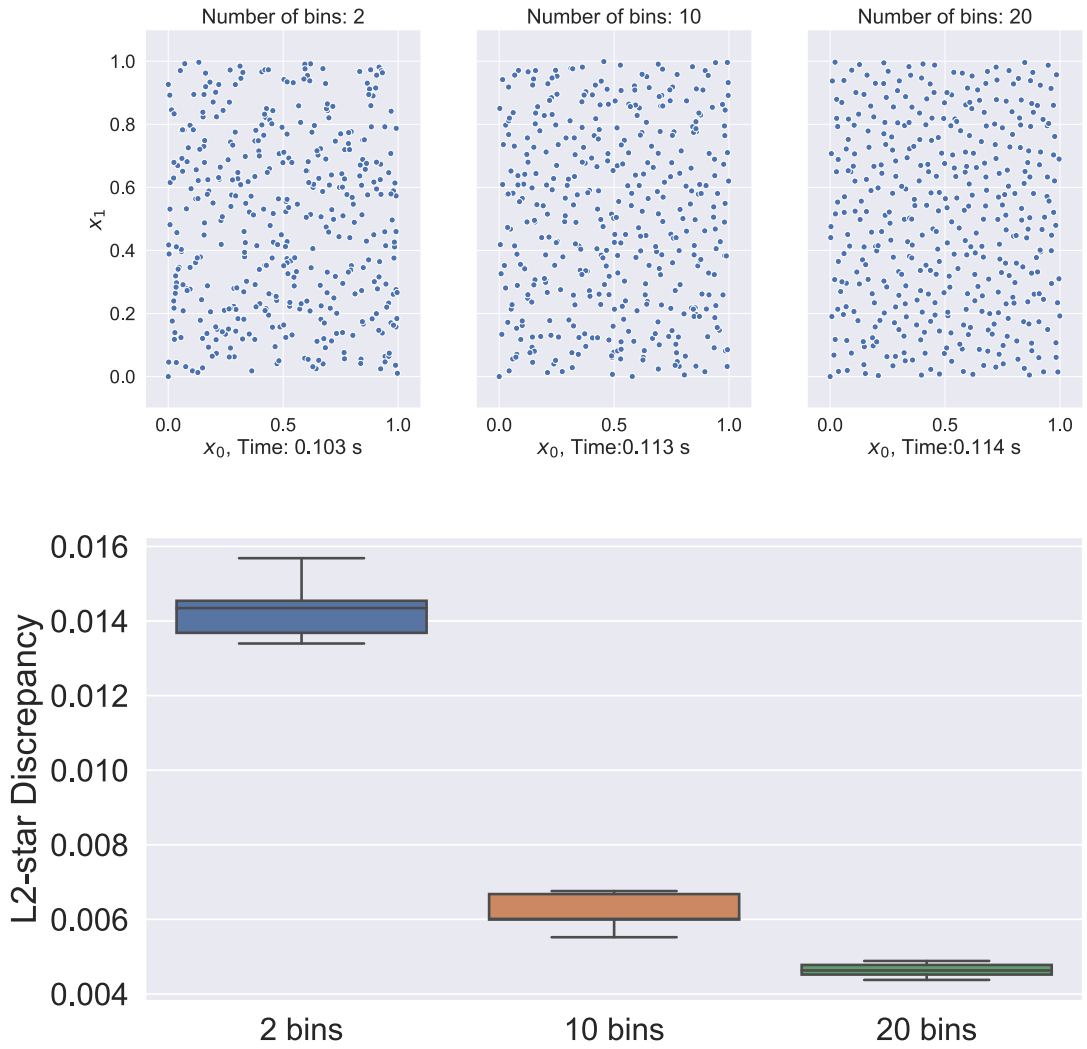


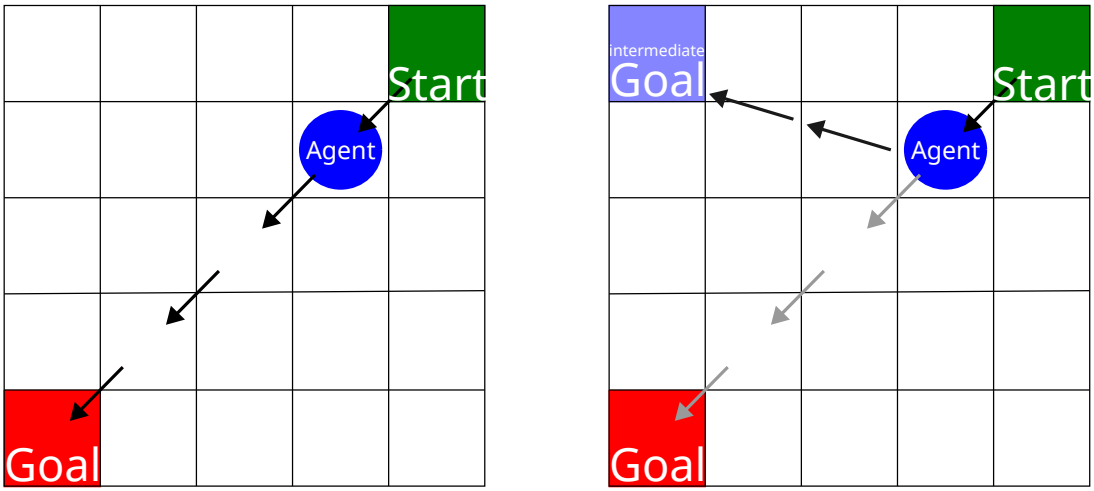
Fig. 5.10: 400 points sampled using HCXA in 2D state space with varying number of bins

5.1 Grid World Example

To demonstrate how HCXA can help with online exploration in reinforcement learning, a simple 5x5 grid world example is considered. The grid is shown in Fig. 5.11. The state and action space are continuous and defined by (5.3).

$$\begin{aligned} \boldsymbol{u} &\in [-1, 1]^2 \\ \boldsymbol{x} &\in [0, 4]^2 \end{aligned} \tag{5.3}$$

During normal exploration, the DDPG agent will use OU noise to learn about the grid and eventually reach the goal state. In the enhanced exploration case, HCXA and MPC are combined to not only reach the goal state but also to explore other regions of state space.



(a) Normal exploration **(b)** Enhanced exploration

Fig. 5.11: Two types of exploration

The dynamics of grid world are defined by a linear state space system with **A** and **B**:

$$\begin{aligned} \boldsymbol{x}[\mathrm{k}+1] &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \boldsymbol{x}[\mathrm{k}] + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \boldsymbol{u}[\mathrm{k}] \\ \boldsymbol{y}[\mathrm{k}] &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \boldsymbol{x}[\mathrm{k}] \end{aligned} \tag{5.4}$$

By using an MPC controller with the linear model shown in Eq. 5.4, any arbitrary reference state $\mathbf{x}^{*,\mathrm{mpc}}$ can be reached by formulating the cost function of MPC as a reference tracking task. Instead of randomly choosing $\mathbf{x}^{*,\mathrm{mpc}}$, the task can be designed more optimally by

using HCXA. Using the count-based approach, HCXA provides a new reference point for MPC such that the uniform distribution can be reached. In the path of reaching this new reference point $\mathbf{x}^{*,\text{mpc}}$, MPC will have to traverse through many intermediate state $\mathbf{x}^{*,\text{inter}}$. These states are also part of state distribution and hence are added to the coverage data set of HCXA. This way when HCXA is queried for next optimal reference state $\mathbf{x}^{*,\text{mpc}}$, the intermediates states $\mathbf{x}^{*,\text{inter}}$ already visited by MPC are also taken into account. This enables more efficient way to explore the state space and collect informative samples for RL agent.

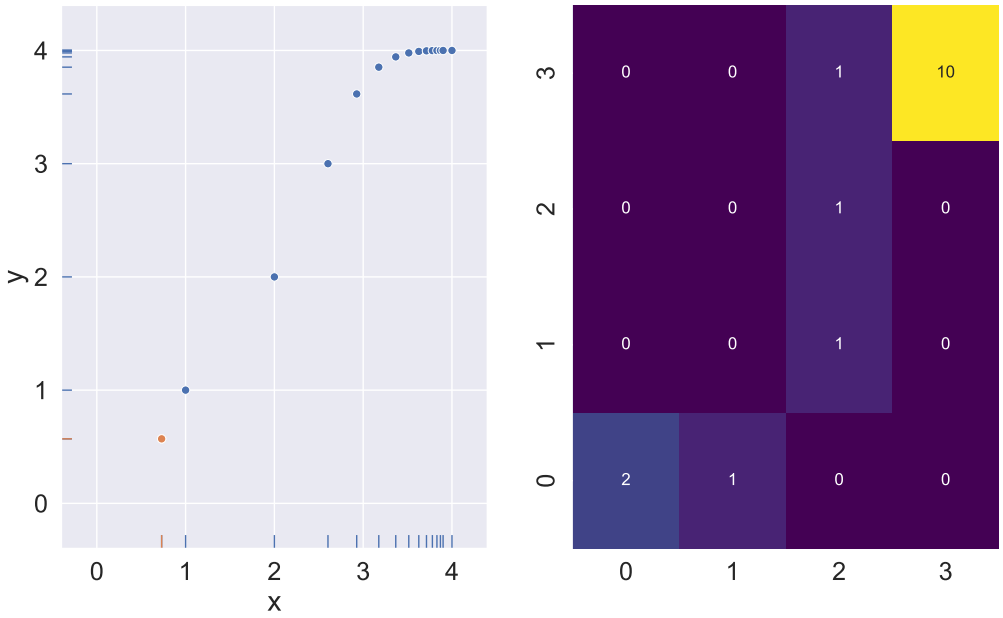


Fig. 5.12: Frame 1. The agent is guided by MPC to reach the goal(orange). The subplot on left represents the points visited by the agent. Blue points are state already visited. The subplot on right is a 2D histogram of the discretized state space using 5 bins in x and y axis

The trajectory in Fig. 5.12 is so-called targeted-exploration trajectory. The target is shown in orange and the states leading to it are shown in blue. Fig. 5.13 through Fig. 5.16, show many such trajectories. These information-rich trajectories are later used by the DDPG agent to effectively and efficiently learn an optimal policy.

