CHALMERS
UNIVERSITY OF TECHNOLOGY

Energy Management in Grid-Connected Photovoltaic Battery Systems Using Degradation-Aware Model Predictive Control-Based Reinforcement Learning

Master's thesis in Systems, Control and Mechatronics

IDA HÄGERYD

STJEPANA SEVER

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's thesis 2025

Energy Management in Grid-Connected Photovoltaic Battery Systems Using Degradation-Aware Model Predictive Control-Based Reinforcement Learning

IDA HÄGERYD

STJEPANA SEVER

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2025

Energy Management in Grid-Connected Photovoltaic Battery Systems
Using Degradation-Aware Model Predictive Control-Based Reinforcement Learning
IDA HÄGERYD
STJEPANA SEVER

Supervisor at Volvo: Huang Zhang, Industrial PhD at Volvo Trucks, Systems and Control
Supervisor at Chalmers: Evelina Wikner, Electric Power Engineering
Examiner: Torsten Wik, Systems and Control

Master's Thesis 2025
Department of Electrical Engineering Division of systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Solar panels on a house

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2025

Energy Management in Grid-Connected Photovoltaic Battery Systems
Using Degradation-Aware Model Predictive Control-Based Reinforcement Learning
IDA HÄGERYD
STJEPANA SEVER
Department of Electrical Engineering
Chalmers University of Technology

Abstract

This thesis presents a degradation-aware Model Predictive Control–based Reinforcement Learning (MPC-RL) framework for optimal energy management in residential grid-connected photovoltaic (PV) battery systems. The goal is to minimize total operating costs while accounting for battery degradation, grid price variability, and forecast uncertainty. The MPC formulation incorporates a parametrized stage cost that includes grid energy costs, battery degradation, and state-of-charge (SoC) penalty window, with parameters tuned using a Q-learning algorithm. Gaussian Process Regression (GPR) is used to forecast the net residual power (PV production minus demand) based on temporal and historical features, enabling a realistic simulation environment and improved MPC decision-making. The proposed method is benchmarked against a baseline non-parametrized MPC controller, purely RL-based strategies from literature, and a simple Gaussian residual model. Results show that while the proposed MPC-RL approach does not outperform all RL benchmarks in short-term simulations, it achieves competitive performance with reduced degradation and improved constraint handling. GPR-based residual forecasts outperform the Gaussian baseline, reducing total monthly costs by up to 2.2 % compared to the random-sampling approach, and closely tracking actual residual trends. These findings suggest that MPC-RL with GPR prediction offers a promising and degradation-conscious approach for residential microgrid energy management.

Keywords: Economic Model Predictive Control, Reinforcement Learning, Photovoltaic, Battery Degradation

Acknowledgments

We would like to express our sincere gratitude to our thesis supervisor, Huang Zhang, for his continuous support and sharing of his expertise in battery systems, reinforcement learning, and Gaussian Process Regression.

We are also very grateful to our supervisor at Chalmers, Evelina Wikner, and our examiner, Torsten Wik, for their valuable insights and constructive feedback.

Our appreciation extends to Olle Friberg, our team manager at Volvo Trucks, and the entire BMS team, for warmly welcoming us into the group.

Finally, we would like to thank our friends and families for their love and support.

Ida Hägeryd and Stjepana Sever, Gothenburg, August 2025

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

BESS	Battery Energy Storage System
DoD	Depth of Discharge
EFC	Equivalent Full Cycle
EMPC	Economic Model Predictive Control
GPR	Gaussian Process Regression
MDP	Markov Decision Process
MG	Microgrid
MPC	Model Predictive Control
PV	Photovoltaic
RL	Reinforcement Learning
RMSE	Root Mean Squared Error
RQ	Rational Quadratic (kernel)
SoC	State of Charge

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Indices

k	Discrete time step index in MPC horizon
π	Control policy index
θ	Index indicating adjustable MPC parameters
g	Grid cost index
b	Battery degradation cost index
s	SoC penalty window index

Parameters

α	Battery replacement threshold (in percentage of the battery rated capacity)
β_{total}	Total battery degradation
β_{cal}	Calendar aging component
β_{cyc}	Cyclic aging component
c	SoC penalty weight
c_b^i	Initial battery unit cost
c_b^r	Replacement battery unit cost
E_b	Battery rated capacity (kWh)
h	Discretization time step (h)
$k_\delta, k_\kappa, k_t, k_T$	Coefficients in battery stress factor models
l_{cal}	Calendar life (years)
l_{cyc}	Cycle life (EFC)

$P_{\text{grid}}^{\text{lim}}$	Maximum grid charge/discharge power (kW)
$SoC_{\max}$	Upper limit of the SoC penalty window
$SoC_{\min}$	Lower limit of the SoC penalty window
σ_{buy}	Electricity buying price
σ_{sell}	Electricity selling price
η	Battery round-trip efficiency
θ_a	Parameter for state transition matrix
θ_b	Parameter for input and residual
θ_d	Parameter for grid cost component in stage cost
θ_e	Parameter for degradation cost component in stage cost
θ_f	Parameter for SoC penalty cost component in stage cost
θ_T	Parameter for terminal cost
θ_0	Parameter for initial cost term λ_θ
ξ_k	Slack variable parameter for the state at timestep k
ξ_N	Slack variable parameter for the terminal state

Variables

a	Action taken by the MPC agent
x	Battery SoC
u	Energy exchanged with the grid (normalized)
u_{lim}	Grid exchange limit (normalized)
w	Net residual power (normalized)
P_{grid}	Grid power flow (kW)
P_{PV}	PV generation (kW)
P_{load}	Load demand (kW)
P_b	Battery charge/discharge power (kW)
r	Rewards to agent
V_f	Terminal cost
L_g	Grid cost
L_b	Battery degradation cost
L_s	SoC penalty cost
T	Operating cell temperature

T_{ref}	Reference cell temperature
κ	SoC in equations for the semi-empirical degradation model
C	Loss of usable battery capacity

Contents

1
Introduction

The growing use of photovoltaic (PV) systems in households presents both opportunities and challenges in energy management. Using solar energy and battery energy storage systems is a crucial step toward reducing reliance on fossil fuels for electricity production. However, one major challenge is optimizing grid transactions to maximize economic benefits. Key factors influencing this include fluctuating grid prices and the costs associated with battery degradation [1]. These battery costs encompass both the initial investment and potential replacement expenses as performance declines over time. By incorporating battery degradation into advanced control strategies, the lifespan of batteries can be extended; this reduces the frequency of battery production and replacement, thereby minimizing the environmental impact associated with manufacturing and disposal.

A promising approach to enhance economic efficiency is Model Predictive Control-based Reinforcement Learning (MPC-based RL), which combines MPC with RL to make informed, data-driven decisions using historical data [2].

MPC and RL are fundamentally different approaches, each with its own advantages and limitations. MPC is a control strategy that relies on a system model and a cost function to optimize a given objective while handling constraints. One major drawback of MPC is its dependence on an accurate model to represent the real system. Even when the model initially reflects the system well, changes over time can lead to performance degradation [3]. In contrast, RL learns decision-making strategies by interacting with the environment. Through trial and error using rewards or penalties, the agent gradually refines its policy to maximize cumulative rewards. However, a key challenge with RL is its high demand for data, as learning through interaction often requires a large number of episodes or iterations before the agent converges to a useful policy. This data inefficiency can make learning both slow and computationally intensive. Achieving efficient and robust learning remains an active and challenging area of research within the field of RL [4].

To address the limitations of both methods, a hybrid approach known as MPC-based RL can be employed. In this framework, a parameterized MPC acts as a function approximator for the optimal policy. An RL algorithm is then used to adjust these parameters based on a performance function, allowing the system to adapt and improve over time while benefiting from the structure and constraint-handling capabilities of MPC [5].

In [1], an economically driven control strategy is proposed for grid-connected PV-battery systems that use batteries previously used in electric vehicles as stationary storage. The approach introduces a stage cost function that incorporates both grid energy costs and battery degradation and learns the optimal policy. However, there is no sure way to guarantee policy convergence or stability for the implicit policy in this work, or that the necessary state and input constraints are hold in every step. In [6], an energy management strategy was proposed for residential microgrids using an MPC-based RL approach to minimize collective electricity costs under uncertainty. Equality and inequality constraints are imposed on the state and the value and action value functions are approximated by solving a constrained MPC problem in every step. The stage cost function used for tuning the parameters, however, does not consider battery degradation, which can negatively impact the long-term economic performance of the PV-battery system. Both works also model the residual production-consumption difference as simple Gaussian noise, without considering, for example, effect of weather or time of day.

This thesis explores a degradation-aware, MPC-based Reinforcement Learning (RL) approach to address the control problem of determining the optimal amount of electricity to buy from or sell to the grid. The primary objective is to minimize economic costs while maintaining system reliability and responsiveness to dynamic environmental and market conditions, all while accounting for battery degradation. A similar approach to [6] is adopted to tune the MPC parameters using a degradation-aware stage cost function, as formulated in [1]. Furthermore, Gaussian Process Regression (GPR) is used to predict user and production residuals, which are integrated into the MPC forecasts to improve predictive accuracy.

1.1 Aim and research questions

The aim of this project is to develop and evaluate an MPC-based RL approach for optimizing the management of residential PV-battery systems, specifically focusing on minimizing economic costs and battery degradation. By combining the predictive capabilities of MPC with the adaptive decision-making strengths of RL, this project seeks to improve energy trading decisions, balancing the purchase and sale of electricity with grid fluctuations and battery constraints. Additionally, the project will explore the application of GPR to enhance the accuracy and efficiency of production and consumption data modeling, providing a more robust framework for real-time decision-making in changing energy markets. The goal is to enhance the long-term economic and operational performance of residential microgrids, accounting for both environmental factors and battery degradation. The following research questions will be answered during the course of this thesis:

1. What is the impact of different parameterizations of the MPC model and cost function on the performance of the reinforcement learning algorithm in terms

of policy optimization and cost reduction?
2. How does the performance of an MPC-based RL approach, trained on historical data, compare to traditional RL-based control in reducing operational costs and battery degradation during simulations?
3. How can Gaussian Process Regression be applied to model production and consumption data in an MPC-based reinforcement learning framework, and what impact does it have on the accuracy and efficiency in predicting system behavior over time?

1.2 Thesis outline

To address these research questions, an MPC-based RL is developed for the system dynamics described in Chapter 2 using the methods outlined in Chapter 3. Various settings of MPC parameterization are evaluated and compared against a baseline, non-parameterized MPC approach in Chapter 4. These configurations are tested both with and without Gaussian Process Regression (GPR) predictions; in the absence of GPR, sampling from a normal distribution based on the available data is used as a substitute. The resulting energy trading strategies are assessed in terms of cost and battery degradation. Finally, the performance of the proposed MPC-based RL methods is compared against a reinforcement learning-only approach developed in [1], providing insights into how different control strategies impact operational efficiency and battery degradation. Degradation evaluation is conducted in two ways: using the same approach as in [1], allowing for fair comparison between the methods, and with the help of a rainflow counting method described in Section 2.2.2, which provides a more accurate evaluation. Eventually, the findings are discussed in Chapter 5.

2
Theory

In this chapter, the theory necessary to understand the problem and the methods in Chapter 3 is laid out, such as MPC and Q-learning. System dynamics and the parametrization of the system are also presented and explained here, as well as the GPR prediction method.

2.1 System dynamics

A typical residential grid-connected PV system is comprised of a PV generation unit and a battery energy storage system (BESS). When PV generation exceeds household demand, the extra energy is either stored in the battery or exported to the grid. When demand surpasses PV output the shortfall is supplied by the battery or, if necessary, drawn from the grid. The dynamic behavior of the PV-battery system is governed by the evolution of the battery's state of charge (SoC), described as follows [1] [6]:

$$\frac{d\text{SoC}}{dt} = \frac{1}{E_b}(P_{\text{grid}} + P_{\text{PV}} - P_{\text{load}}) \cdot \begin{cases} \eta, & \text{if } P_{\text{grid}} + P_{\text{PV}} - P_{\text{load}} > 0, \\ \frac{1}{\eta}, & \text{otherwise.} \end{cases} \tag{2.1}$$

- E_b is the battery rated capacity in kWh
- $P_{\text{grid}} \in [-P_{\text{grid}}^{\text{lim}}, P_{\text{grid}}^{\text{lim}}]$ is the power traded with the grid in kW
- P_{PV} is the power generated by the PV unit
- P_{load} is the household load demand
- η is the battery round-trip efficiency (assumed to be constant)

A positive value of P_{grid} indicates power import from the grid, while a negative value corresponds to power export. Inverter losses are neglected in [1] due to their negligible impact. This simplification is also adopted in this study.

Discretized system dynamics take the form:

$$x(k+1) = x(k) + (u(k) + w(k)) \cdot \begin{cases} \eta, & \text{if } u(k) + w(k) > 0, \\ \frac{1}{\eta}, & \text{otherwise.} \end{cases} \tag{2.2a}$$

with

$$x \triangleq \text{SoC}, \quad u \triangleq \frac{h}{E_b} P_{\text{grid}}, \quad u_{\text{lim}} \triangleq \frac{h}{E_b} P_{\text{grid}}^{\text{lim}}, \tag{2.2b}$$

$$w \triangleq \frac{h}{E_b}(P_{\text{PV}} - P_{\text{load}}), \tag{2.2c}$$

Here, x represents the battery SoC $\in [0, 1]$, and $u \in [-u_{\text{lim}}, u_{\text{lim}}]$ represents the energy exchanged with the grid, normalized by the battery capacity E_b. The term w represents the net residual energy from PV generation and load demand, also normalized with respect to E_b. A schematic of the system is shown in Figure 2.1.

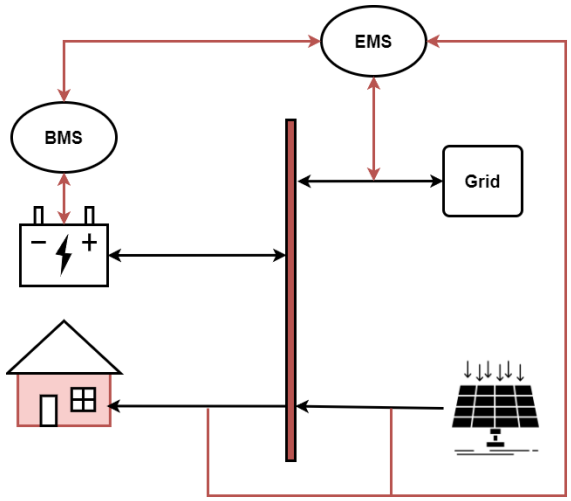

Figure 2.1: Schematic of the PV-battery system. Black arrows represent the flow of electricity, while red arrows indicate the exchange of information between the energy management system and other components.

2.2 Battery degradation

As a key aspect of system modeling, degradation effects must be taken into account. Following subsections describe two different methods used to asses the battery degradation: the energy throughput model and the rainflow cycle-based semi-empirical model.

2.2.1 Energy throughput model

In [1], the battery degradation cost function was given by:

$$L_b(x(k), u(k)) = \frac{0.2 \cdot \beta_{\text{total}}(k)}{1 - \alpha} \cdot (c_i^b + c_r^b) \cdot E_b, \tag{2.3}$$

where α is the replacement threshold factor (which, for less-demanding stationary storage applications is equal to 0.6), c_i^b is the initial unit cost, c_r^b is the replacement cost, and $\beta_{total}(k)$ is the total battery degradation at time step k .

The total battery degradation $\beta_{total}(k)$ is defined as the sum of calendar and cyclic aging. The calendar aging at time step k is estimated as:

$$\beta_{\text{cal}}(k) = \frac{h}{l_{\text{cal}}}, \tag{2.4}$$

where l_{cal} is the number of years until the battery degrades to its end-of-life at a constant storage temperature (without considering the cycling aging). The cycling aging at time step k is given as:

$$\beta_{\text{cyc}}(k) = 0.5 \cdot \frac{|P_b(k)| \cdot h}{l_{\text{cyc}} \cdot E_b}, \tag{2.5}$$

where l_{cyc} is the number of equivalent full cycles until the battery reaches its end-of-life (without considering the calendar aging) and $P_b(k)$ is the charge/discharge power [1].

Even though this model considers both cyclical and calendar aging, it is still fairly simple. It does not consider the inner behavior of the battery cells and the mechanisms that accelerate the degradation such as lithium plating or SEI layer formation, which can be captured with an electrochemical model. It also does not consider the battery capacity knee point, after which the capacity of the battery starts to rapidly decline [1].

2.2.2 Rainflow cycle-based model

Another method for assessing battery degradation is the rainflow cycle-based model, which combines the rainflow counting method together with mechanical fatigue analysis caused by various stress factors.

2.2.2.1 Rainflow counting method

The rainflow counting method is a widely adopted algorithm for fatigue analysis in mechanical and structural systems. It reduces a complex stress–strain or time series signal into a sequence of representative load cycles that are considered to have significant impact on material degradation. Among the several standardized variants of the algorithm, one commonly used implementation is the ASTM E1049-85 three-point rainflow method [7].
Before applying the counting algorithm, the signal is pre-processed through two key filtering steps:

1. Hysteresis filtering: Adjacent peak–valley pairs with a difference below a given threshold are removed. This avoids counting small fluctuations that do not meaningfully contribute to fatigue damage. In the context of battery cycling, this may be less critical depending on the sampling rate and resolution.
2. Peak–valley filtering: The time series is reduced to a sequence of local extrema (peaks and valleys), effectively isolating turning points.

Once filtered, the three-point rainflow algorithm operates as follows [7]:

Given the sequence of extrema $x_0, x_1, x_2, \ldots, x_n$, the algorithm processes points using a double-ended queue that holds the three most recent turning points: x_{i-2}, x_{i-1}, and x_i

Two ranges are calculated:

$$X = |x_i - x_{i-1}|, Y = |x_{i-1} - x_{i-2}|$$

The algorithm then applies the following rules:

1. If $X < Y$: no action is taken; the algorithm reads the next point in the sequence.
2. If $X \geq Y$:
   (a) If the identified cycle is fully enclosed within the series, it is counted as a full cycle with a weight of 1.0.
   (b) If the cycle extends to the beginning or end of the signal, it is counted as a half cycle with a weight of 0.5.
3. After a cycle is counted, the corresponding turning points (those forming Y) are removed from the queue.

This approach enables the decomposition of irregular SoC or stress profiles into standardized cycle representations, which can then be used for fatigue analysis or, in the case of battery applications, for degradation modeling based on cycle characteristics such as depth of discharge and average SoC.

2.2.2.2 Semi-empirical degradation model

In order to model lithium-ion battery aging under realistic operational conditions, this work adopts the semi-empirical degradation model proposed in [8]. This model separates total battery degradation into the two mechanisms mentioned in Section 2.1. Both mechanisms are influenced by stress factors such as temperature (T), depth of discharge (or DoD, marked with δ), state of charge (SoC, marked with κ), and time, and are represented through a unified degradation function.

The degradation sustained during one cycle is expressed as:

$$f_{d,1} = \left[S_\delta(\delta) + S_t(T)\right] S_\kappa(\kappa) S_T(T) \tag{2.6}$$

Here, S_δ captures the influence of discharge depth, S_t accounts for time-related stress during the cycle, while S_κ and S_T represent SoC and temperature stress factors, respectively.

When the battery is not cycled, calendar aging dominates, and the degradation rate simplifies to:

$$f_{d,t} = S_t(t) \cdot S_\kappa(\kappa) \cdot S_T(T) \tag{2.7}$$

The total loss of usable capacity (for batteries with sub 90% of the original battery life remaining) is modeled as an exponential function of the accumulated linearized degradation f_d (the sum of calendar and cycle aging):

$$C = 1 - (1 - C')e^{-f_d} \tag{2.8}$$

The stress functions are defined as follows. The DoD stress factor is empirically modeled to capture the nonlinear increase in degradation with deeper discharges:

$$S_\delta(\delta) = k_{\delta_1} \cdot \delta \cdot \exp\left(\frac{\delta - 1}{k_{\delta_2}}\right) \tag{2.9}$$

where k_{δ_1} and k_{δ_2} are empirical fitting parameters, and δ is the cycle's depth of discharge.

The SoC stress factor is modeled exponentially relative to a reference SoC level:

$$S_\kappa(\kappa) = \exp\left(k_\kappa(\kappa - \kappa_{\text{ref}})\right) \tag{2.10}$$

where k_κ is the SoC stress coefficient and κ_{ref} is the reference SoC at which degradation is considered nominal.

The time-dependent stress factor models calendar aging as a linear function of time:

$$S_t(t) = k_t \cdot t \tag{2.11}$$

with k_t being the degradation rate per unit time.

Finally, the temperature stress factor follows an Arrhenius-type relationship:

$$S_T(T) = \exp\left(k_T(T - T_{\text{ref}})\right) \cdot \frac{T_{\text{ref}}}{T} \tag{2.12}$$

where k_T is the temperature stress coefficient, T is the operating cell temperature, and T_{ref} is the reference temperature (typically 25°C or 298K).

2.3 Model Predictive Control (MPC)

Model predictive control is class of control algorithms that, with the help of a system model, predict the output behavior over a defined prediction horizon. An optimization problem consisting of a cost function along with state and/ or input constraints is solved in order to obtain a trajectory of control inputs of which only the first, u_0, is applied to the real system. The remaining calculated control inputs are then discarded before the optimization starts over for the next time step. This approach follows a receding horizon strategy, so the horizon is shifted further into the future for every time step [9].

The method has a long history of application, particularly in the chemical process industry, where it was originally developed for power plants and petroleum refineries [10]. Over time, it has been adopted across various other sectors, including automotive, food processing, and aerospace industries. Due to its ability to incorporate constraints on states and inputs, MPC is especially well-suited for systems where safety is important.

2.3.1 Classical formulation and economic MPC

Central to MPC is the optimization problem of minimizing the cost function. The cost function to be minimized can be defined in different ways depending on the goal of the control implementation. In conventional MPC, where the objective is to find and hold the system around a steady-state or an optimal set-point, the cost

function is usually quadratic to achieve a faster control response by penalizing the deviation from the steady-state. The MPC problem is usually formulated as:

$$\min_{x,u} \quad \sum_{k=0}^{N-1} L(x_k, u_k) + V_f(x_N), \tag{2.13a}$$

$$\text{s.t.} \quad x_{k+1} = f(x_k, u_k), \quad \forall k = 0, \ldots, N-1, \tag{2.13b}$$

$$g(u_k) \leq 0, \tag{2.13c}$$

$$x_0 = s \tag{2.13d}$$

with L being the cost function, V_f being the terminal cost and g and h being the inequality constraints on the states and inputs. Terminal state x_N must belong to the terminal set. The quadratic objective function usually has the following form:

$$L = x_k^T Q x_k + u_k^T R u_k \tag{2.14}$$

with Q and R being symmetric calibration matrices that are fine-tuned to get the desired controller behavior [9].

In Economic Model Predictive Control (EMPC), the cost function is designed with an economic focus. Unlike traditional approaches, where the objective function is typically quadratic, EMPC accounts for economic considerations where, for example, positive deviations from a target may result in increased profits. As a result, the objective function cannot be formulated solely as a simple quadratic function but must instead balance economic gains and losses [11].

2.4 Reinforcement Learning (RL)

Reinforcement Learning (RL) is a machine learning technique based on trial and error, where an agent interacts with an environment by taking actions and receiving feedback in the form of rewards. The goal of the agent is to minimize or maximize cumulative rewards over time based on the problem formulation [12].

2.4.1 Markov Decision Processes (MDPs) and Reinforcement Learning

Many reinforcement learning (RL) methods operate within environments that can be modeled as Markov Decision Processes (MDPs). An MDP is a mathematical framework for decision-making where outcomes depend on both the agent's actions and inherent randomness. MDPs are characterized by the following components:

- S: A set of states representing the possible situations within the environment. These states can be discrete or continuous.
- A: A set of actions available to the agent from any given state.
- $P_a(s, s')$: The transition probability, which represents the likelihood of moving from state s to state s' when taking action a.
- $R_a(s, s')$: The immediate reward received after transitioning from s to s' via action a.
- $p_1(s_1)$: The initial state distribution, defining the starting conditions of the environment.