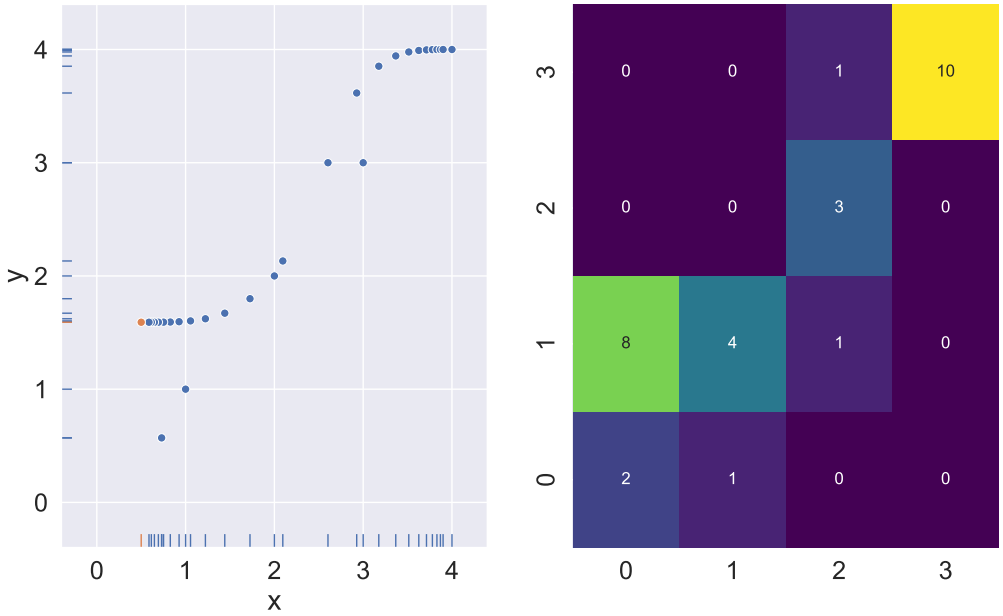


Fig. 5.13: Frame 2

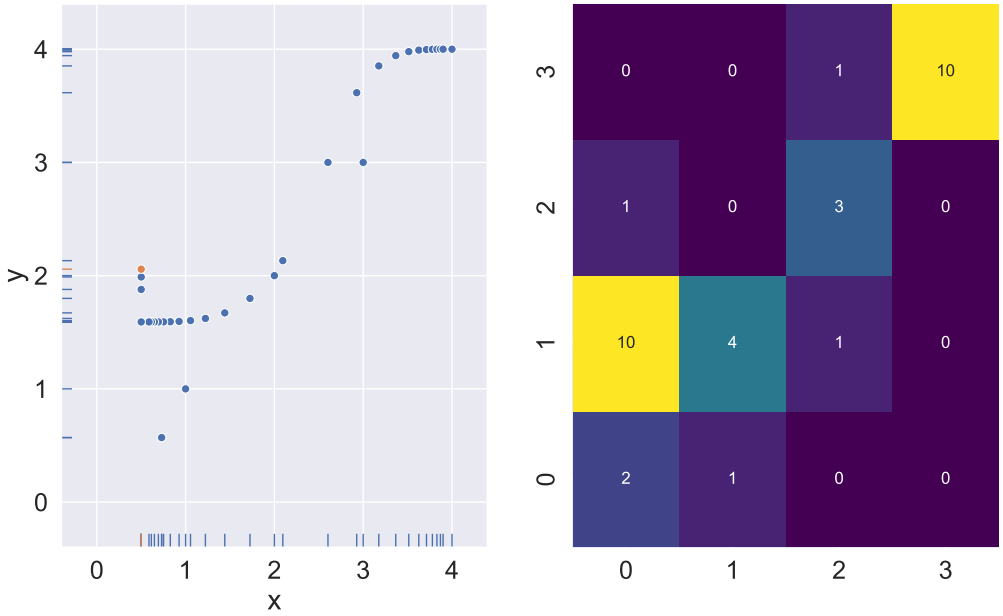


Fig. 5.14: Frame 3

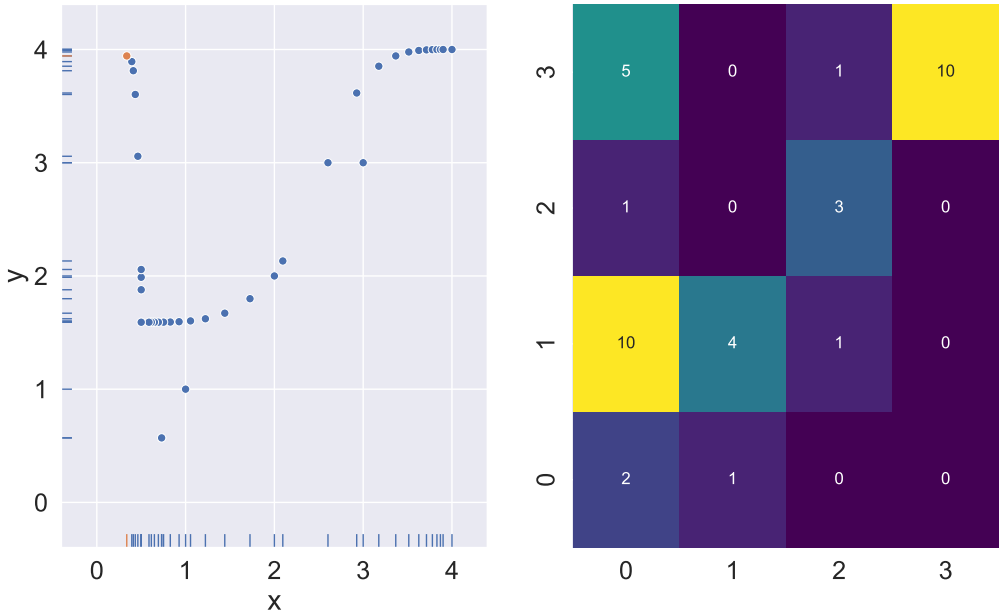


Fig. 5.15: Frame 4

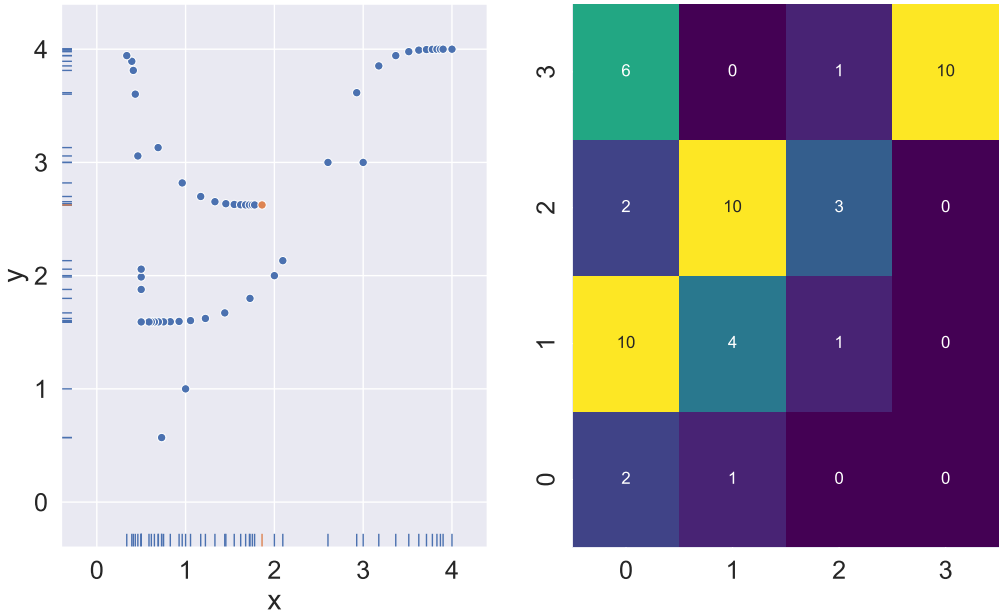


Fig. 5.16: Frame 5

6 Upgraded Reinforcement Learning Agent

This section explains how DDPG, MPC and HCXA, from Sections 3.3.2, 4 and 5 respectively, are combined together to form an upgraded RL agent.

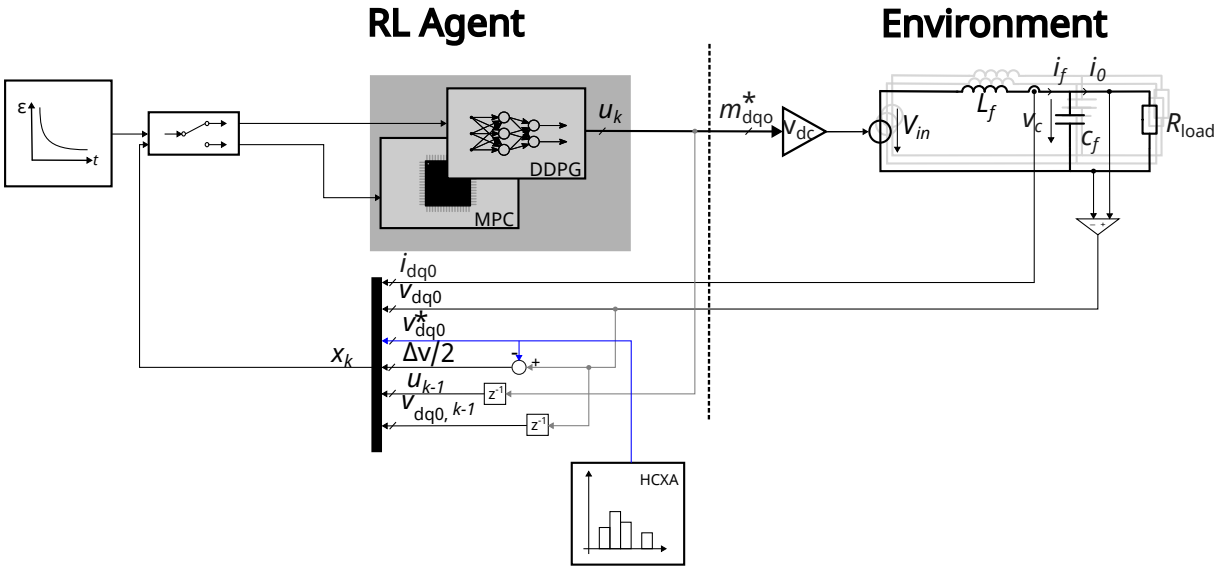


Fig. 6.1: Enhanced RL agent interacting with an electrical power grid application.

The aim of combining model-based predictive control with model-free reinforcement learning is to achieve a policy that takes advantage of both the approaches. At the start, utilize the samples collected using the MPC controller to initiate the RL policy and then slowly reduce the involvement of the MPC controller. From this point onwards, the RL agent has to learn by engaging with its environment. The switching of controller facilitates the possibility of learning a policy which benefits not only from expert knowledge but also with the real-time information gathered by interacting with the environment.

An approach similar to ϵ-greedy (Sec. 3.3.1) is used here to decide when to use actions from MPC and when to use DDPG. An exponentially decaying curve with varying slope can be generated using Eq. (6.1). ϵ_{start}, ϵ_{stop} are starting and end values of ϵ respectively.

N represents the total number of points in the exponential curve and k is current step. Decreasing the value of ϵ_{stop} results in steep decay of exponential curve as shown in Fig 6.2.

$$\epsilon_k = \left(\epsilon_{\text{start}} \cdot \frac{\epsilon_{\text{stop}}}{\epsilon_{\text{start}}}\right)^{\frac{k}{N}} \forall k \in \{1, 2...N\} \tag{6.1}$$

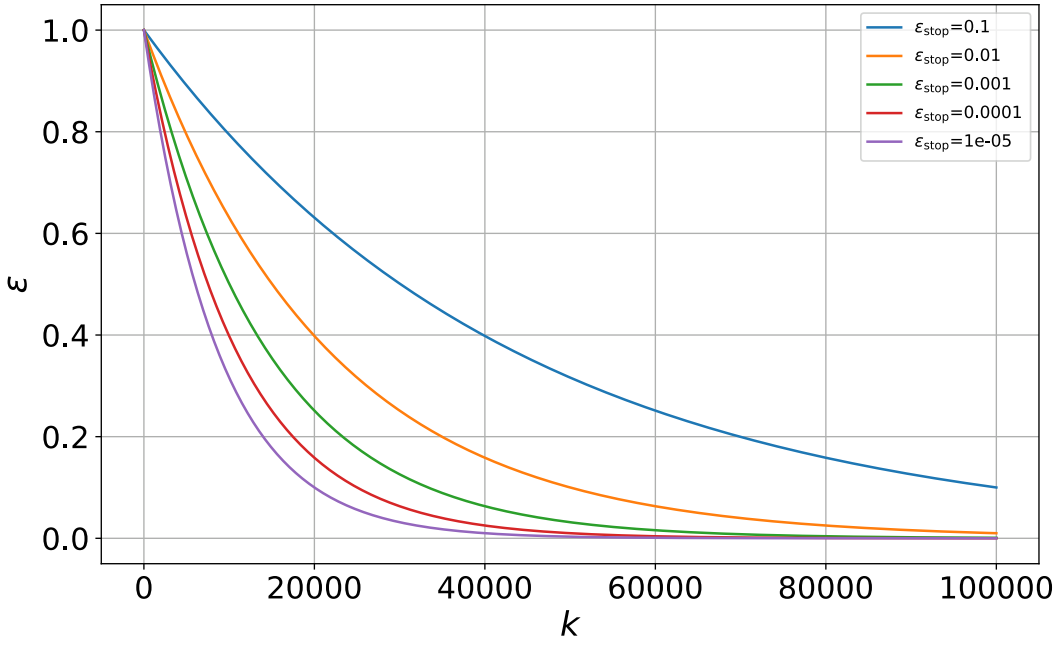


Fig. 6.2: Exponential Curves with different slopes. The curve in violet decays must faster than blue curve

A graphical visualization of how MPC is activated using the exponentially changing value of ϵ is shown in Fig. 6.3 and the logic is given in Eq. 6.2. By choosing a small or large value for ϵ_{stop}, the involvement of MPC can be decreased or increased respectively.

$$\text{MPC Active} = \begin{cases} \text{True}, & \epsilon_k > \text{random value} \in [0, 1) \\ \text{False}, & \text{otherwise.} \end{cases} \tag{6.2}$$