MDPs follow the Markov Property, which states that the next state depends solely on the current state and action, not on the history of prior states or actions [12]:

$$P(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_1, a_1) = P(s_{t+1} \mid s_t, a_t) \tag{2.15}$$

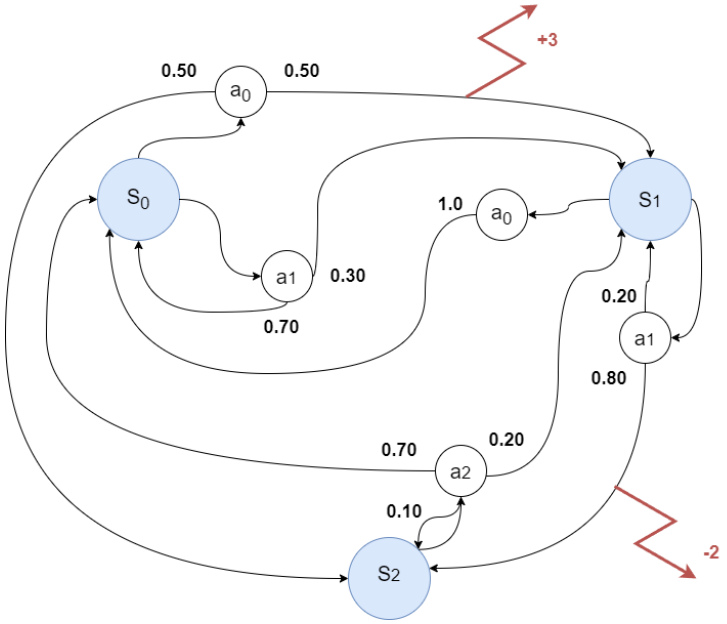

Figure 2.2: An illustrative example showing how an MDP can be structured with $\mathcal{S}$, $\mathcal{A}$, $P_a(s, s')$ and $R_a(s, s')$.

2.4.2 Value functions in MDPs

In RL, many algorithms for MDPs aim to determine optimal policies by estimating value functions. A value function indicates how beneficial it is for an agent to be in a particular state or take a specific action from that state. These value functions are defined relative to specific policies. A policy is a function that, for every state (s) in the state set S computes an action (a) from the action set A.
The state-value function under a policy π is defined as

$$V_\pi(s) = L(s, \pi(s)) + \gamma \mathbb{E}[V_\pi(s^+) \mid s, \pi], \tag{2.16}$$

where $L(x, \pi(s))$ is the stage cost for RL and γ is the discount factor that decides how much weight is being put on immediate reward versus reward further in the future [12]. The action-value function is in turn defined as

$$Q_\pi(s, u) = L(s, u) + \gamma \mathbb{E}[V_\pi(s^+) \mid s, u], \tag{2.17}$$

where s^+ is the next state when taking action a.

2.4.3 Optimal value and policy

The goal in RL is to find the optimal policy π^* that maximizes/minimizes the expected cumulative rewards/penalties. The optimal action-value and state-value functions are given by the Bellman equations:

$$Q^*(s, a) = L(s, a) + \gamma \mathbb{E}[V^*(s') \mid s, a] \tag{2.18}$$

$$V^*(s) = \min_a Q^*(s, a) = Q^*(s, \pi^*(s)). \tag{2.19}$$

The optimal state-value function can also be expressed as

$$V^*(x) = \min_{\pi} V_\pi(x). \tag{2.20}$$

Similarly, the optimal action-value function is

$$Q^*(x,u) = \min_{\pi} Q_\pi(x,u) = Q_{\pi^*}(x,u). \tag{2.21}$$

The optimal policy is derived as

$$\pi^*(x) = \arg\min_{u} Q^*(x,u). \tag{2.22}$$

These Bellman equations are fundamental in RL as they establish a recursive relationship between the value of a state-action pair and the value of the next state. They guide the computation of optimal policies [2]. Note that Q^* and V^* are defined as maximization functions when the goal of the RL is to maximize the reward. In this thesis, formulating them as minimizing functions is necessary since the goal of RL is cost minimization.

One common method to update these functions is Q-learning, which iterates the value functions to converge to the optimal policy.

2.4.4 Q-learning

One of the most popular RL algorithms is Q-learning. Q-learning is from a family of temporal difference (TD) RL methods, i.e. it updates the the value/the action-value function based on the difference between estimates at two different times, $Q(s') - Q(s)$.

The Q-function estimates the cumulative expected reward from taking a specific action in a given state and the general update rule is then usually formulated as [12]:

$$\begin{aligned} Q^+(s,a) &= Q(s,a) + \alpha[R(s,a) + \gamma \min_a Q(s^+,a) - Q(s,a)] \\ &= Q(s,a) + \alpha[R(s,a) + \gamma V(s^+) - Q(s,a)) \end{aligned} \tag{2.23}$$

with α being the learning rate that controls how much newly acquired information overrides old information, and it typically decreases over time to ensure convergence [12].

2.4.5 Gradient descent for action-value function Approximation

In RL problems with large or continuous state and action spaces, it is often impractical to represent the action-value function $Q_\pi(s,a)$ exactly. Instead, we approximate it using a parameterized function $\hat{Q}(s,a,\theta)$, where θ is a vector of tunable parameters. The goal is to find parameter values that make the approximation as close as possible to the true action-value function under policy π.

This objective can be formalized by minimizing the expected squared error between the true value and the estimate:

$$J(\theta) = \mathbb{E}_\pi \left[\left(Q_\pi(s,a) - \hat{Q}(s,a,\theta) \right)^2 \right]. \tag{2.24}$$

To minimize this loss, we apply gradient descent. At each step, the parameters are updated in the direction that reduces the error:

$$\theta_{k+1} = \theta_k + \alpha \left(Q_\pi(s,a) - \hat{Q}(s,a,\theta_k) \right) \nabla_\theta \hat{Q}(s,a,\theta_k), \tag{2.25}$$

where α is the learning rate, and $\nabla_\theta \hat{Q}(s,a,\theta_k)$ is the gradient of the function approximation w.r.t its parameters.
In practice, the true value $Q_\pi(s,a)$ is typically unknown and must be estimated from experience. When used in TD learning, the update instead becomes [13]:

$$\theta_{k+1} = \theta_k + \alpha\, \delta_k\, \nabla_\theta \hat{Q}(s_k, a_k, \theta_k), \tag{2.26}$$

where δ_k is the TD error, defined by:

$$\delta_k = R_{k+1} + \gamma \hat{Q}(s_{k+1}, a_{k+1}, \theta_k) - \hat{Q}(s_k, a_k, \theta_k). \tag{2.27}$$

2.5 MPC-based RL

MPC-based RL combines the strengths of MPC and RL to enhance control performance in uncertain environments. While MPC can handle constraints and ensures stability, it relies on accurate system models. On the other hand, RL excels at learning from interactions but can struggle with stability and constraint satisfaction.

The main idea of MPC-based RL is to use the MPC model as a value function approximation, which is continuously tuned to improve control performance. Unlike classic MPC formulations, the MPC model here is parameterized, introducing parameters θ into the objective function and/or constraints. The resulting parameterized MPC model can be formulated as follows

$$V_\theta(s) = \min_{u,x,\xi} \lambda_\theta(x_0) + \gamma^N V_f^\theta(x_N) + \sum_{k=0}^{N-1} \gamma^k \left(L_\theta(x_k, u_k) + w^T \xi_k \right) \tag{2.28a}$$

$$\begin{aligned} \text{subject to:} \quad & x_{k+1} = f_\theta(x_k, u_k), && \text{(2.28b)} \\ & h_\theta(x_k, u_k) \leq \xi_k, && \text{(2.28c)} \\ & x_0 = s && \text{(2.28d)} \end{aligned}$$

where λ_θ is the extra cost (which can be set to 0 but helps in solving the optimization problem if the objective function is not convex), and ξ_k being the relaxation/slack variable, which is often included in the EMPC formulations. For large enough w,

the solution is identical to the one if there is a feasible trajectory for the initial state [2]. Including the slack variable is going to be important for the RL process, since it will prevent the cost from being infinitely high.

With Q-learning, the parameters are fine-tuned to get the best value function approximation.
The policy is defined as:

$$\pi(s) = u_0^*, \tag{2.29}$$

where u_0^* is the first element of the input sequence $u_0^*, \ldots, u_{N-1}^*$ obtained as the solution of the MPC problem for a given state s. Based on the new parametrized MPC model, the state-action function can then be defined as:

$$Q_\theta(s,a) = \min_{u,x} \lambda_\theta(x_0) + \gamma^N V_f^\theta(x_N) + \sum_{k=0}^{N-1} \gamma^k \left(L_\theta(x_k, u_k) + w^T \xi_k \right) \tag{2.30a}$$

$$\begin{aligned}
\text{subject to:} \quad & x_{k+1} = f_\theta(x_k, u_k), && \text{(2.30b)}\\
& h_\theta(u_k, x_k) \leq \xi_k, && \text{(2.30c)}\\
& h_\theta(u_N, x_N) \leq \xi_N, && \text{(2.30d)}\\
& x_0 = s, && \text{(2.30e)}\\
& u_0 = a, && \text{(2.30f)}
\end{aligned}$$

Notice that the only difference between the parametrized value and the action-value function is the constraint on the initial input u_0. The parameterization satisfies the equalities underlying the Bellman equations:

$$\pi_\theta(s) = \arg\min_a Q_\theta(s,a), \quad V_\theta(s) = \min_a Q_\theta(s,a). \tag{2.31}$$

It is also assumed that the following equalities for converged θ hold:

$$\pi_\theta = \pi^*, V_\theta = V^*, Q_\theta = Q^* \tag{2.32}$$

It is important to note that these equalities do not hold in general, but with the right tuning of the stage cost L_θ, terminal cost V_f^θ and constraint h_θ, this equality can be achieved [2].

2.5.1 Q-learning update step for MPC-based RL

To solve the optimization problem in 2.5, we need to define the Lagrangian:

$$\begin{aligned}\mathcal{L}_\theta(s,y) = \lambda_\theta(x_0) + \gamma^N V_f^\theta(x_N) + \sum_{k=0}^{N-1} \gamma^k L_\theta(x_k,u_k) \\ + \chi_0^\top (x_0 - s) + \zeta^\top (u_0 - a) \\ + \sum_{k=0}^{N-1} \chi_{k+1}^\top (f_\theta(x_k,u_k) - x_{k+1}) \\ + \sum_{k=0}^{N-1} \mu_k^\top (h_\theta(x_k,u_k) + \xi_k)\end{aligned} \tag{2.33}$$

where χ_0 is the multiplier for the initial state constraint x_0, ζ is the multiplier for the fixed initial action $u_0 = a$, χ_{k+1} are the multipliers for the system dynamics constraints, μ_k are the multipliers for the inequality constraints and $y = (x, u, \chi, \mu, \zeta)$ is the set of primal-dual variables.

We can then also conclude that:

$$\nabla_\theta Q_\theta(s,a) = \nabla_\theta \mathcal{L}_\theta(s, y^\star) \tag{2.34}$$

holds for $y^\star$ given by the primal-dual solution of 2.33. The gradient calculation is therefore straightforward since it is a by-product of solving the MPC problem.
From Section 2.4.5 we know that the gradient descent method can be used to update the parameters in the action-value function. The update rule for the RL is then given by:

$$\theta \leftarrow \theta + \alpha \delta_k \nabla_\theta Q_\theta(s_k, a_k). \tag{2.35}$$

where $\nabla_\theta Q_\theta(s_k, a_k)$ is the sensitivity of the action-value function calculated as a bi-product of solving the constrained MPC problem, as seen in the previous section.

Because of the equalities established in Equation 2.31, δ then looks like:

$$\delta_k = L_\theta(s_k, a_k) + \gamma V_\theta(s_{k+1}) - Q_\theta(s_k, a_k) \tag{2.36}$$

with L_θ in this case being equal to:

$$L_\theta(s_k, a_k) = L_\theta(s_k, a_k) + w^T \max(h_\theta(s_k, a_k), 0) \tag{2.37}$$

2.5.2 Stage cost for MPC and RL

The stage cost function used in this thesis, and introduced in [1], is comprised of three main parts:

1) **Grid Cost Function:** The part of the cost function that captures the cost or revenue associated with buying from or selling electricity to the grid at time-step k is given by:

$$L_g(u(k)) = \begin{cases} \sigma_{\text{buy}} u(k), & \text{if } u(k) > 0 \\ \sigma_{\text{sell}} u(k), & \text{otherwise} \end{cases} \tag{2.38}$$

where σ_{buy} and σ_{sell} are the buying and selling prices. It is assumed that $\sigma_{\text{buy}} \geq \sigma_{\text{sell}}$.

2) **Battery Degradation Cost Function:** The component of the cost function that accounts for battery degradation at time step k is given by:

$$L_b(x(k), u(k)) = 0.2 \cdot \frac{\beta_{\text{total}}(k)}{1-\alpha} \cdot (c_b^i + c_b^r) \cdot E_b, \tag{2.39}$$

where α is the replacement threshold, defined as 60% of the battery's rated capacity (i.e., $\alpha = 0.6$), a commonly used value for less-demanding stationary storage applications. The terms c_b^i and c_b^r represent the battery's initial and replacement unit costs, respectively.

3) **Battery SoC Cost Function:** To prolong battery lifetime the cost function penalizes the battery's state of charge (SoC) when it falls outside a specified SoC window $[\text{SoC}_{\min}, \text{SoC}_{\max}]$ using a penalty factor c, i.e.,

$$L_s(x(k)) = c \cdot \max(x(k) - SoC_{\max}, 0) + c \cdot \max(SoC_{\min} - x(k), 0). \tag{2.40}$$

Given the cost functions with the objectives stated in $1) - 3)$, the final economic stage cost is expressed as their summation

$$L(x(k), u(k)) = L_g(u(k)) + L_b(x(k), u(k)) + L_s(x(k)). \tag{2.41}$$

2.6 Gaussian Process Regression

GPR is a supervised learning method and a popular regression technique in machine learning [14]. Unlike many other regression models, GPR does not predict a single output value, but rather a distribution over possible functions that fit the data. This allows GPR to make informed predictions along with uncertainty estimates.

A Gaussian process (GP) is defined as a collection of random variables, any finite number of which have a joint Gaussian distribution. It is fully specified by a mean function $m(x)$ and a covariance function (or kernel) $k(x, x')$, defined as:

$$m(x) = \mathbb{E}[f(x)] \tag{2.42}$$

$$k(x, x') = \mathbb{E}\left[(f(x) - m(x))(f(x') - m(x'))\right] \tag{2.13}$$

This is denoted as:

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')) \tag{2.43}$$

In the regression setting, GPR uses Bayesian inference to update the prior belief $p(f)$ about the function based on observed input-output pairs (x, y). The posterior distribution over functions is given by:

$$p(f \mid x, y) = \frac{p(y \mid x, f)\, p(f)}{p(y \mid x)}, \tag{2.44}$$

where $p(f)$ is the prior over functions, $p(y \mid x, f)$ is the likelihood of the data given the function, and $p(y \mid x)$ is the marginal likelihood [14].

The kernel function $k(x, x')$ plays a central role in GPR. It encodes assumptions about the function's structure—for example, whether it is smooth, periodic, or linear—by measuring similarity between input points. The choice of kernel greatly influences prediction quality, making kernel selection a critical part of model design.

Ultimately, GPR gives a posterior predictive distribution over function values at new input locations, which provides both a predicted mean and a measure of uncertainty.

3
Methods

This chapter focuses on explaining the implementation of the methods whose theoretical background was described in Chapter 2. The specific formulas and libraries used are described here, as well as the main algorithm and how all of the pieces, from solving the optimization problem to GPR, work together to fine-tune the model parameters and in the end return the most economical policy.

3.1 Parametrization

In this subsection, the baseline formulation and the parametrization of the system will be outlined, as well as the formulation of the different test cases.

3.1.1 MPC formulation

Based on the system dynamics described in Section 2.1 and the general MPC-based RL formulation described in 2.5, the following MPC problem definition was obtained:

$$\min_{\hat{x},\hat{u}} \quad \lambda_0 + V_f(\hat{x}(N)) + \sum_{k=0}^{N-1} \Big(L(\hat{x}(k), \hat{u}(k)) \Big) \tag{3.1a}$$

$$\begin{aligned} \text{subject to:} \quad & \hat{x}(k+1) = \hat{x}(k) + (\hat{u}(k) + w(k)), && \text{(3.1b)} \\ & 0 \leq \hat{x}(k) \leq 1, && \text{(3.1c)} \\ & 0 \leq \hat{x}(N) \leq 1, && \text{(3.1d)} \\ & \forall k = 0, \ldots, N-1 \end{aligned}$$

$L(\hat{x}(k), \hat{u}(k))$ is equal to the economic stage cost function in Equation 2.41, and N is the control horizon.

This serves as a baseline formulation that the parametrized versions were compared to. Multiple different parametrization versions of the system were then tested to see if the optimal policy can be achieved and whether MPC-based RL leads to cost savings. Note as well that the roundtrip efficiency, as seen in Equation 2.2, is ignored in this work, similarly to [1], since it impacts the convexity of the MPC formulation.

3.1.2 MPC parametrization

As previously mentioned in 2.5, the parametrization is up to the designer. The θ parameters can both be stand-ins for physical parameters, such as battery capacity, or they can be entirely arbitrary. The complete parametrization of the system dynamics can be seen below:

$$x(k+1) = \theta_a x(k) + \theta_b(u(k) + w(k)), \tag{3.2a}$$

with

$$x \triangleq \text{SoC}, \quad u \triangleq \frac{h}{E_b} P_{\text{grid}}, \quad u_{\text{lim}} \triangleq \frac{h}{E_b} P_{\text{grid}}^{\text{lim}} \tag{3.2b}$$

$$w \triangleq \frac{h}{E_b}(P_{\text{PV}} - P_{\text{load}}). \tag{3.3}$$

The parametrization of the complete stage cost was designed as:

$$L_\theta(x(k), u(k), \theta_d, \theta_e, \theta_f) = \theta_d L_g(x(k), u(k)) + \theta_e L_b(x(k), u(k)) + \theta_f L_s(x(k), u(k)) \tag{3.4}$$

The terminal cost V_f is parametrized similarly to the stage cost with one parameter:

$$V_{f\theta}(\hat{x}(N), \theta_T) = \theta_T(\hat{x}(N) - 0.5)^2. \tag{3.5}$$

The same is done with λ_θ as well:

$$\lambda_\theta(\hat{x}(0), \theta_0) = \theta_0(\hat{x}(0) - 0.5)^2 \tag{3.6}$$

The complete parametrized MPC problem is then given by:

$$\min_{\hat{x},\hat{u}} \quad \lambda_\theta(x_0; \theta_0) + V_{f\theta}(\hat{x}(N); \theta_T) + \sum_{k=0}^{N-1} L_\theta(\hat{x}(k), \hat{u}(k); \theta_d, \theta_e, \theta_f) \tag{3.7a}$$

subject to:

$$\hat{x}(k+1) = \theta_a \hat{x}(k) + \theta_b(\hat{u}(k) + w(k)) \tag{3.7b}$$
$$0 \leq \hat{x}(k) \leq 1 \tag{3.7c}$$
$$0 \leq \hat{x}(N) \leq 1 \tag{3.7d}$$
$$\hat{x}(0) = x(n) \tag{3.7e}$$
$$\forall k = 0, \ldots, N-1$$

Weight for the slack variables was chosen to be 10^6, since this is sufficiently large to give a solution similar to the unrelaxed one, see Section 2.5.

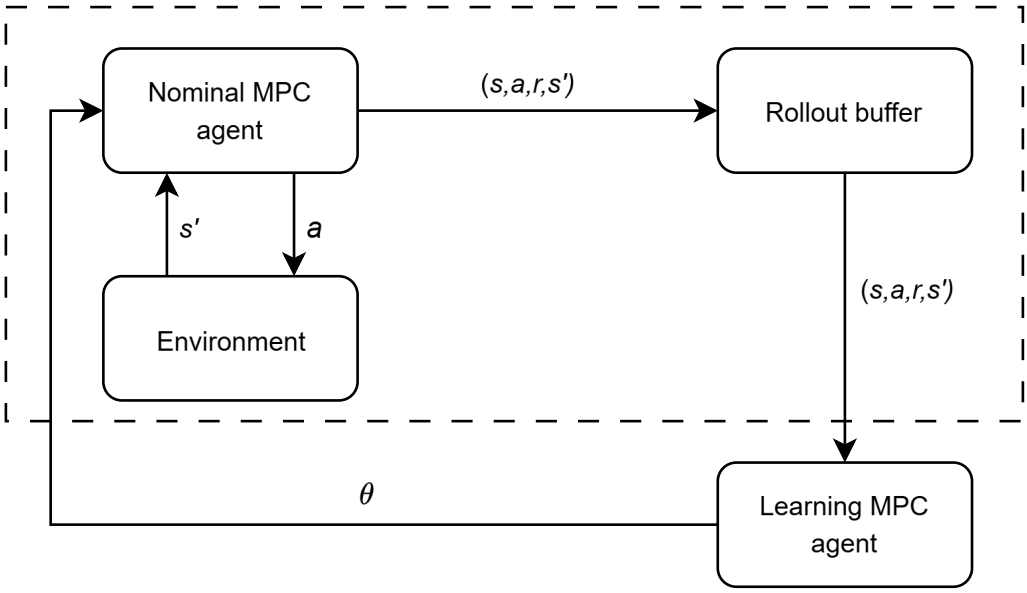

Figure 3.1: A flowchart representing the simulation setup. Rollouts are created by the nominal MPC agent that interacts with the environment, with 720 rollouts being created for every parameter update. The rollouts are then used by the learning agent to learn the optimal policy.

3.2 Simulation setup

The mpcrl library, created in [15], was used for this thesis. Some of the library's classes and methods were altered to accommodate the change in prices and residual power. The overarching structure of the algorithm is presented below.

Originally, tuning was planned to cover six months for each of the three test cases (see Section 3.2.3). Due to the significant time required for tuning over an extended period, performing six months of tuning for all cases was not feasible. The tuning period was therefore reduced to one month per case. This approach allowed for a balance between computational resources and the ability to effectively evaluate the results of different parameterization cases.

3.2.1 Agent

The agent is responsible for learning. It operates in two phases: a nominal MPC agent phase, where the policy π is fixed and parameters are not updated and a learning phase, where value functions are computed and parameters are updated.

In the nominal MPC phase, the following steps were executed for each timestep:

1. Solve the MPC problem to obtain the optimal action (u_0^*)
2. Execute the action and record the stage cost $L(s_k, a_k)$
3. Transition to the next state s'.
4. Save the tuple (s, a, s', L) into the rollout buffer. One rollout is one such

transition tuple.
This rollout generation was repeated until the full dataset is collected. In this case it consists of 720 rollout steps, representing one month of operation with a sampling time of one hour.

After all rollouts were collected, the learning phase began. Here, the MPC agent was ran again, but for computing value functions. For each tuple (s, a, s', L) in the rollout the following steps were executed:

1. Fix s and a as constraints and solve the MPC again. The result gives $Q_\theta(s, a)$ and the gradient ∇Q_θ, a by-product of solving the MPC (see Section 2.5.1).
2. Solve the MPC with s' as the initial state (no action constraint) to obtain $V_\theta(s')$ (see Section 2.5).
3. Compute the temporal difference (TD) error, as per Equation 2.37
4. Update the parameters in the MPC scheme

This full process—generating rollouts, computing value functions, and updating parameters was repeated for a specified number of episodes until each of the learnable parameters converge to a steady value. In this thesis, 300 episodes for each of the cases were sufficient to reach convergence of the parameters, as well as the stage cost convergence and stabilization of the TD error.

3.2.2 Main algorithm

The main algorithm was then structured in the following way:
1. Define the environment class with the system dynamics, as well as the right spot prices and residual for every time instance.

2. Define the MPC controller class. This includes setting the correct dynamics, setting the constraints, defining the learnable parameters, defining the stage cost function, setting spot prices and the residual as per 3.1.
3. Define the rollout generator.
4. Run the simulation. This included the following sub-steps:
   1. Instantiate the MPC controller instance
   2. Instantiate the Q-learning agent.
   3. Run the offline training for the agent
   4. Plot the results

Battery-related constants used in the dynamics and cost calculations are presented in Table 3.1.