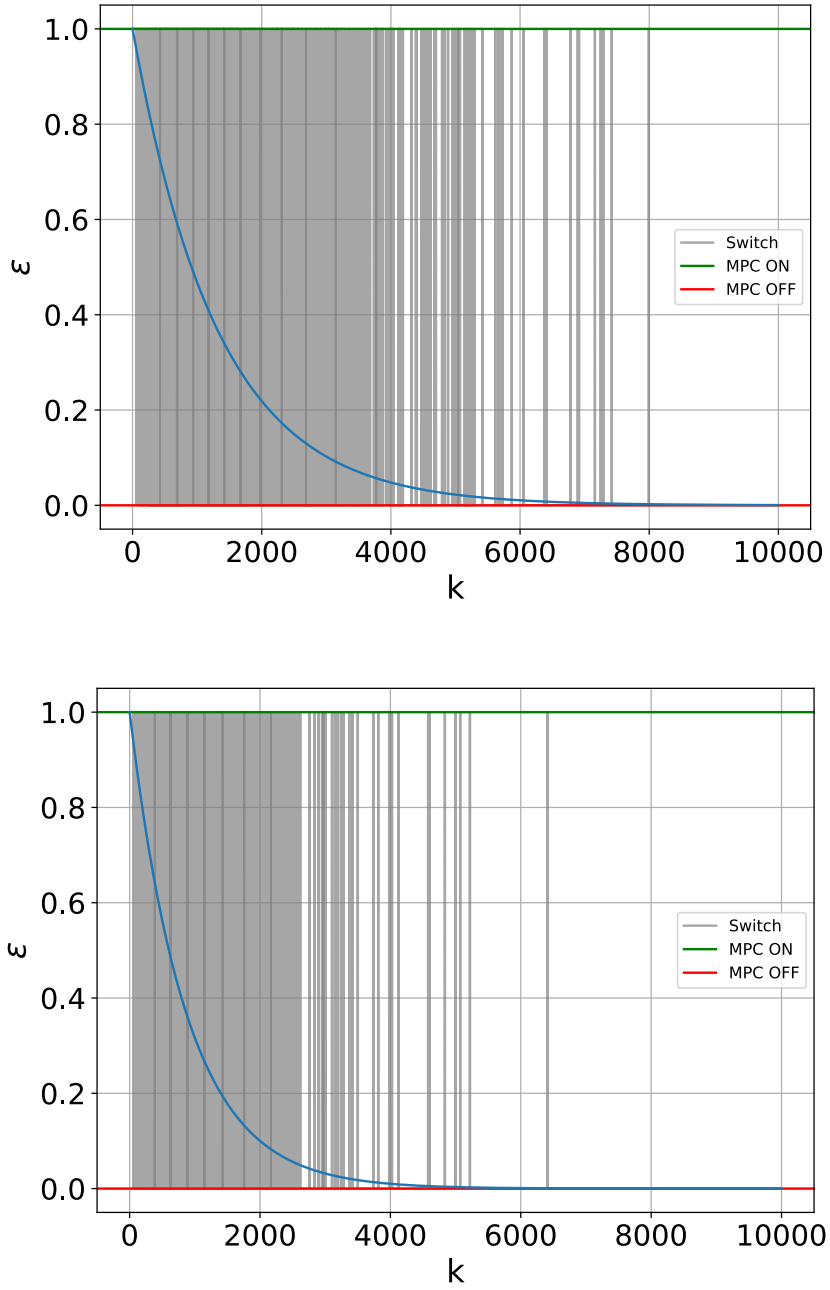


Fig. 6.3: The blue curve shows the value of ϵ changing with respect to learning step k. The red line represents the outcome of condition (6.2) if *false* and green line represents the outcome in case of *true*. The grey line indicates the state of the switch at that learning step k.

The pseudocode for the upgraded RL agent in given by Alg. 2. Prior to training, the replay buffer of DDPG agent is loaded with transition samples collected by using the

MPC to accelerate the learning. During the training, at each timestep k, condition (6.2) is checked. If True, action $\boldsymbol{u}_k$ applied to the environment is given by MPC. In case the condition is False, deterministic policy of DDPG, $\pi_\phi(\boldsymbol{x}_k)$ is used with OU noise. However, the variance of noise is significantly decreased compared basic RL agent. After applying the action $\boldsymbol{u}_k$ to the environment, the transition sample is stored to replay buffer and the next state $\boldsymbol{x}_{k+1}$ is added to HCXA's coverage data.

Algorithm 2 RL agent 2.0 pseudocode

Input: initialize HCXA, MPC controller and DDPG.
collect rollouts using MPC and store the transition to replay buffer $\mathcal{D}$.
repeat
  observe x_k.
  if MPC active according to (6.2)
    apply $u_k = \text{MPC}(x_k)$
  else:
    apply $u_k = \pi_\theta(x_k) + v_k$
  observe r_{k+1}, x_{k+1} and d_{k+1}
  add x_{k+1} to HCXA's coverage data.
  store experience $\langle x_k, u_k, r_{k+1}, d_{k+1} \rangle$ to replay buffer $\mathcal{D}$
  if x_{k+1} is terminal, reset environment
  if time to update **then**
    sample mini-batch $\mathcal{B} \subset \mathcal{D}$
    update θ, ϕ, θ_{target} and ϕ_{target}.
  end if
  k $\leftarrow$ k + 1
until convergence condition is met

In order to generate targeted-trajectories when using MPC, the reference voltage $\boldsymbol{v}_{dq0}^{*,\text{mpc}}$ is generated by HCXA. Since HCXA is updated at every timestep k, when queried it generates a new reference voltage $\boldsymbol{v}_{dq0}^{*,\text{mpc}}$ as target, which is sampled from region of low density according to coverage data at timestep k. Once the MPC is on, it stays on consecutively for K timesteps, a hyperparameter, before checking (6.2). The reason to postpone the (6.2) condition check by K timesteps, is to allow enough time for MPC algorithm to reach the reference voltage from any arbitrary state and generate informative samples.

The featured vector given to RL agent consists of Fig. 6.4 show an episode from training phase of a DDPG agent where $\tilde{\boldsymbol{u}}_{\text{abc}}$ are the normalized modulation indices and r is reward. The green line indicates whether MPC is on/off.

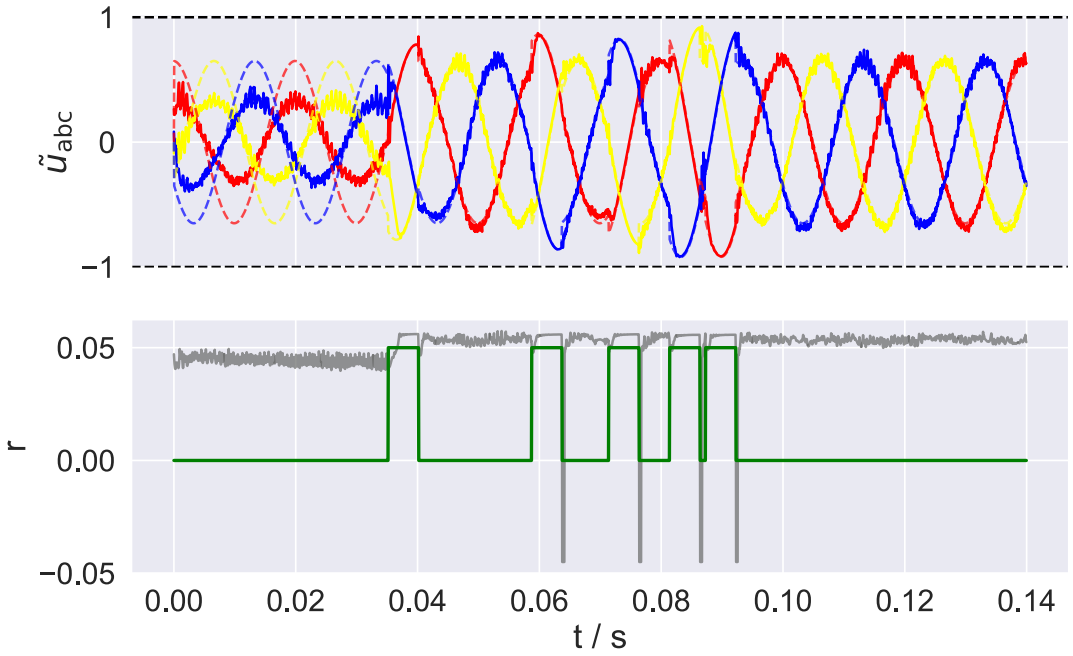


Fig. 6.4: Calculation of reward based on whether MPC or DDPG is active. The subplot on top shows modulation index realized(solid) vs what they should be(dashed). The second subplot show the reward(grey) and whether MPC is active

7 Evaluation and Conclusion

The control application considered in this work is three-phase voltage source inverter providing 3-phase AC voltage with root mean square value (RMS) of 230 volts at 50 Hz and peak amplitude of 320 volts. A stochastic unkown resistive load is connected to the inverter. An exemplary load curve is shown in Fig. 7.1. In section 7.1, a training and testing results of basic version of DDPG-based agent is shown. Section 7.2 shows performance of RL agent with enhanced exploration. In order to compare the dynamic response of both the agents their performance is demonstrated for two different type of load steps.

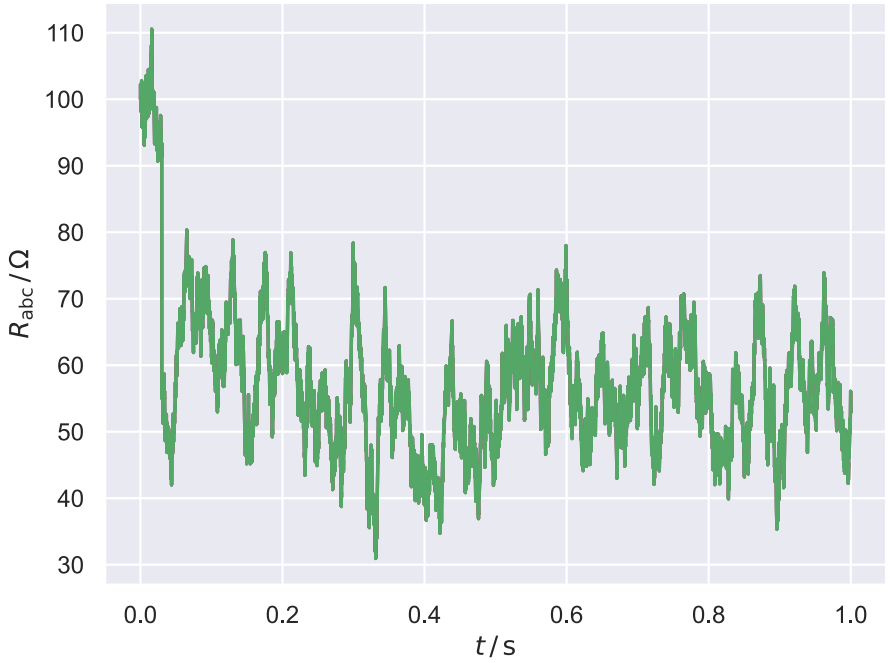


Fig. 7.1: Exemplary power grid load curve used for validation

	Symbol	Values
Filter inductance	L_1	70mH
Filter capacitance	C_1	250μF
Internal resistance inductor	R_1	1.1$\mu\Omega$
Internal resistance capacitor	R_2	7$\mu\Omega$
DC-link voltage	$_{\text{DC}}$	1000V
Sample time	T_s	0.1ms
Nominal current	i_nom	300A
Nominal voltage	v_nom	$230 \cdot \sqrt{2}$V
Current limit	i_lim	400A
Voltage limit	v_nom	500V
Maximal voltage	v_max	1.2$\cdot v_\text{nom}$
Grid frequency	f	50Hz

Tab. 7.1: Considered system parameters and operating values.

A safeguard[4] is used to ensure that actions taken by agent are safe and do not terminate the environment.[1,2].

[1]The code is provided by the author https://github.com/Webbah/Safe-Reinforcement-Learning-Based-Control-in-Power-Electronic-Systems.git.

[2]Please refer to appendix A.2 for more details.