Table 3.1: Battery and cost related parameters

Parameter	Value	Unit
Battery rated capacity (E_b)	200	kWh
Maximum charge/discharge power (P_b^{lim})	70	kW
Battery round-trip efficiency (η)	96	%
SoC window ($\text{SoC}_{\text{min}} - \text{SoC}_{\text{max}}$)	20–85	%
Calendar life (l_{cal})	13.5	years
Cycle life (l_{cyc})	6000	EFC
Battery replacement threshold (α)	60	%
Battery initial unit cost (c_b^i)	463	EUR/kWh
Battery replacement unit cost (c_b^r)	413	EUR/kWh
SoC penalty weight (c)	1	EUR

3.2.3 Different test cases

To evaluate how the different parametrization strategies affect the performance some different test cases were simulated. The layout of the testing setup is described in the table below, as well as the initial values for the learnable parameters:

Table 3.2: Parameter update configurations across different cases and their initial values. Red numbers mark the learnable parameters while the other values are not updated.

	Baseline	System dynamics	Stage cost	All together
θ_a	1	1	1	1
θ_b	1	1	1	1
θ_d	1	0.1	0.1	0.01
θ_e	1	0.0025	0.0025	0.0025
θ_f	1	1	1	0.1
θ_T	1	0	0	10
θ_0	1	0	0	10

The four main subgroups of tests are, as seen in Table 3.2:

- **Baseline case**: no learnable parameters are defined and all parameters are initialized as 1. This is so the other test cases can be compared to it.
- **System dynamics only**: parameters related to the system dynamics θ_a and θ_b, as well as terminal cost parameter θ_T and λ_0, are defined as learnable and updated.
- **Cost functions only**: parameters related to the stage cost θ_d, θ_e and θ_f, as well as terminal cost parameter θ_T and λ_0, were set as learnable.
- **System dynamics and cost functions**: all parameters were set as learnable.

The initial values were decided through experimentation. Since the learning rate step size was very small for most parameters to prevent overshoot and divergence, having appropriate initial values was important for stability and the overall cost.

3.3 Implementation of Gaussian Process Regression for residual power prediction

Once the policy is learned, the policy can be evaluated for future use. The issue is that the consumption-production difference needs to be known for the next 24 hours (from the timestep k) in order for MPC controller to be able to decide the optimal input at time k. To forecast the residual the GPR method, outlined in Section 2.6, was used.

A GPR model can have a multi-dimensional input that maps to one singular output. In this case, the input values are week of the year, day of the week, hour of the day and residual load values for the previous 24 hours; the output is the predicted residual load for the next hour. This is done recursively to get the predictions for the next 24 hours, since the prices are released once a day at 13:00. This is then done for multiple days/months, depending on the number of iterations/number of rollouts in the main RL loop, which in this case is half a year.

Pseudo-code for the method can be seen below:

Algorithm 1 GPR-Based Residual Power Prediction

Input: $P_{\text{res\_window}} = [P_{\text{res}}(k-23), \ldots, P_{\text{res}}(k)]$, Week W, Day D, Hour H
for $t = 1$ to prediction horizon **do**
  $x_{\text{input}} \leftarrow [W, D, H, P_{\text{res\_window}}]$
  $P_{\text{next}} \leftarrow \text{model.predict}(x_{\text{input}})$
  Remove $P_{\text{res\_window}[0]}$
  Append P_{next} to $P_{\text{res\_window}}$
  $H \leftarrow (H+1) \mod 24$
  if $H == 0$ **then**
    $D \leftarrow (D+1) \mod 7$
    if $D == 0$ **then**
      $W \leftarrow (W+1) \mod 52$
    end if
  end if
end for

The kernel function was chosen based on the best performance, i.e. the best match between the predicted residual values and actual values. Rational Quadratic (RQ) kernel gave the best results for performance metrics such as Root Mean Squared Error (RMSE), which is often used to evaluate machine learning/regression models:

$$k_{\text{RQ}}(x, x') = \sigma_f^2 \left(1 + \frac{(x - x')^2}{2\alpha\ell^2}\right)^{-\alpha}.$$

The available dataset was comprised of one half year and two full years of PV production and consumption data. Data for one month in the beginning of second half of the year 2021 was used for training the GPR. Data for the same time in 2022 was

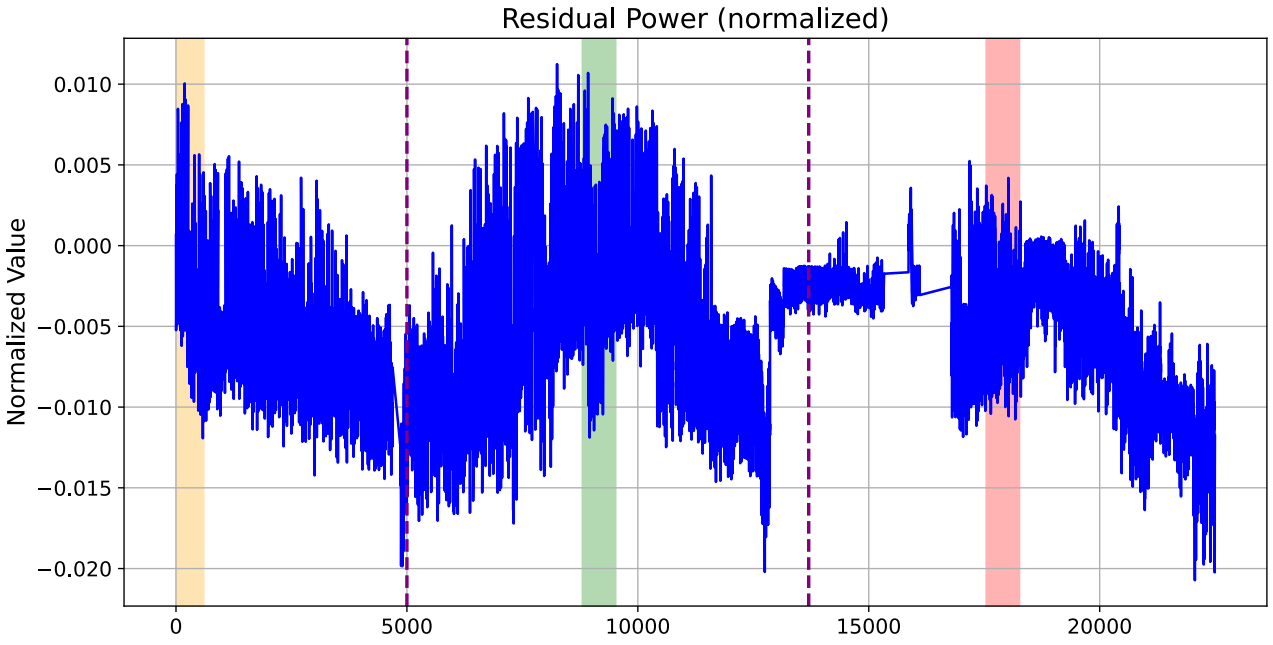

Figure 3.2: Plot over the residual data. The yellow slice was used for training the GPR, the green for training the MPC-RL model and the red one for evaluation. The vertical lines mark the end/beginning of the year.

used for training the MPC-based RL model, while the corresponding data of 2023 data was used for evaluation of the GPR method and cost comparison. Data for the first part of 2022 and 2023 was not used. This was due to two reasons: to give a fair comparison with [1], where the method was evaluated on the same portion of data (second half of 2023), and due to the data from first part of 2023 having low quality and not being very representative, which would give worse GPR predictions (see Figure 4.1).

This method was benchmarked against random sampling from a simple Gaussian distribution similarly to [1]. Mean value and standard deviation was taken from residuals data for June 2021, same data that the GPR was trained on, and the residuals data was sampled from this Gaussian distribution. Random seed was used for replicability and fair comparison between the cases.

4
Results

In this chapter the results of the thesis are presented and discussed. MPC-based RL method is compared to the baseline non-parametrized MPC method and the RL method used in [1]. Different settings of parametrization are compared with each other and the baseline model/RL model. Lastly, the contribution of the GPR method of forecasting is also evaluated by comparing it to only using a constant mean value for residual.

4.1 Simulation results

In this section, the resulting parameter and state/input trajectory graphs for all of the cases are presented, as well as tables containing learning rates used for training and final cost.

4.1.1 Baseline MPC

The baseline MPC operation over a month (June 2023) without any parameter tuning is displayed in Figure 4.1. The input and state constraints are shown by the blue shaded areas. The tuned controllers are compared to this case as a reference point.

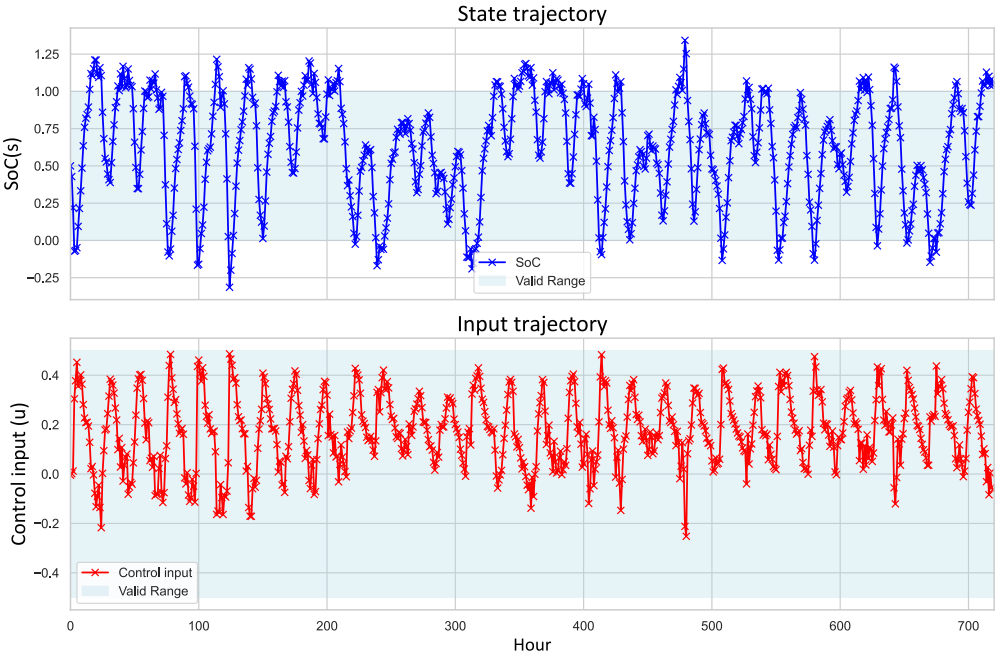

Figure 4.1: Plot over the state and input trajectories for the baseline MPC case with no tunable parameters for one month of operation. The shaded blue area are the input/state constraints

This plot demonstrates multiple constraint violations, whereas no violations occurred in the other cases. This is mainly because the SoC window cost, terminal cost, and λ-cost had a weaker influence compared to the other scenarios. These costs share the common purpose of discouraging the SoC profile from deviating significantly from 0.5 in either direction. In a real-life scenario these constraints are critical. This is another positive aspect that was revealed when tuning using MPC-based RL.

4.1.2 Case 1

Case 1 corresponds to the "system dynamics only" test setup from Section 3.2.3. In this configuration, only θ_a and θ_b from the system dynamics were updated, together with the terminal cost parameter θ_T and θ_0. The motivation was to see whether tuning parameters directly tied to the system model could deliver noticeable improvements, even without modifying the stage cost terms. Table 4.1 lists the learning rates and the final converged parameters for Case 1.

Table 4.1: Learning rates used for the training and the resulting converged parameters for Case 1.

Parameter	Learning rate	Resulting parameter
θ_a	5×10^{-5}	1.025
θ_b	1.6×10^{-4}	0.128
θ_T	6×10^{-3}	-6.579
θ_0	8×10^{-2}	25.236

The evolution of the tunable parameters during training is shown in Figure 4.2. Figure 4.3 shows the corresponding MPC trajectories, which show reduced oscillations compared to the baseline.

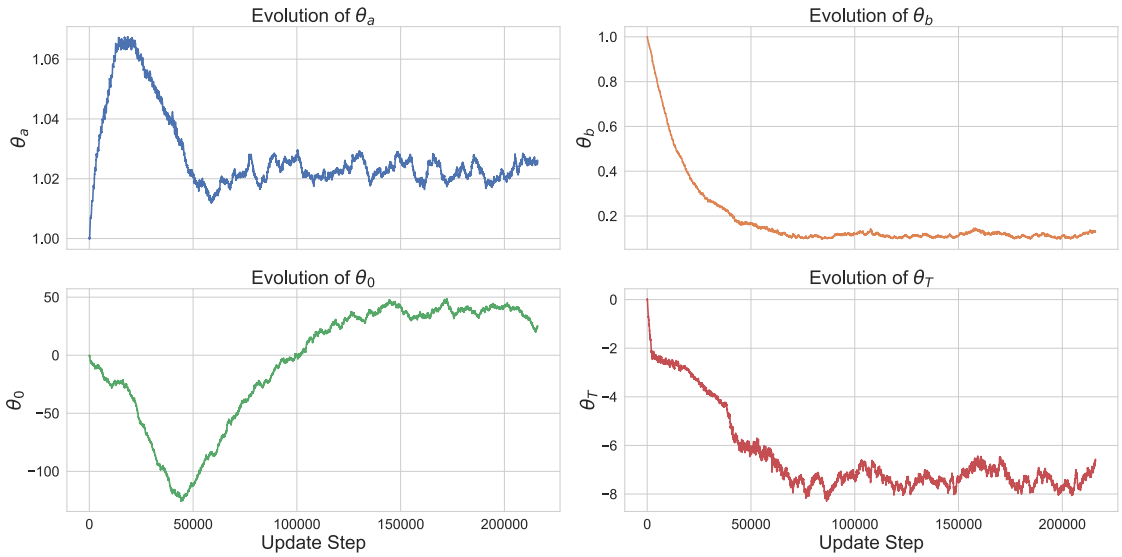

Figure 4.2: Evolution of the learnable parameters for Case 1.

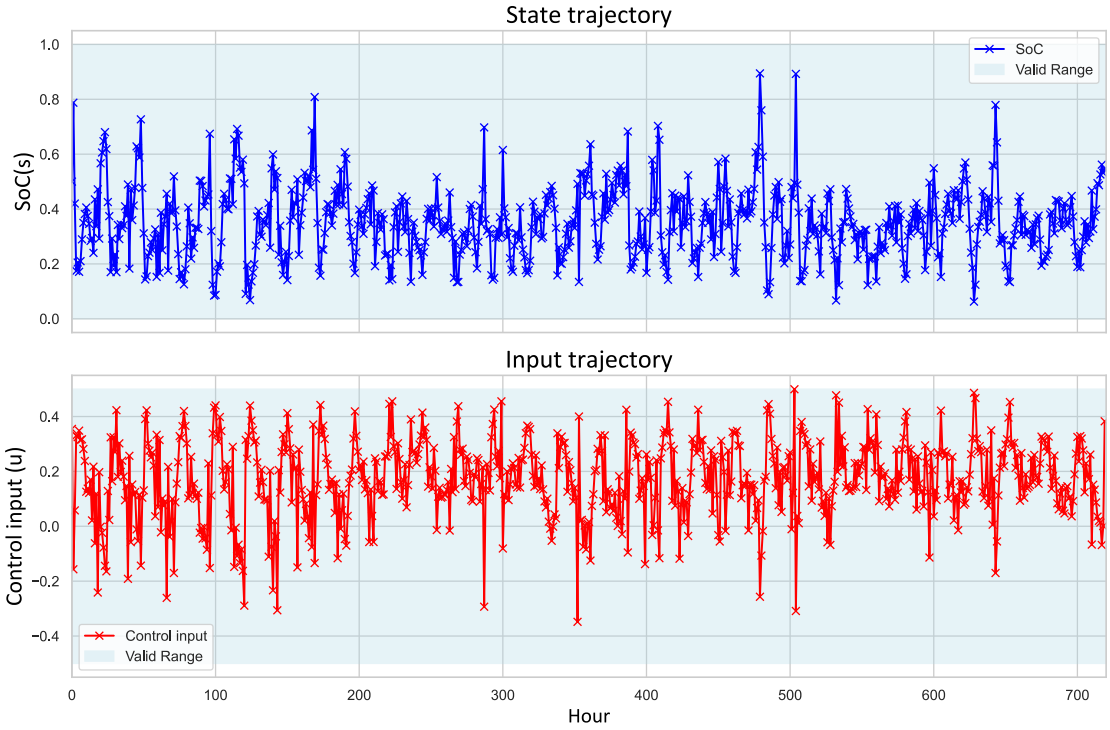

Figure 4.3: State and input trajectories for MPC with parameters from Case 1.

4.1.3 Case 2

Case 2 is the cost functions only setup. Here, the cost-related parameters θ_d, θ_e, and θ_f were updated, together with θ_T and θ_0. The purpose was to check if tuning the cost terms alone could improve the balance between grid cost and battery degradation. Table 4.2 lists the learning rates and the final converged parameters for Case 2. It is worth noting that θ_f became extremely large, converging to 3.19×10^6. This can be attributed to a scale mismatch between the parts of the stage cost, defined in Equation 3.4. If the cost L_s is much smaller than L_g and L_b over the trajectory, the algortihm increases θ_f to make L_s contribute more to the total cost. This parameter was given a higher learning rate than the others to speed up convergence since tests indicated it would otherwise take too long to settle.

Table 4.2: Learning rates used for the training and the resulting converged parameters for Case 2.

Parameter	Learning rate	Resulting parameter
θ_d	5×10^{-3}	-4.441
θ_e	5×10^{-5}	-0.055
θ_f	5×10^{5}	3187443.209
θ_T	8×10^{-3}	-48.978
θ_0	6×10^{-2}	2.927

The evolution of the tunable parameters during training is shown in Figure 4.4.

Figure 4.5 shows the corresponding MPC trajectories, which also show reduced oscillations compared to the baseline.

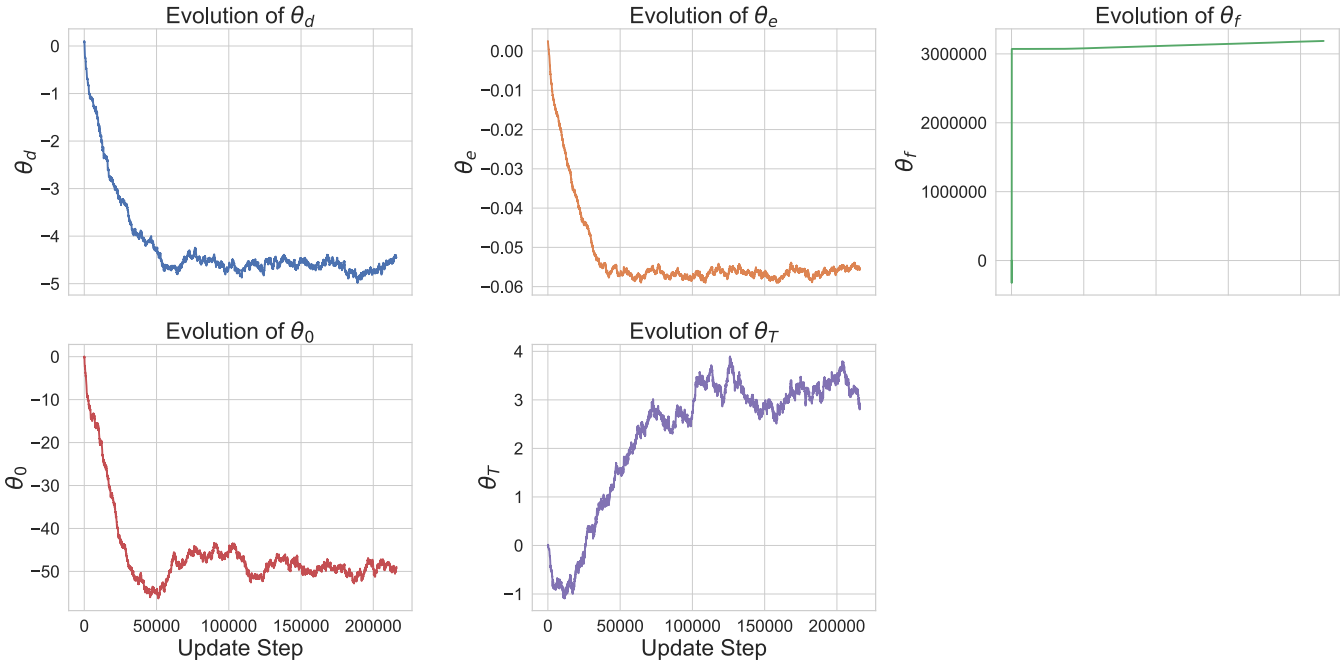

Figure 4.4: Evolution of the learnable parameters for Case 2.

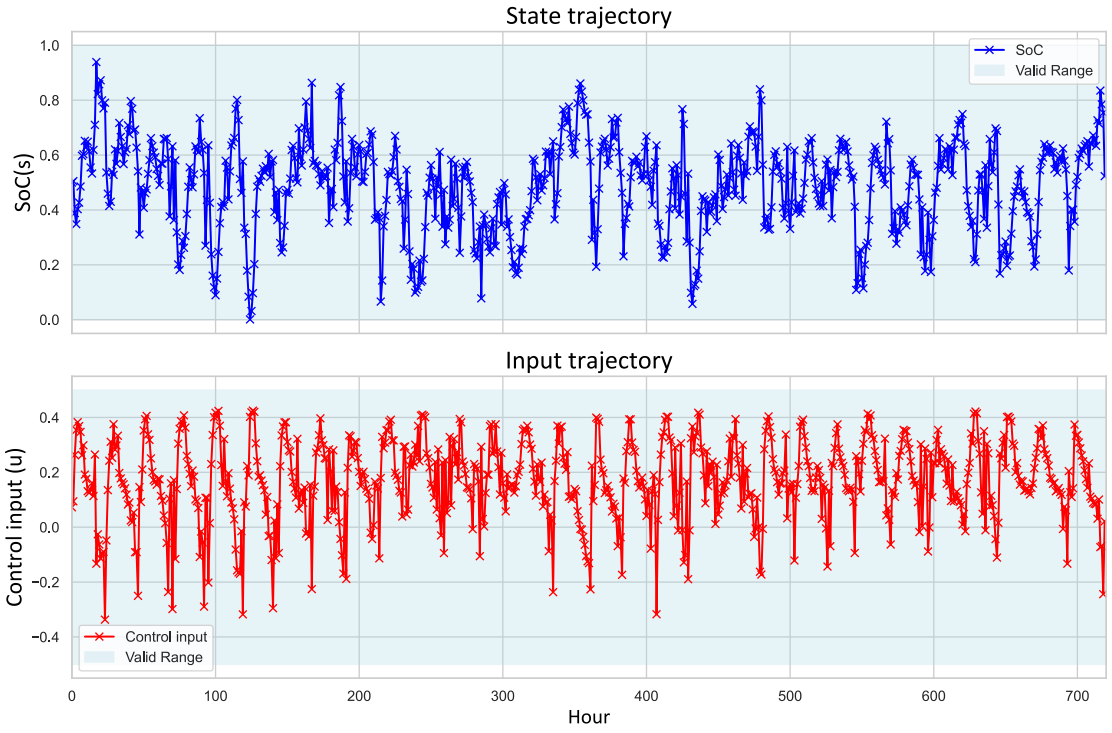

Figure 4.5: State and input trajectories for MPC with parameters from Case 2.

4.1.4 Case 3

Case 3 is the system dynamics and cost functions together setup. Here, both groups of parameters were updated, which gave the most freedom to adapt the controller. The learning rate used for the training and the resulting parameters can be seen in Table 4.3. Unlike Case 2, θ_f stayed within a reasonable range. The algorithm could adjust multiple parameters including the system dynamics and other cost parameters, so θ_f did not have to compensate on its own. The learning rate for θ_f was also smaller and the cost terms were better balanced, which helped keep its value lower.

Table 4.3: Learning rates used for the training and the resulting converged parameters for Case 3.

Parameter	Learning rate	Resulting parameter
θ_a	1×10^{-9}	1.000013
θ_b	1×10^{-9}	0.999
θ_d	1×10^{-8}	0.00997
θ_e	1×10^{-7}	-3.602×10^{-5}
θ_f	1×10^{-9}	0.10000070
θ_T	1×10^{-3}	0.0715
θ_0	1×10^{-5}	9.745

The evolution of the tunable parameters during training is shown in Figure 4.6. Figure 4.7 shows the corresponding MPC trajectories, which also show a reduction in constraint violations and a smoother input profile when compared to the baseline.