7.1 Performance of basic RL agent

Basic RL agent refers to a DDPG agent that uses just OU noise to perform exploration. Stable-baselines [27] is used to implement the agent in python that can interact with OMG (the environment). Fig. 7.2 show the average reward of 5 basic DDPG agent trained for 100,000 steps. In order to analyze the training and evaluate the learning curve, a moving average of both average reward and one standard deviation was calculated using a window size of 1000 steps. With γ set to 0.94, the maximum reward possible is $r_{\max} = 0.06$. From the Fig. 7.2, it can be inferred that basic DDPG learns a suboptimal policy with $\overline{r} \approx 0.04$. In the situations when the actions selected by the agent are unsafe according A.2, and safeguard has to intervene the reward is set to lowest operation limit in (3.17). This value corresponds to $r = -0.75 \cdot (1 - \gamma) = -0.045$.

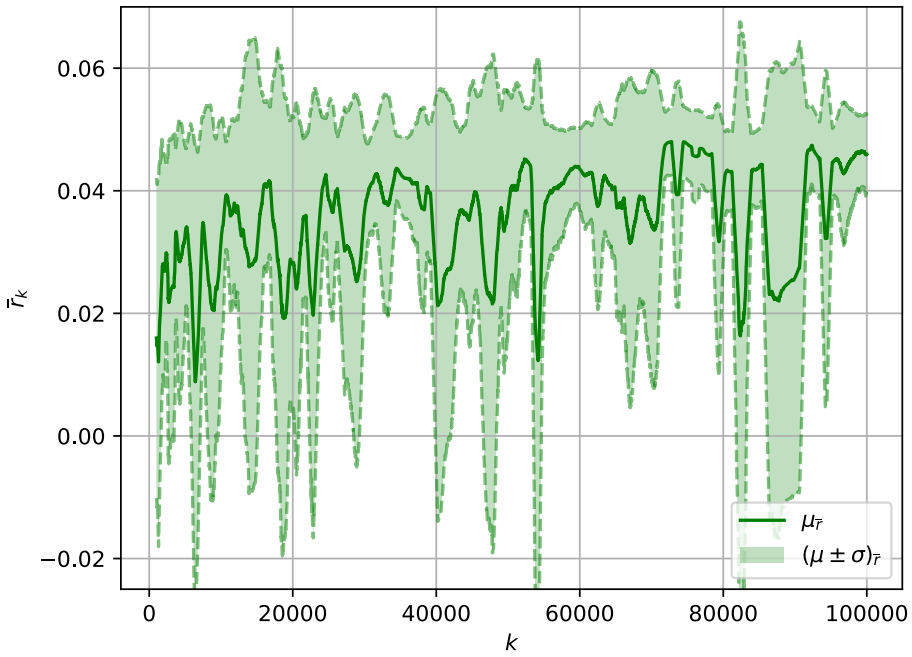


Fig. 7.2: Averaged mean reward and one standard deviation per learning step of the 5 trained agents. A moving average with window size of 1000 steps was a applied to the averaged mean.

Fig. 7.3 show that the trained basic agent is able to track the reference voltage, however, with significant harmonics both in current and voltage waveform. Looking at the modulation indices $\tilde{\boldsymbol{u}}_{\mathrm{dq0}}$, it is clear why. The d-component(red) and q-component(yellow) are oscillating leading to voltage and current harmonics in the inverter.

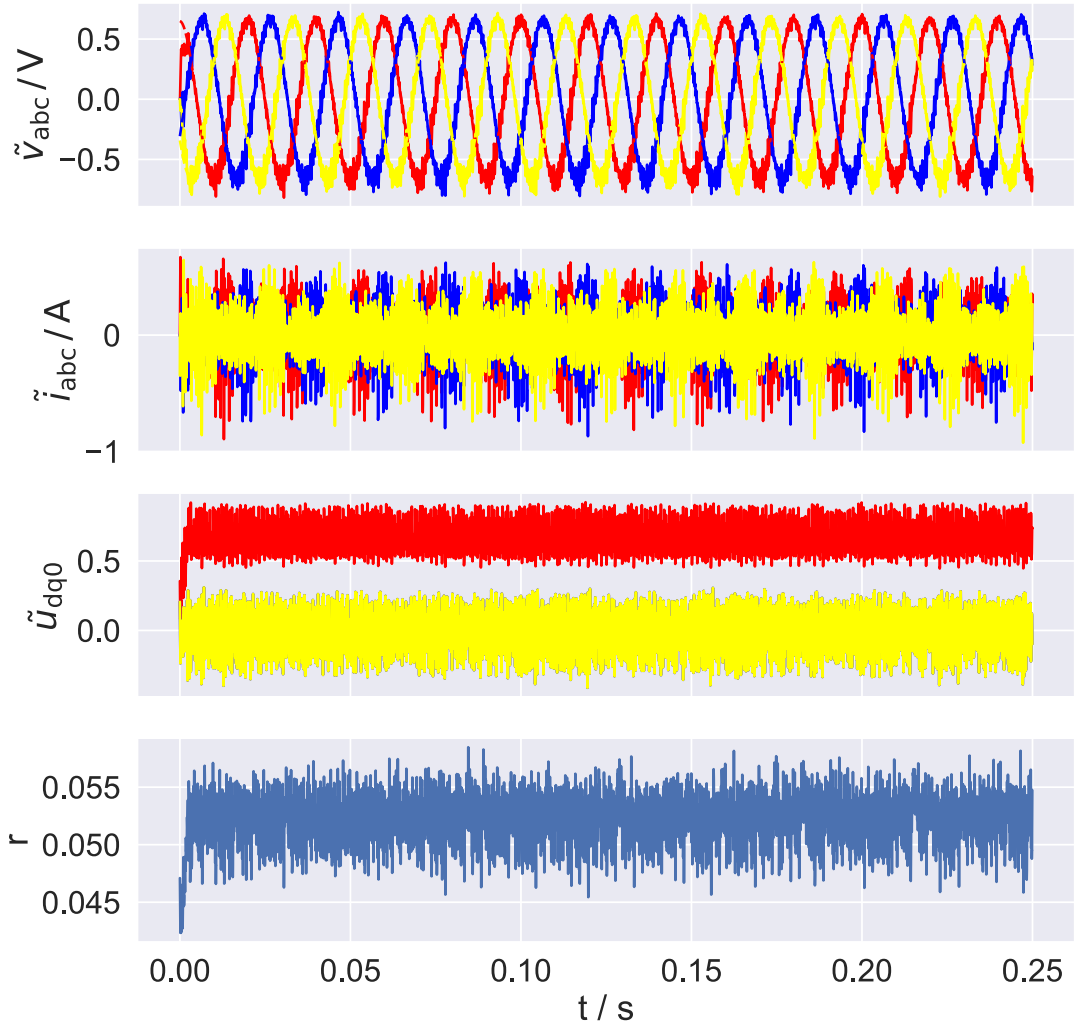


Fig. 7.3: Beginning of the testcase with DDPG agent

7.2 Performance of upgraded RL agent

Using the methods and tools described in the earlier sections, a DDPG agent with enhanced exploration technique is also trained for 100,000 steps using similar setup as for basic DDPG agent before it. Fig. 7.4 shows how the reward for 5 such trained agents during the training. For simplicity and better inference, the reward and one standard deviation is averaged over a moving window of size 1000. The upgraded agent was load with pre-computed replay buffer of size 100,000 steps which was computed using only MPC and HCXA. The $\epsilon_{\text{start}} = 0.005$ and $\epsilon_{\text{stop}} = 1e^{-5}$. In normal setting, the Fig. 7.4 would have a perfect way to make conclusion about the performance of the RL agent. However, in this case, RL agent is working along side MPC, the reward is skewed and is not a key performance indicator of the controller.

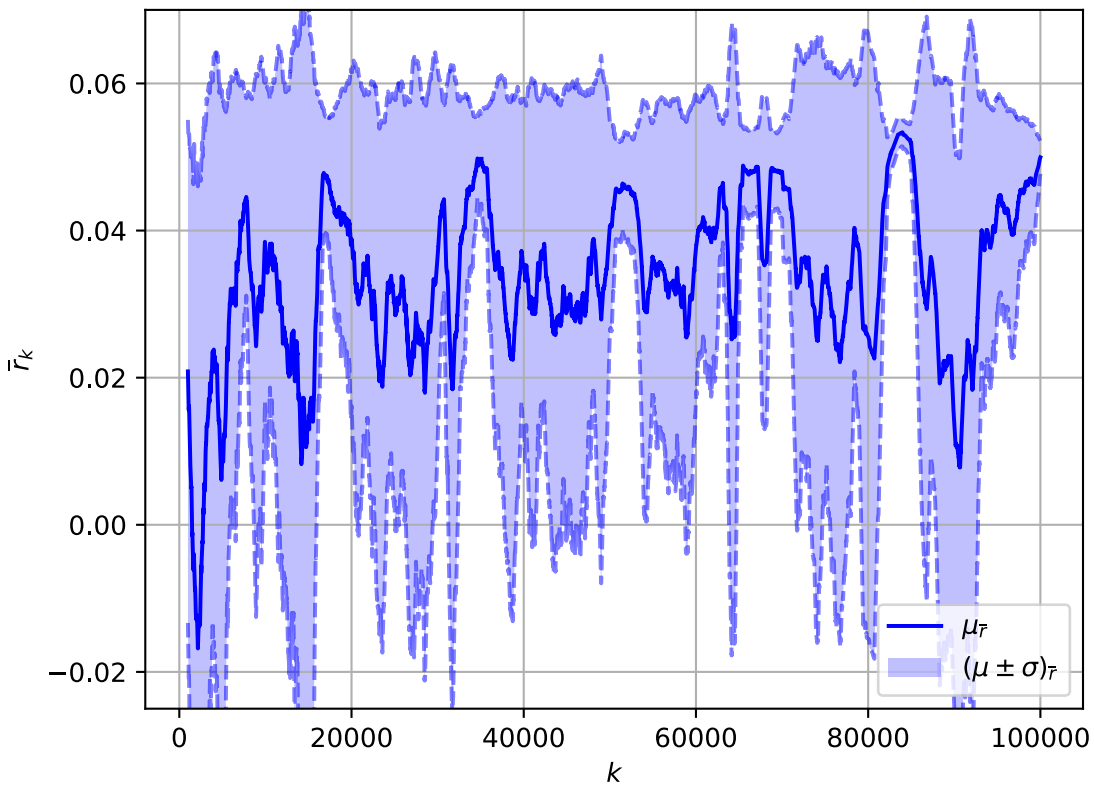


Fig. 7.4: Averaged mean reward and one standard deviation per learning step of the 5 trained enhanced agents. A moving average with window size of 1000 steps was a applied to the averaged mean.

The involvement of MPC is exponentially decreased over the training phase. Hence, towards the end of the training, only RL agent is responsible for selecting the action chosen

to interact with the environment. From Fig. 7.4, we can see around ≈ 85,000 steps, the reward increases significantly across all the 5 agents. However, after few more training steps, there is large oscillation. These oscillations can be seen almost throughout the whole training and are result of change in policy caused due large variations in actor network parameters. This phenomenon, where the RL agent forgets the optimal policy and starts behaving poorly after every few episodes, is commonly referred to as 'policy degradation' or 'catastrophic forgetting'[28].

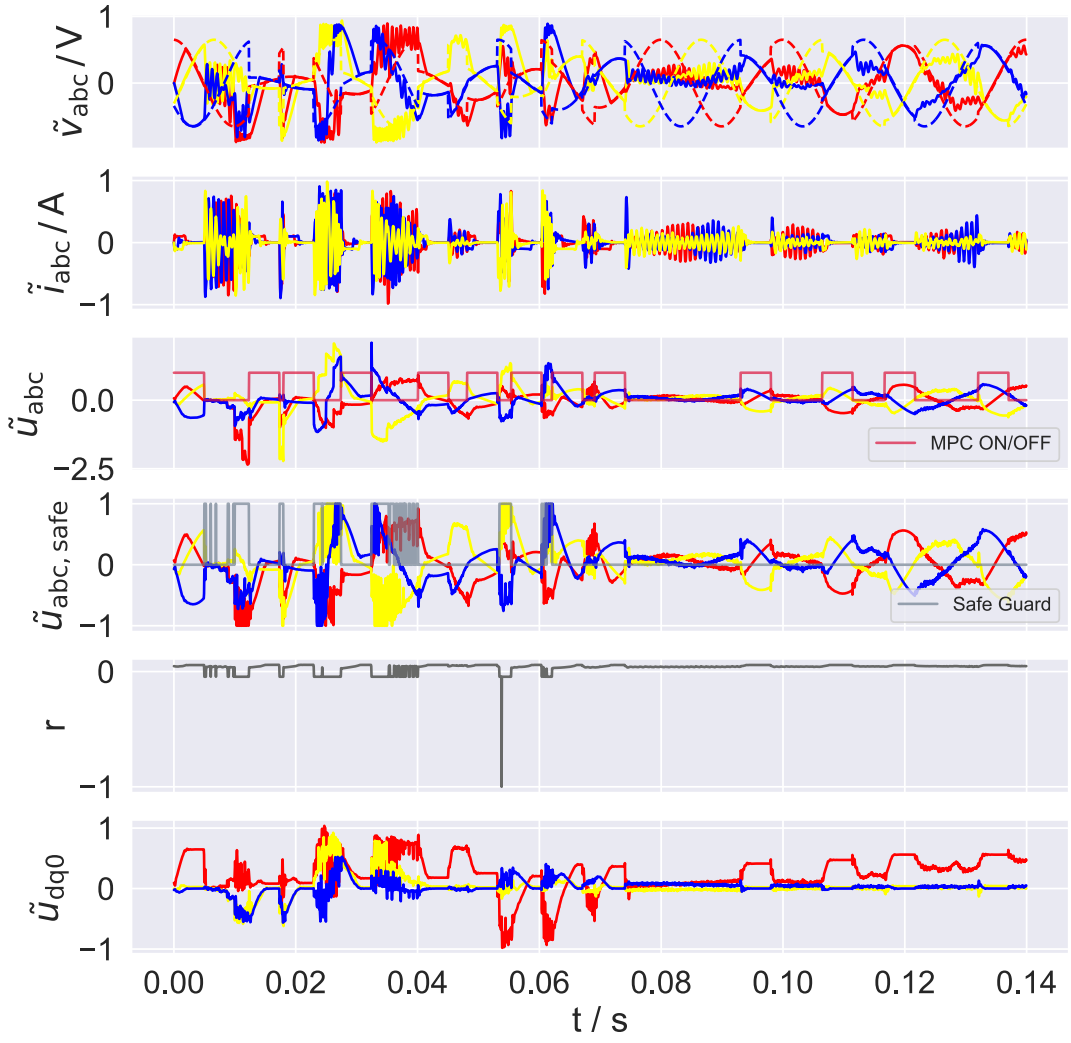


Fig. 7.5: Training episode 1, k =2800 steps

In Fig. 7.5, the first training episode is shown. The top subplot $\tilde{\boldsymbol{v}}_{\text{abc}}$ shows the normalized value of voltage in abc reference frame. Similarly, the second subplot shows the normalized current values $\tilde{\boldsymbol{i}}_{\text{abc}}$ through the inductor in abc reference frame. The third subplots illustrates the normalized modulation indices $\tilde{\boldsymbol{u}}_{\text{abc}}$ chosen by the controller. Additionally, it also displays a scaled binary signal(crimson red) indicating instances when MPC is active. Fourth subplot shows how safeguard[4] ensure that actions provided by the controller to the system are safe. Every time safeguard is active, the gray curve switches state from logic 0 to logic 1. The last two subplots show the reward r at each time step and the actions $\tilde{\boldsymbol{u}}_{\text{dq0}}$ taken by agent in dq0 reference frame respectively.

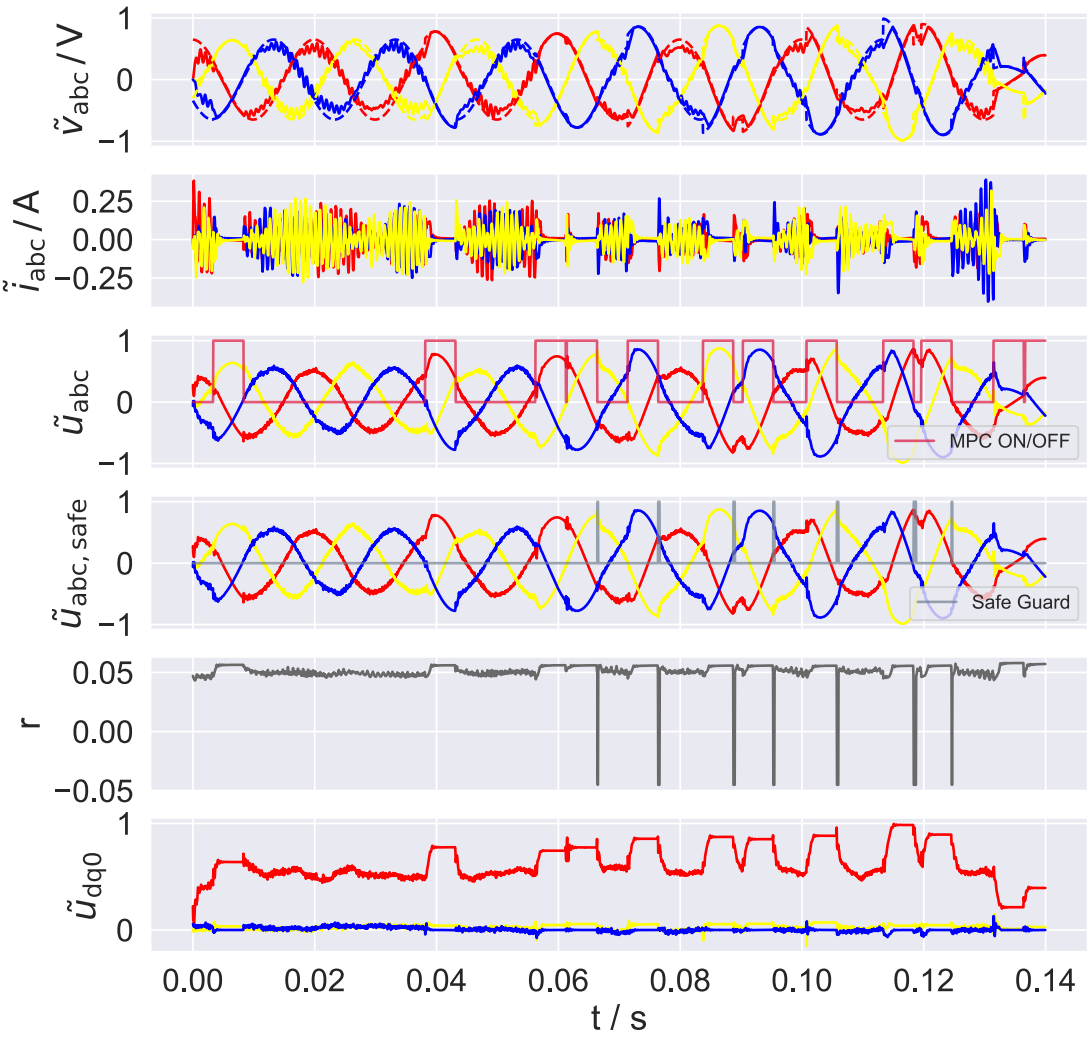


Fig. 7.6: Training episode 2, k =5600 steps

The RL agent, by the end of first episode, is able to learn a good policy which is only improves in the second episode. However, as the training progresses, the policy fluctuations occur, stopping the RL agent to converge to optimal policy. This can be seen in Fig. 7.7.

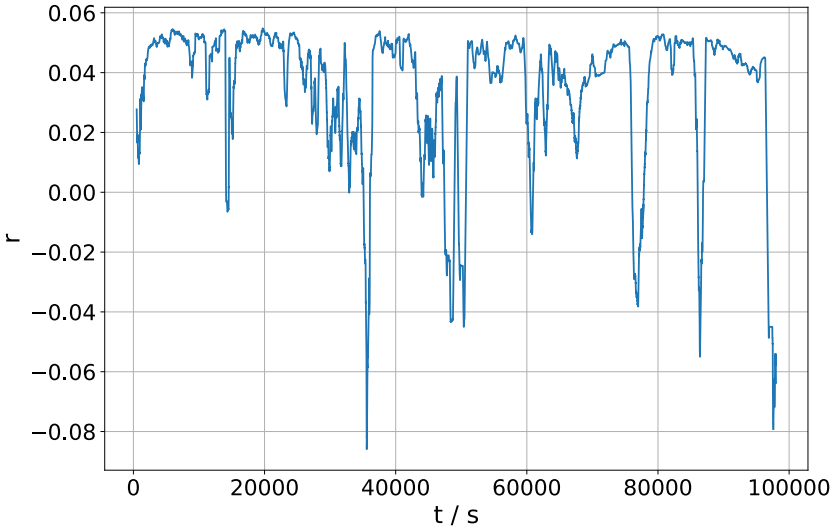


Fig. 7.7: Reward over 100K steps

Fig. 7.8 shows the validation of a trained agent in OMG. The trained agent is able to achieve a steady state with peak voltage of 325V. Furthermore, the agent is able to achieve a great sinusoidal shape in current waveforms as well indicating the voltage variations in dq0 frame is minimum. This results in almost zero current ripple.

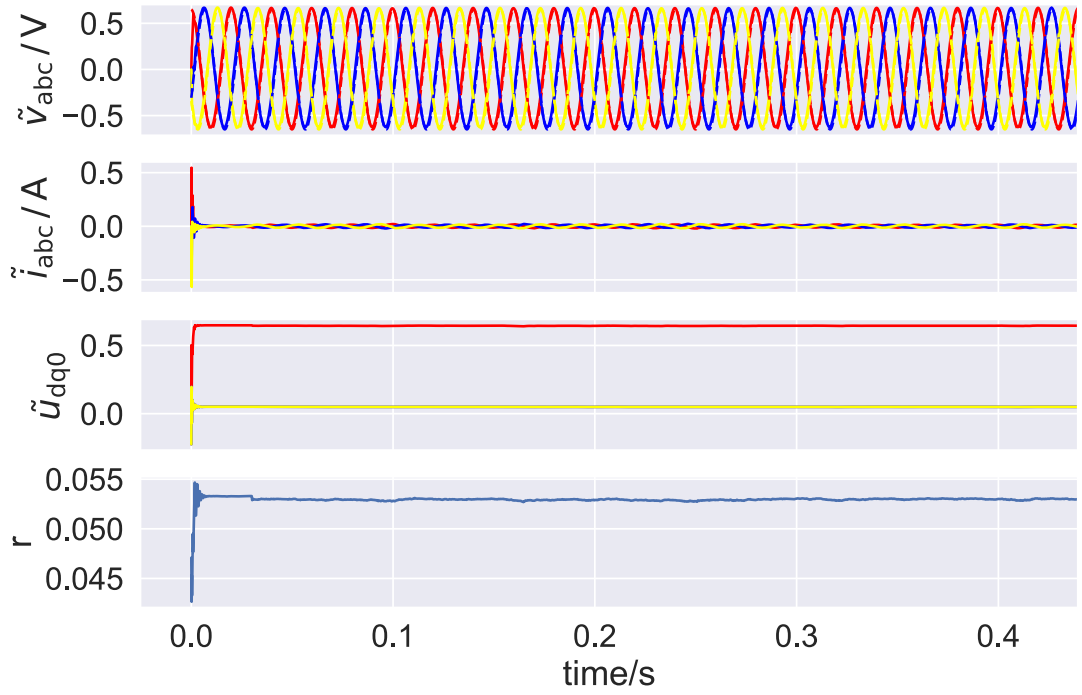
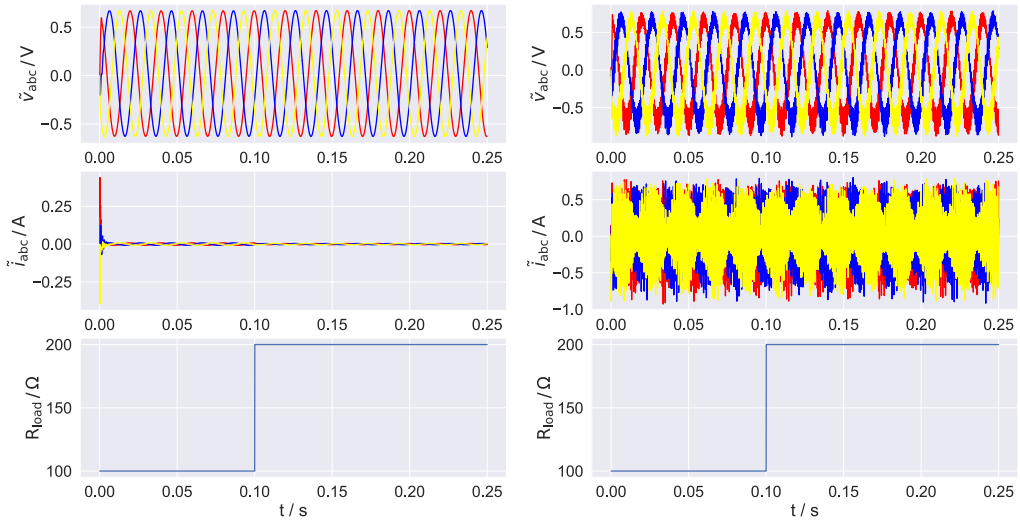
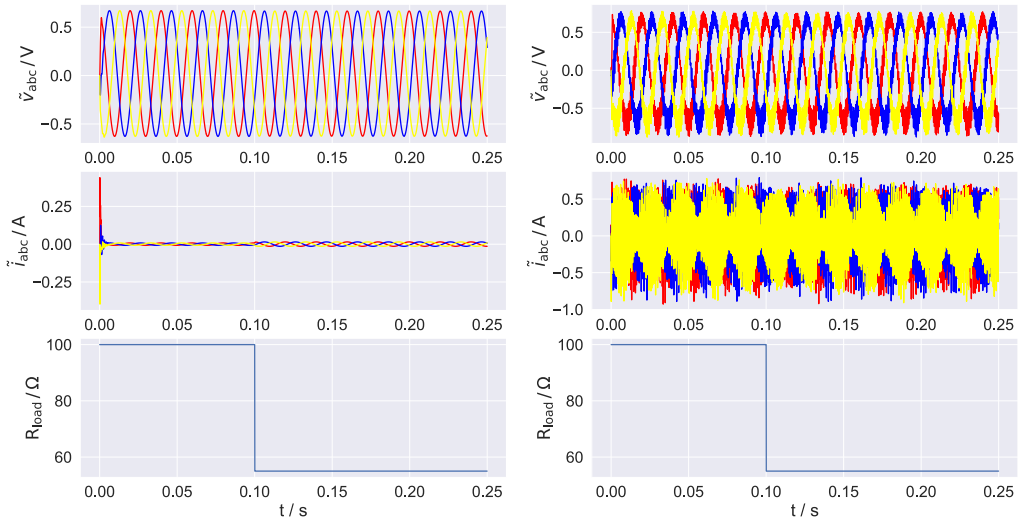


Fig. 7.8: Testing episode 1, k =20000 steps

The enhanced RL agent shown in Fig. 7.9 is trained for only 50,000 steps, which is half compared to its counter part. However, it is important to note that the comparison is not entirely fair as the enhanced RL agent had a pre-loaded buffer. Nevertheless, the learning has improved significantly. In order to demonstrate the dynamic response of both the controllers, two exemplary load steps are performed. These represent some of the more demanding test cases for a voltage forming inverter in the field [29]. In Fig. 7.9a, the load changes in a step like manner from 100 Ω to 200 Ω at exactly 0.1s. The upgraded RL agent(left) is able to maintain constant voltage with very slight increase in inductor current. A similar test is performed again with load changing from 100 Ω to 45 Ω and is shown in Fig. 7.9b. In all the test and validation case, only RL controller are used to select an action. Additionally, no action noise is used as the exploration is stopped.



(a) Voltage and current for both controllers when load is increased in step-like manner.



(b) Voltage and current for both controllers when load is decreased in step-like manner.

Fig. 7.9: Exemplary loadsteps. Upgraded RL agent on left and basic RL agent on right

7.3 Conclusion and Outlook

The focus of this work was to improve the learning by enhancing the current method of exploration used in reinforcement learning. An alternative, in form of targeted-exploration trajectories was suggested. These target-exploration trajectories were generated using a model predictive controller with perfect knowledge about the environment. In order to perform online-exploration, a histogram-based count maximization algorithm was proposed. Compared to the fast time of DESSCA, HCXA is 6x times faster. The computation of DESSCA at 1000 points is on average 0.06s where as HCXA maintains a constant time of 3ms. This shows HCXA is fast and applicable to real-world scenarios. During the training of the enhanced RL agent, it was observed that the agent is able to learn near-optimal policy very quickly. One of the trained agent shows 50% reduction in training time compared to basic RL agent.

However, the policy keeps on changing every few episodes due to a phenomenon known as catastrophic forgetting. There are several factors that can lead to this degradation in policy such as high learning rate or frequent updates of the target networks. To solve this issue, hyperparameter optimization needs to be performed (using Optuna[30]). The proposed method bears the result for further investigation and can significantly improve the training time. Another alternative is to use twin-delayed deep deterministic policy gradient TD3[12], which is an extension of DDPG but is less sensitive to hyperparameters. This can stablize the learning and yield an optimal policy.

In this work, the model of the system was assumed to known. This is not a very practical assumption and estimation methods such recursive least square should be employed in the training phase to estimate the system parameters. In reinforcement learning, reward design is very important and determines the behaviour of the agent. As a next step, a more complex reward function should be considered that leverages more of the expert knowledge.

Appendix

A.1 Iteration of Exploration in Grid World

For the sake of completeness, the rest of the frames for the grid world example are shown here.

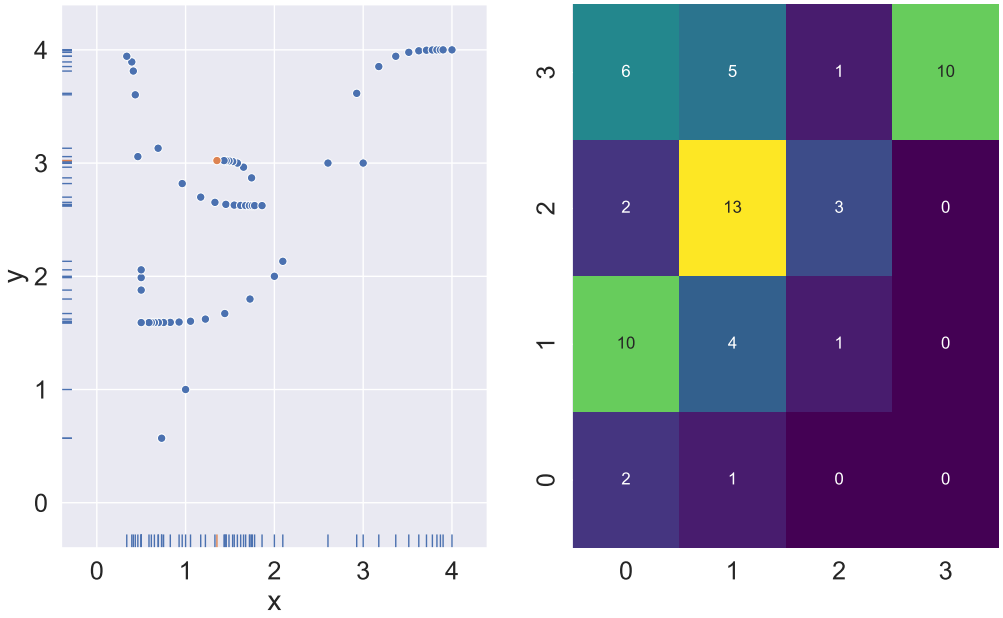


Fig. A.1: Frame 6

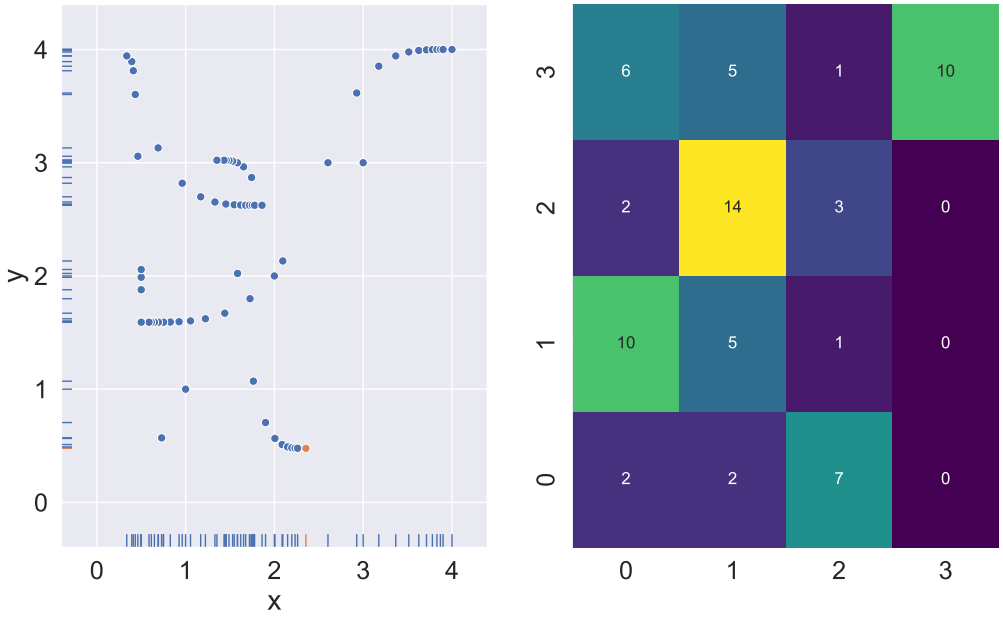


Fig. A.2: Frame 7

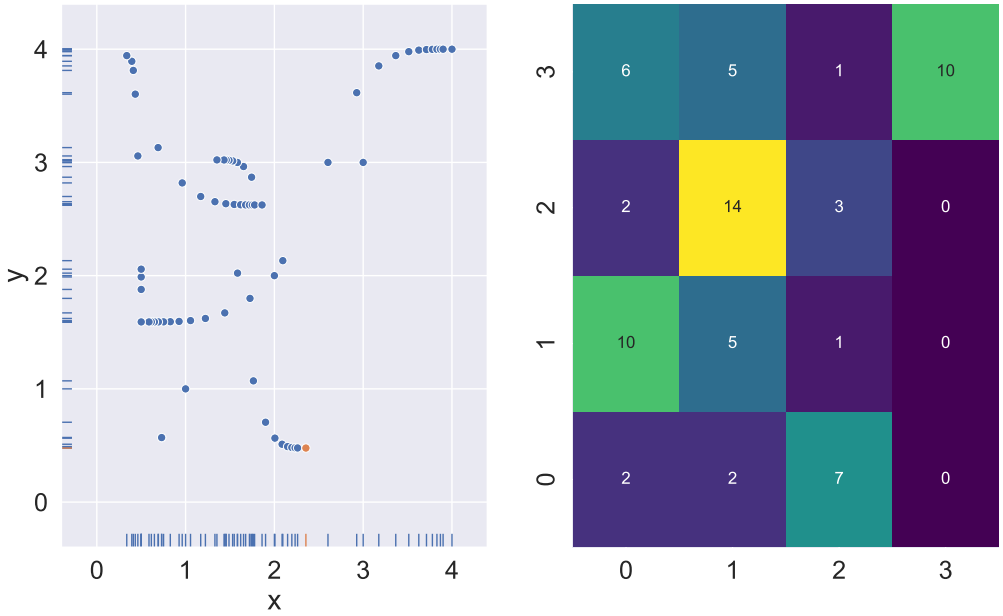


Fig. A.3: Frame 8

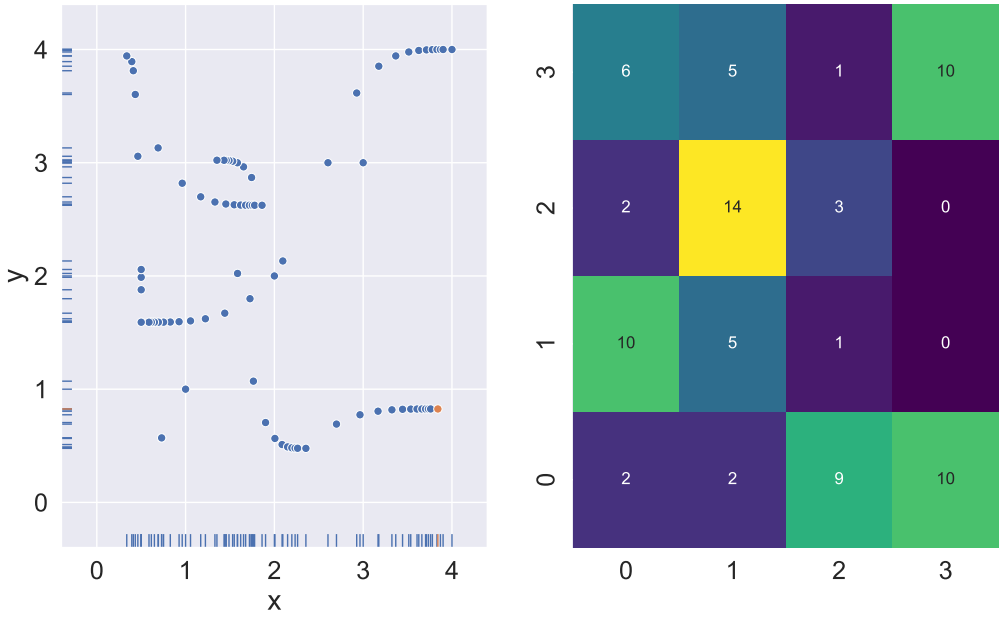


Fig. A.4: Frame 9

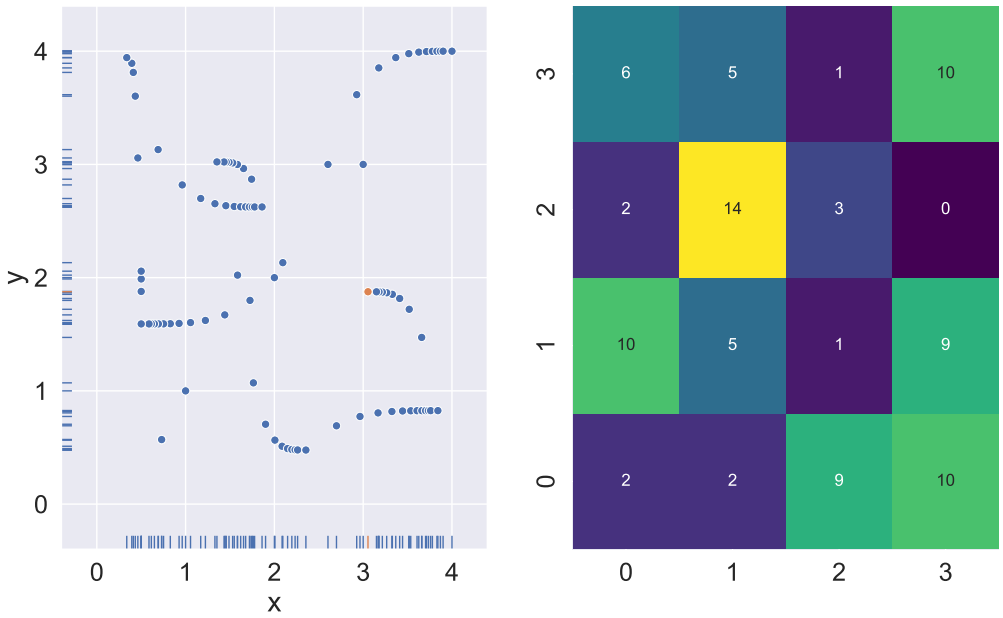


Fig. A.5: Frame 10

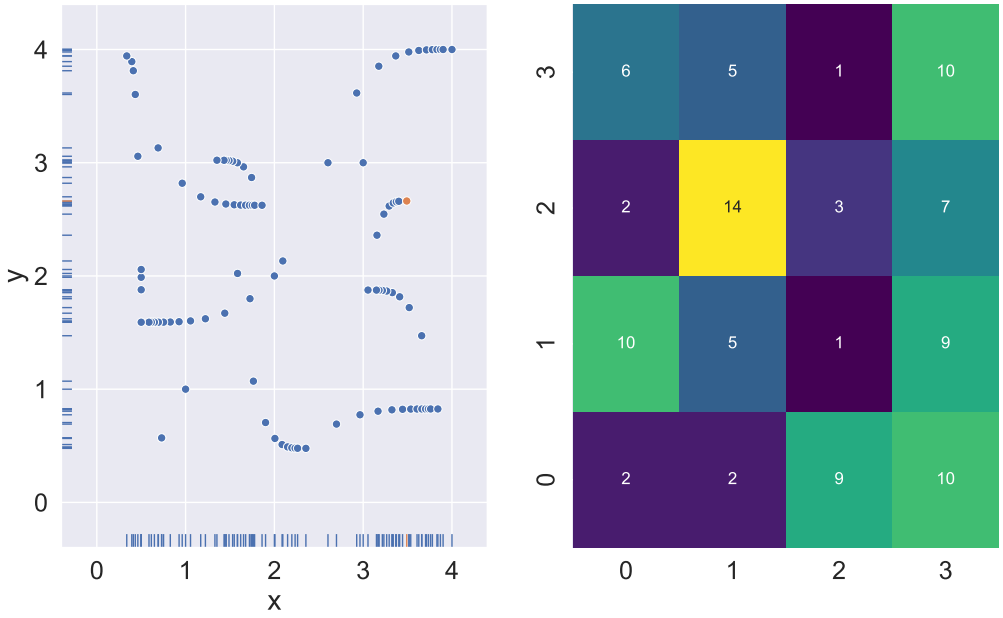


Fig. A.6: Frame 11

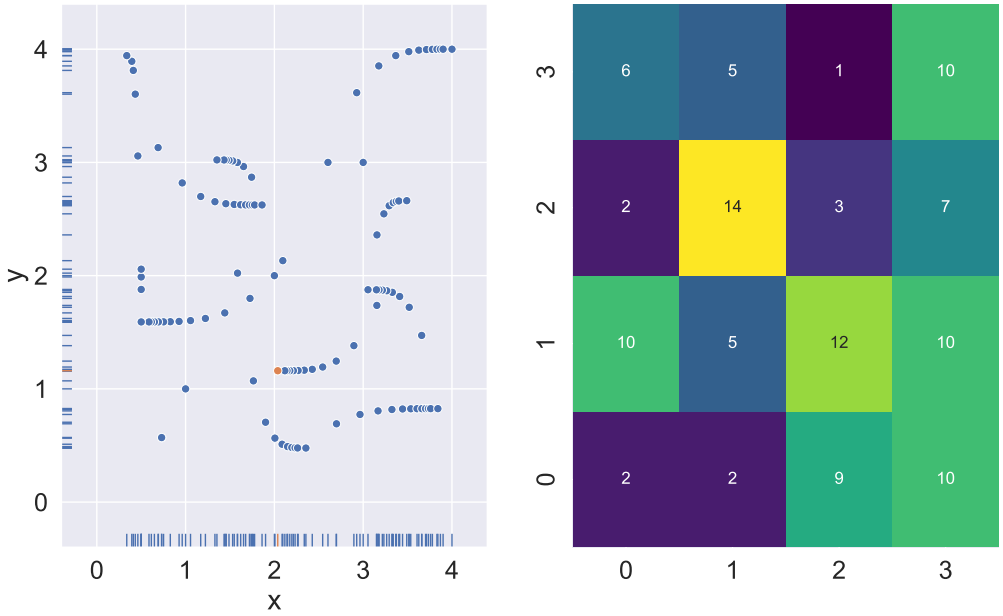


Fig. A.7: Frame 12

A.2 Safeguard

In [4], author successfully guarantees safety using constrained optimization approach of MPC in electrical power grid setting identical to the one used in this thesis. Assuming a symmetrical balanced load, the three-phase system can defined as linear discrete state-space system per phase(A.1).

$$\boldsymbol{x}_{\mathrm{k+1}} = \boldsymbol{A}_{\mathrm{d}}\boldsymbol{x}_{\mathrm{k}} + \boldsymbol{b}_{\mathrm{d}}u_{\mathrm{k}} \quad \forall p \in \{\mathbf{a},\mathbf{b},\mathbf{c}\} \tag{A.1}$$

with

$$\mathbf{A} = \begin{bmatrix} -\frac{R_1}{L_1} - \frac{R_2 R_{\mathrm{load}}}{L_1(R_2+R_{\mathrm{load}})} & -\frac{1}{L_1} + \frac{R_2}{L_1(R_2+R_{\mathrm{load}})} \\ \frac{R_{\mathrm{load}}}{C_1 R_2 + R_{\mathrm{load}}} & \frac{1}{C_1 R_2 + R_{\mathrm{load}}} \end{bmatrix}, \mathbf{b} = \begin{bmatrix} \frac{1}{L_1} \\ 0 \end{bmatrix} \tag{A.2}$$

.

The work of the safeguard is to check whether the action selected by agent $u_{k,\mathrm{RL}}$ satisfies the input and state constraints for all future states. In the case, the action selected by DDPG agent is not safe, safeguard needs to provide an action $u_{k,\mathrm{SG}}$. A state-action pair that satisfies the bounds for all future states are considered feasible. A set of feasible states $\mathcal{X}_0$ can be iteraively determined while optimizing a quadratic programming (QP) [15]:

$$\min_{\boldsymbol{u}} \frac{1}{2}\boldsymbol{u}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{u} + \boldsymbol{x}\boldsymbol{F}^{\mathrm{T}}\boldsymbol{u}, \quad \text{s.t.}\, \boldsymbol{G}\boldsymbol{u} \leq \boldsymbol{E}\boldsymbol{x} + \mathbf{e}. \tag{A.3}$$

Here, $\boldsymbol{H}, \boldsymbol{G}, \boldsymbol{F}, \boldsymbol{E}, \boldsymbol{W}_{\boldsymbol{x}}, \boldsymbol{e}$ are the condensed QP matrices and vectors that incorporate the future system trajectories subject to the given plant model, cost function and constraints[4](c.f. [15] for theoretical reference).

All states for which an action $\boldsymbol{u}$ can be found that satisfies Eq. (A.3) make up the feasible set: $\mathcal{X}_0 = \{\boldsymbol{x}_0 \in \mathcal{X} | \boldsymbol{u} \in \mathcal{U} : \boldsymbol{G}\boldsymbol{u} \leq \boldsymbol{E}\boldsymbol{x}_0 + \boldsymbol{e}\}$. The QP in Eq. (A.3) is solved using a convex cost function to a safe action:

$$\boldsymbol{u}_{k,\mathrm{SG}} = \mathrm{argmin}_{\boldsymbol{u}_k} \|\boldsymbol{u}_k - {\boldsymbol{u}_{k,\mathrm{RL}}}^2\|, \quad \mathbf{s.t.} \quad \boldsymbol{u}_k \in \mathcal{F}_0. \tag{A.4}$$

where $\mathcal{F}_0$ is a combined polytope for feasible set of $\boldsymbol{x}$ and $\boldsymbol{u}$ defined as:

$$\mathcal{F}_{\mathbf{0}} = \left\{ \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{u} \end{bmatrix} \in \mathbb{R}^{n+m} \middle| \begin{bmatrix} \boldsymbol{W}_x & 0 \\ \boldsymbol{E} & \boldsymbol{G} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{u} \end{bmatrix} \leq \begin{bmatrix} \omega_{\mathbf{x}} \\ \boldsymbol{e} \end{bmatrix} \right\} \tag{A.5}$$

.

An exemplary feasible set $\mathcal{F}_0$ for the system described in Eq. (A.1) is visualized in Fig. A.8 as green polyhedron. All the states and action pairs inside the green region are within the feasible set(i.e., the blue dot, but not the red dot).

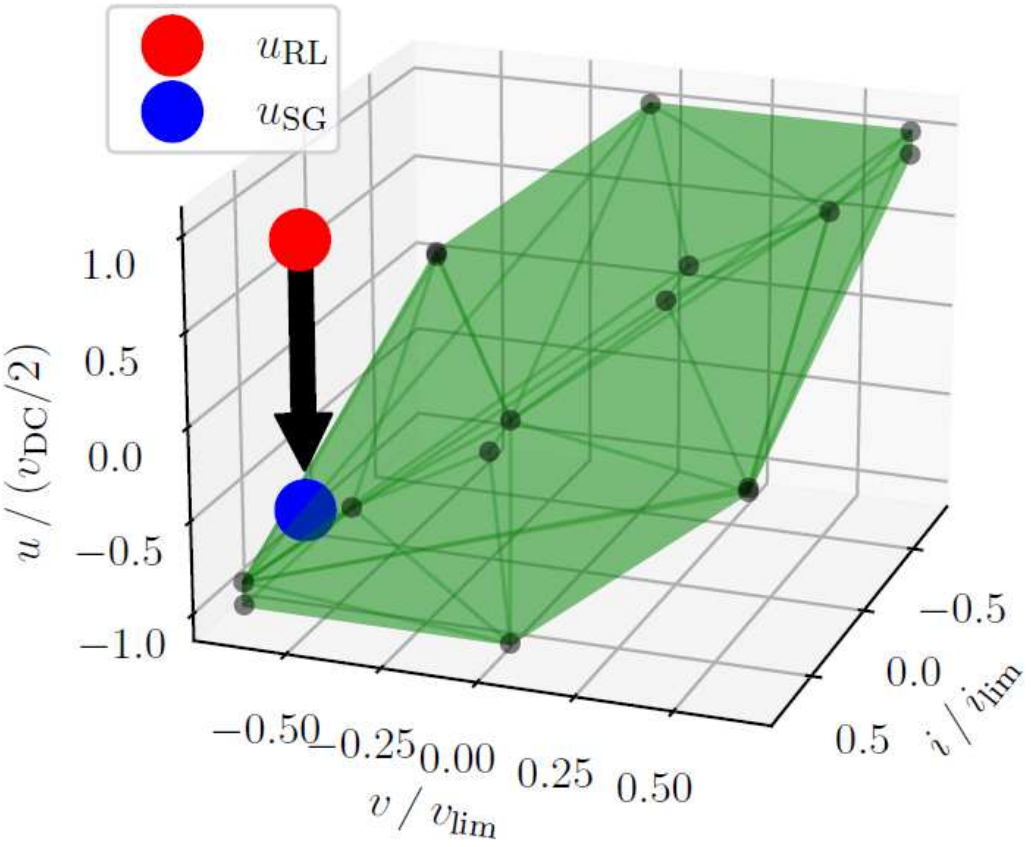


Fig. A.8: Feasible set $\mathcal{F}_0$ for a single phase of the three-phase voltage source inverter[4]

Lists

List of Tables

7.1 Considered system parameters and operating values. 44

List of Figures

2.1 Hierarchical control levels in microgrids[5] 3
2.2 Three-phase four-wire voltage source inverter connected to a load via LC filter. 4
2.3 The Park-Clarke Transformation. 6
2.4 Overview of the interconnections between the different parts of the OMG toolbox. The OpenModelica and OpenAI Gym logos are the property of their respective owners. 6
2.5 Schematic view of example grid in OpenModelica. 7

3.1 The agent-environment interaction in a Markov decision process 9
3.2 Simplified control diagram integrating DDPG with 3-phase Voltage Source Inverter 16

4.1 State feedback model predictive controller. 17
4.2 Receding horizon idea (Derivative of the work from Martin Behrendt licensed under CC BY-SA 3.0) 18
4.3 Model predictive controller combined with HCXA to perform reference tracking in OMG 21

5.1 Flow Chart of histogram-based count maximization algorithm. The red and green line from decision block represents *True* and *False* respectively . 23
5.2 An example case of HCXA discretizing a 2D state space using 5 bins and 4 bins in x_0, x_1 axis respectively. The subplots on diagonal axis are 1D representation of state-space along its respective axis. On the bottom left corner is the scatter plot of points sampled in the state space and on top right corner is discretized state space shown using 2D histogram. 24

5.3 An exemplary case where state space is 4D. Similar to 2D case, bottom triangle of 4x4 matrix is a scatter plot shown in 2D and upper triangle is its 2D histogram representation. The diagonal subplots represent the count when seen in 1D 25
5.4 Computational time to receive next optimal point in state space as a function of size of coverage data over 10 iterations. 26
5.5 10 points sampled from 3 different algorithms. The top group of scatter plots shows the co-ordinates of points in 2D space along with total computation time. The box plot at the bottom shows the uniformity measure of 5 trials using HCXA, DESSCA and Halton sequence 28
5.6 100 points sampled using HCXA, DESSCA and Halton sequence 29
5.7 1000 points sampled using HCXA, DESSCA and Halton sequence 30
5.8 10 points sampled using HCXA in 2D state space with varying number of bins 31
5.9 100 points sampled using HCXA in 2D state space with varying number of bins 32
5.10 400 points sampled using HCXA in 2D state space with varying number of bins 33
5.11 Two types of exploration 34
5.12 Frame 1. The agent is guided by MPC to reach the goal(orange). The subplot on left represents the points visited by the agent. Blue points are state already visited. The subplot on right is a 2D histogram of the discretized state space using 5 bins in x and y axis 35
5.13 Frame 2 36
5.14 Frame 3 36
5.15 Frame 4 37
5.16 Frame 5 37

6.1 Enhanced RL agent interacting with an electrical power grid application. . 38
6.2 Exponential Curves with different slopes. The curve in violet decays must faster than blue curve 39
6.3 The blue curve shows the value of ϵ changing with respect to learning step k. The red line represents the outcome of condition (6.2) if *false* and green line represents the outcome in case of *true*. The grey line indicates the state of the switch at that learning step k. 40
6.4 Calculation of reward based on whether MPC or DDPG is active. The subplot on top shows modulation index realized(solid) vs what they should be(dashed). The second subplot show the reward(grey) and whether MPC is active 42

7.1 Exemplary power grid load curve used for validation 43
7.2 Averaged mean reward and one standard deviation per learning step of the 5 trained agents. A moving average with window size of 1000 steps was a applied to the averaged mean. 45
7.3 Beginning of the testcase with DDPG agent 46

7.4 Averaged mean reward and one standard deviation per learning step of the 5 trained enhanced agents. A moving average with window size of 1000 steps was a applied to the averaged mean. 47
7.5 Training episode 1, k =2800 steps 48
7.6 Training episode 2, k =5600 steps 49
7.7 Reward over 100K steps 50
7.8 Testing episode 1, k =20000 steps 51
7.9 Exemplary loadsteps. Upgraded RL agent on left and basic RL agent on right 52

A.1 Frame 6 54
A.2 Frame 7 55
A.3 Frame 8 55
A.4 Frame 9 56
A.5 Frame 10 56
A.6 Frame 11 57
A.7 Frame 12 57
A.8 Feasible set $\mathcal{F}_0$ for a single phase of the three-phase voltage source inverter[4] 59

Acronyms

AC actor-critic

ANN artificial neural network

CFTOC constrained finite-time optimal control

DDPG deep deterministic policy gradient

DESSCA density estimation-based state-space coverage acceleration

DG distributed generators

DP dynamic programming

DQN deep Q networks

DRL deep reinforcement learning

ES exploring starts

HCXA histogram-based count maximization algorithm

KDE kernel density estimator

LTI linear time invariant

MC Markov chain

MDP Markov decision process

MG microgrid

ML machine learning

MPC model predictive control

OMG OpenModelica Microgrid Gym

OU ornstein–uhlenbeck

QP quadratic programming

RES renewable energy sources

RL reinforcement learning

RLS recursive least squares

TD temporal difference

TD3 twin-delayed deep deterministic policy gradient

VSI voltage source inverter

References

[1] D. Weber, M. Schenke, and O. Wallscheid, *Steady-state error compensation in reference tracking and disturbance rejection problems for reinforcement learning-based control*, 2022. arXiv: 2201.13331 [eess.SY].

[2] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, *Continuous control with deep reinforcement learning*, 2019. arXiv: 1509.02971 [cs.LG].

[3] J. Rawlings, D. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2017. [Online]. Available: https://books.google.de/books?id=MrJctAEACAAJ.

[4] D. Weber, M. Schenke, and O. Wallscheid, "Safe reinforcement learning-based control in power electronic systems," in *2023 International Conference on Future Energy Solutions (FES)*, 2023, pp. 1–6.

[5] "Hierarchical microgrid control," in *Microgrid Dynamics and Control*. John Wiley and Sons Ltd, 2017, ch. 5, pp. 221–265. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119263739.ch5.

[6] D. Weber, S. Heid, H. Bode, J. H. Lange, E. Hüllermeier, and O. Wallscheid, "Safe bayesian optimization for data-driven power electronics control design in microgrids: From simulations to real-world experiments," *IEEE Access*, vol. 9, pp. 35 654–35 669, 2021.

[7] R. R. Deshmukh, M. S. Ballal, and H. M. Suryawanshi, "A fuzzy logic based supervisory control for power management in multibus dc microgrid," *IEEE Transactions on Industry Applications*, vol. 56, no. 6, pp. 6174–6185, 2020.

[8] P. García-Triviño, J. P. Torreglosa, L. M. Fernández-Ramírez, and F. Jurado, "Decentralized fuzzy logic control of microgrid for electric vehicle charging station," *IEEE Journal of Emerging and Selected Topics in Power Electronics*, vol. 6, no. 2, pp. 726–737, 2018.

[9] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html.

[10] R. Bellman, *Dynamic Programming*, 1st ed. Princeton, NJ, USA: Princeton University Press, 1957.

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

[12] S. Fujimoto, H. van Hoof, and D. Meger, *Addressing function approximation error in actor-critic methods*, 2018. arXiv: 1802.09477 [cs.AI].

[13] D. Silver, G. Lever, N. M. O. Heess, T. Degris, D. Wierstra, and M. A. Riedmiller, "Deterministic policy gradient algorithms," in *International Conference on Machine Learning*, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:13928442.

[14] L. UPB, *Reinforcement learning course*. [Online]. Available: https://github.com/upb-lea/reinforcement_learning_course_materials.

[15] F. Borrelli, A. Bemporad, and M. Morari, *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, 2017.

[16] D. O. Wallschied, *Model predictive control - basics and a quadratic programming solution*, Advanced Control, University Paderborn, Paderborn, 2021.

[17] J. Andersson, J. Gillis, J. E. Horn, A. Jameson, C. Kirches, and et al., *Casadi: A symbolic package for automatic differentiation and optimal control*, https://web.casadi.org, Accessed: August 13, 2023, 2021.

[18] A. Wächter and L. T. Biegler, "On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006.

[19] S. Lucia, A. Tatulea-Codrean, C. Schoppmeyer, and S. Engell, "Rapid development of modular and sustainable nonlinear model predictive control solutions," *Control Engineering Practice*, vol. 60, pp. 51–62, Mar. 2017.

[20] S. Thrun, "Efficient exploration in reinforcement learning.," Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-CS-92-102, Jan. 1992.

[21] C. Wang and K. W. Ross, "On the convergence of the monte carlo exploring starts algorithm for reinforcement learning," *CoRR*, vol. abs/2002.03585, 2020. arXiv: 2002.03585. [Online]. Available: https://arxiv.org/abs/2002.03585.

[22] M. Schenke and O. Wallscheid, "Improved exploring starts by kernel density estimation-based state-space coverage acceleration in reinforcement learning," *CoRR*,

vol. abs/2105.08990, 2021. arXiv: 2105.08990. [Online]. Available: https://arxiv.org/abs/2105.08990.

[23] L. Kuipers and H. Niederreiter, *Uniform Distribution of Sequences* (Dover Books on Mathematics). Dover Publications, 2012. [Online]. Available: https://books.google.de/books?id=mnY8LpyXHM0C.

[24] Wikipedia, *Low-discrepancy sequence*, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Low-discrepancy_sequence#Definition_of_discrepancy.

[25] P. V. *et al*, "SciPy 1.0: Fundamental algorithms for scientific computing in python," *Nature Methods*, vol. 17, no. 3, pp. 261–272, 2020. [Online]. Available: https://doi.org/10.1038%2Fs41592-019-0686-2.

[26] W. contributors, *Halton sequences — wikipedia, the free encyclopedia*, https://en.wikipedia.org/wiki/Halton_sequence.

[27] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, *Stable baselines*, https://github.com/hill-a/stable-baselines, 2018.

[28] A. Cahill, "Catastrophic forgetting in reinforcement-learning environments," Master of Science thesis, University of Otago, 2011. [Online]. Available: http://hdl.handle.net/10523/1765.

[29] J. Lange, D. Schmies, K. S. Stille, J. Böcker, and O. Wallscheid, "Experimental comparison of fpga-implemented model predictive voltage control to cascaded proportional resonant control for a three-phase four-wire three-level grid-forming inverter of 250 kva," in *2022 24th European Conference on Power Electronics and Applications (EPE'22 ECCE Europe)*, 2022, pp. 1–9.

[30] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.