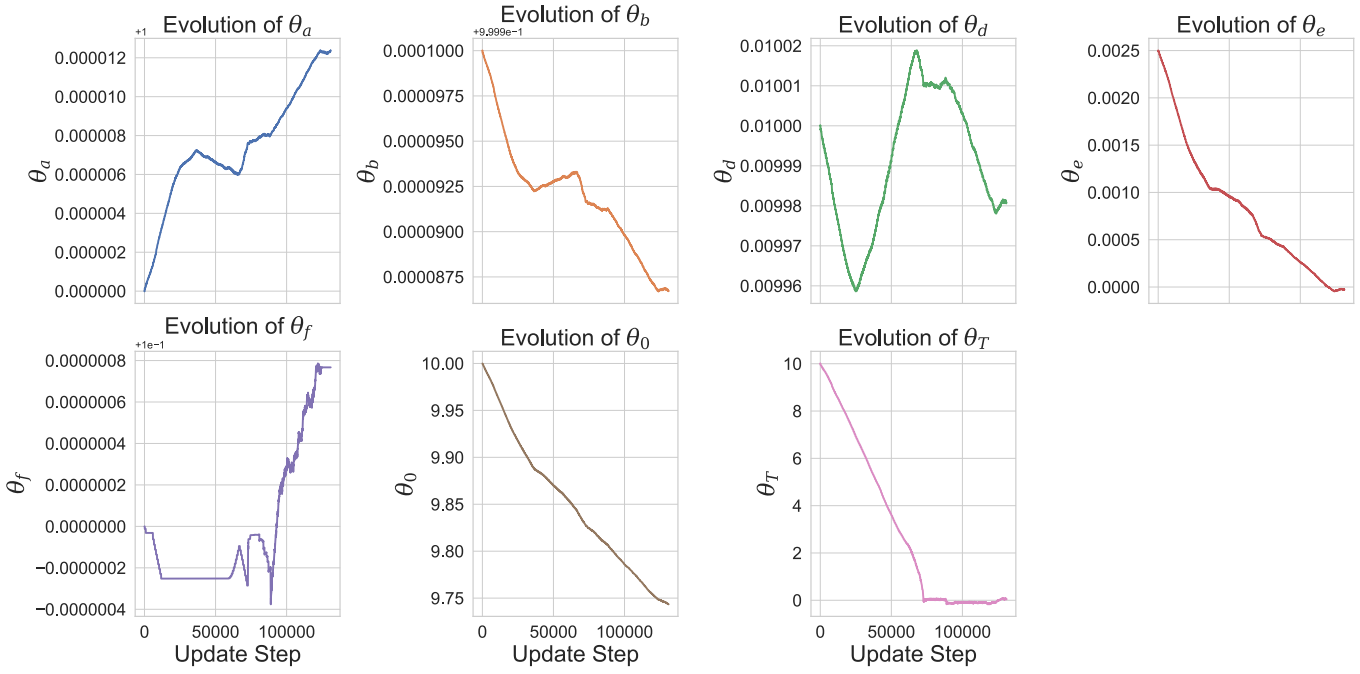

Figure 4.6: Evolution of the learnable parameters for Case 3.

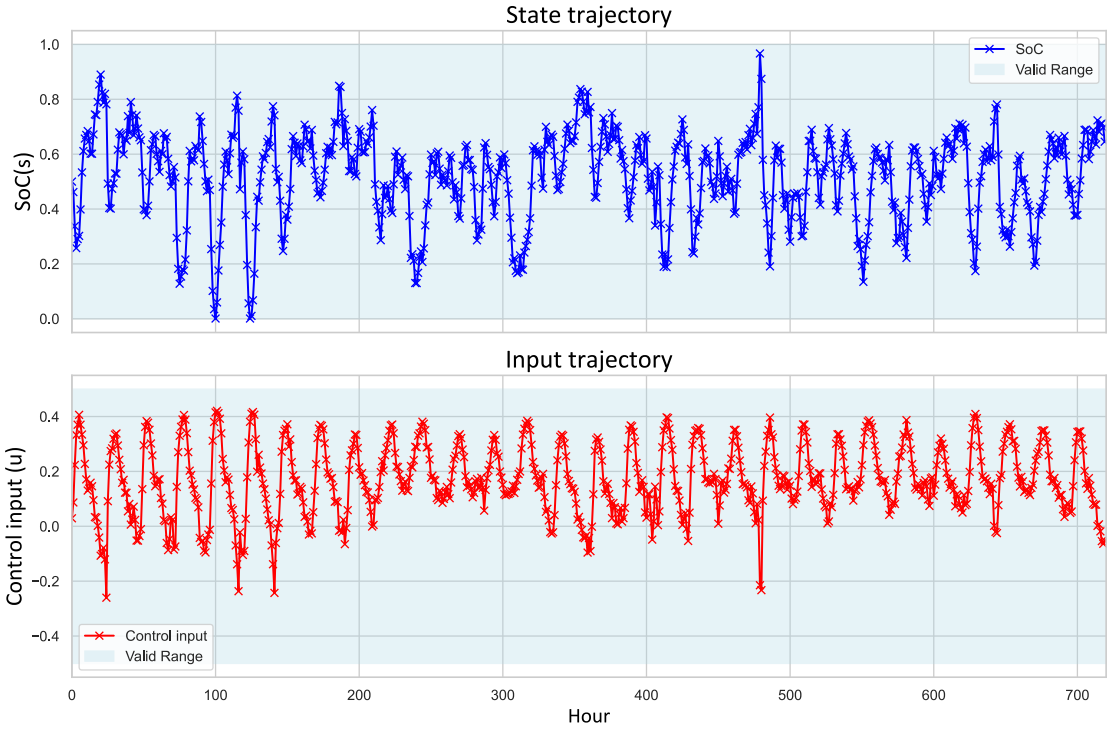

Figure 4.7: State and input trajectories for MPC with parameters from Case 3.

4.1.5 Cost comparison

In the table below, the grid, battery degradation and total cost are presented for all four cases, as well as costs for rule-based, implicit and explicit RL cases from [1] for comparison. The rule-based strategy follows simple heuristics — it charges the battery when PV generation exceeds the load and discharges when the load exceeds PV power, without using forecasts or optimization. The difference between the two RL strategies is that the explicit policy is a fixed, time-invariant control strategy learned under constant electricity prices, where the power-production difference is modeled as a Gaussian random variable estimated from historical data. In contrast, the implicit policy is time-varying, uses forecasted electricity prices, and models the power-production difference with a conditional probability distribution [1].

Battery degradation and battery degradation cost in the table are based on the energy-throughput model.

Table 4.4: Cost comparison of different control strategies. The first three strategies from the table are from [1]

Control strategy	Grid cost (EUR)	Battery degradation (%)	Battery degradation cost (EUR)	Total cost (EUR)
Rule-based (MSC)	2975.0	0.13	583.6	3558.6
Explicit	2977.6	0.20	861.9	3839.5
Implicit	2952.4	0.18	809.8	3762.2
Baseline MPC	3047.4	0.22	979.9	4027.1
Case 1	3058.3	0.21	942.4	4000.7
Case 2	3128.0	0.20	900.9	4028.9
Case 3	3045.2	0.19	813.9	3859.1

It is evident that the 4 simulated cases did not result in a better overall cost than the RL control strategies simulated by [1]. Case 2 gave the highest overall cost, with Case 3 having the best overall cost out of the four strategies. When comparing Case 3 to the explicit and implicit RL strategies, the total cost is only 0.5 % higher that the explicit case, and 2.6 % higher than the implicit case.

All four cases had a worse grid cost than the previous strategies, with Case 2 having the highest grid cost and Case 3 the lowest. Cases 1, 2 and 3 all had a lower degradation percentage and cost compared to the baseline case, with Case 3 even having a lower battery degradation than the explicit RL case.

It is, however, important to note the difference in results for [1] when simulating for only one month compared to half a year. Neither the explicit nor the implicit strategy resulted in a lower overall cost for one month compared to the rule-based strategy, while the economical gain is quite significant when simulated over half a year. This indicates that the economic gain could be higher even for the MPC-RL cases when simulated over a longer period of time. As mentioned in [1], the rule-based strategy also has the lowest battery degradation of all because the battery is not used as much.

Results show a downward trend for the degradation cost with the increasing amount of tunable parameters. This does not, however, seem to be the case for the grid cost, with Case 1 having a lower grid cost than Case 2. This could indicate that the choice of parameters connected to the dynamics has a bigger impact on the grid/total cost than the cost function parameters. It could also indicate that the set of parameters in Case 2 are sub-optimal and that learning rates/initial values should be further fine-tuned.

The MPC-based RL approach got slightly worse results than the more traditional RL approach used in [1]. Both the grid cost and degradation cost were slightly worse for Case 3, even though more advanced methods were used for obtaining the policy

and predicting the residuals during online operation. Since the total cost for Case 3 is close to the total cost for the methods from [1], it is possible that further tuning or a different parametrization could make the results as good or better. The limitation in the need for tuning and the manual choice of parametrization is discussed in Section 5.1.

4.2 GPR

The predictions of the GPR model for the first month of the year 2023 can be seen in Figure 4.8.

From the figure, it is evident that the GPR curve closely follows the actual residual curve with som minor miss-alignments in places where the actual residual values have either very low or very high amplitude.

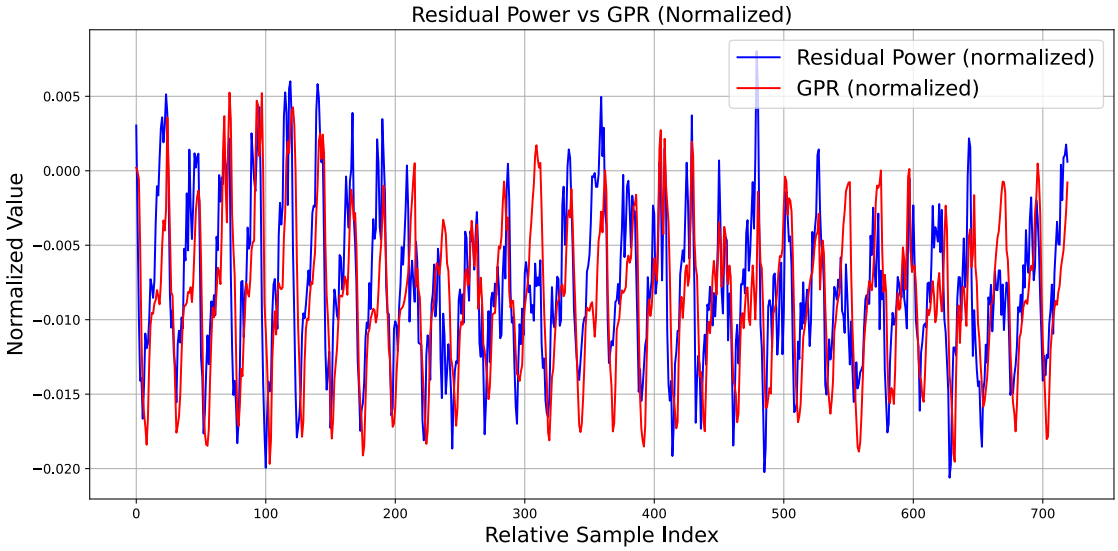

Figure 4.8: Comparison of the actual residual data vs the predicted residual data for the first month of 2023.

In the table below, comparison between the GPR and only using a simple Gaussian distribution is shown and contrasted with the performance when actual values for June 2023 were used.

Table 4.5: Cost comparison between cost using residual values predicted by GPR, from a simple Gaussian distribution and actual residual values

Cases	Cost with GPR			Cost with Gaussian sampling			Cost with observed data		
	Grid	Deg	Total	Grid	Deg	Total	Grid	Deg	Total
Baseline MPC	3047.4	979.9	**4027.1**	3062.8	1205.0	**4267.7**	3022.2	807.8	**3830.0**
Case 1	3058.3	942.4	**4000.7**	3101.4	1050.8	**4152.2**	3064.6	970.1	**4034.8**
Case 2	3128.0	900.9	**4028.9**	3082.7	1066.9	**4149.6**	3082.3	965.6	**4047.9**
Case 3	3045.2	814.3	**3859.5**	3004.3	939.5	**3943.9**	3011.8	725.9	**3737.7**

Using GPR-predicted residuals leads to mixed results. When compared to the last column (actual/historical residual values), the grid and degradation cost (and consequently the overall cost) is lower for Case 1. This means that the energy demands may not be met in practice due to the discrepancy between the predicted residual and the actual energy demand during that time period. For Case 2 the grid cost is 1.4 % higher but the degradation cost is 7.2 % lower.

The overall cost is only higher for the baseline case and Case 3 since both the grid and degradation costs are higher. For Case 3, the total cost when using GPR predictions is only 3.3% higher than with actual residuals, corresponding to a difference of 121.8 euros over one month. The difference in battery degradation between these two cases is minimal, at just 0.002 percentage points. This suggests that the GPR predictions cause slightly increased grid power consumption: too much electricity is bought from the grid that will not be used, which will also lead to higher battery usage and degradation

However, compared to residuals generated by random sampling from a simple Gaussian distribution, the GPR approach performs better across all cases in terms of total cost. For instance, the total cost difference for Case 3 between the Gaussian residuals and actual residuals is approximately 5.5%, equating to an extra 206.2 euros per month—about 84.4 euros more than with the GPR method. Additionally, degradation costs are higher across all cases with the Gaussian residuals compared to both GPR and actual residual cases. Although the grid cost is lower for the Gaussian residuals than for GPR, it is also lower than the real residual case for Case 3, which could result in the electricity demands not being met in practice.

Overall, GPR is the more economically viable and safer choice, even though it may lead to slight overspending and overuse of the battery. To get even better results, experimentation with some more complex kernels may be needed.

4.3 Degradation assessment

Table 4.6 shows the battery degradation in percent and degradation costs calculated both with the energy-throughput method and the rainflow cycle-based method. Important to note here is that the results cannot be fully compared: the two methods

were developed by different sources and the equations therefore contain battery parameters from two different batteries. Nevertheless, the two batteries have the same chemistry (LFP) and the comparison can therefore still be useful.

Table 4.6: Comparison between original degradation calculation and rainflow counting method.

Cases	Energy-throughput model		Rainflow cycle-based model	
	Degradation (%)	Degradation cost	Degradation (%)	Degradation cost
Baseline MPC	0.22	979.9	0.87	3815.4
Case 1	0.21	942.4	0.32	1385.2
Case 2	0.20	900.9	0.32	1391.9
Case 3	0.19	813.9	0.27	1174.1

The results for the rainflow method showed more degradation for all cases, and also a bigger difference in degradation and degradation cost between the different cases in general. This demonstrates its capacity to detect degradation information that the other method might have overlooked. Since total damage is dependent on cycle amplitude and sequence rather than just total throughput, this leads to a more accurate evaluation of battery aging.

Compared to the original approach the rainflow results demonstrate significantly greater variations in degradation and cost across cases. This suggests that modeling with degradation awareness is probably more important than first believed when looking at the other degradation results. By ensuring that only important cycles are counted, the pre-processing steps increase accuracy. In general, using rainflow counting offers a more detailed foundation for predicting battery degradation, showing the importance of accurate cycle-based analysis for efficient battery management.

4.4 Key influencing factors

The main elements influencing the results of the suggested approach are discussed in this section. The complexity and structure of the parametrization, the choice of algorithm hyper-parameters like learning rates and initial values, and the precision of residual predictions made with Gaussian Process Regression (GPR) are some of these important factors. To make the method more reliable and help guide future improvements, it is important to understand how these factors influence the training process and the final results.

4.4.1 Step sizes and initial parameters

The quality of training was found to be significantly impacted by the choice of step sizes for the different parameters. Excessively large step sizes in testing resulted in graphs that looked erratic and indicated unstable parameter updates. This illustrates that larger step sizes do not always lead to faster convergence. Performance

was occasionally lowered as a result of the parameters converging to values that were substantially different from those obtained with smaller steps due to excessively large steps. Thus, a significant drawback of this approach is that precise step size adjustment is required to guarantee convergence in a manageable amount of time. Choosing relatively small step sizes and letting the algorithm run for a long time is a workable alternative if enough time is available.

The choice of initial values also played a significant role and could have as much impact as the choice of learning rate, especially for the parameters that had to have a lower learning rate due to their volatility. Certain combinations of learning rates and initial values produced a very high cost due to a lot of constraint violations, or resulted in the MPC solver failing after only a few episodes. For example, in Case 3, changing the initial value of θ_e from 0.0025 to 0.025 resulted in the cost steadily going up during training and the solver failing after only 22 episodes. This means that the amount of variables that need tuning then almost doubles. For cases 1 and 2, when not all parameters are varied, the choice of the non-tunable parameters can also have an impact on the total RL cost. Striking this balance between initial values and step sizes then adds to the already existing time complexity of finding appropriate learning rates.

4.4.2 Richness and structure for parametrization

The number of tunable parameters had a clear impact on how well the MPC-based RL approach performed. Case 1 had four tunable parameters, Case 2 had five, and Case 3 had seven. Based on the results it seems like having more tunable parameters can help lower the total cost. This seems logical since a richer parametrization means the algorithm has more degrees of freedom and thereby more possibilities for lowering the grid cost and the degradation cost.

However, the results also showed that more parameters do not always guarantee better performance. For example, Case 1, which had one less tunable parameter than Case 2 actually performed slightly better in terms of total cost. This suggests that the placement and initial value of the non-tunable parameters play a more significant role than the number of tunable parameters.

4.4.3 Choice of training data

As described in Section 3.3, the historical data from 2022 was used for training the MPC-RL model. The model was then evaluated with residual values predicted by the GPR to emulate the actual usage of the model in the real world. This choice possibly influenced the model's accuracy: using predicted values for both training and evaluation could have led to better performance because the residual distributions would be more similar. However, this approach would likely perform worse when tested against real observed data, which is why it was not adopted. Exploring this setup in future work could provide some more insight into how prediction uncertainty affects model performance.

5

Conclusion

In order to optimize the operational performance of residential photovoltaic battery systems, a degradation-aware model predictive control-based reinforcement learning (MPC-based RL) framework was investigated. Gaussian Process Regression (GPR) was used to improve the forecasting of residual production/consumption differences. The goal was to minimize both battery degradation costs and grid-related costs during dynamic environmental and market conditions. The effect of several selected parameterizations was examined, and the findings were compared to those from conventional RL-based control.

Based on the tuning process performed to reach the results the method shows high sensitivity to hyper-parameters, especially the learning rates and the starting parameter values. Even though using more rich parameterizations usually gave better cost performance, the amount of tuning increased with each added tunable parameter. The best setup (Case 3) managed to reduce battery degradation compared to the baseline MPC while still being cost-effective, with total economical costs similar to more traditional RL methods. When compared to simple Gaussian sampling, the GPR forecasts showed better results, helping to lower costs and battery degradation.

The original degradation calculations were replaced with rainflow counting after the parameter tuning to increase the accuracy of the degradation evaluation. It calculated higher and more varying degradation levels, demonstrating that cycle depth and sequence are important factors in degradation. This suggests that degradation-aware control might be more important than what the simpler model indicated.

Since the method is reliant on time consuming fine-tuning and high-quality historical data, the scalability is limited in its current form. Despite these challenges, the study showed that when degradation is properly considered combining MPC and RL can create energy management strategies that are safe and cost-effective. Future development could focus on testing wider range of datasets and battery chemistries, automated selection of hyperparameters and parametrizations, and evaluation over longer horizons. By addressing these issues MPC-based RL may become a more reliable tool for PV-battery management in the real world.

5.1 Limitations of the chosen method

As discussed in Section 4.4.1, the algorithm is sensitive to the choice of learning rates and initial parameter values. Other hyperparameters such as weights for the RL cost also have an impact. This dependence on hyperparameters increases the risk of suboptimal convergence. Finding the right hyperparameter combinations requires complex tuning, further complicated by the fact that the algorithm takes a substantial amount of time to run. It frequently includes iterative trial-and-error experimentation. There is also a possibility that the ideal set of hyperparameters will differ based on the dataset or the accuracy of the prediction model. These elements restrict the method's ability to be scaled to various system configurations without requiring extra work. This tuning limitation has likely lead to sub-optimal results in this instance. This problem could be addressed through further work, which will be discussed in Section 6.

The fact that the designer controls the parametrization is another drawback of the selected approach. The results indicate that better costs are not necessarily provided by a more rich parametrization. The issue is that there are a lot of different ways to parametrize, and finding a good parametrization that produces favorable results may involve a lot of trial and error.

The availability of high-quality historical data is also crucial to the approach. Only particular time windows with comparatively constant operating conditions were covered by the dataset used in this study. Both the RL agent's training and the precision of the GPR forecasts may suffer from low quality data. The statistical patterns found in the training set influence the learned policy of the controller. Large differences from real-world operation (such as different household demand or unusual weather) may result in decreased performance.

6
Future work

Building on the previously discussed limitations and insights, a number of important topics for future research are presented here.

Testing the strategy over a longer time period could be crucial. Longer simulations beyond the one-month window will produce a clearer picture of the method's actual performance and useful advantages since economic gains and degradation patterns can vary significantly over extended horizons. Other than testing the strategy over a longer period of time, to confirm the generalizability of the method would require expanding the validation to additional datasets that represent various operating conditions. The current approach relies heavily on having historical data that is both high in quality and representative, as discussed in Section 5.1. To better assess how robust the method is in real-world situations, it should be evaluated on more datasets with different demands and environmental conditions.

In order to make the method more applicable to a wide range of energy storage functions, it could be expanded in order to take into account different cell chemistries since each type can have its own degradation patterns and system behavior.

The effects of parametrization on performance are discussed in Sections 4.4 and 5.1. To determine which structures provide the best trade-off between complexity and quality, future research could investigate varied parameterizations. Another possibility is to look into automated parametrization design methods.

Automated hyperparameter optimization techniques, such as grid search or gradient descent, could also be integrated into this approach to speed up the tuning process and ensure optimality.

Future work can make the MPC-based RL framework for battery management more effective and practical by addressing the issues discussed here.

Bibliography

[1] H. Zhang, F. Altaf, and T. Wik, "Comparative Study of Aging-Aware Control Strategies for Grid-Connected Photovoltaic Battery Systems," no. Cdc, pp. 3501–3507, 2024.

[2] S. Gros and M. Zanon, "Data-driven economic NMPC using reinforcement learning," *IEEE Transactions on Automatic Control*, vol. 65, no. 2, pp. 636–648, 2020.

[3] G. Ji, K. Zhang, and Y. Zhu, "A method of MPC model error detection," *Journal of Process Control*, vol. 22, no. 3, pp. 635–642, 2012.

[4] D. Mayfrank, M. Velioglu, A. Mitsos, and M. Dahmen, "Sample-Efficient Reinforcement Learning of Koopman eNMPC," pp. 1–24, 2025.

[5] M. Zanon, S. Gros, and A. Bemporad, "Practical reinforcement learning of stabilizing economic MPC," *2019 18th European Control Conference, ECC 2019*, pp. 2258–2263, 2019.

[6] W. Cai, A. B. Kordabad, and S. Gros, "Energy management in residential microgrid using model predictive control-based reinforcement learning and Shapley value," *Engineering Applications of Artificial Intelligence*, vol. 119, no. January, p. 105 793, 2023.

[7] ASTM, "E1049-85: Standard Practices for Cycle Counting in Fatigue Analysis," *E1049 - 85*, vol. 85, no. Reapproved 2011, pp. 1–10, 2011.

[8] B. Xu, A. Oudalov, A. Ulbig, G. Andersson, and D. S. Kirschen, "Modeling of lithium-ion battery degradation for cell life assessment," *IEEE Transactions on Smart Grid*, vol. 9, no. 2, pp. 1131–1140, 2018.

[9] K. S. Holkar and L. M. Waghmare, "An overview of model predictive control," *International Journal of Control and Automation*, vol. 3, no. 4, p. 7, 2010.

[10] S. Joe Qin and Badgwell Thomas A, "A survey of industrial model predictive control technology," *Control Engineering Practice*, vol. 11, pp. 733–764, 2003.

[11] M. Ellis, H. Durand, and P. Christofides, "A tutorial review of economic model predictive control methods," *Journal of Process Control*, vol. 24, 2014.

[12] M. Wiering and M. van Otterlo, *Conclusions, future directions and outlook*. 2012, vol. 12, pp. 613–630, ISBN: 9783642015267.

[13] E. F. Morales and J. H. Zaragoza, "An introduction to reinforcement learning," *Decision Theory Models for Applications in Artificial Intelligence: Concepts and Solutions*, pp. 63–80, 2011.

[14] M. Seeger, *Gaussian processes for machine learning*. The MIT Press, 2004, vol. 14, pp. 69–106.

[15] F. (C. f. S. Airaldi and Control), *mpc-reinforcement-learning*, 2024.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY