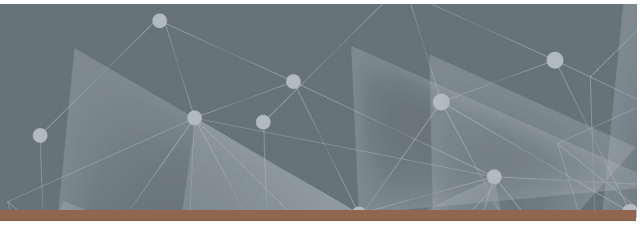
CHALMERS
UNIVERSITY OF TECHNOLOGY



Learning-Enhanced Nonlinear Model Predictive Control for Battery Thermal Management Systems

Master's thesis in Systems Control and Mechatronics

Lech Kazimierz Kula, Daniel Joseph McCauley

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026
www.chalmers.se

Master's thesis 2026

Learning-Enhanced Nonlinear Model Predictive Control for Battery Thermal Management Systems

DANIEL JOSEPH MCCAULEY & LECH KAZIMIERZ KULA

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2026

Learning-Enhanced Nonlinear Model Predictive Control for Battery Thermal Management Systems

DANIEL JOSEPH MCCAULEY & LECH KAZIMIERZ KULA

Supervisor: Prashant Lokur, Geely Technology Europe
Examiner: Nikolce Murgovski, Department of Electrical Engineering Chalmers University of Technology

Master's Thesis 2026
Department of Electrical Engineering
Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2026

Learning-Enhanced Nonlinear Model Predictive Control for Battery Thermal Management Systems
Daniel Joseph McCauley & Lech Kazimierz Kula
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Battery thermal management (BTM) systems in electric vehicles are required to regulate the temperature of the battery powering the vehicle. Model predictive control (MPC) is an optimization-based control strategy that has proven useful in nonlinear control tasks across many different domains, and is therefore a promising candidate for BTM. However, battery thermal management systems are difficult to model due to nonlinearities, and simplified control models that do not fully capture the true dynamics are often employed, which can result in reduced control performance.

In this thesis, an adaptive control framework is proposed for learning model residuals using a neural network. The learned residuals are used within the control model of the controller, resulting in a control model that adapts to the system. Specifically, the neural network is trained using two distinct loss functions, resulting in two distinct adaptive controllers. Both adaptive controllers are compared against a nominal controller relying solely on a physics-based model, on both matched and mismatched systems. The framework is initially tested on a benchmark reference tracking cascaded tank system, where it successfully learns the mismatch in dynamics and achieves improved closed-loop control performance. The framework is subsequently evaluated for both reference tracking and economic MPC formulations in BTM systems.

For reference tracking, the adaptive controllers yielded mixed results, in some scenarios decreasing cost by up to 44 %, whereas in other scenarios increasing cost by up to 409 %. Similarly the root-mean-squared tracking error was reduced in some cases, and substantially increased in others. In economic MPC, the adaptive controller achieved cost reductions of 23 % to 35 % for all mismatched models, while incurring up to 11 % higher cost in a scenario with a matched model.

Model adaptation via neural network residuals is therefore not automatically beneficial, as the approach is sensitive to the loss design, hyperparameters, and the training data. The proposed framework does improve performance in some scenarios, and enhancing its robustness and generalizability warrants further investigation.

Keywords: Battery Thermal Management, Model Predictive Control, Adaptive Control, Neural Networks, Residual Dynamics, Electric Vehicles

Acknowledgements

We would like to express sincere gratitude to our supervisor Prashant Lokur. Throughout the thesis he has provided great support and invaluable advice. His deep insight and immense experience in battery thermal energy management has proven indispensable for our thesis work and for our personal growth as engineers. He has also continuously inspired us to be diligent and thorough with research. We would also like to extend thanks to our examiner, Professor Nikolce Murgovski, who has been extremely kind, patient, and helpful by providing his knowledge of optimization and control. We would also like to thank Geely Technology Europe for hosting us and providing excellent resources. The people at Geely Technology Europe have been welcoming and contributed to our work by providing a kind working environment where learning and research is supported. Finally, we would like to thank the other thesis students at Geely Technology Europe, with whom we have shared successes and troubles, and with whom we have had many thoughtful discussions.

Daniel Joseph McCauley & Lech Kazimierz Kula, Gothenburg, June 2026

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis:

AdamW	Adaptive Moment Estimation with Weight Decay
BTM	Battery Thermal Management
EMPC	Economic Model Predictive Control
ENMPC	Economic Nonlinear Model Predictive Control
EV	Electric Vehicle
FIM	Fisher Information Matrix
FTP-75	Federal Test Procedure 1975
HPIPM	High-Performance Interior Point Method
L4CasADi	Learning for CasADi Framework
LLM	Large-Language Model
MLP	Multilayer perceptron
MPC	Model Predictive Control
N-S	Nonlinear saturation function
NLP	Nonlinear Programming
NN	Neural Network
NTU	Number of Transfer Units
P-E	Product-Exponential term
QP	Quadratic Program
RMS	Root Mean Square
RMSE	Root-mean Squared Error
RNN	Recurrent Neural Network
RTI	Real-time Iteration
SINDy	Sparse Identification of Nonlinear Systems
SQP	Sequential Quadratic Programming
UKF	Unscented Kalman Filter

Nomenclature

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

Indices

e	Index for epoch
i	Index for input, neuron, or data sample
k	Index for sampling stage
l	Index for neural network layer
n	Index for discrete stage

Sets

$\mathcal{U}, \bar{\mathcal{U}}$	Constraint set for input, scaled constraint set
$\mathcal{U}_{\text{cct}}$	Input constraint set for cascaded tanks
$\mathcal{X}, \bar{\mathcal{X}}$	State constraint set, scaled constraint set
$\mathcal{X}_f, \bar{\mathcal{X}}_f$	Terminal set, scaled constraint set
$\mathcal{X}_{\text{cct}}$	State constraint set for cascaded tanks
$\mathcal{T}, \bar{\mathcal{T}}$	Constraint set for battery temperature, scaled set
$\bar{\mathcal{T}}_{\text{steady}}$	Operating region constraint set for battery
$\mathcal{T}_f, \bar{\mathcal{T}}_f$	Terminal constraint set for battery temperature, scaled set
$\mathcal{T}_{\text{cool}}$	Constraint set on coolant in and out temperatures

Parameters

A, B	Linearized dynamics matrices
A_1, A_2	Cross sectional area of tank 1 and tank 2

a_1, a_2	Cross section of connecting pipe 1 and out pipe 2
$\tilde{a}_1, \tilde{a}_2$	Mismatched cross sections for cascaded tanks
η	Learning rate
$\alpha_0, \alpha_1, \alpha_2, \alpha_3$	Fitting parameters for heat exchange effectiveness
$b, b^{[l]}$	Neural network bias
c	Confidence parameter for neural network
c_0, c_1, c_2, c_3	Polynomial fit coefficients for pump power
c_{bat}	Battery cell specific heat capacity
c_{cool}	Coolant specific heat capacity
D	Pump displacement
$\Delta t, \Delta T$	Time step size, sampling period
$\varepsilon_{\text{thresh}}$	Error threshold for trajectory loss
γ	Fitting parameter for environmental heat losses
γ_{Jac}	Gradient max norm clipping threshold
g	Gravitational constant
hA_{bat}	Heat transfer between battery and coolant
$k_{\text{conv,cct}}$	Unit conversion constant for pump in cascaded tanks
k_{nl}	Coefficient for heater mismatch
λ_{wd}	Weight decay hyperparameter
λ_{jac}	Jacobian weight for trajectory loss
m	Number of data samples
m_{bat}	Battery cell mass
N	Number of stages / prediction horizon
n	Number of inputs
N_{batch}	Batch size for neural network training
N_{epochs}	Number of epochs for training
N_{rolling}	Teacher forcing interval
$N_{\text{simul,end}}$	Sample of the simulation ending time
N_{steady}	Sample at which the controller reaches the reference
P_∞	Terminal cost matrix / Cost-to-go matrix
Q, R	Cost weighting matrices
$Q_{\text{cct}}, R_{\text{cct}}$	Weighting matrices for cascaded tanks
T	Finite horizon
T_{env}	Environment temperature

T_f	Time at the end of the planning horizon
$T_{\text{simul,end}}$	Ending time of the simulation
Θ_0	True parameters of dynamics
Θ_{nom}	Nominal parameters
Θ_{res}	Residual parameters
$\sigma_{T_{\text{bat}}}$	Gaussian disturbance / prediction noise
w	Weighting for trajectory loss
$w_i, W^{[l]}$	Neural network weight / weight matrix
W, W_{e}	Cost matrices for economic MPC
Z_l, Z_u	Weighting matrices for lower and upper slack variables
Z_l^e, Z_u^e	Weighting matrices for lower and upper terminal slack variables
$T_{\text{bat,scale}}, I_{\text{bat,scale}}$	Scaling parameters for temperature and current
$\omega_{\text{scale}}, Q_{\text{scale}}$	Scaling parameters for pump velocity and heater power

Variables

a	Output of the neuron
d	Disturbance trajectories
$f, \hat{f}$	True state dynamics / Estimated state dynamics
f_{nom}	Nominal control model dynamics
f_{res}	Residual dynamics
f_{SG}	Savitzky-Golay filtered estimated true dynamics
f_{cct}	Cascaded tanks dynamics
g_t	Gradient of the objective
h_1, h_2	Water level in tank 1 and tank 2
$I_{\text{bat}}, \bar{I}_{\text{bat}}$	Current / Normalized current
$J(\theta)$	Empirical risk
$L(\cdot)$	Loss function
$\mathcal{L}_{\text{der}}, \mathcal{L}_{\text{traj}}$	Derivative loss / Trajectory loss
ℓ	Stage cost
ℓ_{econ}	Economic stage cost
ℓ_{cct}	Stage cost for cascaded tanks
$\dot{m}_{\text{cool}}$	Mass flow rate of the coolant

NTU_{bat}	Number of thermal units
P_{pump}	Pump power
$Q, \bar{Q}$	Power supplied by the heating-cooling unit / Normalized
Q_{cool}	Heat supplied to battery through convective heat transfer
Q_{heat}	Power supplied by the heater
$Q_{\text{true}}, Q_{\text{nl}}$	True / Nonlinear heating power applied to the system
$r_{\text{x}}, r_{\text{u}}$	Reference for the state / Reference for the input
$r_{\bar{\text{T}}_{\text{bat}}}, r_{\bar{\text{u}}}$	References for normalized battery temperature / input
s_l, s_u	Concatenated lower and upper slack variables
$\bar{s}_{\text{T}_{\text{steady}},\text{l}}, \bar{s}_{\text{T}_{\text{steady}},\text{u}}$	Normalized slacks for operating region
$s_{\bar{\text{T}}_{\text{bat}},\text{l}}, s_{\bar{\text{T}}_{\text{bat}},\text{u}}$	Lower and upper slack variables for battery temperature
$s_{\text{T}_{\text{cool}},\text{l}}, s_{\text{T}_{\text{cool}},\text{u}}$	Lower and upper slack variables for coolant temperatures
t	Time
t_{steady}	Time the first controller reaches the reference
$T_{\text{bat}}, \bar{T}_{\text{bat}}$	Battery temperature / Normalized battery temperature
$T_{\text{bat,traj}}$	Trajectory of battery temperature
$\bar{T}_{\text{steady,min}}, \bar{T}_{\text{steady,max}}$	Normalized bounds for operating region
$T_{\text{cool,in}}, T_{\text{cool,out}}$	Temperature of the coolant flowing into / out of the battery
$u, \bar{u}$	Control input trajectories / Normalized inputs
$\bar{u}_{\text{ss}}, \bar{T}_{\text{bat,ss}}$	Steady state inputs and steady state temperature
u_{cct}	Control input for cascaded tanks
v	Speed of liquid through orifice
V	Volume
$V_\infty(x, u)$	Infinite-horizon cost function
$V_f(x)$	Cost-to-go / terminal cost
V_T	Finite-horizon cost
$x, \hat{x}$	State trajectories / Estimated state trajectories
x_{cct}	State for cascaded tanks
x_0, X_0	Initial condition / True regressors of dynamics
$X_{\text{nom}}, X_{\text{res}}$	Nominal regressors / Residual regressors
z	Weighted sum in neural network
ε	Prediction error
σ	Activation function
θ	Neural network parameters

$\omega, \bar{\omega}$	Pump angular velocity / Normalized pump angular velocity

Contents

List of Acronyms **ix**

Nomenclature **xi**

List of Figures **xix**

List of Tables **xxiii**

1 Introduction **1**
1.1 Objectives 4
1.2 Limitations 5
1.3 Policy Regarding Use of Large-language Models (LLMs) 5

2 Theory **7**
2.1 Model Predictive Control 7
2.2 Neural Networks and Multilayer Perceptron Networks 10
2.3 Neural Networks for Adaptive Model Predictive Control 13

3 Methods **17**
3.1 Modeling of a Battery Thermal Management System 17
3.2 Model Predictive Control Formulation of Reference Tracking for Battery Thermal Management 22
3.3 Model Predictive Control Formulation of Economic Energy for Battery Thermal Management 26
3.4 Neural Network Formulation for Learning Model Residuals in Battery Thermal Management 28
3.4.1 Formulation For Derivative Loss 30
3.4.2 Formulation for Trajectory Loss 32
3.4.3 Integration of Neural Networks for Residual Dynamics into a Model Predictive Control Framework 36
3.4.4 Tuning for Neural Networks in Adaptive Model Predictive Control 36
3.5 Formulation of Cascaded Tank Benchmark 39
3.5.1 Modeling of a Cascaded Tanks System 39
3.5.2 Model Predictive Control Formulation of Tracking Problem for Cascaded Tanks 41
3.5.3 Neural Network Architecture for Cascaded Tanks 43

4 Results **45**
4.1 Benchmark: Cascaded Tanks Using Derivative Loss and Trajectory Loss . . . 45
4.2 Reference Tracking for Battery Thermal Management Using Nominal and Adaptive Model Predictive Control . . . 48
4.2.1 Reference Tracking in Battery Thermal Management Systems with Trajectory Loss Adaptive Model Predictive Control . . . 50
4.2.1.1 Investigation into the Impact of the Residual Neural Network Trained on Trajectory Loss . . . 57
4.2.2 Reference Tracking in Battery Thermal Management Systems with Derivative Loss Adaptive Model Predictive Control . . . 61
4.2.3 Computation Times and Solver Errors . . . 65
4.3 Economic Model Predictive Control for Battery Thermal Management Using Nominal and Adaptive Controllers . . . 66

5 Discussion **77**
5.1 Summary of Adaptive Controller Results . . . 77
5.2 Evaluation of Adaptive Controller Method and Future Work . . . 78

6 Conclusion **81**

Bibliography **83**

List of Figures

2.1 Architecture of a generic neural network with n inputs, m outputs and k hidden layers. 11
2.2 Framework for adaptive model predictive control. The feedback nature of the controller can be seen as well. 14

3.1 Overview of battery thermal management system. The heating-cooling unit, pump and battery pack in turn consist of subsystems. 18
3.2 Current drawn from the FTP-75 drive cycle. 19
3.3 Nominal simulation using the model derived here for heating from 12.5 °C for 1000 s. 21
3.4 Fitted pump power compared to the true data. The fit follows the underlying data closely and is positive in the operating range of the pump. 27
3.5 Diagram of Cascaded Water Tanks. 40

4.1 Reference tracking for cascaded tanks on a matched model for water level in the first tank. Adaptive controller uses derivative as loss function. 45
4.2 Reference tracking for cascaded tanks on a matched model for water level in the second tank. Adaptive controller uses derivative as loss function. 46
4.3 Reference tracking for cascaded tanks on a mismatched model for water level in the first tank. Adaptive controller uses derivative as loss function. 46
4.4 Reference tracking for cascaded tanks on a mismatched model for water level in the second tank. Adaptive controller uses derivative as loss function. 47
4.5 Reference tracking for cascaded tanks on a mismatched model for water level in the first tank. Adaptive controller uses trajectory as loss function. 47
4.6 Reference tracking for cascaded tanks on a mismatched model for water level in the second tank. Adaptive controller uses trajectory as loss function. 48

4.7 Adaptive and nominal controller tracking behavior for a matched model in a heating scenario. The nominal controller tracks the reference closely while letting the temperature drop between current spikes. The adaptive controller undershoots the reference generally and also has a large spike around 3500 s. 50
4.8 Adaptive and nominal controller tracking behavior for heating P-E scenario. The nominal controller has a clear undershoot of 1 °C to 1.5 °C. The adaptive controller instead tracks the reference closely. . 51
4.9 Adaptive and nominal controller tracking behavior for a matched model in a cooling scenario. The nominal controller has a general undershoot of 0.25 °C. The adaptive shows a stronger undershoot, except for times 3000 s to 4000 s where it tracks the reference closely. 52
4.10 Adaptive and nominal controller tracking behavior for cooling P-E mismatch. The nominal controller has an offset of approximately 1.26 °C. The adaptive controller follows the reference closely. 53
4.11 Battery temperature when controlled by adaptive and nominal controllers under nonlinear output saturation. 54
4.12 Heating power signal generated by the controllers. During much of the period between 3000 s and 4000 s the battery temperature for the adaptive controller was below the reference, but cooling inputs are supplied. Between 5000 s and 6000 s the controller sometimes sends heating inputs even when in this region the battery temperature is above the reference. 54
4.13 Adaptive and nominal controller tracking behavior for a mismatched model in a heating scenario using with reduced input weighting. Due to the decreased weighting on inputs, the controller uses more control effort and thus the nominal exhibits a lower steady-state error. The adaptive controller has worse performance than when using lower weightings on inputs. 56
4.14 Simulation of battery temperature for 1000 s using nominal model, adaptive model with higher confidence and adaptive model used in control in a mismatched system. The nominal control model has an accumulating error that grows with each time step. 57
4.15 Residual value at different heating powers with the rest of inputs fixed, blue line, together with the Jacobian of the residual with respect to heating power, red line. 59
4.16 Residual value at different angular velocities with the rest of inputs fixed, blue line, together with the Jacobian of the residual with respect to angular velocities, red line. 59
4.17 Residual value at different current values with the rest of inputs fixed, blue line, together with the Jacobian of the residual with respect to current, red line. 60
4.18 Residual value at different temperature values with the rest of inputs fixed, blue line, together with the Jacobian of the residual with respect to temperature, red line. 60

4.19 Adaptive, trained on derivative loss, and nominal controller tracking behavior for a matched model in a heating scenario. The nominal controller tracks the reference closely while letting the temperature drop between current spikes. The adaptive controller undershoots the reference more than the nominal controller and also starts driving the temperature rapidly downward after 7000 s. 61

4.20 Adaptive, trained on derivative loss, and nominal controller tracking behavior for mismatched P-E model in a heating scenario. The nominal controller has a steady-state offset while the adaptive controller not only has this offset but even dips below it at 3700 s and again at 6000 s. 62

4.21 Adaptive, trained on derivative loss, and nominal controller tracking behavior for a matched model in a cooling scenario. The adaptive controller reaches the reference slightly faster but then undershoots it and subsequently overshoots it. The nominal controller has a steady-state offset below the reference. 63

4.22 Adaptive, trained on derivative loss, and nominal controller tracking behavior for a P-E mismatched model in a cooling scenario. The adaptive controller reaches the reference while the nominal stays above the reference. 63

4.23 Adaptive, trained on derivative loss, and nominal controller tracking behavior for a mismatched model in a cooling scenario with nonlinear saturation of the heater power. The adaptive controller reaches the reference while the nominal stays above the reference. 64

4.24 Economic model predictive control in a heating case for current turned on and off. 67

4.25 Economic model predictive control in a cooling case for current turned on and off. 68

4.26 Closed-loop temperature trajectories obtained using nominal and adaptive controllers for an economic formulation in a matched heating scenario. The performance of the nominal and adaptive controllers is nearly identical. The trajectory of the nominal controller is dashed for clarity. 70

4.27 Closed-loop temperature trajectories obtained using nominal and adaptive controllers for an economic formulation in a P-E mismatched heating scenario. The nominal controller stays outside of the target region for much of the simulation, while also overcompensating and shooting further into the target region than the adaptive controller. . 71

4.28 Closed-loop temperature trajectories obtained using nominal and adaptive controllers for an economic formulation in a matched cooling scenario. As opposed to heating the two controllers have noticeably different trajectories, highlighting that this is a more involved control problem. 72

4.29 Closed-loop temperature trajectories obtained using nominal and adaptive controllers for an economic formulation in a P-E mismatched cooling scenario. The nominal controller stays outside of the target region for much of the simulation, similar to the mismatched heating case, while the adaptive controller reaches the target region. 73
4.30 Closed-loop temperature trajectories obtained using nominal and adaptive controllers for an economic formulation in a N-S mismatched cooling scenario. The nominal controller stays outside of the target region for much of the simulation, similar to the mismatched heating case, while the adaptive controller reaches the target region. 74

List of Tables

3.1 Parameters for battery thermal management system. 19
3.2 Hyperparameters for neural network for tracking battery temperature using derivative loss. 30
3.3 Hyperparameters for neural network for reference tracking of battery temperature using trajectory loss. 33
3.4 Hyperparameters for neural network for thermal management using trajectory loss. 33
3.5 Description of tunable hyperparameters for neural networks. Starred * hyperparameters are only used in trajectory loss. 37
3.6 True parameter values. 42
3.7 Simulation parameters. 43
3.8 Hyperparameters for neural network for cascaded tanks using derivative loss. 44
3.9 Hyperparameters for neural network for cascaded tanks using trajectory loss. 44

4.1 Simulation parameters for reference tracking. 49
4.2 Results for battery temperature tracking using trajectory loss. 55
4.3 Fixed values for evaluating Jacobians 58
4.4 Results for battery temperature tracking using derivative loss. 65
4.5 Computation times for the nominal controller and the two adaptive controllers using different loss functions. 65
4.6 Simulation parameters for economic model predictive control. 69
4.7 Costs for economic controllers. 74

1
Introduction

Electric vehicles (EVs) are experiencing rapid growth in deployment, with the European Commission highlighting the role of EVs in reducing transport-related emissions [1]. However, widespread adoption is hampered by several factors, notably range anxiety [2]. The performance and range of lithium-ion batteries are highly dependent on the temperature at which the battery operates. Low temperatures affect performance negatively [3], making operation within an appropriate temperature range important. In [4], the preferred operating range is identified as between 20 – 40 °C. Moreover, it is necessary to regulate the battery temperature to ensure compliance with safety standards, with the temperature kept below 50 °C to avoid thermal runaway and above 10 °C to avoid battery degradation. Maintaining these boundaries ensures optimal performance, safety, and maximum lifetime. Consequently, thermal energy management is integral to the performance of batteries in electric vehicles, with [5] highlighting the need for advanced control methods.

Model predictive control (MPC) is an optimization-based control strategy that computes control inputs by minimizing a cost function subject to system dynamics and operational constraints [6]. It achieves this by approximating an infinite-horizon problem with a finite-horizon problem supplemented by a cost-to-go that accounts for future behavior. This makes MPC suitable for battery thermal management, where the controller must balance temperature regulation, energy consumption, actuator constraints, and safety limits. MPC has previously been applied to battery thermal management in [7], [8] and [9]. A common challenge in implementing MPC for EV applications is the requirement of accurate modeling of dynamics. Models for EV battery systems are complex, consisting of nonlinear coupled partial differential equations [10]. Consequently, the dynamics of an EV battery system are subject to nonlinearities, uncertainties in model parameters, measurement errors, and unmodeled dynamics.

A model can be obtained using system identification [11], where data from the system is collected and a model is constructed using methods such as least-squares parameter identification and empirical transfer function estimates. More recent approaches to system identification have proposed using neural networks, particularly to model nonlinear dynamics [12]. System identification and modeling are traditionally performed offline. However, a model developed offline will typically only be valid for the conditions under which the data was collected, and if the system changes there will be a mismatch between the model and the system.

To capture system changes, parameter identification and modeling can be performed online, a topic studied in the field of adaptive control [13] and, for MPC applications, in learning-based model predictive control [14]. In [14], three main areas of learning-based model predictive control are highlighted. These areas are learning the system dynamics, learning the controller design, e.g. terminal components, tuning of cost and constraints, and MPC for safety learning. Furthermore, the need to ensure that learning-based MPC can be used in real-time through effective approximations and computational speed-ups is mentioned as an important area for future research. The first area is especially relevant in this context, as the system dynamics are complex. The general setting for learning system dynamics presented in [14] is to express the model as a nominal system model plus an additive learned term. The additive learned term, or learned residual, is then identified online based on collected data.

In [15], the approach of expressing the model with a known model and an additive learned residual term was used. The additive term represented unknown model dynamics. The known dynamics were obtained by physical modeling, whereas a neural network was used to represent the unknown dynamics. This forms the basis of an adaptive MPC scheme in which residual dynamics are computed online. The results demonstrated an improvement over nominal MPC methods, both in computational time and error reduction. The pipeline for adaptive training and solving the optimal control problem was based on the Learning for CasADi Framework (L4CasADi) developed in [16], [17]. This framework allows neural network models to be used efficiently in numerical optimization.

In [18], a nonlinear model predictive control policy applied to the cascaded two-tank system is presented where the system dynamics are considered unknown. Instead, the model was derived by a recurrent neural network (RNN) learning framework that used measurement data to approximate the system dynamics. Further, the model, constraints, stage cost, and terminal costs were parametrized. The parametrized model predictive control problem was then used in a model predictive controller to calculate optimal inputs, and the parameters updated online.

In [19], a physics-informed neural network was used for the problem of electric water heating. A physics-informed neural network was defined there as a neural network that was trained on both past data and physical laws. The physical laws were added into the loss function, for instance through boundary conditions and state evolutions, to ensure that the network follows physical laws. Furthermore, this approach allows training of accurate neural networks even when the amount of training data is comparatively low.

Model mismatch can not only be addressed by adapting the model. In [20], it was demonstrated that using data-driven economic nonlinear MPC (ENMPC) can obtain optimal control laws even when the model is wrong [20], by modifying parameters in the stage and terminal cost. The parameters were modified using reinforcement

learning. This approach was applied for trajectory tracking of autonomous surface vehicles in [21], together with updating model parameters, and was found to increase performance compared to a mismatched model.

In [22], a method for safety-driven battery charging utilizing adaptive MPC with real time parameter identification is proposed. This paper introduces the concept of the Fisher Information Matrix (FIM) as a method for parameter identification. FIM is defined as the expected value of the Hessian matrix of the log-likelihood function with respect to the unknown parameters. Maximizing this quantity improves parameter identification. This concept is utilized in the online parameter updating of the controller. However, a limitation of this method is that if there are uncertainties in the nominal model parameters, then the offline design phase will introduce suboptimal trajectories, since the FIM is maximized based on these nominal model parameters. This highlights the need for more robust parameter estimation.

Another method that does not use reinforcement learning or neural networks is presented in [23]. An economic MPC formulation is used for a nonlinear system, together with least-mean squares adaptation of parameters and tube-based MPC to ensure constraint satisfaction. Constraint satisfaction is ensured both with respect to varying parameters and to external disturbances. Performance guarantees are given under the assumption of finite-energy disturbances and finite parameter variations.

In [24], a method for representing the dynamics of unstable systems is presented by utilizing recurrent neural networks (RNNs) where a neural adaptive MPC method is introduced. Essentially, a neural network provides state estimates that are used for the observer as well as parameter estimations. This is achieved by using a joint Unscented Kalman Filter (UKF) which updates the model by updating the last linear layer. The updated parameters and states are then input into the MPC formulation.

In [25], a data-driven control scheme is presented that integrates sparse identification of nonlinear systems (SINDy) with feedback linearization. SINDy determines the prominent contributing terms in the model, and these are used for state feedback realization. The state-observation problem is resolved by using a neural network to estimate unmodeled dynamics. This scheme is used for an inverted pendulum and a single-link robot manipulator. The results show improved tracking and convergence with only minimal additional computational overhead.

For applications in thermal systems, neural networks have been used in [26] to construct an adaptive control scheme for refrigeration units. Not only model parameters and control were updated, but also the characteristics of the neural network itself. The neural network characteristics that were adaptive were the network's weights, and the center points and widths of the basis functions. The results were promising, showing an improvement over traditional neural networks in both simulation and experimental trials.

Another application of neural networks in modeling thermal systems was presented in [27], where a neural network is used to adapt a parameter-varying MPC framework used to control the temperature and energy consumption of a building via its heating, ventilation, and air conditioning system. The framework improves parameter-varying linear control by using a physics-inspired Bayesian neural network to estimate the parameter-varying linear control model. This is achieved by the neural network providing uncertainty estimates that are used in online training to update parameters. The results demonstrate 28.39 % higher accuracy than typical linear MPC frameworks and result in 36.23 % better control performance without increasing complexity.

Overall, the literature shows that learning-based MPC can improve control performance under model mismatch, either by learning system dynamics, adapting model parameters, modifying cost functions, or incorporating safety-oriented uncertainty handling. Using learning-based residual models for adaptive control appears promising due to their ability to adaptively update models and to accurately capture nonlinear dynamics. Expressing the dynamics with a physically derived nominal model allows physical knowledge to be incorporated, while an adaptive residual term captures the unmodeled dynamics and varying parameters. Motivated by these developments, online learning of residual dynamics is employed in this thesis. The residual dynamics are modeled using lightweight neural networks. A learning-based model predictive control framework is developed and tested on a cascaded tanks benchmark for reference tracking. The learning-based framework is then applied to both reference tracking and economic model predictive control in a battery thermal management system.

1.1 Objectives

The objective of this thesis is to investigate whether a neural-network residual model can improve the performance of MPC controllers under model mismatch. More specifically, the following research questions are addressed:

1. How well does an adaptive MPC scheme perform for reference tracking using controller cost and root mean squared tracking error as a metric?
2. How well does an adaptive MPC scheme perform with an economic cost function as a metric?

Evaluating whether neural network-learned system dynamics and can be used effectively on a real-world problem is particularly interesting, as these neural network methods are relatively new and unexplored in practical MPC implementation. The focus in this work is on evaluating the method rather than developing a full vehicle thermal management controller. The cascaded-tank system is used as a benchmark nonlinear system, to which a reference tracking model predictive controller is applied. The battery thermal management case is used to study the method in a more application-oriented setting, where both reference-tracking MPC and economic MPC are considered to evaluate the adaptive framework under different control objectives.

1.2 Limitations

A primary limitation for both cascaded tank and battery thermal management is that the data will be generated by simulation and not provided by hardware measurements. This means that computational metrics will only be estimates and are not necessarily indicative of hardware performance. Additionally battery thermal management refers to only the battery coolant loop subsystem in [28].

Another limitation is due to the limited size of the neural network. Since the neural network is updated online, it is impossible to create a deep neural network (i.e. many layers) and thus due to computational times the network will be limited to only a few layers. Hence there is a limitation imposed upon the learned residual dynamics. Only one drive cycle, the Federal Test Procedure 1975, will be used. Performance under other drive cycles will therefore not be tested. Perfect state observation is assumed, together with the controller knowing the drive cycle beforehand and thus knowing the current drawn by the battery over the prediction horizon, to limit sources of errors.

1.3 Policy Regarding Use of Large-language Models (LLMs)

During the course of this thesis, the use of large-language models (LLMs) has only been employed for the purposes of language, formatting, and debugging code. All output from LLMs has been checked. The usage of LLMs has been performed with the approval of our supervisor and was conducted in accordance with Chalmers' master's thesis policy regarding AI use. The authors take full responsibility for the employment of LLMs.

2

Theory

This chapter presents the theoretical background required for the thesis. First, optimal control and its implementation through model predictive control are introduced. Second, neural networks, with a focus on multilayer perceptron (MLP) networks, are reviewed. Finally, the use of neural networks in adaptive model predictive control frameworks is discussed.

2.1 Model Predictive Control

A nonlinear system can be described in terms of state trajectories x, control input trajectories u and disturbance trajectories d. The state evolves according to the associated dynamics. For a nonlinear system at time t this can be described by a nonlinear function

$$\dot{x}(t) = f(x(t), u(t), d(t)). \tag{2.1}$$

An optimal control problem consists of finding inputs and corresponding state trajectories, subject to the state dynamics, state constraints $x \in \mathcal{X}$, an initial condition x_0 and input constraints $u \in \mathcal{U}$ to minimize a cost function $V_\infty(x, u)$ [6]. The form of the cost function determines what type of problem is solved. For reference tracking, the cost is of the form

$$V_\infty(x, u) = \int_0^\infty \left(\begin{bmatrix} x(t) \\ u(t) \end{bmatrix} - \begin{bmatrix} r_\mathrm{x}(t) \\ r_\mathrm{u}(t) \end{bmatrix} \right)^T \begin{bmatrix} Q & 0 \\ 0 & R \end{bmatrix} \left(\begin{bmatrix} x(t) \\ u(t) \end{bmatrix} - \begin{bmatrix} r_\mathrm{x}(t) \\ r_\mathrm{u}(t) \end{bmatrix} \right) dt \tag{2.2}$$

with $r_\mathrm{x}(t)$ being the reference for the state, $r_\mathrm{u}(t)$ the reference for the input with Q and R being positive definite weighting matrices. The state reference determines which states should be tracked, and references on the input are usually set as the inputs required to keep this state. A higher Q relative to R means that tracking the state is considered more important than tracking the input. The requirement that the matrices are positive definite implies that the cost penalizes deviations from $x(t) = r_\mathrm{x}(t)$ and $u(t) \to r_\mathrm{u}(t)$ and that the cost is minimized when the state and input converge to their references. The optimization formulation of the reference

tracking problem becomes

$$\begin{aligned} &\min_{x,\ u} V_\infty(x,u) \\ \text{s.t. } &\dot{x}(t) = f(x(t),u(t),d(t)) \\ &x \in \mathcal{X} \\ &u \in \mathcal{U}. \\ &x(0) = x_0. \end{aligned} \tag{2.3}$$

The system is generally assumed to continue for infinite time and therefore the optimization problem (2.3) has infinite state trajectories and input trajectories. For systems where the state dynamics are linear, that have no additional constraints and for which the Algebraic Riccati equation converges the solution to (2.3) can be calculated analytically. However, for systems with constraints or nonlinear dynamics the problem (2.3) needs to be solved using an optimizer. In infinite-horizon form, this will result in infinite optimization variables, which is computationally infeasible.

Model predictive control (MPC) is a control method for solving optimal control problems that uses a finite horizon T together with a stage cost $\ell(x,u)$ and a cost-to-go $V_f(x)$ to approximate an infinite-horizon cost while remaining computationally feasible [6]. Defining the cost from time $t=0$ to $t=T$ as $V_T = \int_0^T \ell(x(t),u(t))\,dt$ then the sum

$$V_T + V_f \tag{2.4}$$

is intended to approximate the infinite-horizon cost. Since the integral from 0 to T is finite this cost V_T can be calculated exactly as

$$V_T = \int_0^T \ell(x(t),u(t))\,dt \tag{2.5}$$

where the stage cost $\ell(x(t),u(t))$ is defined as

$$\ell(x(t),u(t)) = \left(\begin{bmatrix} x(t) \\ u(t) \end{bmatrix} - \begin{bmatrix} r_{\mathrm{x}}(t) \\ r_{\mathrm{u}}(t) \end{bmatrix}\right)^T \begin{bmatrix} Q & 0 \\ 0 & R \end{bmatrix} \left(\begin{bmatrix} x(t) \\ u(t) \end{bmatrix} - \begin{bmatrix} r_{\mathrm{x}}(t) \\ r_{\mathrm{u}}(t) \end{bmatrix}\right) \tag{2.6}$$

for tracking. The cost-to-go needs to be defined correctly to ensure a good approximation of the infinite-horizon cost. For reference tracking and under linear dynamics this can be done with the solution to the algebraic Riccati equation P_∞ as a weighting matrix. For nonlinear problems this is approximated using the linearized system [6]. Note that the cost-to-go is positive definite if the algebraic Riccati equation has a solution. At the final time T, only the state is a decision variable and the cost-to-go becomes

$$V_f(x(T)) = (x(T) - r_{\mathrm{x}}(T))^T P_\infty (x(T) - r_{\mathrm{x}}(T)). \tag{2.7}$$

Since the problem will be computed only until time T there also needs to be a requirement on the state at T, namely that the state ends in a feasible region for the optimization problem. Denote the set of feasible states as $\mathcal{X}_f$, where $\mathcal{X}_f$ is called

the terminal set. With the stage cost, cost-to-go and the terminal set defined the model predictive control problem can be defined as

$$\begin{aligned} \min_{x,u} & \int_0^T \ell(x(t), u(t))\, dt + V_f(x(T)) \\ \text{s.t. } & \dot{x}(t) = f(x(t), u(t)) \\ & x(t) \in \mathcal{X} \\ & x(T) \in \mathcal{X}_f \\ & u(t) \in \mathcal{U} \\ & x(0) = x_0 \end{aligned} \tag{2.8}$$

which is a nonlinear programming (NLP) problem and can be solved using sequential quadratic programming (SQP). In SQP the NLP is approximated by a quadratic program (QP) [29]. These in turn can be solved using a quadratic solver such as the High-Performance Interior Point Method (HPIPM) [30]. To make computations quicker a real-time iteration (RTI) formulation of the SQP method can be used, known as SQP-RTI [31].

The problem (2.8) is defined in continuous time. For it to be computationally solvable it needs to be discretized into time steps of size $\Delta t = T/N$ with N being the number of stages. The discretization involves splitting the problem into discrete stages $0,\ 1, \ldots, N$ with each stage corresponding to a sample time $0,\ \Delta t,\ \ldots, N\Delta t$. In each stage the input and state are assumed constant. The dynamics are discretized by integrating them using a suitable integration method, for instance a Runge-Kutta integration

$$x(n+1) = x(n) + \mathrm{RK}(x(n), u(n)). \tag{2.9}$$

The cost will be replaced by a sum. The discretized problem is then a computationally feasible implementation of (2.3). When solving the problem the state and input trajectory for N steps will be obtained (x^*, u^*). Only the first control will be applied $u^*(0)$. Then at the next time step the problem is solved again starting from the new state obtained after applying the input. This manner of applying control, called the receding horizon principle, is more expensive than solving the optimization once and applying all N inputs. However, it implements feedback into the control, since the initial state at each optimization will be read from the plant.

The formulation for reference tracking was presented above. However, there are other possible cost functions which will yield other behavior. Tracking is not always the desired behavior. In some systems the goal is to minimize a cost that has some economic meaning. In these cases the cost functions can be defined in such a way as to minimize the economic cost instead of the target error. Define the economic stage cost as $\ell_{\text{econ}}(\cdot)$ and the economic cost-to-go as $V_{f,\text{econ}}(\cdot)$. The cost functions need to be defined to have a lower bound, otherwise there will be no solution to the optimization problem. However, the economic costs do not have the requirement that they are positive definite. This means that the cost is not necessarily decreasing when the state or input approaches the equilibrium. Instead the optimal state and

input might consist of cyclical trajectories. In turn, the cost-to-go is not necessarily decreasing which removes guarantees on recursive feasibility and stability.

2.2 Neural Networks and Multilayer Perceptron Networks

Neural networks are machine learning models that map inputs to outputs. The mapping is accomplished by stacking computational units called neurons into layers. Neurons perform a linear transformation of inputs in the form of a weighted sum, and then pass these through a transformation called an activation function.

$$z = \sum_{i=1}^{n} w_i x_i + b \tag{2.10}$$

$$a = \sigma(z) \tag{2.11}$$

where subscript i denotes the i-th input, x_i denotes a neuron's input, w_i denotes the weight, b denotes the bias, z denotes the weighted sum, $\sigma(z)$ denotes the activation function as a function of the weighted sum, and a denotes the output of the neuron. The neuron computes the sum over a span of n inputs.

A neural network is composed of stacks of neurons organized into layers. The first layer is the input layer and is composed of the input data that we wish to map to the output. The final layer is the output layer which is composed of the network's prediction. In between these layers are intermediate layers called hidden layers and this is where neurons perform the weighted sum and activation calculations. An arbitrary number of hidden layers can be placed in between the input and output layers. Figure 2.1 is a visual representation of a generic neural network.

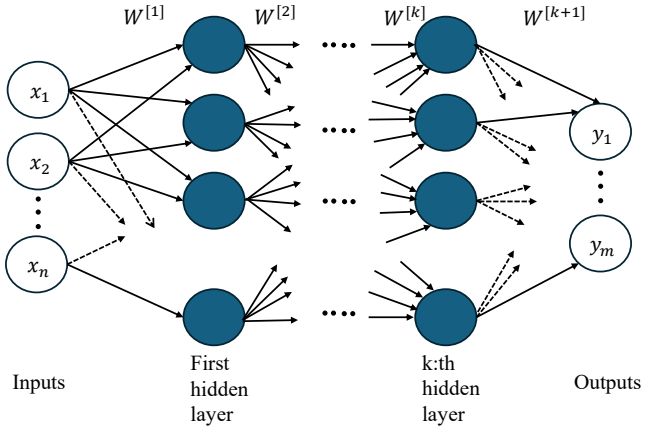


Figure 2.1: Architecture of a generic neural network with n inputs, m outputs and k hidden layers.

The activation function of a neural network calculates the neuron output via a transformation of the weighted sum of inputs. This essentially translates the weighted sum into some kind of prediction of the network. Typically neural networks employ step, ReLU, sigmoid, or hyperbolic tangent functions as activation functions, depending upon the application.

Neural networks learn the weights and biases from data by comparing the network's prediction of output with the true output data. This is done by means of a loss function, shown in equation (2.12).

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} L(f(x^i; \theta), y^i) \tag{2.12}$$

θ denotes the parameters to be optimized, $J(\theta)$ denotes the empirical risk and $L(f(x^i; \theta), y^i)$ denotes the loss function for data at sample i. The loss function essentially gauges how closely the neural network's prediction matches the true data. The goal is for the neural network to minimize this loss, which is equivalent to minimizing the difference between predictions and true data. To minimize this loss the neural network uses a backpropagation algorithm. This algorithm works in four distinct steps.

1. Forward pass: inputs go through the network and an output prediction is computed.
2. Loss calculation: The loss function computes the difference between prediction and true data.
3. Backward pass: the error is propagated backwards through the network. Each neuron adjusts weight and bias to account for the error.
4. Parameter update: the weights and biases are updated to reduce error.

The key to the optimization process is the backwards pass. The backwards pass is performed by gradient descent. Gradient descent uses the chain rule from calculus to determine how inputs change outputs, by computing derivatives backwards through the network. To perform gradient descent it is necessary to compute:

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta L(f(x^i; \theta), y^i) \tag{2.13}$$

where ∇_θ denotes the gradient with respect to the parameters θ. This is done through the chain rule. Suppose there is a three layer network composed of one neuron each where x denotes the input layer neuron, z the hidden layer neuron, and y the output layer neuron. Then to determine the derivative of y with respect to x it is necessary to compute:

$$\frac{dy}{dx} = \frac{dy}{dz} \cdot \frac{dz}{dx} \tag{2.14}$$

In a more generic case, this is performed for all neurons within a neural network. Generally speaking, the updated weights are obtained via the following equation:

$$W^{[l]} = W^{[l]} - \eta \cdot dW^{[l]} \tag{2.15}$$

$$b^{[l]} = b^{[l]} - \eta \cdot db^{[l]} \tag{2.16}$$

Where $W^{[l]}$ and $b^{[l]}$ denote the weight and bias in layer l, $dW^{[l]}$ and $db^{[l]}$ denote the updates to the weigh and bias in the layer, and η denotes the learning rate of the parameters. The learning rate is an extremely important hyperparameter in the neural network as it determines the rate of convergence to a solution and computational time required to train a neural network. Typically this gradient descent is performed not for the totality of the dataset, but instead for small samples of data called batches are selected to pass through the neural network for a given cycle called an epoch. Batching increases computational efficiency. It is another important hyperparameter for a neural network that requires tuning.

An important concept in neural networks is regularization. Regularization is a set of methods used to reduce overfitting. A particularly useful method is L2 regularization, also known as weight decay. Weight decay penalizes large weights by adding the square of the magnitude of coefficients to the loss function.

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} L(f(x^i; \theta), y^i) + \frac{\lambda}{2m} \cdot \|W\|_2^2 \tag{2.17}$$

Where λ is a hyperparameter that is tuned. The additional cost on the weight matrix W is determined by the following formula.

$$\|W\|_2^2 = \sum_{j=1}^{n} w_j^2 = W^T \cdot W \tag{2.18}$$

Here w_j denotes an element of the weight matrix W. This norm is known as the Frobenius norm and is used to determine the magnitude of a signal. By adding this

quantity to the cost function, large weights are penalized via the minimization of the cost function. Subsequently the gradient descent needs to account for this quantity.

$$W^{[l]} = W^{[l]} - \eta \cdot (dW^{[l]} + \frac{\lambda}{m} W^{[l]}) \tag{2.19}$$

Regularization helps to generalize the network by punishing large weights and therefore reducing overfitting. The result is a more robust architecture.

The choice of an optimizer for the neural network is important for the overall network training. A commonly used optimizer is Adaptive Moment Estimation with Weight Decay (AdamW). AdamW is a robust method for performing gradient descent because it combines the concept of root mean square (RMS) propagation, gradient descent with momentum, and weight decay. The pseudocode for AdamW is available at [32].

Multilayer perceptron (MLP) networks are a type of feedforward neural networks that consist of neurons that are connected via nonlinear activation functions. These activation functions are typically chosen to be continuous. Multilayer perceptrons are often employed in deep machine learning and can be used for a diverse set of applications. An MLP consists of three or more layers where the neurons have nonlinear activation functions. MLPs are powerful because they can approximate any continuous function. A requirement is that input features be numeric.

A diverse set of activation functions can be used. A commonly used activation function is the hyperbolic tangent, This function maps inputs between -1 and 1 while being centered on 0. This provides fast training, but because the function levels off past -1 and 1 the activation can saturate for very high or low values. This means the gradient becomes quite small, which translates to the vanishing gradient problem for deep networks. However, for networks with few layers this is less of a problem.

2.3 Neural Networks for Adaptive Model Predictive Control

An overview of the adaptive model predictive control framework used here can be seen in Figure 2.2.

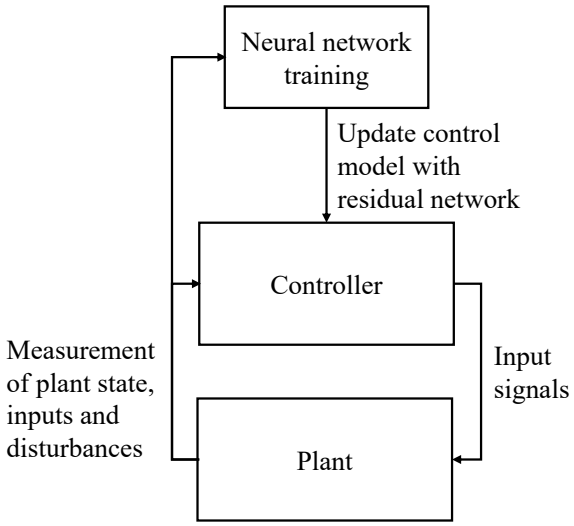


Figure 2.2: Framework for adaptive model predictive control. The feedback nature of the controller can be seen as well.

The controller has feedback with the plant where measurements from the plant are used in the controller. On top of this feedback architecture, a neural network is used to learn the residual of the controller model and then update the control model. Keeping the neural network training parallel to the controller ensures that, even if the neural network training takes longer than sampling time of the controller, the controller will still produce an output. Denote the true dynamics for a system, corresponding to the plant, as $f(X_0, \Theta_0)$ where X_0 are all regressors of the dynamics and Θ_0 are the true parameters of the dynamics. A model should correspond as well as possible to the true dynamics, but obtaining such a model can be time-consuming or even impossible. The model used in model predictive control is therefore a simplified model where not all regressors and parameters are included. Denote this model as the nominal model $f_{\text{nom}}(X_{\text{nom}}, \Theta_{\text{nom}})$ with nominal regressor X_{nom} and nominal parameters Θ_{nom}. Note that the nominal regressor does not necessarily include all true regressors, similarly the nominal parameters do not include all parameters. The nominal model therefore has a mismatch that can be modeled using the residual dynamics $f_{\text{res}}(X_{\text{res}}, \Theta_{\text{res}})$ with the residual regressors X_{res} and residual parameters Θ_{res}. The residual regressor and parameters do not have to be the same as for the nominal dynamics. The goal with the residual model is to try to accurately approximate the true dynamics as

$$f(X_0, \Theta_0) \approx f_{\text{nom}}(X_{\text{nom}}, \Theta_{\text{nom}}) + f_{\text{res}}(X_{\text{res}}, \Theta_{\text{res}}). \tag{2.20}$$

where $X = \begin{bmatrix} X_{\text{nom}} & X_{\text{res}} \end{bmatrix}$ and $\Theta = \begin{bmatrix} \Theta_{\text{nom}} & \Theta_{\text{res}} \end{bmatrix}$. Denote the nominal control model together with the learned residual as

$$\hat{f}(X, \Theta) = f_{\text{nom}}(X_{\text{nom}}, \Theta_{\text{nom}}) + f_{\text{res}}(X_{\text{res}}, \Theta_{\text{res}}) \tag{2.21}$$

which is called the adaptive control model.

Residual dynamics can be modeled in different ways. One possibility is to use a neural network. In a neural network the input features will be X_{res} and the model parameters Θ_{res}. The neural network should be trained to approximate the residual. There are different ways of doing this. One method is using derivative loss, used in [15] and [16]. The derivative loss is calculated

$$\mathcal{L}_{\text{der}}(X, \Theta, X_0, \Theta_0) = (f(X_0, \Theta_0) - \hat{f}(X, \Theta))^2. \tag{2.22}$$

By finding the neural network $f_{\text{res}}(X_{\text{res}}, \Theta_{\text{res}})$ that minimizes (2.22) the residual dynamics will be modeled.

Another method is using trajectory loss, which is obtained by integrating the estimated dynamics for the state $\hat{x}$

$$\hat{x}(t) = x_0 + \int_0^T \hat{f}(X, \Theta)\, dt \tag{2.23}$$

where x_0 is the initial state and whose true value is assumed to be known and T the final time, to obtain the trajectory $\hat{x}(t)$. Denote the true trajectory for the state as $x(t)$. The squared trajectory loss can be written as

$$\mathcal{L}_{\text{traj}}(x(t), \hat{x}(t)) = (x(t) - \hat{x}(t))^2. \tag{2.24}$$

This is essentially a prediction error method, well studied in system identification [11]. Trajectory loss has been used extensively in adaptive model predictive control [33] and in the field of system identification [34]. Applications include identifying models for cascaded tanks [18], unmanned surface vehicles [35] and vehicle-dynamics models [36]. Minimizing (2.24) corresponds to

$$\min \mathcal{L}_{\text{traj}}(x(t), \hat{x}(t) = \min \left(x(t) - \left(x_0 + \int_0^T \hat{f}(X, \Theta)\, dt \right) \right)^2 = \tag{2.25}$$

$$= \min \left(x(t) - \left(x_0 + \int_0^T f_{\text{nom}}(X_{\text{nom}}, \Theta_{\text{nom}}) + f_{\text{res}}(X_{\text{res}}, \Theta_{\text{res}})\, dt \right) \right)^2. \tag{2.26}$$

Minimizing (2.25) is not necessarily the same as finding the underlying residual dynamics, since

$$\min \left(x(t) - \left(x_0 + \int_0^T \hat{f}(X, \Theta)\, dt \right) \right)^2 \neq \min \left(f(X_0, \Theta_0) - \hat{f}(X, \Theta) \right)^2 \tag{2.27}$$

which comes from how integration is not injective. This can easily be seen in a proof by counterexample. Assume that trajectory loss is to be minimized on the interval $[0, 1]$. Assume that the true dynamics are given by

$$f_{0,\text{ex}}(x(t)) = 1 + \frac{\cos t}{3 \cdot \sin 1} \tag{2.28}$$

with initial condition $x_{0,\text{ex}} = 0$ this gives the trajectory

$$x_{\text{ex}}(t) = x_{0,\text{ex}} + \int_0^1 1 + \frac{\cos t}{3 \cdot \sin 1}\, dt = 1 + \frac{1}{3}. \tag{2.29}$$

Let the nominal dynamics be

$$f_{\text{nom,ex}}(x(t)) = 1. \tag{2.30}$$

The trajectory loss (2.24) is obviously minimized by finding the true residual $\frac{\cos t}{3 \cdot \sin 1}$. However, for the residual

$$f_{\text{res,ex}}(x(t)) = t^2 \tag{2.31}$$

the trajectory loss will also be minimized since

$$\mathcal{L}_{\text{traj,ex}} = \left(1 + \frac{1}{3} - (x_{0,\text{ex}} + \int_0^1 1 + t^2\, dt)\right)^2 = \left(1 + \frac{1}{3} - \left(1 + \frac{1}{3}\right)\right)^2 = 0 \tag{2.32}$$

and since the loss (2.24) is greater or equal to zero for a convex problem this is also a minimizer. Therefore, minimizing (2.24) is not guaranteed to give the true residual. This has the consequence that outside of the minimization interval the trajectories will diverge. This is a limitation of the trajectory loss. To combat this, either the horizon for the trajectory loss T must be increased or the training has to be updated often.

3

Methods

In this chapter the modeling of a battery thermal management (BTM) system is presented, together with model predictive control formulations for reference tracking and minimizing economic costs in this thermal management system. Further, the theory in Section 2.3 is applied to the problem of finding the residual dynamics in the model of the BTM. The cascaded tank benchmark is also introduced, including modeling, reference tracking model predictive control and learning residual dynamics.

3.1 Modeling of a Battery Thermal Management System

The battery thermal management (BTM) system under consideration here consists of a battery pack, a heating-cooling unit, a pump and a controller, for an electric vehicle driving according to a drive cycle. Additionally, heat losses to the environment are included. The model is based on [28]. The controller sends inputs to the pump and heating-cooling unit to control the temperature of the battery pack T_{bat}. The inputs are the pump angular velocity, ω, and the power supplied by the heating-cooling unit Q. The heating-cooling unit heats or cools the coolant that flows through the system and resulting in heating or cooling of the battery through convection in a cooling plate that sits above the battery. The battery pack consists of four battery cells. During driving current is drawn from the battery to power the electric vehicle and this heats the battery from resistive losses due to the batteries internal resistance R_{bat}. The current drawn is taken from a drive cycle. For an overview of the battery thermal management system see Figure 3.1.

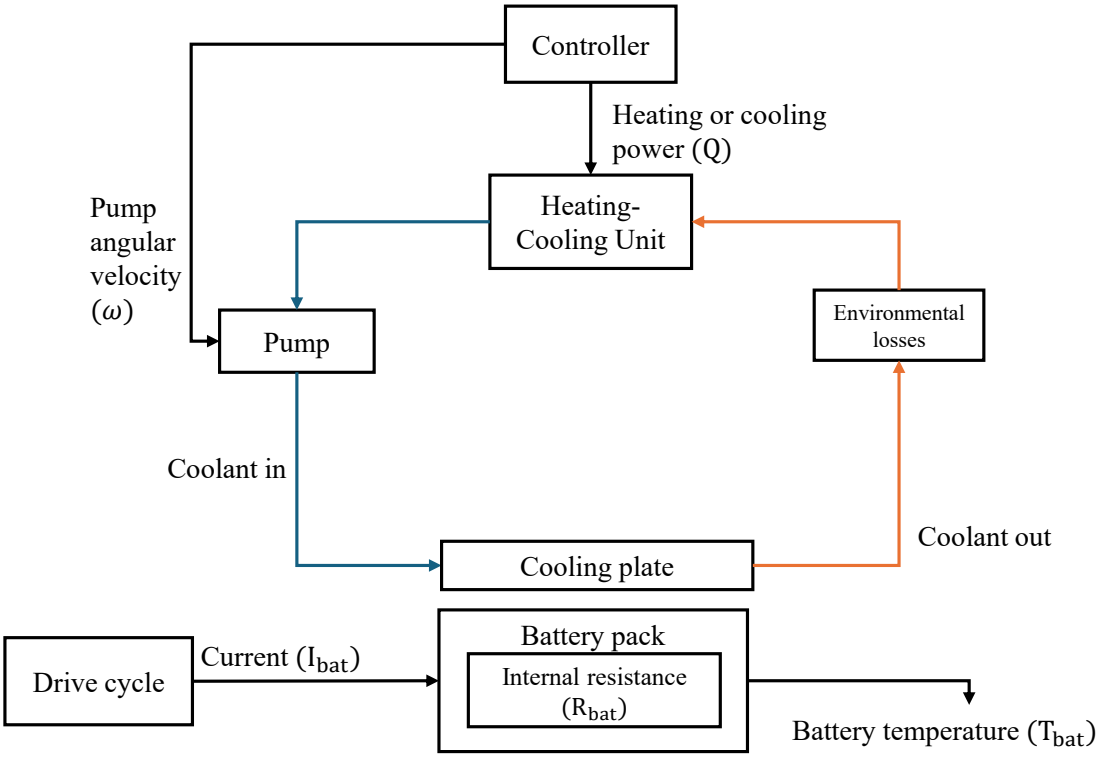


Figure 3.1: Overview of battery thermal management system. The heating-cooling unit, pump and battery pack in turn consist of subsystems.

The pump is driven by a shaft whose angular velocity is the input, and this input is affected by a one-second input delay. The heating-cooling unit is modeled with a heat flow rate source. The heat flow rate source heats a thermal mass while transferring energy to the coolant through convective heat transfer. The heat input can both be positive, modeling heating, and negative, modeling cooling and is also affected by a one second delay. For simulating driving, the Federal Test Procedure 1975 (FTP-75) drive cycle is used [37]. This drive cycle is meant to model city driving. The FTP-75 was originally defined for combustion engines, but the dataset available in the model also includes the corresponding current. The cycle includes data for 2474 s. To allow for longer simulations, the data is repeated 10 times, which gives a data set that can be used for a maximum of roughly 6 hours and 52 minutes. The current profile is presented in Figure 3.2

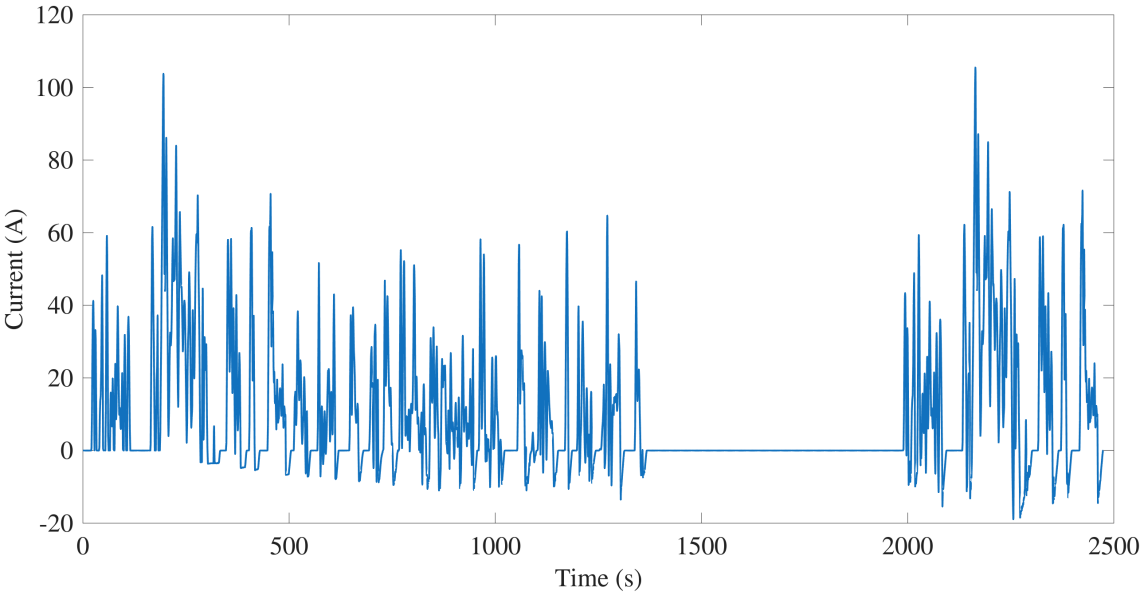


Figure 3.2: Current drawn from the FTP-75 drive cycle.

This drive cycle presents a challenging setting for the controller, since the current varies strongly.

For parameters of the battery thermal management system see Table 3.1.

Table 3.1: Parameters for battery thermal management system.

Parameter	Symbol	Value
Battery cell mass	m_{bat}	10 kg
Battery cell specific heat capacity	c_{bat}	795 J/kgK
Battery cell resistance	R_{bat}	0.0109 Ω
Heat transfer between battery and coolant	hA_{bat}	2500 W/C°
Coolant specific heat capacity	c_{cool}	3.5 kJ/kgK
Coolant density	ρ	1060 kg/m^3
Pump displacement	D	40 cm^3/rev

Next, the mathematical modeling of the battery thermal management system is presented. The heat generation comes from three different terms. Heat is generated by the current according to

$$R_{\text{bat}} I_{\text{bat}}^2, \tag{3.1}$$

which is Joule's law of heating. The coolant supplies heat to the battery through convective heat transfer in the cooling plate

$$Q_{\text{cool}} = \dot{m}_{\text{cool}} c_{\text{cool}} \left(T_{\text{cool,out}} - T_{\text{cool,in}} \right) \tag{3.2}$$

where $\dot{m}_{\text{cool}}$ is the mass flow rate of the coolant, $T_{\text{cool,in}}$ is the temperature of the coolant flowing into the battery and $T_{\text{cool,out}}$ is the temperature of the coolant flowing out of the battery. The mass flow rate of the coolant is determined by the pump,

which in turn depends on the angular velocity signal sent by

$$\dot{m}_{\text{cool}} = \rho D \omega. \tag{3.3}$$

The battery also has losses to the environment, modeled as

$$\gamma(T_{\text{env}} - T_{\text{bat}}), \tag{3.4}$$

where γ is a fitting parameter and T_{env} is the environment temperature. Together, (3.1), (3.2), and (3.4) contribute to the change in battery temperature by

$$\frac{dT_{\text{bat}}}{dt} = \frac{\alpha_0}{m_{\text{bat}} \cdot c_{\text{bat}}} \left(R_{\text{bat}} \cdot I_{\text{bat}}^2 - Q_{\text{cool}} + \gamma(T_{\text{env}} - T_{\text{bat}}) \right), \tag{3.5}$$

where α_0 is a fitting parameter. The coolant in and out temperatures in (3.2) are modeled using the number of transfer units (NTU) method [38]. The coolant temperatures are then modeled as depending on the mass flow rate and the power supplied by the heater. The coolant in temperature is modeled as

$$T_{\text{cool,in}} = \alpha_1 \left(T_{\text{bat}} + \frac{1}{1 - \exp\{(-NTU_{\text{bat}})\}} \cdot \frac{Q_{\text{heat}}}{\dot{m}_{\text{cool}} c_{\text{cool}}} \right), \tag{3.6}$$

where α_1 is a fitting parameter and $\exp\{-NTU_{\text{bat}}\}$ is the heat exchange effectiveness and NTU is the number of thermal units

$$NTU_{\text{bat}} = \alpha_3 \frac{hA_{\text{bat}}}{\dot{m}_{\text{cool}} c_{\text{cool}}}, \tag{3.7}$$

with α_3 as another fitting parameter The coolant out temperature is modeled as

$$T_{\text{cool,out}} = (T_{\text{cool,in}} - T_{\text{bat}}) \cdot \alpha_2 \cdot \exp\{(-NTU_{\text{bat}})\} + T_{\text{bat}}, \tag{3.8}$$

where α_2 is an additional fitting parameter. The coolant temperature is bounded between the freezing point $T_{\text{cool,min}}$ and the boiling point $T_{\text{cool,max}}$.

In total, there are five fitting parameters, $\alpha_0,\ \alpha_1,\ \alpha_2,\ \alpha_3$ and γ that model heat exchange effectiveness in different parts of the battery thermal management system. These fitting parameters are obtained by running a heating cycle of the system and collecting battery temperature, angular velocity, heating power and current data. The data is collected for T_{end} with a sampling period of ΔT. This data is used to simulate the heating of the battery using the dynamics (3.5). Denote the dynamics as a function $\hat{f}(\alpha, \gamma)$ depending on the parameters $\alpha = [\alpha_0,\ \alpha_1,\ \alpha_2,\ \alpha_3]$, and γ

$$f(\alpha, \gamma, t) = \frac{\alpha_0}{m_{\text{bat}} \cdot c_{\text{bat}}} \left(R_{\text{bat}} \cdot I_{\text{bat}}(t)^2 - Q_{\text{cool}}(t) + \gamma(T_{\text{env}} - T_{\text{bat}}(t)) \right). \tag{3.9}$$

Values for ω and Q are taken from the collected data. The optimal fit of these parameters is obtained by solving the optimization problem (3.10)

$$
\begin{aligned}
&\min_{\alpha,\gamma} \sum_{k\Delta T}^{T_{end}} (T_{\text{bat}}(k\Delta T) - \hat{T}_{\text{bat}}(k\Delta T))^2 \\
&\text{s.t. } \hat{T}_{\text{bat}}(k\Delta T) = \int_{((k-1)\Delta T}^{k\Delta T} f(\alpha, \gamma, t)\ dt \\
&\quad T_{\text{bat,min}} \leq T_{\text{bat}}(k\Delta T) \leq T_{\text{bat,max}} \\
&\quad T_{\text{cool,in,min}} \leq T_{\text{cool,in}} \leq T_{\text{cool,in,max}} \\
&\quad T_{\text{cool,out,min}} \leq T_{\text{cool,out}} \leq T_{\text{cool,out,max}} \\
&\quad 0.5 \leq \alpha \leq 1.5 \\
&\quad 0 \leq \gamma \leq 1000 \\
&\quad \hat{T}_{\text{bat}}(0) = T_{\text{bat}}(0)
\end{aligned}
\tag{3.10}
$$

where $T_{\text{bat}}(k\Delta T)$ is the true temperature at sampling point k. The parameters α and γ are constrained to limit the search space. α is constrained around 1 since this should model efficiency, while γ must be positive, and the upper bound is to avoid exploding values. The integral is evaluated using fourth-order Runge-Kutta integration. The fitting parameters were found to be

$$
\alpha = \begin{bmatrix} 0.635039 \\ 0.915692 \\ 0.919681 \\ 1.47275 \end{bmatrix}, \gamma = 7.38325. \tag{3.11}
$$

A simulation of the trajectory using the nominal model, with the fitting parameters (3.11) is presented below

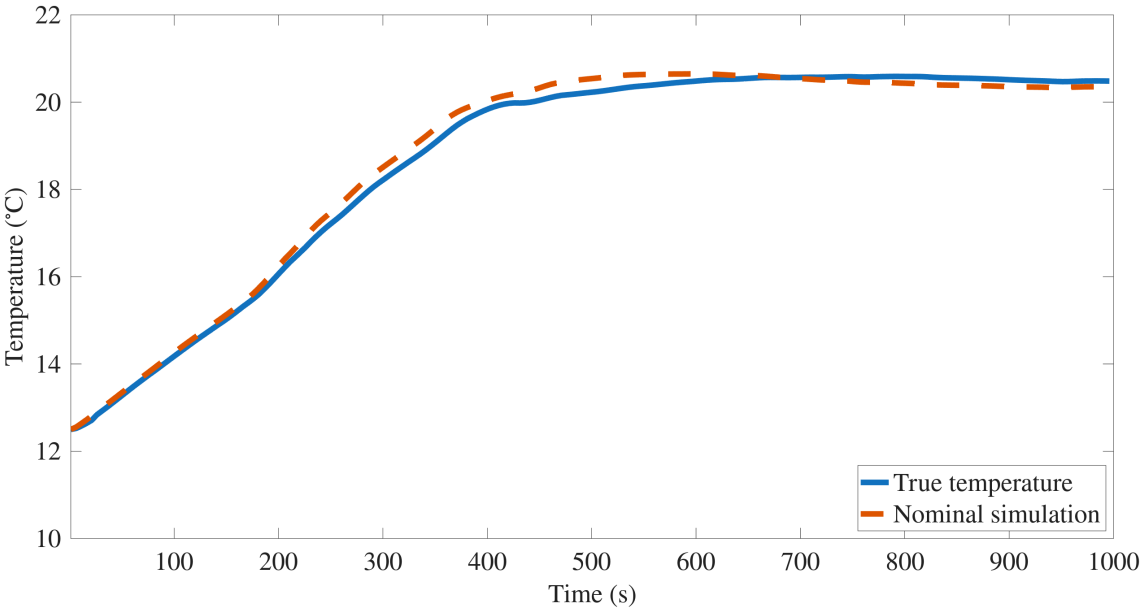


Figure 3.3: Nominal simulation using the model derived here for heating from 12.5 °C for 1000 s.

From the figure, it is clear that the model derived here follows the shape of the true temperature trajectory. Some mismatch can be seen between the nominal simulation and the true temperatures, especially when the battery temperature starts nearing 20.5 °C. The temperature mismatch is at most 0.33 °C, and the root mean squared error of the simulation is 0.19 °C. In closed loop, the controller will be supplied with new temperature measurements after 5 s, meaning that this error will not compound.

The model fitting is performed on the model presented above. The fitting would be done during the controller design stage. With time, the battery thermal management system will change, e.g. due to wear and tear of components. The fitting would also differ depending on which vehicle is used to generate data. Different vehicles, even from within the same vehicle model, could have different parameters due to equipment having some design tolerances. To model this, mismatches can be implemented. Mismatches could be changing parameters either in the control model or in the simulator. For modeling mismatch in the heater, two functions are applied to the heating power signal sent by the controller. Firstly the heater power Q is multiplied by the exponential of the heater power with a negative coefficient,

$$Q_{\text{true}} = Q \exp\{-k_{\text{nl}} Q\} \tag{3.12}$$

with $k_{\text{nl}} = \frac{1}{50000}$, where Q_{true} is the true heating power applied to the system. This models a heater that has a nonlinear saturation in the heating regime while being more sensitive and outputting more power in the cooling regime. This is later called the product-exponential mismatch. Additionally a nonlinear saturation function is also applied

$$Q_{\text{nl}} = Q \exp\{-k_{\text{nl}} |Q|\} \tag{3.13}$$

which models a heater with decreasing efficiency when the power approaches its maximum absolute value, i.e., in both cooling and heating. Note that this function is the same as (3.12) for positive heater power.

3.2 Model Predictive Control Formulation of Reference Tracking for Battery Thermal Management

The model developed in Section 3.1 is used in a model predictive control formulation for reference tracking in a battery thermal management system. The state to be controlled is battery temperature T_{bat}, the inputs are pump angular velocity ω and the heater power Q. The current I_{bat} is a disturbance. To improve numerical

accuracy, these variables are scaled to the same order of magnitude

$$\bar{T}_{\text{bat}} = \frac{T_{\text{bat}}}{T_{\text{bat,scale}}} \tag{3.14}$$

$$\bar{I}_{\text{bat}} = \frac{I_{\text{bat}}}{I_{\text{bat,scale}}} \tag{3.15}$$

$$\bar{\omega} = \frac{\omega}{\omega_{\text{scale}}} \tag{3.16}$$

$$\bar{Q} = \frac{Q}{Q_{\text{scale}}}. \tag{3.17}$$

with $T_{\text{bat,scale}} = 100, I_{\text{bat,scale}} = 25, \omega_{\text{bat,scale}} = 100, Q_{\text{bat,scale}} = 1000$. For brevity, the inputs are denoted as a row vector

$$\bar{u} = \begin{bmatrix} \bar{\omega} \\ \bar{Q} \end{bmatrix}. \tag{3.18}$$

The dynamics are also scaled

$$\frac{d\bar{T}_{\text{bat}}}{dt} = f_{\text{nom}}\left(\bar{T}_{\text{bat}}(t), \bar{u}(t), \bar{I}_{\text{bat}}\right) \tag{3.19}$$

which constitutes the nominal control model. Note that the coolant in and out dynamics are implicitly part of the dynamics $f_{\text{nom}}\left(\bar{T}_{\text{bat}}(t), \bar{u}(t), \bar{I}_{\text{bat}}\right)$. The current $\bar{I}_{\text{bat}}$ is a disturbance. It is assumed that the root mean square value of the current over the controller sampling time is given as a parameter for the entire planning horizon. In practice, this would be done by a predictor that predicts the drive cycle.

The stage cost $\ell(\bar{T}_{\text{bat}}(t), \bar{u}(t))$ for tracking is defined as

$$\ell(\bar{T}_{\text{bat}}(t), \bar{u}(t)) = \left(\begin{bmatrix} \bar{T}_{\text{bat}}(t) \\ \bar{u}(t) \end{bmatrix} - \begin{bmatrix} r_{\bar{\text{T}}_{\text{bat}}}(t) \\ r_{\bar{\text{u}}}(t) \end{bmatrix}\right)^T \begin{bmatrix} Q & 0 \\ 0 & R \end{bmatrix} \left(\begin{bmatrix} \bar{T}_{\text{bat}}(t) \\ \bar{u}(t) \end{bmatrix} - \begin{bmatrix} r_{\bar{\text{T}}_{\text{bat}}}(t) \\ r_{\bar{\text{u}}}(t) \end{bmatrix}\right) \tag{3.20}$$

with references $r_{\bar{\text{T}}_{\text{bat}}}(t)$ and $r_{\bar{\text{u}}}(t)$ and weighting matrices Q and R. The weighting matrices are defined as the following

$$Q = 10 \cdot \left[T^2_{\text{bat,scale}}\right] \tag{3.21}$$

$$\tag{3.22}$$

$$R = \begin{bmatrix} 1 & 0 \\ 0 & 10 \end{bmatrix}. \tag{3.23}$$

The terminal cost is defined as

$$V_{\text{f}}(\bar{T}_{\text{bat}}(T_f)) = (\bar{T}_{\text{bat}}(T_f) - r_{\bar{\text{T}}_{\text{bat}}}(T_f))^T P_\infty (\bar{T}_{\text{bat}}(T_f) - r_{\bar{\text{T}}_{\text{bat}}}(T_f)) \tag{3.24}$$

with terminal cost matrix P_∞, where T_f is the time at the end of the planning horizon.
The model predictive control problem is solved from the initial time t_0 to T_f with initial condition $\bar{T}_{\text{bat}}(0) = \bar{T}_{\text{bat},0}$. $\bar{\mathcal{T}}$ is the constraint set, and $\bar{\mathcal{T}}_{\text{f}}$ is the terminal

constraint set for battery temperature, $\mathcal{T}_{\text{cool}}$ is the constraint on the coolant in and out temperatures. To ease computation, these constraints are formulated with slack variables. Slack variables for battery temperatures are denoted as $s_{\bar{\text{T}}_{\text{bat,l}}}$ for the lower constraint and $s_{\bar{\text{T}}_{\text{bat,u}}}$ for the upper constraint. Similarly the slack variables for coolant are denoted as $s_{\text{T}_{\text{cool,l}}}$ for the lower constraint and $s_{\text{T}_{\text{cool,u}}}$ for the upper constraint. There is also slack on the terminal constraint for temperature and these are denoted as $s^{\text{e}}_{\bar{\text{T}}_{\text{bat,l}}}$ and $s^{\text{e}}_{\bar{\text{T}}_{\text{bat,u}}}$ These are constrained such that

$$0 \leq s_{\bar{\text{T}}_{\text{bat,l}}}, s_{\bar{\text{T}}_{\text{bat,u}}}, s_{\text{T}_{\text{cool,l}}}, s_{\text{T}_{\text{cool,u}}} \leq \infty \tag{3.25}$$

The constraint sets for the battery temperature are thereafter formulated as

$$\bar{\mathcal{T}} = \left[\bar{T}_{\text{bat,min}} - s_{\bar{\text{T}}_{\text{bat,l}}}, \bar{T}_{\text{bat,max}} + s_{\bar{\text{T}}_{\text{bat,u}}}\right] \tag{3.26}$$

and the terminal constraints are

$$\bar{\mathcal{T}}_{\text{f}} = \left[\bar{T}_{\text{bat,min}} - s^{\text{e}}_{\bar{\text{T}}_{\text{bat,l}}}, \bar{T}_{\text{bat,max}} + s^{\text{e}}_{\bar{\text{T}}_{\text{bat,u}}}\right]. \tag{3.27}$$

For coolant in and out temperatures, the constraints are defined as

$$\mathcal{T}_{\text{cool}} = \left[T_{\text{cool,min}} - s_{\text{T}_{\text{cool,l}}}, T_{\text{cool,max}} + s_{\text{T}_{\text{cool,u}}}\right]. \tag{3.28}$$

To ensure that the constraints are still followed, the slack variables have to be penalized heavily. They are penalized using the weighting matrices Z_{l} for lower slack, and Z_{u} for upper slack, with the cost

$$\frac{1}{2}\begin{bmatrix} s_{\text{l}}(t) \\ s_{\text{u}}(t) \end{bmatrix}^T \begin{bmatrix} Z_{\text{l}} & 0 \\ 0 & Z_{\text{u}} \end{bmatrix} \begin{bmatrix} s_{\text{l}}(t) \\ s_{\text{u}}(t) \end{bmatrix} \tag{3.29}$$

where s_{l} is the concatenation of the lower slack variables $s_{\text{l}}(t) = \begin{bmatrix} s_{\bar{\text{T}}_{\text{bat,l}}} & s_{\text{T}_{\text{cool,l}}} \end{bmatrix}^T$ and $s_{\text{u}}(t)$ the concatenation of the upper slack variables $s_{\text{l}}(t) = \begin{bmatrix} s_{\bar{\text{T}}_{\text{bat,u}}} & s_{\text{T}_{\text{cool,u}}} \end{bmatrix}^T$. Similarly, the terminal slack variables have cost

$$\frac{1}{2}\begin{bmatrix} s^{\text{e}}_{\text{l}} \\ s^{\text{e}}_{\text{u}} \end{bmatrix}^T \begin{bmatrix} Z^{\text{e}}_{\text{l}} & 0 \\ 0 & Z^{\text{e}}_{\text{u}} \end{bmatrix} \begin{bmatrix} s^{\text{e}}_{\text{l}} \\ s^{\text{e}}_{\text{u}} \end{bmatrix} \tag{3.30}$$

where $s^{\text{e}}_{\text{l}} = s^{\text{e}}_{\bar{\text{T}}_{\text{bat,l}}}$ and $s^{\text{e}}_{\text{u}} = s^{\text{e}}_{\bar{\text{T}}_{\text{bat,u}}}$. The slack weighting matrices are chosen such that $Z_{\text{l}}, Z_{\text{u}}, Z^{\text{e}}_{\text{l}}, Z^{\text{e}}_{\text{u}} >>> Q, R, P_{\infty}$. The constraint set on inputs $\bar{\mathcal{U}}$ is formulated as

$$\bar{\mathcal{U}} = \left[\begin{bmatrix} \bar{\omega}_{min} \\ \bar{Q}_{min} \end{bmatrix}, \begin{bmatrix} \bar{\omega}_{max} \\ \bar{Q}_{max} \end{bmatrix}\right]. \tag{3.31}$$

Finally, the model predictive control problem is formulated as

$$
\begin{aligned}
\min_{\bar{\mathrm{T}},\bar{\mathrm{u}}} & \int_{t_0}^{T_f} \ell\left(\bar{T}_{\mathrm{bat}}(t), \bar{u}(t)\right) + \frac{1}{2}\begin{bmatrix} s_{\mathrm{l}}(t) \\ s_{\mathrm{u}}(t) \end{bmatrix}^T \begin{bmatrix} Z_{\mathrm{l}} & 0 \\ 0 & Z_{\mathrm{u}} \end{bmatrix} \begin{bmatrix} s_{\mathrm{l}}(t) \\ s_{\mathrm{u}}(t) \end{bmatrix} dt + \\
& + V_{\mathrm{f}}\left(\bar{T}_{\mathrm{bat}}\right) + \frac{1}{2}\begin{bmatrix} s_{\mathrm{l}}^{\mathrm{e}} \\ s_{\mathrm{u}}^{\mathrm{e}} \end{bmatrix}^T \begin{bmatrix} Z_{\mathrm{l}}^{\mathrm{e}} & 0 \\ 0 & Z_{\mathrm{u}}^{\mathrm{e}} \end{bmatrix} \begin{bmatrix} s_{\mathrm{l}}^{\mathrm{e}} \\ s_{\mathrm{u}}^{\mathrm{e}} \end{bmatrix} \\
\text{s.t. } & \dot{\bar{T}}_{\mathrm{bat}}(t) = f_{\mathrm{nom}}\left(\bar{T}_{\mathrm{bat}}(t), \bar{u}(t), \bar{I}_{\mathrm{bat}}(t)\right) \\
& \bar{T}_{\mathrm{bat}}(t) \in \mathcal{T} \\
& \bar{T}_{\mathrm{bat}}(T) \in \mathcal{T}_{\mathrm{f}} \\
& \bar{u}(t) \in \mathcal{U} \\
& T_{\mathrm{cool,in}},\ T_{\mathrm{cool,out}} \in \mathcal{T}_{\mathrm{cool}} \\
& \bar{T}_{\mathrm{bat}}(0) = \bar{T}_{\mathrm{bat},0}.
\end{aligned}
\tag{3.32}
$$

where t_0 is the time at which (3.32) is solved and T_f is the time at the end of the planning horizon. The problem is formulated in acados and internally discretized. The final time T_f is therefore changed to the number of shooting nodes N where

$$T_f = N \cdot \Delta T \tag{3.33}$$

and ΔT is the sampling time. To integrate the dynamics and cost, explicit Euler integration is used. The current $I_{\mathrm{bat}}(t)$ changes with time. To provide the controller with the current during planning, it is treated as a parameter and set for the planning horizon. The current values are taken from future values in the drive cycle, and it is assumed that these are provided by a perfect estimator. The references $r_{\bar{T}_{\mathrm{bat}}}$ and $r_{\bar{u}}$ are also treated as parameters and are set for the entire prediction horizon. The reference for the battery temperature is set constant to

$$r_{\bar{\mathrm{T}}_{\mathrm{bat}}} = \bar{T}_{\mathrm{bat,ref}}. \tag{3.34}$$

For input references, the steady-state inputs are calculated. In steady state, the temperature should ideally have the value $\bar{T}_{\mathrm{bat,ss}} = \bar{T}_{\mathrm{bat,ref}}$. To keep it at this value, the steady state inputs $\bar{u}_{\mathrm{ss}}$ should be such that the following holds

$$\bar{T}_{\mathrm{bat,ss}} = \bar{T}_{\mathrm{bat,ref}} + \int_0^{\Delta T} f\left(\bar{T}_{\mathrm{bat,ref}}, \bar{u}_{\mathrm{ss}}, \bar{I}_{\mathrm{bat,ss}}\right) \tag{3.35}$$

where $\bar{I}_{\mathrm{bat,ss}}$ is the current at the end of the planning horizon. To solve for the steady state inputs $\bar{u}_{\mathrm{ss}}$, the following optimization problem is solved

$$\min_{\bar{u}_{\mathrm{ss}}\ \bar{T}_{\mathrm{bat,ss}}} \quad R_{\mathrm{ss}}\bar{u}_{\mathrm{ss}} + (\bar{T}_{\mathrm{bat,ss}} - \bar{T}_{\mathrm{bat,ref}})Q_{\mathrm{ss}}(\bar{T}_{\mathrm{bat,ss}} - \bar{T}_{\mathrm{bat,ref}}) \tag{3.36}$$

$$\text{s.t.} \quad \bar{T}_{\mathrm{bat,ss}} = \bar{T}_{\mathrm{bat,ref}} + \int_0^{\Delta T} f\left(\bar{T}_{\mathrm{bat,ref}}, \bar{u}_{\mathrm{ss}}, \bar{I}_{\mathrm{bat,ss}}\right) \tag{3.37}$$

$$
\begin{aligned}
& \bar{T}_{\mathrm{bat,ss}} \in \mathcal{T} \\
& \bar{T}_{\mathrm{bat,ss}} \in \mathcal{T}_{\mathrm{f}} \\
& \bar{u}_{\mathrm{ss}} \in \mathcal{U} \\
& T_{\mathrm{cool,in}},\ T_{\mathrm{cool,out}} \in \mathcal{T}_{\mathrm{cool}}
\end{aligned}
\tag{3.38}
$$

where $\bar{T}_{\text{bat,ss}}$ is also an optimization variable to account for cases where the reference cannot be reached, e.g. because the current is too high. With the steady-state inputs and state, it is also possible to find the cost-to-go matrix P_∞. This is done by linearizing the dynamics (3.19) around the steady state

$$A = \left.\frac{df\left(\bar{T}_{\text{bat}}(t), \bar{u}(t), \bar{I}_{\text{bat}}(t)\right)}{d\bar{T}_{\text{bat}}}\right|_{\bar{T}_{\text{bat,ss}}, \bar{u}_{\text{ss}}, \bar{I}_{\text{bat,ss}}} \tag{3.39}$$

$$B = \left.\frac{df\left(\bar{T}_{\text{bat}}(t), \bar{u}(t), \bar{I}_{\text{bat}}(t)\right)}{d\bar{u}}\right|_{\bar{T}_{\text{bat,ss}}, \bar{u}_{\text{ss}}, \bar{I}_{\text{bat,ss}}} \tag{3.40}$$

and calculating the cost-to-go using the continuous algebraic Riccati equation

$$P_\infty = \text{CARE}(A, B, Q, R). \tag{3.41}$$

3.3 Model Predictive Control Formulation of Economic Energy for Battery Thermal Management

In the economic formulation for battery thermal management, the battery temperature is to be kept within an operating region defined by the normalized bounds $\bar{T}_{\text{steady,min}}$ and $\bar{T}_{\text{steady,max}}$. Here, $\bar{T}_{\text{steady,min}}$ and $\bar{T}_{\text{steady,max}}$ denote the normalized temperature limits, respectively. The goal is to keep the battery temperature between these two limits while minimizing the pump power and heater power.

The formulation of economic MPC for battery thermal management follows a similar MPC framework as in Section 3.2. The normalization of variables, the input vector $\bar{u}$, and the normalized dynamics (3.19) still hold. The primary difference is in stage cost and cost-to-go. Instead of having the cost determined by tracking a reference state, the cost is determined by minimizing energy consumption within the constraints. The stage cost is defined by the heater power and the pump power. Heater power is already one of the inputs, whereas the pump power is not readily available but is obtained via a third-order polynomial fit of the normalized pump angular velocity $\bar{\omega}$

$$P_{\text{pump}}(\bar{\omega}) = c_0 + c_1\bar{\omega} + c_2\bar{\omega}^2 + c_3\bar{\omega}^3 \tag{3.42}$$

which is obtained by sampling data of the pump power $P_{\text{pump,true}}(k\Delta T)$ and the pump angular velocity $\bar{\omega}(k\Delta T)$ for $k = 0, \ldots, N_{\text{data,pump}}$ with sampling time ΔT, and solving the following optimization problem

$$\min_{c_0,\ c_1,\ c_2,\ c_3} \sum_{k=0}^{N_{\text{data,pump}}} \left(P_{\text{pump,true}}(k\Delta T) - P_{\text{pump}}(\bar{\omega}(k\Delta T))\right)^2 \tag{3.43}$$

$$\begin{aligned} \text{s.t. } & 0 \leq P_{\text{pump}}(\bar{\omega}(k\Delta T)) \\ & 0 \leq c_1,\ c_2,\ c_3 \\ & 1000 \geq c_0, c_1, c_2, c_3. \end{aligned} \tag{3.44}$$

Pump power must be non-negative to preserve its physical meaning, while the constants $c_1,\ c_2,\ c_3$ are bounded below by 0 to keep the pump power monotonic in $\bar{\omega}$. The constants are upper bounded to limit the search space. The following expression is obtained

$$P_{\text{pump}}(\bar{\omega}) = \frac{1}{1000}(-0.21574 + 10.0501\bar{\omega}^2 + 6.84987\bar{\omega}^3) \tag{3.45}$$

for the pump power in kilowatt. For validation, the pump power expression is plotted alongside the true data

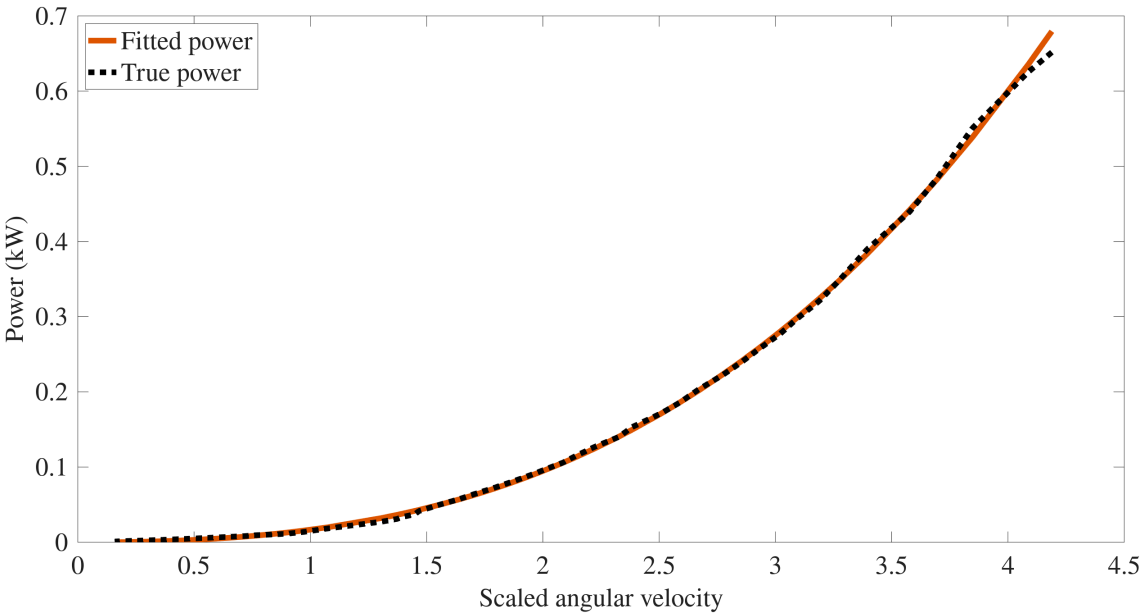


Figure 3.4: Fitted pump power compared to the true data. The fit follows the underlying data closely and is positive in the operating range of the pump.

Furthermore, the heater power should be included in the cost function. The economic stage cost function $\ell_{\text{econ}}(\bar{T}_{\text{bat}}, \bar{u})$ is defined, depending on normalized battery temperature $\bar{T}_{\text{bat}}$ and normalized inputs $\bar{u} = [\bar{\omega}, \bar{Q}]^T$, as the weighted norm

$$\ell_{\text{econ}}(\bar{T}_{\text{bat}}, \bar{u}) = \left\| y_{\text{econ}}(\bar{T}_{\text{bat}}, \bar{u}) - r_{\text{ref,econ}} \right\|_W^2$$

where $y_{\text{econ}}(\bar{T}_{\text{bat}}, \bar{u})$ is the expression for the power and battery temperature

$$y_{\text{econ}}(\bar{T}_{\text{bat}}, \bar{u}) = \begin{bmatrix} \overbrace{\frac{1}{1000}(-0.21574 + 10.0501\bar{\omega}^2 + 6.84987\bar{\omega}^3)}^{P_{\text{pump}}(\bar{\omega})} \\ \bar{Q} \\ \bar{T}_{\text{bat}} \end{bmatrix} \tag{3.46}$$

with $r_{\text{ref,econ}} = [0,\ 0,\ \bar{T}_{\text{bat,target}}]^T$, where $\bar{T}_{\text{bat,target}}$ is the normalized target temperature. The economic cost-to-go $V_{f,\text{econ}}$ is defined as

$$V_{f,econ} = (\bar{T}_{\text{bat}} - r^e_{\text{ref,econ}})^2 \tag{3.47}$$

and $r^{\mathrm{e}}_{\mathrm{ref,econ}} = [\bar{T}_{\mathrm{bat,target}}]$. The cost matrices W and W_{e} are defined as

$$W = \begin{bmatrix} 15 & 0 & 0 \\ 0 & 15 & 0 \\ 0 & 0 & 0 \end{bmatrix}, W_{\mathrm{e}} = 0.$$

The purpose of including the battery temperature in the cost function is to potentially alleviate issues with automatic generation of Jacobians and Hessians.

To keep the battery within the operating region, constraints are imposed to keep the battery temperature between $\bar{T}_{\mathrm{steady,min}}$ and $\bar{T}_{\mathrm{steady,max}}$. The battery might be initialized outside of this range; therefore the normalized slacks $\bar{s}_{\mathrm{T_{steady},l}}$ and $\bar{s}_{\mathrm{T_{steady},u}}$ are added to the constraints. The constraint formulation for the battery temperature is then

$$\bar{\mathcal{T}}_{\mathrm{steady}} = \left[\bar{T}_{\mathrm{steady,min}} - \bar{s}_{\mathrm{T_{steady},l}}, \bar{T}_{\mathrm{steady,max}} + \bar{s}_{\mathrm{T_{steady},u}}\right]. \tag{3.48}$$

The slack variables might take large values if the battery is initialized far from the operating range. A high quadratic penalty on the slack was found to lead to solver errors because of exploding costs. Therefore, the slack penalty includes a smaller weight on the square of the slacks and, as opposed to the formulation in Section 3.2, also includes a linear slack penalty. The slack penalty for the state in the economic case is therefore

$$\frac{1}{2}\begin{bmatrix} \bar{s}_{\mathrm{T_{steady},l}} \\ \bar{s}_{\mathrm{T_{steady},u}} \\ 1 \end{bmatrix}^T \begin{bmatrix} Z_{\mathrm{econ,T_{bat},l}} & 0 & z_{\mathrm{econ,T_{bat},l}} \\ 0 & Z_{\mathrm{econ,T_{bat},u}} & z_{\mathrm{econ,T_{bat},u}} \\ z_{\mathrm{econ,T_{bat},l}} & z_{\mathrm{econ,T_{bat},u}} & 0 \end{bmatrix} \begin{bmatrix} \bar{s}_{\mathrm{T_{steady},l}} \\ \bar{s}_{\mathrm{T_{steady},u}} \\ 1 \end{bmatrix} \tag{3.49}$$

with $Z_{\mathrm{econ,T_{bat},l}}$ and $Z_{\mathrm{econ,T_{bat},u}}$ being costs on the quadratic slack and $z_{\mathrm{econ,T_{bat},l}}$ with $z_{\mathrm{econ,T_{bat},u}}$ being the cost on the linear slack.

Economic model predictive control for battery thermal management can be characterized as reference tracking with high penalties on overshoots (in the case of cooling) and undershoots (in the case of heating), together with a cost on inputs that better corresponds to the actual power usage. It is assumed that the cost on inputs and the costs on slack formulated here balance this trade-off properly. For a proper thermal management formulation it would be necessary to evaluate the degraded performance of the battery when it is outside the target region.

3.4 Neural Network Formulation for Learning Model Residuals in Battery Thermal Management

The model developed in Section 3.1 and used in the model predictive control formulation in Section 3.2 is assumed to have a mismatch with respect to the true dynamics of the system. The mismatch might arise from parameter errors or hidden dynamics. To improve controller performance, the nominal model is extended with a neural network trained to model the residual dynamics. Denote the neural network for the residual dynamics as

$$f_{\mathrm{res}}(X_{\mathrm{res}}(k), \Theta_{\mathrm{res}}) \tag{3.50}$$

where $X_{\text{res}}(k)$ is the vector of inputs to the neural network at sample k and Θ_{res} is the vector of parameters consisting of weights and biases. The vector of inputs comes from data collected in batches of size N_{batch} with a sample time of ΔT that is the same as the discretization time for the model predictive controller. Denote the batches of data as X which include scaled samples of the battery temperature, current, angular velocity, and heating power

$$X(k) = \begin{bmatrix} \bar{T}_{\text{bat}}(k) & \bar{I}_{\text{bat}}(k) & \bar{\omega}(k) & \bar{Q}(k) \end{bmatrix}. \tag{3.51}$$

In the battery thermal management case, the batches of data (3.51) are used both as the inputs to the neural network X_{res} and inputs used in the nominal model X_{nom}, hence inputs to both neural network and nominal model will be denoted as X hereafter. After a batch has been collected, the neural network is trained. Training then continues every time a new batch is collected. The neural networks are initially trained offline on data collected from a simulation without additionally added mismatch in heating power or parameters. This initializes the neural network with reasonable parameters in the online learning. The network is a Multilayer Perceptron (MLP) network. Denote the adaptive control model as consisting of the nominal dynamics (3.5) and the neural network (3.50), as

$$\hat{f}(X(k), \Theta_{res}) = f_{\text{nom}}(X(k)) + s \cdot f_{\text{res}}(X(k), \Theta_{\text{res}}) \tag{3.52}$$

where $s \in \{0, 1\}$ is an integer switch that turns on the neural network. The switch is given as a parameter. Note that the nominal dynamics could also be described using a parameter vector Θ_{nom}; however, the parameters in the nominal dynamics do not change, and therefore Θ_{nom} is dropped from the arguments for notational convenience. For the adaptive controller, the model (3.52) is used within the model predictive control formulation in place of the nominal dynamics. The adaptive model is not used in the steady-state selection problem or when calculating the cost-to-go, since these problems are sensitive to small perturbations, and the neural network cannot be assumed to have converged at any given time step.

For training the neural network two different loss functions are used, derivative loss and trajectory loss. In both cases, the output of the final layer, $r_{\text{res}}(X, \Theta_{\text{res}})$ is passed through a hyperbolic tangent function, resulting in the residual of the neural network being

$$f_{\text{res}}(X(k), \Theta_{\text{res}}) = c \tanh r_{\text{res}}(X(k), \Theta_{\text{res}}). \tag{3.53}$$

where c is a confidence parameter. By applying a hyperbolic tangent to the last layer, the residual dynamics become bounded, where the bound should maximally correspond to the bound of the mismatch. Due to the true dynamics not being available, the value of the confidence parameter c is empirically obtained by tuning. The confidence parameter additionally limits how much the neural network impacts the nominal model, meaning that it can be set low if the neural network should only make minor adaptations to the model. Although using a hyperbolic tangent saturates the residual, prior work using HardTanh, a piecewise-linear approximation of the hyperbolic tangent, has been shown to still be able to have explanatory power [39].

3.4.1 Formulation For Derivative Loss

It is necessary to obtain the dynamics of the plant such that the loss function described in Section 2.3 can be utilized by the neural network. The dynamics can be calculated directly from temperature data, using e.g., an Euler derivative; however this operation amplifies quick changes and results in a non-smooth and poorly conditioned derivative. To learn the underlying physics of the dynamics, a Savitzky-Golay filter is used to smooth the collected temperature data. The Savitzky-Golay filter fits this data to a polynomial and then calculates the derivative [40]. This filter functions as a low-pass filter but with small attenuation in the pass-band and therefore results in less information loss compared to using a standard low-pass filter. This is how the true plant dynamics $f(X_0, \Theta_0)$ are approximated, and the approximation is denoted by $f_{\text{SG}}(X_0, \Theta_0)$.

A neural network that uses derivative loss, described in Section 2.3, is trained for the adaptive updates of the battery thermal model.

$$\mathcal{L}_{\text{der}} = \frac{1}{N_{batch}} \sum_{k=0}^{N_{\text{batch}}-1} \left(f_{\text{SG}}(X_0(k), \Theta_0) - \hat{f}(X(k), \Theta_{\text{res}}) \right)^2 \tag{3.54}$$

where $\mathcal{L}_{\text{der}}$ is the loss function defined as the mean squared error (MSE), and N_{batch} is the batch size of the collected data. By minimizing (3.54), the adaptive model $\hat{f}(X(k), \Theta_{\text{res}})$ approximates the filtered dynamics, and hence the residual learns the mismatch. The loss (3.54) is minimized by the AdamW optimizer. To avoid rapid changes in the neural network parameters, the gradient norm $\nabla_\Theta \mathcal{L}_{\text{der}}$ is clipped if it exceeds a threshold γ_{Jac}. The optimizer runs for N_{epochs} epochs per update. The network settings are presented in Table 3.2.

Table 3.2: Hyperparameters for neural network for tracking battery temperature using derivative loss.

Parameter	Value
Number of hidden layers	2
Hidden dimension	16
Learning rate (η)	10^{-3}
Batch size (N_{batch})	200
Epochs (N_{epochs})	200
Weight decay (λ_{wd})	0.1
Confidence c	0.001
Gradient max norm γ_{Jac}	0.01

A pseudocode algorithm of the derivative loss formulation is shown in Algorithm 1.

Algorithm 1 Neural Network Training with Derivative Loss for the BTM system

Require: Collected batch of temperature and input data $\{T_{\text{bat}}(k), X(k)\}_{k=0}^{N_{\text{batch}}-1}$, where $X(k)$ is the regressor vector used in the adaptive model.
Require: Savitzky–Golay filter parameters (window length, polynomial order).
Require: Neural network parameters Θ_{res} (initialized offline), optimizer AdamW with learning rate η, weight decay λ_{wd}, gradient norm threshold γ_{Jac}, and number of epochs N_{epochs}.
Ensure: Updated residual network parameters Θ_{res}.

Step 1: Estimate plant dynamics using Savitzky–Golay filter
for $k = 0$ to $N_{\text{batch}} - 1$ **do**
    Apply Savitzky–Golay filter to $\{T_{\text{bat}}(j)\}$ to obtain smoothed derivative $\dot{T}_{\text{bat,SG}}(k)$
    Set $f_{\text{SG}}(X_0(k), \Theta_0) \leftarrow \dot{T}_{\text{bat,SG}}(k)$
end for

Step 2: Train residual network using derivative loss
for $e = 1$ to N_{epochs} **do** ▷ Epoch loop
    Initialize accumulated loss: $\mathcal{L}_{\text{der}} \leftarrow 0$
    for $k = 0$ to $N_{\text{batch}} - 1$ **do**
        Compute adaptive model output
            $\hat{f}(X(k), \Theta_{\text{res}}) = f_{\text{nom}}(X(k)) + f_{\text{res}}(X(k), \Theta_{\text{res}})$
        Compute sample loss
            $\ell_k \leftarrow \left(f_{\text{SG}}(X_0(k), \Theta_0) - \hat{f}(X(k), \Theta_{\text{res}})\right)^2$
        Accumulate loss
            $\mathcal{L}_{\text{der}} \leftarrow \mathcal{L}_{\text{der}} + \ell_k$
    end for
    Normalize loss:
        $\mathcal{L}_{\text{der}} \leftarrow \mathcal{L}_{\text{der}}/N_{\text{batch}}$
    Compute gradient w.r.t. Θ_{res}:
        $g \leftarrow \nabla_{\Theta_{\text{res}}} \mathcal{L}_{\text{der}}$
    Clip gradient norm if necessary:
    if $\|g\|_2 > \gamma_{\text{Jac}}$ **then**
        $g \leftarrow \gamma_{\text{Jac}} \cdot g/\|g\|_2$
    end if
    Update parameters with AdamW:
        $\Theta_{\text{res}} \leftarrow \text{AdamW}(\Theta_{\text{res}}, g, \eta, \lambda_{\text{wd}})$
end for
return Θ_{res}

After running Algorithm 1, the residual neural network $f_{\text{res}}(X(k), \Theta_{\text{res}})$ is then tested on data that were not part of its training to see if it outperforms the null hypothesis. The null hypothesis is here considered to be the nominal controller, i.e. a residual that is zero. If the residual neural network outperforms the null hypothesis, the switch parameter in the model $\hat{f}(X(k), \Theta_{\text{res}})$, equation (3.52), is turned on by $s = 1$, otherwise the switch is turned off by $s = 0$.

3.4.2 Formulation for Trajectory Loss

A neural network using trajectory loss, described in Section 2.3, is trained to provide adaptive updates to the battery thermal model. Compared to the theoretical framework in Section 2.3, the hyper trajectory loss formulation is discretized, data on battery temperatures and currents filtered through a Savitzky-Golay filter, and additional steps are introduced to improve training as described below. The predicted temperature is calculated by Euler integration using step size ΔT as

$$\begin{aligned} T_{\text{bat,traj}}(k+1) &= T_{\text{bat,traj}}(k) + \Delta T \cdot (f_{\text{nom}}(X(k)) + f_{\text{res}}(X(k), \Theta_{\text{res}})) + \sigma_{T_{\text{bat}}}(k) = \\ &= T_{\text{bat,traj}}(k) + \Delta T \cdot \hat{f}(X(k), \Theta_{\text{res}}) + \sigma_{T_{\text{bat}}}(k) \end{aligned} \tag{3.55}$$

which gives a trajectory $T_{\text{bat,traj}}$ to be used in the loss formulation, and where $\sigma_{T_{\text{bat}}}(k)$ is a small Gaussian disturbance added to the prediction to teach the neural network not to overreact to small changes and to instead learn the underlying physics. For $k = 0$, the true battery temperature T_{bat} at this sample is given. The temperature is then simulated forward. The simulation can be performed for the entire batch size N_{batch}. However, to prevent the model from learning to minimize its own drift, after N_{rolling} steps the trajectory is reinitialized with the true battery temperature T_{bat}

$$T_{\text{bat,traj}}(k+1) = \begin{cases} T_{\text{bat}}(k+1) \text{ if } (k+1) \bmod N_{\text{rolling}} = 0 \\ T_{\text{bat,traj}}(k) + \Delta T \cdot \hat{f}(X(k), \Theta_{\text{res}}) + \sigma_{T_{\text{bat}}}(k) \text{ otherwise} \end{cases} \tag{3.56}$$

which is an application of teacher forcing. The prediction error ε is calculated as

$$\varepsilon(k) = \begin{cases} (T_{\text{bat}}(k) - T_{\text{bat,traj}}(k)) & \text{if } |T_{\text{bat}}(k) - T_{\text{bat,traj}}(k)| > \varepsilon_{\text{thresh}} \\ 0 & \text{otherwise} \end{cases} \tag{3.57}$$

where $\varepsilon_{\text{thresh}}$ is the threshold above which the neural network should identify an error. The threshold parameter is set to limit the neural network from overcompensating for small errors and thereby increase robustness. The trajectory loss is then calculated as

$$\mathcal{L}_{\text{traj}} = \sum_{k=0}^{N_{\text{batch}}-1} w^k \varepsilon(k)^2 \tag{3.58}$$

where w is a weighting factor that is greater than 1. This weighting reduces the dominance of early errors in the loss, since early errors compound over the trajectory. In addition to the trajectory loss, a loss on the norm of the Jacobian with respect to the battery temperature is employed. Denote the Jacobian loss as

$$\mathcal{L}_{\text{jac}} = \sum_{k=0}^{N_{\text{batch}}-1} \lambda_{\text{jac}} \|\nabla_{T_{\text{bat}}} f_{\text{res}}(X(k))\|_2^2 \tag{3.59}$$

where λ_{jac} is the weight on this loss. The Jacobian loss is introduced to steer the network away from large gradients, since these could lead to aggressive control actions and numerical instability. The total loss function thus becomes

$$\mathcal{L}_{\text{total,traj}} = \mathcal{L}_{\text{traj}} + \mathcal{L}_{\text{jac}}. \tag{3.60}$$

The AdamW optimizer then takes steps to reduce this loss. The norm of the gradient of the neural network parameters is clipped above γ_{Jac} to avoid rapid changes and to improve generalization. This idea is inspired by [41], where a neural network was developed that had a limit on the spectral norm of each layer to constrain their Lipschitz constant. The loss is then recalculated using the updated neural network, and this process repeated for N_{epochs} epochs. In contrast to the derivative loss training, the switch in the adaptive control model (3.52), is always turned on $s = 1$ after initial training. For an overview of the hyperparameters in the neural network for reference tracking of battery temperature using trajectory loss, see Table 3.3.

Table 3.3: Hyperparameters for neural network for reference tracking of battery temperature using trajectory loss.

Parameter	Value
Number of hidden layers	2
Hidden dimension	32
Learning rate (η)	10^{-2}
Batch size (N_{batch})	200
Epochs (N_{epochs})	200
Weight decay (λ_{wd})	0.1
Confidence (c)	0.001
Jacobian weight (λ_{jac})	0.1
Prediction noise ($\sigma_{T_{\text{bat}}}$)	0.005
Error threshold ($\varepsilon_{\text{thresh}}$)	0.1
Teacher forcing interval (N_{rolling})	20
Gradient max norm (γ_{Jac})	0.1

The hyperparameters vary slightly for the thermal management implementation, since the nature of the problem is different, and can be seen in Table 3.4.

Table 3.4: Hyperparameters for neural network for thermal management using trajectory loss.

Parameter	Value
Number of hidden layers	2
Hidden dimension	32
Learning rate (η)	10^{-3}
Batch size (N_{batch})	200
Epochs (N_{epochs})	200
Weight decay (λ_{wd})	0.01
Confidence (c)	0.001
Jacobian weight (λ_{jac})	0.1
Prediction noise ($\sigma_{T_{\text{bat}}}$)	0.001
Error threshold ($\varepsilon_{\text{thresh}}$)	0.2
Teacher forcing interval (N_{rolling})	20
Gradient max norm (γ_{Jac})	0.1

In the thermal management problem, after the battery enters the target region, the control inputs are naturally smaller since the battery temperature is allowed to vary more; therefore, the corrections should be more careful. A pseudocode for the trajectory training algorithm is shown in Algorithm 2.

Algorithm 2 Neural Network Training with trajectory Loss for BTM

Require: Batch of measured data $\{T_{\text{bat}}(k), X(k)\}_{k=0}^{N_{\text{batch}}-1}$
Require: Nominal model f_{nom}, residual NN $f_{\text{res}}(\cdot, \Theta_{\text{res}})$, adaptive model $\hat{f}(X, \Theta_{\text{res}}) = f_{\text{nom}}(X) + f_{\text{res}}(X, \Theta_{\text{res}})$
Require: Hyperparameters: ΔT, $w > 1$, λ_{jac}, $\varepsilon_{\text{thresh}}$, N_{rolling}, prediction noise std. $\sigma_{T_{\text{bat}}}$, gradient norm threshold γ_{Jac}, AdamW settings $(\eta, \lambda_{\text{wd}})$, number of epochs N_{epochs}
Ensure: Updated NN parameters Θ_{res}

for $e = 1$ to N_{epochs} **do** ▷ Epoch loop
  Initialize trajectory loss: $\mathcal{L}_{\text{traj}} \leftarrow 0$
  Initialize Jacobian loss: $\mathcal{L}_{\text{jac}} \leftarrow 0$
  Set initial predicted temperature $T_{\text{bat,traj}}(0) \leftarrow T_{\text{bat}}(0)$
  for $k = 0$ to $N_{\text{batch}} - 1$ **do**
    if $(k+1) \mod N_{\text{rolling}} = 0$ **then**
      $T_{\text{bat,traj}}(k+1) \leftarrow T_{\text{bat}}(k+1)$ ▷ Teacher forcing
    else
      Sample prediction noise $\nu_k \sim \mathcal{N}(0, \sigma^2_{T_{\text{bat}}})$
      $T_{\text{bat,traj}}(k+1) \leftarrow T_{\text{bat,traj}}(k) + \Delta T \cdot \hat{f}(X(k), \Theta_{\text{res}}) + \nu_k$
    end if
    Compute prediction error
      $\delta_k \leftarrow T_{\text{bat}}(k) - T_{\text{bat,traj}}(k)$
    if $|\delta_k| > \varepsilon_{\text{thresh}}$ **then**
      $\varepsilon(k) \leftarrow \delta_k$
    else
      $\varepsilon(k) \leftarrow 0$
    end if
    Accumulate trajectory loss
      $\mathcal{L}_{\text{traj}} \leftarrow \mathcal{L}_{\text{traj}} + w^k \varepsilon(k)^2$
    Compute Jacobian penalty term
      $J_k \leftarrow \nabla_{T_{\text{bat}}} f_{\text{res}}(X(k), \Theta_{\text{res}})$
      $\mathcal{L}_{\text{jac}} \leftarrow \mathcal{L}_{\text{jac}} + \lambda_{\text{jac}} \|J_k\|_2^2$
  end for
  Compute total loss:
    $\mathcal{L}_{\text{total,traj}} \leftarrow \mathcal{L}_{\text{traj}} + \mathcal{L}_{\text{jac}}$
  Compute gradient $g \leftarrow \nabla_{\Theta_{\text{res}}} \mathcal{L}_{\text{total,traj}}$
  if $\|g\|_2 > \gamma_{\text{Jac}}$ **then**
    $g \leftarrow \gamma_{\text{Jac}} \cdot g / \|g\|_2$ ▷ Gradient clipping
  end if
  Update parameters with AdamW:
    $\Theta_{\text{res}} \leftarrow \text{AdamW}(\Theta_{\text{res}}, g, \eta, \lambda_{\text{wd}})$
end for
return Θ_{res}

3.4.3 Integration of Neural Networks for Residual Dynamics into a Model Predictive Control Framework

The neural networks in this section are formulated using PyTorch. The formulations of the model predictive control problems in Section 3.2 and Section 3.3 are implemented in acados, which is built on top of CasADi. CasADi has support for symbolic representations of variables and functions, as well as the algorithmic differentiation of these. Algorithmic differentiation is required to compute the Jacobians and the Hessians used within the optimization problem. A multilayer perceptron consists of layers, where each layer performs an affine transformation (which in turn consists of matrix multiplications and summations) that is then passed through an activation function. If the output is fed into another hidden layer, the same operations are applied. All of these operations can be represented symbolically. Therefore, the network can also be represented symbolically by constructing it using symbolic variables for the inputs and parameters. From this symbolic representation, it is then possible to obtain Jacobians and Hessians by differentiating the network. Parameters of the network can then be supplied as parameters to the optimization problem.

However, rebuilding the network symbolically in CasADi introduces overhead, for example, copying parameters from the neural network. Therefore, the L4CasADi framework is used to more efficiently perform these operations. L4CasADi [17] provides functionality to convert a neural network formulated in PyTorch into the CasADi framework. L4CasADi constructs the symbolic representation of the network and calculates the Jacobian and the Hessian of the network. These are then stored as a shared library. From the shared library, the framework then loads these symbolic representations into CasADi functions, which can then be used as standard CasADi functions. The exported functions are used in the dynamics model of the MPC problem. L4CasADi supports efficient online updates of the MLP parameters. Further, the L4CasADi-generated functions can be supplied to acados through the shared libraries. The precursor to the L4CasADi framework has been used by its creators in adaptive model predictive control of quadrotors and agile robotic platforms [16]. The code developed here was based on [15] and example files from the GitHub repository for L4CasADi [42].

3.4.4 Tuning for Neural Networks in Adaptive Model Predictive Control

The neural networks presented in this section have architectural design choices and hyperparameters that require tuning. For the architecture, the key considerations are the size of the network (including the number of layers and the hidden dimension) and the activation functions. The design of the architecture hinged on the following ideas:

1. Lightweight architecture for real-time MPC
2. Differentiable output for reliable differentiation and optimization

which were considered constraints on the design choices. Making the network architecture as lightweight as possible is necessary both to make the model predictive

control optimization feasible with the given sampling time and to avoid overfitting. The choice of the hyperbolic tangent as an activation function hinged on the necessity of having a differentiable neural network output. Other common activation functions, such as rectified linear units (ReLUs), are non-differentiable at zero and produce non-smooth mappings, which may be problematic for MPC solvers. The hyperbolic tangent, by contrast, is smooth and differentiable everywhere.

Since the goal of adaptive model predictive control is generalizability, the hyperparameters were kept constant for different scenarios, e.g. heating or cooling. However, differences were allowed between reference tracking and economic model predictive control, since these controllers have different performance objectives. Available hyperparameters to tune can be seen in Table 3.5. Starred hyperparameters (*) correspond to hyperparameters that exist only in the trajectory loss formulation.

Table 3.5: Description of tunable hyperparameters for neural networks. Starred * hyperparameters are only used in trajectory loss.

Hyperparameter	Description
Number of hidden layers	Determines the expressivity of the neural network
Hidden dimension	Determines the expressivity of the neural network
Learning rate (η)	Sizes of gradient descent steps
Epochs (N_{epochs})	Number of times learning is performed
Batch size (N_{batch})	Number of data points on which to train the network
Weight decay (λ_{wd})	Decay on neural network weights
Confidence (c)	Scaling of the residual output
Gradient max norm (γ_{Jac})	Clipping threshold for the gradient of parameters
* Jacobian weight (λ_{jac})	Weight of the Jacobian loss
* Prediction noise ($\sigma_{T_{\text{bat}}}$)	Noise added during training
* Error threshold ($\varepsilon_{\text{thresh}}$)	Size of errors to be included in training
* Teacher forcing interval (N_{rolling})	Simulation steps before re-initialization

The number of hidden layers, together with the hidden dimension, determines the function space of the neural network. Larger values allow the neural network to output functions with a larger range and thus more complex behavior. Consequently, using such a neural network in an MPC optimization will increase the solver's computational time. Additionally, large values for the number of hidden layers and the hidden dimension risk overfitting the network. However, a neural network with few layers and a small hidden dimension becomes a low-order nonlinear mapping, which may not justify the use of neural networks. Since the goal of the thesis is to evaluate neural networks in model predictive control, the minimal architecture considered uses two hidden layers with a minimum hidden dimension of 16.

The number of epochs determines how many steps are taken in the AdamW optimizer. More epochs should result in better-trained neural networks but also increase the training time. Ultimately, the model update should be performed as close as possible to the time when the data were collected; therefore, the number of epochs was small compared to traditional deep learning. The learning rate was kept as high as

possible to ensure that the neural network optimizer takes meaningful steps towards the minimum while not overshooting any minima. The learning rate is relatively large compared to traditional deep learning, partly because the number of epochs was relatively small. The tuning approach treated the batch size as something that should be as large as possible while still allowing the neural network to update often enough to incorporate new information. For the drive cycle under consideration here, and with the controllers used here, roughly the first 1000 s will involve a pure heating or cooling scenario, where a high power and a high coolant flow rate are used. After the first 1000 s, the battery will enter a region where the pump power and coolant flow are decreased, since the controller wants to maintain its purported steady state. The weight decay was kept relatively large to ensure that the network is regularized.

Confidence in the nominal model is high, since it performs well in simulations for matched models. Thus, the confidence c was set low to bound the maximum correction made by the neural network. Clipping the maximum gradient norm of the parameters (γ_{Jac}) restricts the magnitude of the gradients the neural network produces, which limits rapid changes in the learned residual and helps maintain stability and smoothness.

The Jacobian weight is a penalty on the Jacobian of the residual neural network with respect to the battery temperature, and it should be set such that the residual network becomes smooth to facilitate its use in the MPC optimization. The prediction noise $\sigma_{T_{\text{bat}}}$ determines how much noise is added to the predictions generated by the neural network. This should teach the neural network not to overreact to slight deviations but instead to learn the underlying physics. The error threshold $\varepsilon_{\text{thresh}}$ determines which errors the neural network should attempt to remedy. If this is set too low, the neural network will try to explain every variance between the nominal and true values. Trying to remedy small errors is meaningless because of the closed-loop approach used. Therefore, this was set towards higher values. The teacher forcing interval determines how many steps the trajectory loss formulation takes before being reinitialized with the true temperature. The residual network should learn to perform simulations that are accurate over the entire MPC horizon. However, during training, errors are partially due to the neural network residual. If this is too high, then the neural network will contribute too much of the error itself and will only learn to counteract its own bias, instead of learning to remedy the error in the physics.

Tuning of these values was initially performed offline. The neural network was trained on a batch of data, and its performance was then evaluated on a test dataset outside the training data. Because of the neural network's ability to model complex relationships, it was found that very low errors on both the training and test data could be obtained. However, this does not guarantee good closed-loop performance. It is possible that relations that might explain residuals on the training and test data are learned but that then degrade control performance. An example would be that the neural network, instead of learning that the current contributes much

heating, instead believes that the heating power is stronger than it actually is.

Such behavior was found when testing the adaptive controller. Hyperparameters that fit the data perfectly on the test set were then applied in an online context and were found to decrease performance. Therefore, the system identification cycle was employed by first tuning hyperparameters offline and then testing them online, using closed-loop performance to iteratively adjust them. In the online setting for reference tracking, the tracking error together with the cost was used as a performance indicator. For economic model predictive control, the cost, together with potential violations of the temperature constraints, was used as a performance indicator.

3.5 Formulation of Cascaded Tank Benchmark

Cascaded tanks are used as a benchmark problem to evaluate the adaptive model predictive control method developed. Cascaded tanks are a common benchmark in nonlinear system identification, being part of three proposed benchmarks in [43]. Instead of using a dataset, a model of the cascaded tanks will be used and then used in a model predictive control framework. The residual neural networks for adaptive control of cascaded tanks will also be defined.

3.5.1 Modeling of a Cascaded Tanks System

The benchmark is a cascaded tank system. It consists of two tanks aligned vertically, where the outlet of one tank feeds into the inlet of another. The governing dynamics are determined through Torricelli's law and conservation of mass. The first tank is filled with water through the inlet pipe. The cross sectional area of this tank is A_1. At the bottom of this tank is a connecting pipe with cross section a_1 that connects the first tank to the second tank. Water flows out from the second tank through a pipe with cross section a_2. The cross section of tank 2 is A_2. The water level in each tank, with regards to the bottom of the respective tank, is h_1 and h_2 respectively. The flow of water into the first tank can be controlled using the input u_{cct}. The cascaded tanks are illustrated in Figure 3.5.

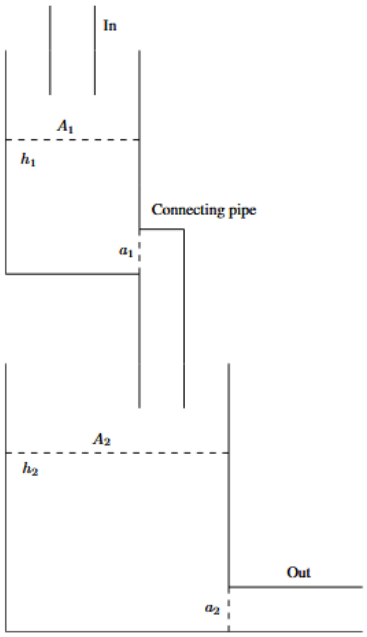


Figure 3.5: Diagram of Cascaded Water Tanks.

Torricelli's law relates the speed of a liquid flowing out of an orifice to the height from the liquid surface to the center of the hole as the following:

$$v = \sqrt{2gh} \tag{3.61}$$

where v is the speed of the liquid through the orifice, g is the gravitational constant, and h is the height from the liquid surface to the center of the hole. The change in volume with respect to time of a tank constant surface area is defined as

$$\dot{V} = A\dot{h} \tag{3.62}$$

where A is the cross sectional area of the tank and $\dot{h}$ is the change in water level with respect to time. The change in volume with respect to time of a pipe with constant surface area is defined as

$$\dot{V} = a \cdot v = a \cdot \sqrt{2gh} \tag{3.63}$$

where a denotes the cross sectional area of the pipe. Mass flow rate is related to volume flow through the following relation

$$\dot{m} = \rho \cdot \dot{V} \tag{3.64}$$

with ρ is the density of the liquid. Using this the mass flow rate through the inlet/outlet pipe can be described as the following

$$\dot{m}_{\text{tank}} = \rho \cdot A \cdot \dot{h} \tag{3.65}$$

$$\dot{m}_{\text{pipe}} = \rho \cdot a \cdot \sqrt{2gh}. \tag{3.66}$$

Additionally, water flows into the first tank by a pump. This term is defined as

$$\dot{m}_{\text{input}} = k_{\text{conv,cct}} \cdot u_{\text{cct}} \tag{3.67}$$

with u being the voltage applied to the pump, $k_{\text{conv,cct}}$ being a unit conversion constant. The law of conservation of mass dictates that

$$\dot{m}_{\text{tank}} + \dot{m}_{\text{pipe}} + \dot{m}_{\text{input}} = 0 \tag{3.68}$$

This is the mass balance equation. Note that a mass balance applies to each tank. Rearranging the mass balance and solving for $\dot{h}$ the following dynamics are obtained

$$f_{\text{cct}}(h_1, h_2, u_{\text{cct}}) = \begin{bmatrix} \dot{h}_1 \\ \dot{h}_2 \end{bmatrix} = \begin{bmatrix} \frac{k_{\text{conv,cct}}}{\rho A_1} u_{\text{cct}} - \frac{a_1}{A_1}\sqrt{2gh_1} \\ \frac{a_1}{A_1}\sqrt{2gh_1} - \frac{a_2}{A_2}\sqrt{2gh_2} \end{bmatrix} \tag{3.69}$$

where the mass balance has been performed for each tank. Note that the input is applied upon the first tank. For the equation to hold it is important that $a_1 << A_1$ and $a_2 << A_2$. Equation (3.69) is a nonlinear state equation due to the square root.

3.5.2 Model Predictive Control Formulation of Tracking Problem for Cascaded Tanks

The model developed in Section 3.5.1 is used in a model predictive control formulation for reference tracking of water level in the cascaded tanks. The states are defined as the water level of each tank and are denoted as $x_{\text{cct}} = [h_1 \ h_2]^T$ and u_{cct} being the control input. The dynamics are denoted as

$$\dot{x}_{\text{cct}} = f_{\text{cct}}(h_1, h_2, u_{\text{cct}}) = f_{\text{cct}}(x_{\text{cct}}, u_{\text{cct}}) \tag{3.70}$$

The stage cost $\ell_{\text{cct}}(x_{\text{cct}}(t), u_{\text{cct}}(t))$ is defined as

$$\begin{aligned} &\ell_{\text{cct}}(x_{\text{cct}}(t), u_{\text{cct}}(t)) = \\ &= \left(\begin{bmatrix} x_{\text{cct}}(t) \\ u_{\text{cct}}(t) \end{bmatrix} - \begin{bmatrix} r_{\text{x,cct}}(t) \\ r_{\text{u,cct}}(t) \end{bmatrix} \right)^T \begin{bmatrix} Q_{\text{cct}} & 0 \\ 0 & R_{\text{cct}} \end{bmatrix} \left(\begin{bmatrix} x_{\text{cct}}(t) \\ u_{\text{cct}}(t) \end{bmatrix} - \begin{bmatrix} r_{\text{x,cct}}(t) \\ r_{\text{u,cct}}(t) \end{bmatrix} \right) \end{aligned} \tag{3.71}$$

with references $r_{\text{x,cct}}(t), r_{\text{u,cct}}(t)$ and weighting matrices Q_{cct} and R_{cct}. The weighting matrices are defined as

$$Q_{\text{cct}} = 10 \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.72}$$

$$\tag{3.73}$$

$$R_{\text{cct}} = 0.1 \tag{3.74}$$

The terminal cost is defined to be the same as the stage cost, i.e. the Riccati based terminal cost is not computed here. The model predictive control problem is solved from an initial time t_0 to a final time T_f with the initial condition $x_{\text{cct}}(0) = x_{0,cct}$.

$\mathcal{X}_{\text{cct}}$ is the state constraint set and $\mathcal{U}_{\text{cct}}$ is the input constraint set. The constraints are formulated as the following

$$\mathcal{X}_{\text{cct}} = [h_{\min}, h_{\max}] \tag{3.75}$$
$$\mathcal{U}_{\text{cct}} = [u_{\min}, u_{\max}] \tag{3.76}$$

Where $h_{\min} = [h_{\min,1}, h_{\min,2}]^T$ denotes the minimum liquid level for both states, $h_{\max} = [h_{\max,1}, h_{\max,2}]^T$ denotes the maximum liquid level for both states, $u_{\min,\text{cct}}$ denotes the minimum pump voltage, and $u_{\max,\text{cct}}$ denotes the maximum pump voltage. The model predictive control problem is then formulated as the following

$$\begin{aligned} \min_{x_{\text{cct}}, u_{\text{cct}}} & \int_{t_0}^{T_f} \ell_{\text{cct}}\left(x_{\text{cct}}(t), u_{\text{cct}}(t)\right) dt \\ \text{s.t. } & \dot{x}_{\text{cct}}(t) = f_{\text{cct}}\left(x_{\text{cct}}(t), u_{\text{cct}}(t)\right) \\ & x_{\text{cct}}(t) \in \mathcal{X}_{\text{cct}} \\ & u_{\text{cct}}(t) \in \mathcal{U}_{\text{cct}} \\ & x_{\text{cct}}(0) = x_{0,\text{cct}} \end{aligned} \tag{3.77}$$

where t_0 is the time at which (3.77) is solved and T_f is the time at the end of the planning horizon. The problem is formulated in acados and internally discretized. The final time T_f is there changed to the number of shooting nodes N where

$$T_f = N \cdot \Delta T \tag{3.78}$$

where ΔT is the sampling time. The integration of dynamics and cost the classic Runge-Kutta 4 scheme is used. The state and input references are set as the following

$$r_{\text{x,cct}} = \begin{bmatrix} h_{1,\text{ref}} \\ h_{2,\text{ref}} \end{bmatrix} \tag{3.79}$$
$$r_{\text{u,cct}} = u_{\text{ref,cct}} \tag{3.80}$$

Note that model mismatch is introduced meaning that there is a difference between the control model dynamics and the true dynamics. The true model parameters are presented in table 3.6.

Table 3.6: True parameter values.

Parameter	Value	Units
g	9.81	$\frac{m}{s^2}$
a_1	0.1	m^2
a_2	0.1	m^2
A_1	1	m^2
A_2	1	m^2
k	1000	$\frac{kg \cdot m}{V}$
ρ	1000	$\frac{kg}{m^3}$

Mismatch is introduced by setting the cross-section are of the connecting pipe to $\tilde{a}_1 = 0.7 \cdot a_1$ and the cross section area of the outflow pipe as $\tilde{a}_2 = 0.7 \cdot a_2$ in the

control model, and additionally introducing a nonlinear saturation on the output in the plant model

$$\tilde{u}_{\text{cct}} = u_{\text{cct}} \cdot \exp\left\{\frac{-|u_{\text{cct}}|}{10}\right\}. \tag{3.81}$$

Relevant simulation parameters are presented in table 3.7.

Table 3.7: Simulation parameters.

Parameter	Value	Units
ΔT	0.5	s
N	50	N/A
$h_{1,\text{init}}$	0.5	m
$h_{2,\text{init}}$	0.5	m
$h_{1,\text{ref}}$	1.0	m
$h_{2,\text{ref}}$	1.0	m
$u_{\text{ref,cct}}$	0.0	V
$h_{1,\text{min}}$	0	m
$h_{1,\text{max}}$	2.0	m
$h_{2,\text{min}}$	0.0	m
$h_{2,\text{max}}$	2.0	m
u_{min}	0.0	V
u_{max}	0.8	V

3.5.3 Neural Network Architecture for Cascaded Tanks

For the cascaded tanks the neural network has inputs $X_{\text{cct}} = [h_1, h_2, u]$, parameters $\Theta_{\text{res,cct}}$ and two outputs $f_{\text{res,h}_1}(X_{\text{cct}}, \Theta_{\text{res,cct}})$ and $f_{\text{res,h}_2}(X_{\text{cct}}, \Theta_{\text{res,cct}})$ corresponding to the residual on the water level in tank 1 and in tank 2. The control model in the adaptive controller thus becomes

$$\hat{f}_{\text{cct}}(X_{\text{cct}}, \Theta_{\text{res,cct}}) = f_{\text{nom,cct}}(X_{\text{cct}}) + f_{\text{res,cct}}(X_{\text{cct}}, \Theta_{\text{res,cct}}) \tag{3.82}$$

where $f_{\text{res,cct}}(X_{\text{cct}}, \Theta_{\text{res,cct}})$ is the concatenation of the two neural network outputs and $f_{\text{nom,cct}}(X_{\text{cct}})$ is the nominal model with mismatched parameters $\tilde{a}_1$ and $\tilde{a}_2$. Training and implementation is done analogous to Section 3.4, except that no filtering is done since true dynamics can be queried, no clipping of the gradient of the parameters is performed and no noise is added in the training of the trajectory loss. Since this is a benchmark the architecture of the neural network was not restricted to be larger than necessary. For hyperparameters of the derivative loss formulation consult table 3.8.

Table 3.8: Hyperparameters for neural network for cascaded tanks using derivative loss.

Parameter	Value
Number of hidden layers	2
Hidden dimension	8
Learning rate (η_{cct})	10^{-2}
Batch size ($N_{\text{batch,cct}}$)	50
Epochs ($N_{\text{epochs,cct}}$)	100
Weight decay ($\lambda_{\text{wd,cct}}$)	0.3
Confidence c_{cct}	2

For hyperparameters of the trajectory loss formulation consult table 3.9.

Table 3.9: Hyperparameters for neural network for cascaded tanks using trajectory loss.

Parameter	Value
Number of hidden layers	2
Hidden dimension	8
Learning rate (η_{cct})	10^{-2}
Batch size ($N_{\text{batch,cct}}$)	50
Epochs ($N_{\text{epochs,cct}}$)	100
Weight decay ($\lambda_{\text{wd,cct}}$)	0.01
Confidence (c_{cct})	0.2
Jacobian weight ($\lambda_{\text{jac,cct}}$)	0.01
Teacher forcing interval ($N_{\text{rolling,cct}}$)	10

Note that for this benchmark the confidence parameter could be chosen to match the maximum magnitude of the known model mismatch. In practice, due to the simple nature of this problem this was not deemed necessary and an arbitrary confidence was assigned that performed well in simulation.

4

Results

This chapter presents results for the cascaded tank benchmark, the reference tracking for the battery thermal management system and economic cost formulations for battery thermal management.

4.1 Benchmark: Cascaded Tanks Using Derivative Loss and Trajectory Loss

The cascaded tanks benchmark is used to evaluate the methods for adaptive model predictive control described in section 3.4.1 and 3.4.2. The trajectories are simulated using Runge-Kutta 4 integration of the true plant function. Since the plant is simulated using an explicit model, the instantaneous derivative and trajectories can be evaluated directly, which removes the need for filtering. The inputs to the neural network are the water level in the two tanks h_1 and h_2 together with the input u. Both tanks are initialized at heights of 0.05 m and the reference for both tanks is 1 m. In Figure 4.1 the water level for tank 1 is presented controlled by the adaptive controller using derivative loss, blue line, and the nominal controller, red line.

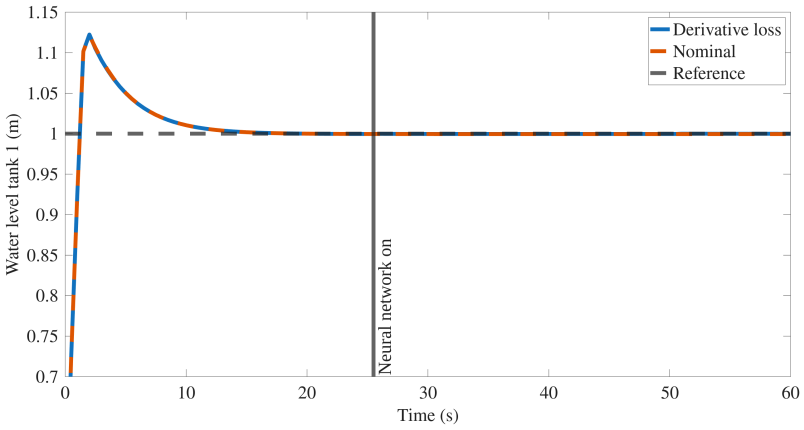


Figure 4.1: Reference tracking for cascaded tanks on a matched model for water level in the first tank. Adaptive controller uses derivative as loss function.

After an initial overshoot, both controllers start tracking the reference closely. Similarly, for the water level in the second tank, Figure 4.2 shows that both controllers follow the reference closely.

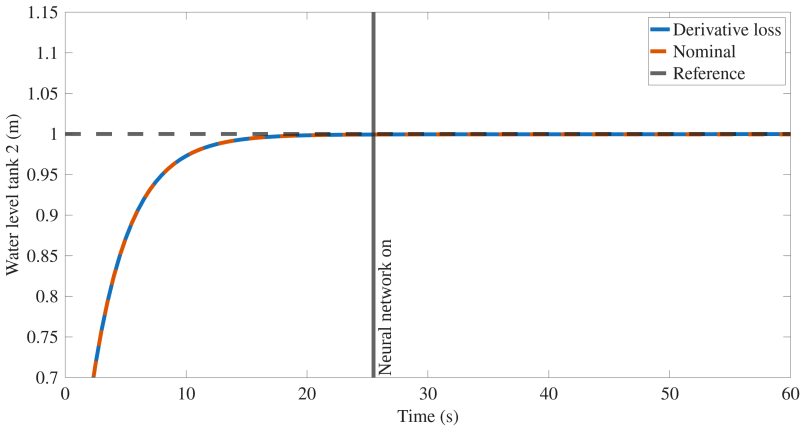


Figure 4.2: Reference tracking for cascaded tanks on a matched model for water level in the second tank. Adaptive controller uses derivative as loss function.

That the adaptive controller operates similarly to the nominal controller, on a matched model, indicates that the neural network correctly learns that the residual dynamics should be zero. This is, however, not a guarantee that, with limited training, a neural network converges to the true solution. Nevertheless, the case presented here is simple and indicates that neural networks can be used on matched models without necessarily degrading control performance. For a mismatched model, presented in Figure 4.3, the mismatch between plant and control model leads to degraded performance of the nominal controller. The nominal controller overestimates the water level, both because it assumes that the water outflow is lower and because it assumes that the input is more effective. In turn this leads to a steady state error of roughly 0.05 m.

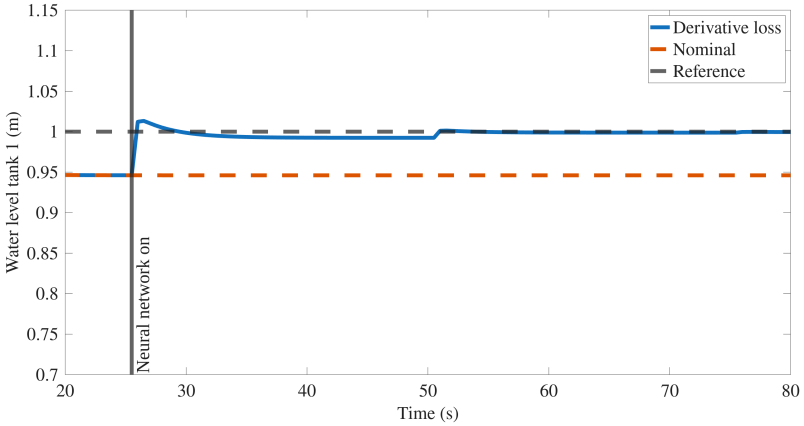


Figure 4.3: Reference tracking for cascaded tanks on a mismatched model for water level in the first tank. Adaptive controller uses derivative as loss function.

The adaptive controller instead identifies this mismatch, updates the model, and after several training instances starts tracking the reference. The same behavior can be observed for the water level in tank 2.

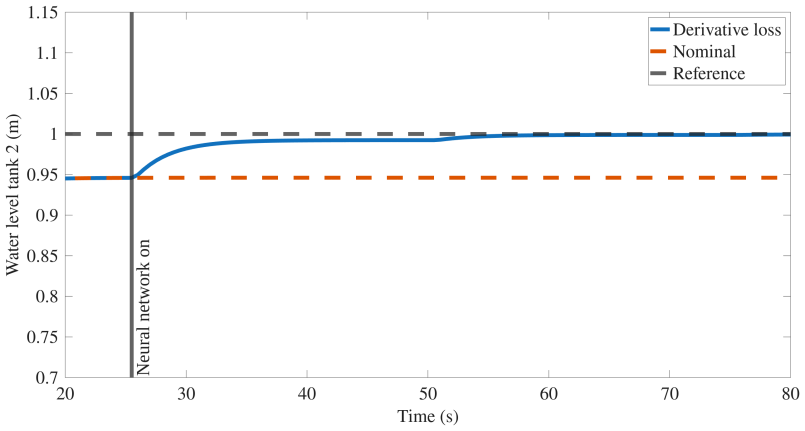


Figure 4.4: Reference tracking for cascaded tanks on a mismatched model for water level in the second tank. Adaptive controller uses derivative as loss function.

These results indicate that the derivative loss based adaptive MPC can successfully identify and compensate for model mismatch in this benchmark. suggesting that the method is promising for more complex control tasks. Next the performance of trajectory loss is evaluated. The water level of tank 1 for both adaptive controller using trajectory loss and the nominal controller is presented in Figure 4.5 on a mismatched plant.

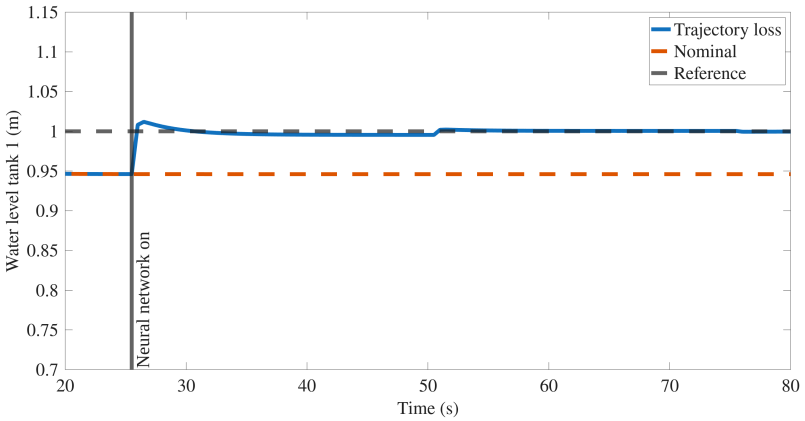


Figure 4.5: Reference tracking for cascaded tanks on a mismatched model for water level in the first tank. Adaptive controller uses trajectory as loss function.

In the same way as when using derivative loss the adaptive controller identifies the mismatch and starts tracking the reference. It is interesting to note however that the trajectory for the adaptive controller using trajectory loss is not the same as for the adaptive controller using derivative loss, with there being slight differences. The same behavior is seen for the water level in tank 2.

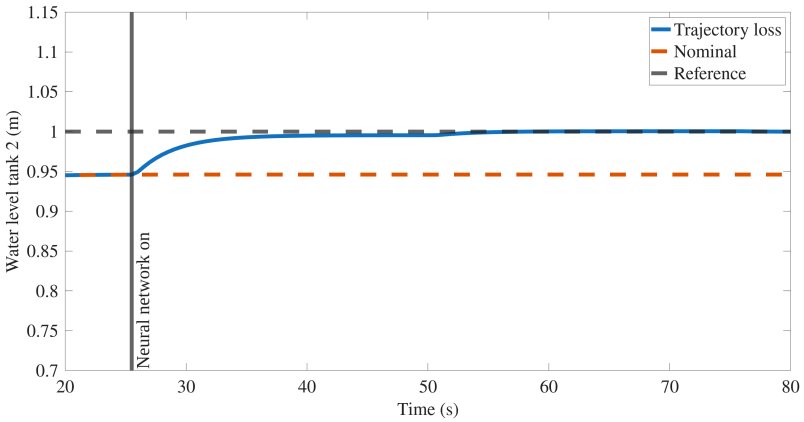


Figure 4.6: Reference tracking for cascaded tanks on a mismatched model for water level in the second tank. Adaptive controller uses trajectory as loss function.

The water level reaches the reference for both tanks, even with this being a mismatched system. As for the derivative loss, the trajectory loss formulation shows promise for more complex control tasks.

The benchmark adaptive control problem presented here is a simple case of mismatch being introduced. The mismatch in the model is based purely on mismatches in the dynamics, i.e. in the derivatives. Additionally the mismatch is time-invariant, meaning it is the same for all steps in the prediction horizon. The similar performance of the derivative loss and trajectory loss controllers here should not be seen as a general result. As discussed in section 2.3, the minimization of the two different loss functions is not equivalent and care should be taken when deciding between the two formulations. Additionally, that both controllers performed well on matched models, cannot be seen as a general result. Since the models are matched, and the loss calculation has access to function descriptions of the dynamics, the loss will be zero. Even if it could take time to converge to a network with zero weights in each layer, there will at least not be additional errors introduced from trying to estimate the dynamics.

4.2 Reference Tracking for Battery Thermal Management Using Nominal and Adaptive Model Predictive Control

This section presents results for reference tracking using nominal and adaptive model predictive control (MPC). The reference is set to $T_{\text{bat,ref}} = 20.5$ °C. Both heating and cooling performance are compared, heating corresponds to an environment temperature of 5 °C, and cooling corresponds to an environment temperature of 36 °C. These temperatures are chosen to both be the same distance to the reference. The initial battery temperature is set equal to the environment temperature. Model mismatch is introduced in some scenarios as described in Section 3.1, by increasing the environmental losses and modifying the effective heater power. Two nonlinear

heater characteristics are considered. In the first, the heater power is multiplied by an exponential term, denoted as P-E. In the second, a nonlinear saturation function is applied to the heater power, denoted as N-S. For positive heater power, the P-E and N-S functions coincide. Therefore N-S is only used in cooling cases, even though it would still have a minor effect in heating since the controller might still send negative heater input. The simulation scenarios are presented in Table 4.1.

Table 4.1: Simulation parameters for reference tracking.

Scenario	Initial temperature	Heater function	Environmental losses
Heating	5 °C	None	20 W/(m^{2}K)
Heating, P-E	5 °C	P-E	600 W/(m^{2}K)
Cooling	36 °C	None	20 W/(m^{2}K)
Cooling, P-E	36 °C	P-E	600 W/(m^{2}K)
Cooling, N-S	36 °C	N-S	600 W/(m^{2}K)

Note that, although the scenarios are named according to the heater function, the the environmental losses are the dominant source of model mismatch.

All simulations are run for 7422 s which corresponds to three full drive cycles. To collect sufficient data for training, the neural network is turned off for the first 1000 s, i.e. the controller is run on a purely nominal model. Costs are compared by evaluating the cost functions defined in Section 3.2. To avoid being overly reliant on the steady state inputs, which are calculated based on the control model, the input references are set to 0. Additionally the root-mean squared error (RMSE) of the tracking error is presented. Since the battery temperature might still be far away from the steady state at 1000 s the RMSE is calculated on data from the time step after the first controller reaches the reference, so that the steady-state tracking is not overshadowed by behavior far away from the steady state. Denote the time that the first controller reaches the reference as t_{steady}, then the sample this happens at is $N_{\text{steady}} = t_{\text{steady}}/\Delta T$. Denote the ending time of the simulation as $T_{\text{simul,end}}$ and the sample of this is $N_{\text{simul,end}} = T_{\text{simul,end}}/\Delta T$. The RMSE is then calculated as

$$RMSE = \sqrt{\frac{1}{N_{\text{simul,end}} - N_{\text{steady}}} \sum_{k=N_{\text{steady}}}^{N_{\text{simul,end}}} \left(T_{\text{bat,ref}} - T_{\text{bat}}(k)\right)^2} \tag{4.1}$$

where $T_{\text{bat}}(k)$ is the battery temperature at sample k. In the following subsections, the performance of the nominal and adaptive MPC schemes for the heating and cooling scenarios listed in Table 4.1 is compared using these metrics.

4.2.1 Reference Tracking in Battery Thermal Management Systems with Trajectory Loss Adaptive Model Predictive Control

This section presents the reference tracking behavior of both nominal and trajectory loss adaptive controllers in the battery thermal management system. The adaptive controller employs a neural network pretrained on a matched heating scenario. In Figure 4.7 the results for the matched heating scenario are presented. The adaptive controller is shown in blue, the nominal controller in red, and the temperature reference is indicated as a dashed line.

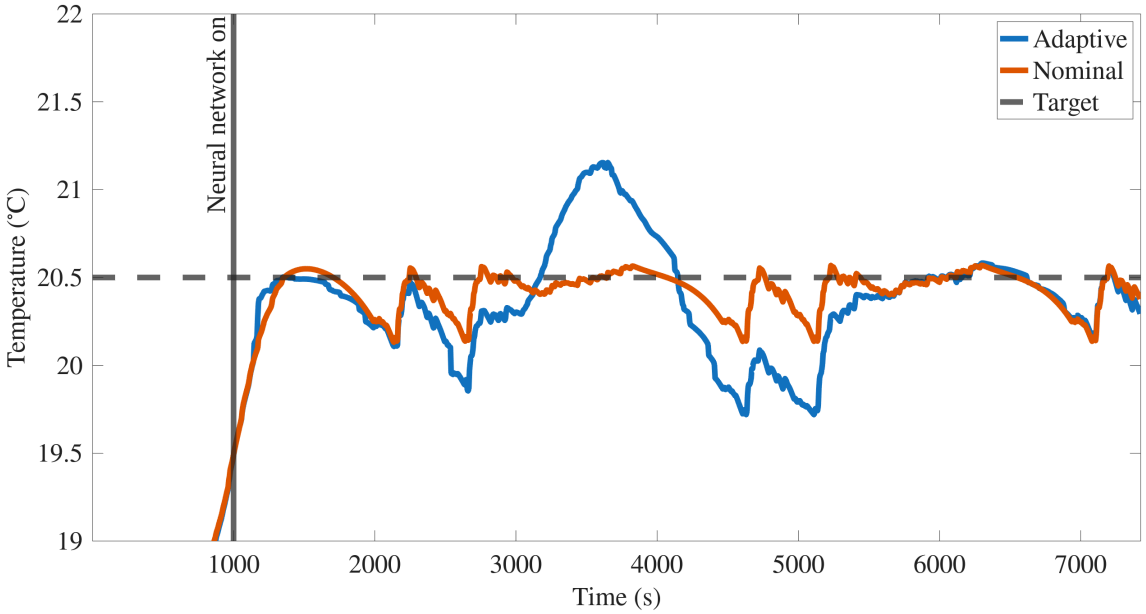


Figure 4.7: Adaptive and nominal controller tracking behavior for a matched model in a heating scenario. The nominal controller tracks the reference closely while letting the temperature drop between current spikes. The adaptive controller undershoots the reference generally and also has a large spike around 3500 s.

The figure shows that both controllers converge to a neighborhood of the reference, with both controllers intersecting the reference after roughly 2200 s. The RMSE for the nominal is 0.13 °C while for the adaptive it is 0.33 °C. Thus, neither controller follows the reference perfectly. The nominal controller exhibits dips followed by small overshoots at roughly 3000 s, 5000 s and at 7000 s. These are instances prior to spikes in current where the temperature dips. These deviations signify that the controller lets the temperature decrease and then use the current heating to reach the target again. For perfect reference tracking it would be desirable to follow the reference with no such deviations. However, the MPC cost penalizes input usage, so that the high cooling required to stay at the reference might not be worthwhile. In tracking battery temperature, small deviations are acceptable, since the actual battery performance under slight deviations is not heavily impacted. The adaptive controller follows a similar pattern, but exhibits a strong overshoot at 3500 s and otherwise generally undershoots the reference, particularly between 4000 s and 6000

s. Computing the online costs for both controller shows that the adaptive has a roughly 160 % higher cost. Together with the higher RMSE, this indicates that, in the matched model case considered here, the online adaptation of the model degrades closed-loop performance.

The heating P-E scenario is presented in Figure 4.8. Because of the exponential saturation on the heater, and due to the increased environmental losses, both controllers take longer to regulate the temperature towards the reference, with the nominal controller never intersecting the reference. The adaptive controller intersects the reference at roughly 5000 s, while being in the neighborhood of the reference after 3000 s. Similar to the matched case, both controllers exhibit the same dipping behavior between current spikes. The effect of model mismatch on the nominal controller becomes apparent. It has a RMSE of 1.22 °C and exhibits a persistent undershoot of the reference. The control model predicts that with the current input it should be able to heat up the battery, but instead it is only enough to not lose temperature. If the controllers were not operating in feedback, this error would thus have been much larger. The adaptive controller instead maintains the temperature much closer to the reference, with an RMSE of only 0.21 °C.

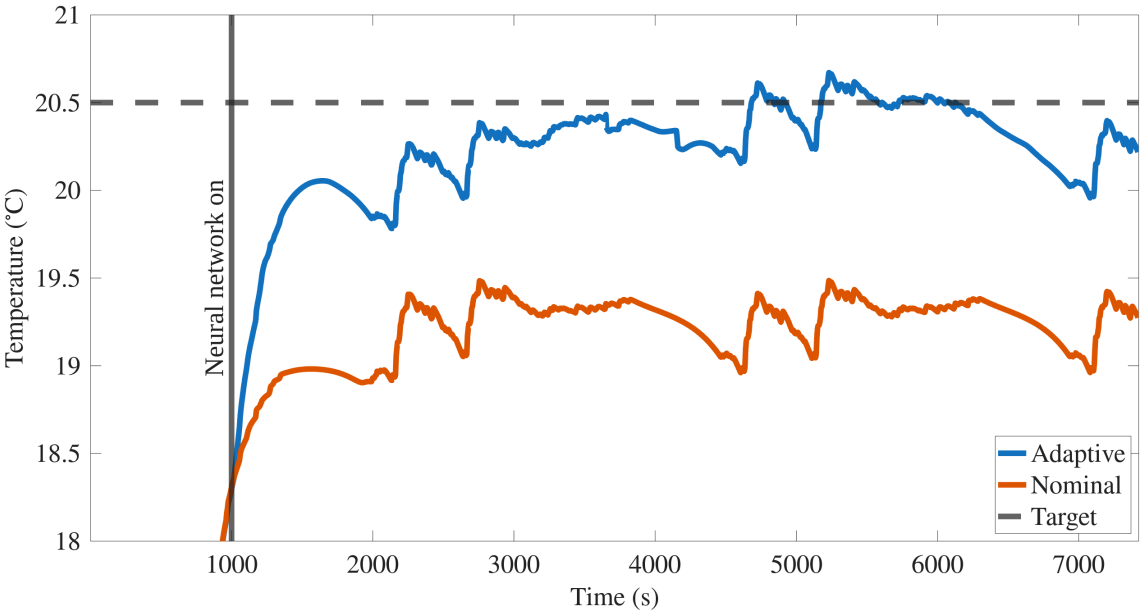


Figure 4.8: Adaptive and nominal controller tracking behavior for heating P-E scenario. The nominal controller has a clear undershoot of 1 °C to 1.5 °C. The adaptive controller instead tracks the reference closely.

Comparing costs gives that the adaptive achieves a roughly 44 % cost decrease compared to the nominal controller. Moreover, it does not appear to undershoot the reference in the same way it did for the matched case. Under high model mismatch, which this scenario simulates, the neural networks is exposed to more informative data because the residuals are larger. This behavior may be explained by the increased informativeness of the data under mismatch.

Cooling on a matched model is presented next. The nominal model parameters were fitted for a heating scenario, meaning that some degradation in performance for cooling should be expected. This is corroborated by the RMSE which is 0.25 °C compared to 0.13 °C for the matched heating case. However, as can be seen in Figure 4.9, the nominal controller still performs reasonably well. It does not track the reference as well as it did in the heating case, generally maintaining a slight undershoot relative to the reference. Additionally, the adaptive controller reaches the reference quicker at 1500 s compared to the nominal reaching it roughly 300 s later. However, the adaptive then drifts below the reference, undershooting down to a temperature of about 19 °C.

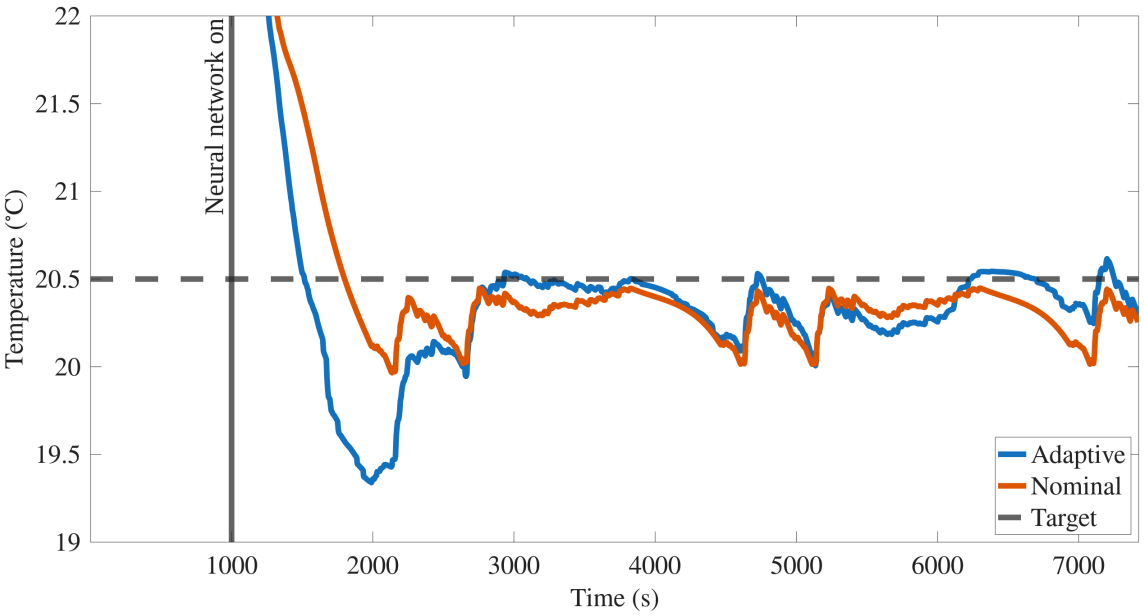


Figure 4.9: Adaptive and nominal controller tracking behavior for a matched model in a cooling scenario. The nominal controller has a general undershoot of 0.25 °C. The adaptive shows a stronger undershoot, except for times 3000 s to 4000 s where it tracks the reference closely.

The undershooting behavior for the adaptive controller may be the result of the neural network overfitting to the previous time steps. In the batch before the 1000 s mark the heater is close to the maximum cooling. Similarly the pump flow rate is at its highest to ensure high coolant flow and that the heat flow is distributed to the battery. When the battery temperature gets closer to the reference the residual network enters an input combination, low cooling and reduced flow, that were not present in the training data. By 2000 s the adaptive appears to correct this behavior and maintains the temperature closer to the reference, though it undershoots between 4000 s and 6000 s. In total, the nominal controller again outperforms the adaptive controller in the matched cooling case. The adaptive controller incurs a cost that is approximately 20 % higher than that of the nominal controller, mainly attributable to the undershooting of the reference.

Next a case of cooling P-E model mismatch is shown in Figure 4.10. Similar behavior

to as in the case of mismatch for heating is observed. The nominal controller has an RMSE error of roughly 1.26 °C, whereas the adaptive controller maintains the temperature close to the reference, with only slight deviations.

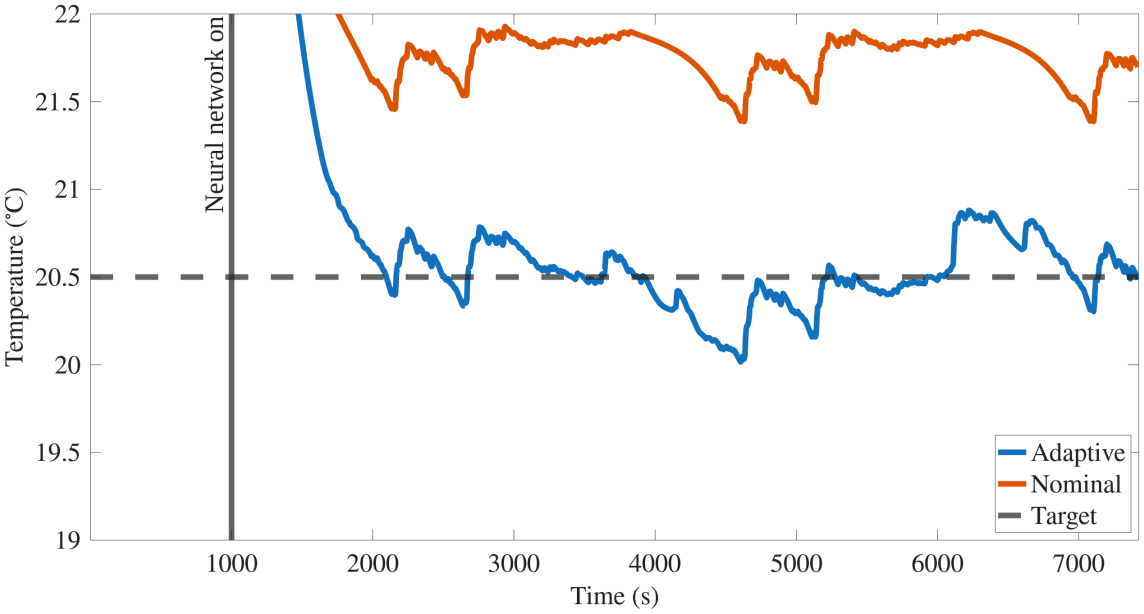


Figure 4.10: Adaptive and nominal controller tracking behavior for cooling P-E mismatch. The nominal controller has an offset of approximately 1.26 °C. The adaptive controller follows the reference closely.

Comparing costs gives that the adaptive controller has a cost that is 26 % lower than the nominal. Comparing RMSE values shows that the nominal has an RMSE of 1.26 °C while the adaptive controller has a smaller of 0.18 °C.

Nonlinear saturation is presented in Figure 4.11. Similar to other cases of the mismatch the nominal controller has a steady state offset. It is larger than the cooling P-E scenario presented above, since here the heater is less rather than more powerful. The adaptive controller reaches the reference but at around 5000 s the battery temperature starts increasing and drifts away from the reference.

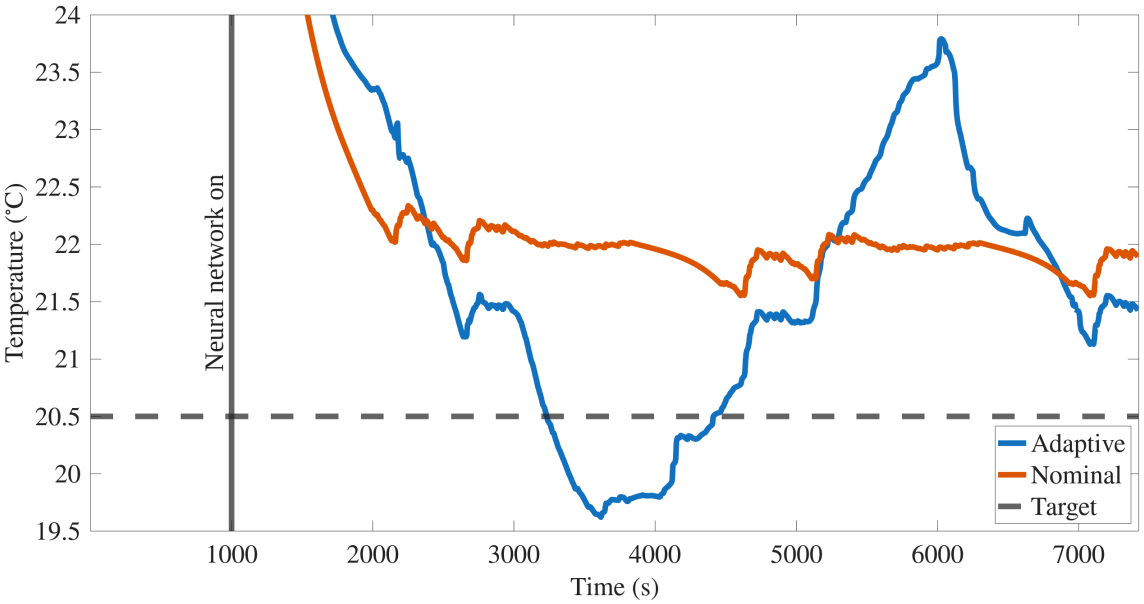


Figure 4.11: Battery temperature when controlled by adaptive and nominal controllers under nonlinear output saturation.

To better understand this behavior, the heater power inputs generated by the controllers are shown in Figure 4.12.

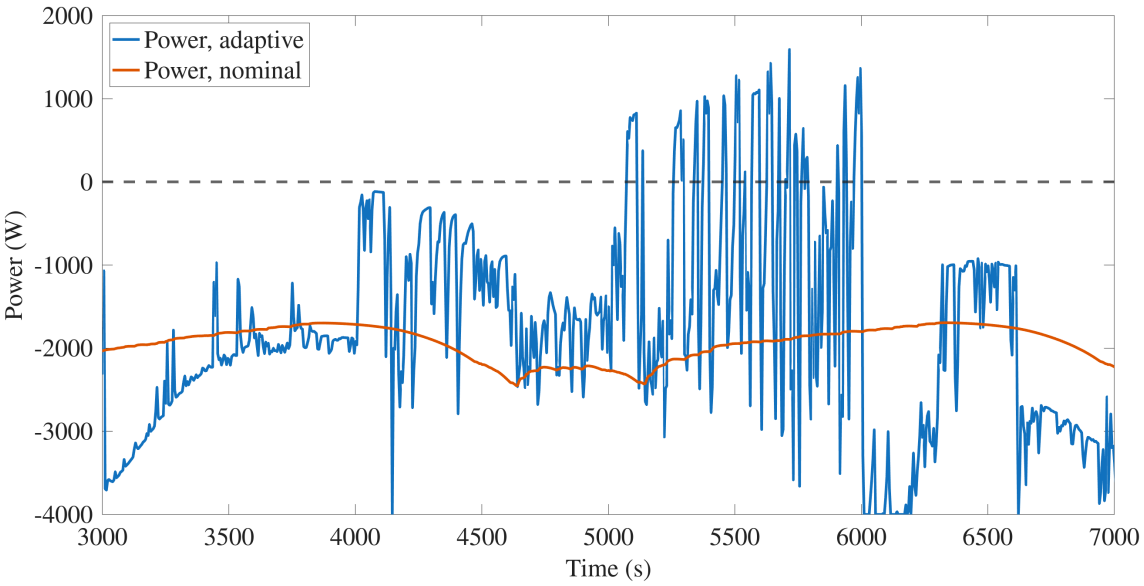


Figure 4.12: Heating power signal generated by the controllers. During much of the period between 3000 s and 4000 s the battery temperature for the adaptive controller was below the reference, but cooling inputs are supplied. Between 5000 s and 6000 s the controller sometimes sends heating inputs even when in this region the battery temperature is above the reference.

Noticeably the adaptive controller supplies positive power between 5000 s and 6000 s, even when in this region the battery temperature is above the reference. In this scenario, the controller has mainly sent signals to cool the system. Even during

3000 s to 4000 s, when the battery was below the reference, it supplied negative power. Consequently the neural network has little training data on the effect of positive heater power. The heating between 3000 s and 4000 s was mainly due to current. It is therefore plausible that the neural network associated the observed heating with negative heating power, effectively learning a sign error on the effects of heating power. At the next training for the neural network, at 6000 s, the adaptive controller starts cooling rapidly again, perhaps indicating that this error in learning was partially corrected. In comparison the nominal controller keeps a negative input that varies slightly in the entire interval of 3000 s and 6000 s. Even if the nominal controller uses a much simpler strategy it achieves an RMSE of 1.41 °C, compared to an RMSE of 1.51 °C for the adaptive controller. The cost on the adaptive controller is additionally 31 % higher. An important takeaway is that learning a more complex model might degrade control performance. The nominal model ultimately results in a quite simple control strategy. Neglecting current for intuition, if the battery temperature is below the reference, the controller will supply positive heating power, and if it is above the reference it supplies negative heating power. The residual neural network can instead learn incorrect dependencies when the training data are not sufficiently informative, producing counter-intuitive control actions.

Table 4.2 summarizes the reference tracking results for the nominal controller and the adaptive trajectory loss controller. Recall that Heating and Cooling correspond to matched models, whereas Heating, P-E, Cooling, P-E correspond to increased temperature losses together with a product-exponential function applied to the heater power, and Cooling, N-S corresponds to a model with increased temperature losses and a nonlinear saturation of the heater power.

Table 4.2: Results for battery temperature tracking using trajectory loss.

Scenario	$\text{RMSE}_{\text{nom}}^{\text{traj}}$	$\text{COST}_{\text{nom}}^{\text{traj}}$	$\text{RMSE}_{\text{ada}}^{\text{traj}}$	$\text{COST}_{\text{ada}}^{\text{traj}}$	$\text{COST}_{\text{ada}}^{\text{traj}}/\text{COST}_{\text{nom}}^{\text{traj}}$
Heating	0.13 °C	1860	0.33 °C	4860	2.61
Heating, P-E	1.22 °C	36100	0.21 °C	20300	0.56
Cooling	0.25 °C	17800	0.36 °C	21300	1.20
Cooling P-E	1.26 °C	92400	0.18 °C	68700	0.74
Cooling, N-S	1.41 °C	118000	1.51 °C	154000	1.31

The nominal controller outperforms the adaptive controller in all matched cases. For the mismatched cases the adaptive controller outperforms on the P-E scenarios while having worse performance with nonlinear saturation. In its current form, the adaptive scheme therefore does not have the generalizability required for deployment in a real battery thermal management system.

The results above focused on improving performance via model adaptation and were obtained under a specific weighting on state and inputs. However, reference tracking performance can also improved by increasing weighting on state relative to inputs.

Intuitively, decreasing weighting on the inputs delays the point when the controller starts reducing inputs. As a result the steady state tracking error will be lower. To explore this behavior a run where the costs on inputs were decreased by 10 is presented in Figure 4.13 under the mismatched heating scenario.

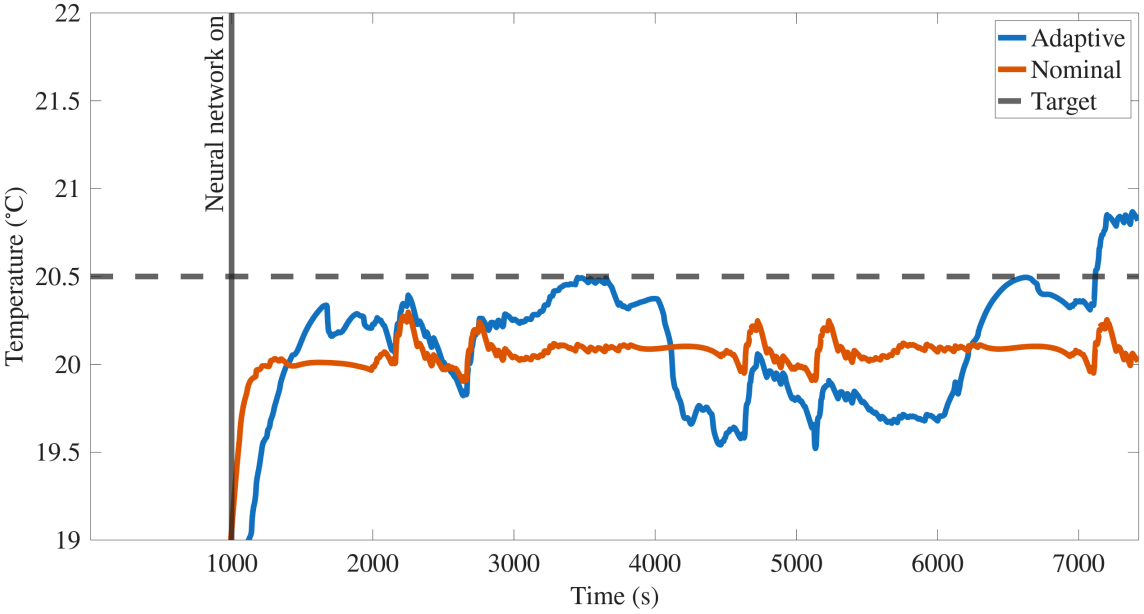


Figure 4.13: Adaptive and nominal controller tracking behavior for a mismatched model in a heating scenario using with reduced input weighting. Due to the decreased weighting on inputs, the controller uses more control effort and thus the nominal exhibits a lower steady-state error. The adaptive controller has worse performance than when using lower weightings on inputs.

Compared to the original weighting in Figure 4.8, the nominal controller now shows a smaller deviation from the reference. The battery temperature hovers around 20.0 °C for the nominal controller, instead of around 19.3 °C as in the case with higher input costs. While the nominal controller's performance improves, the adaptive controller's performance deteriorates. Instead of staying close to the reference it dips to approximately 19.5 °C around 4500 s and again shortly after 5000 s. The degraded performance of the adaptive controller can be explained by the increased input sensitivity when input weights are reduced. Lowering the cost on inputs increases the threshold at which the controller starts applying smaller inputs. This, in turn, causes more frequent switching between operating regimes, which is detrimental to the adaptive controller. Higher input costs, by contrast, dampens the inputs naturally letting the controller work in the regime it was trained on. Reducing input costs too much makes the control actions more bang-bang like, which reduces the advantage of MPC and raises the question of whether MPC should be used. These results indicate that tuning cost weights can result in increased closed-loop control performance and may be more reliable than the model adaptation investigated here.

4.2.1.1 Investigation into the Impact of the Residual Neural Network Trained on Trajectory Loss

The neural network affects the controller, as can be seen in the different trajectories between the nominal controller and adaptive controller. In this section the impact of the neural network is analyzed in more detail. The scenario under consideration is the product-exponential (P-E) mismatched heating case presented above. The neural network is trained on the first 1000 s. Three cases are considered:

1. Simulation using the nominal control model. This case is referred to as "Nominal simulation".
2. Simulation using an adaptive control model with the confidence parameter increased to 0.1. This case is referred to as "Adaptive simulation".
3. Simulation using the adaptive control model used in the reference tracking. This case will is referred to as "Adaptive control", since this is the network used in the controller.

Including case (2.), an adaptive control model with a higher confidence parameter, serves to better demonstrate the explanatory power of neural networks. In all simulations, control inputs supplied by the nominal controller operating in closed-loop are used. The simulations are run for the next 1000 s, corresponding to the planning horizon of the controller, and are shown in Figure 4.14.

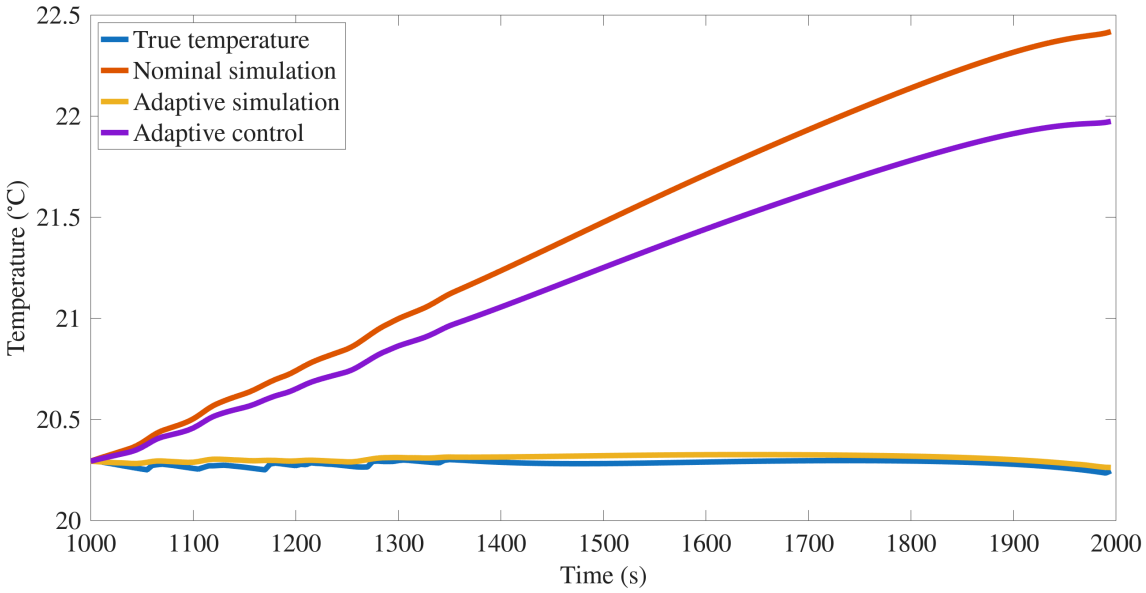


Figure 4.14: Simulation of battery temperature for 1000 s using nominal model, adaptive model with higher confidence and adaptive model used in control in a mismatched system. The nominal control model has an accumulating error that grows with each time step.

The nominal controller predicts that the temperature is much higher than the true temperature. In the closed-loop the nominal controller measures at each time step that it is far away from the reference and applies large inputs at each step. In the mismatched P-E heating case, the effective heater power is reduced and the battery loses more heat to the environment. Consequently the nominal simulation overes-

timates the battery temperature. By contrast, the adaptive simulation, case (2), has identified that there is less heating power supplied and more heat loss to the environment, instead following the true temperature much closer. Simulation using the adaptive model used in control, case (3.), lies between the nominal simulation and the adaptive simulation with more confidence. These results might suggest that a higher confidence would be beneficial. However, a high confidence was found to decrease performance, especially in matched model scenarios. Therefore, the confidence parameter used in the controller was set lower as a trade-off.

Good simulation performance is not enough to guarantee good control performance for model predictive control. When the MPC optimizer tries to find the optimum it uses Jacobians of the model with respect to states and inputs. Therefore, the Jacobians with respect to the different neural network inputs are presented, alongside the corresponding residual values. The neural network has four inputs. When evaluating the Jacobian with respect to one of those inputs the other three are fixed to the values listed in Table 4.3.

Table 4.3: Fixed values for evaluating Jacobians

	Fixed value
$T_{\text{bat,fix}}$	20 °C
$I_{\text{bat,fix}}$	40 A
ω_{fix}	200 rad/s
Q_{fix}	2000 W

The Jacobians are computed in the scaled input space to conform with the input scaling of the neural network.

First, the Jacobian of the neural network $f_{\text{res}}(X_{\text{res}})$ with respect to Q is evaluated by calculating

$$\left.\frac{df_{\text{res}}(X_{\text{res}})}{dQ}\right|_{T_{\text{bat,fix}},I_{\text{bat,fix}},\omega_{\text{fix}}} \tag{4.2}$$

for all potential values of Q, i.e. -4000 W to 4000 W or -4 to 4 in scaled units. The residual values and Jacobian are presented in Figure 4.15.

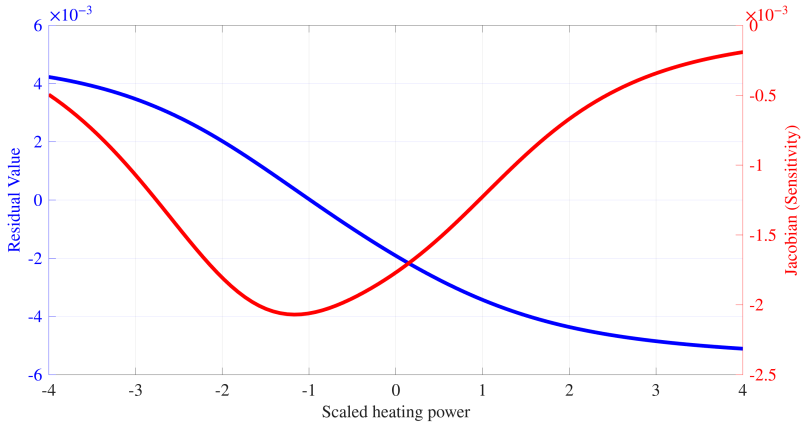


Figure 4.15: Residual value at different heating powers with the rest of inputs fixed, blue line, together with the Jacobian of the residual with respect to heating power, red line.

Both the residual and its Jacobian are smooth functions of Q. This is desirable for MPC, as it avoids discontinuities in the sensitivities used by the optimizer. The magnitude of both is small but comparable to those of the nominal model derivatives. The neural network correctly identifies that the heating is stronger than modelled for $Q < 0$ and weaker than modeled for $Q > 0$, consistent with the nonlinear function applied to the heating power. Second, the Jacobian with respect to angular velocity ω

$$\left.\frac{df_{\text{res}}(X_{\text{res}})}{d\omega}\right|_{T_{\text{bat,fix}},I_{\text{bat,fix}},Q_{\text{fix}}} \tag{4.3}$$

was evaluated at all possible angular velocities 150 rad/s to 418 rad/s, or 0.15 to 4.18 in scaled units.

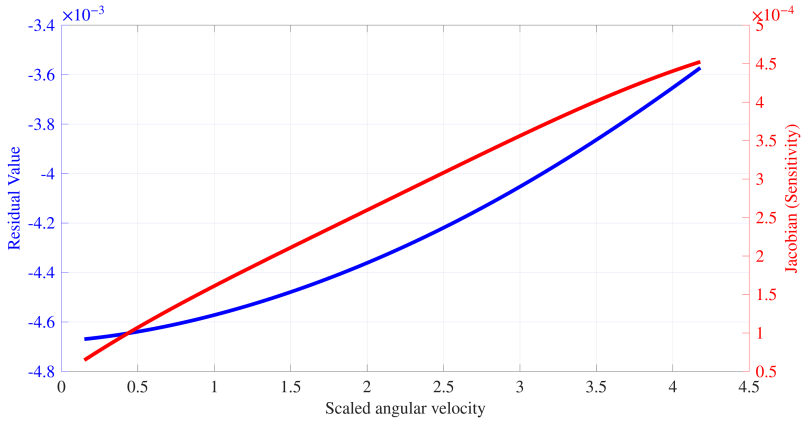


Figure 4.16: Residual value at different angular velocities with the rest of inputs fixed, blue line, together with the Jacobian of the residual with respect to angular velocities, red line.

A similar level of smoothness is observed for the Jacobian with respect to ω. No explicit mismatch was introduced in the pump model. Still, the observed residual and Jacobian could be the neural network has identified mismatch in how the pump

was modeled. Otherwise, it is also possible that the neural network tries to explain some variance with ω even though it is not the true cause. This latter corresponds to a potential failure mode for the neural network. Third, the Jacobian with respect to the current is calculated

$$\left.\frac{df_{\text{res}}(X_{\text{res}})}{dI_{\text{bat}}}\right|_{T_{\text{bat,fix}},\omega_{\text{fix}},Q_{\text{fix}}} \tag{4.4}$$

and evaluated between 0 A and 80 A, or 0 and 3.2 in scaled units.

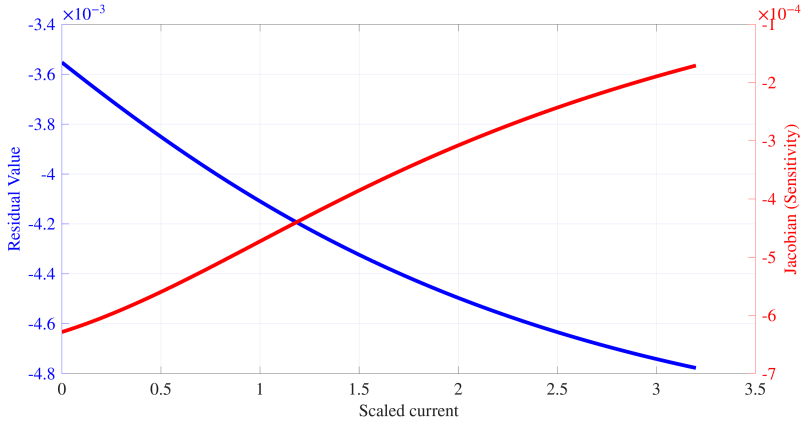


Figure 4.17: Residual value at different current values with the rest of inputs fixed, blue line, together with the Jacobian of the residual with respect to current, red line.

Again, the Jacobians are smooth. No explicit mismatch was implemented on current either, but this part of the model surely is imperfect. The controller uses a 5 s RMS current, whereas the true current changes every 1 s. This is enough to induce mismatch. Last, the Jacobian with respect to the battery temperature is calculated

$$\left.\frac{df_{\text{res}}(X_{\text{res}})}{dT_{\text{bat}}}\right|_{I_{\text{bat,fix}},\omega_{\text{fix}},Q_{\text{fix}}} \tag{4.5}$$

at temperatures between 263.15 K and 303.15 K, or 2.6315 and 3.0315 in scaled formulation.

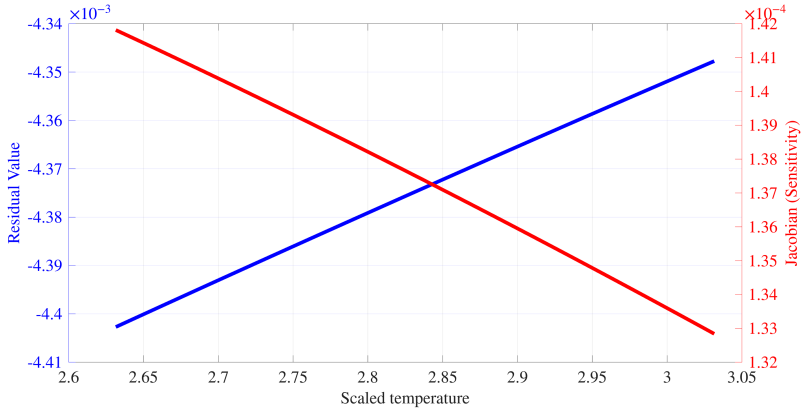


Figure 4.18: Residual value at different temperature values with the rest of inputs fixed, blue line, together with the Jacobian of the residual with respect to temperature, red line.

Both the residual and the Jacobian are smooth, nearly linear. The mismatch implemented on battery was increasing losses to the environment, which should scale linearly since the heat loss to the environment is a linear term. However, the battery temperature also impacts the dynamics in a nonlinear manner by its coupling with the coolant temperature. The almost linear residual therefore represents a simplification rather than a full representation of the nonlinear mismatch.

Overall, the Jacobians of the residual model are smooth and well behaved functions. This suggest that the MPC optimizer should be able to evaluate and use them without numerical difficulty. However, smoothness is not enough to ensure good control performance. Due to the absence of a ground-truth residual function, it is not possible to fully verify if the learned residuals model accurately captures the true model mismatch.

4.2.2 Reference Tracking in Battery Thermal Management Systems with Derivative Loss Adaptive Model Predictive Control

This section presents the results for the adaptive controller trained using derivative loss. The nominal controller-results are repeated for convenience and to allow for easier comparison between adaptive and nominal controllers. Figure 4.19 shows the results for heating in a matched scenario.

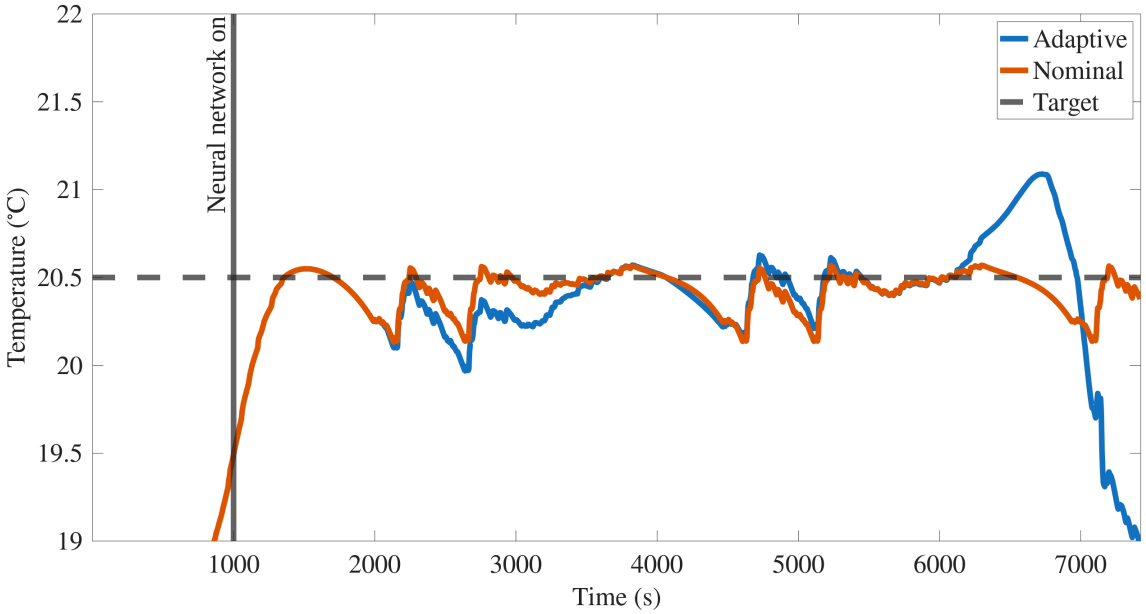


Figure 4.19: Adaptive, trained on derivative loss, and nominal controller tracking behavior for a matched model in a heating scenario. The nominal controller tracks the reference closely while letting the temperature drop between current spikes. The adaptive controller undershoots the reference more than the nominal controller and also starts driving the temperature rapidly downward after 7000 s.

The adaptive controller performs worse than the nominal controller. Only between

4000 s and 6000 s are their performances similar. Outside of this interval, the adaptive controller either undershoots the reference more than the nominal controller, or exhibits strong overshoots. Towards the end of the simulation the adaptive controller also starts to cool the battery rapidly. This occurs after the spike between 6000 s and 7000 s and can be seen as an over-correction from the adaptive controller. Overall, the adaptive controller has a cost that is approximately 409 % higher than the nominal controller. Worse performance can also be seen in the RMSE values, with the nominal achieving an RMSE of 0.13 °C while the adaptive has an RMSE of 0.41 °C. For the mismatched heating P-E scenario, shown in Figure 4.20, the adaptive controller once again behaves poorly and performs worse than the nominal controller.

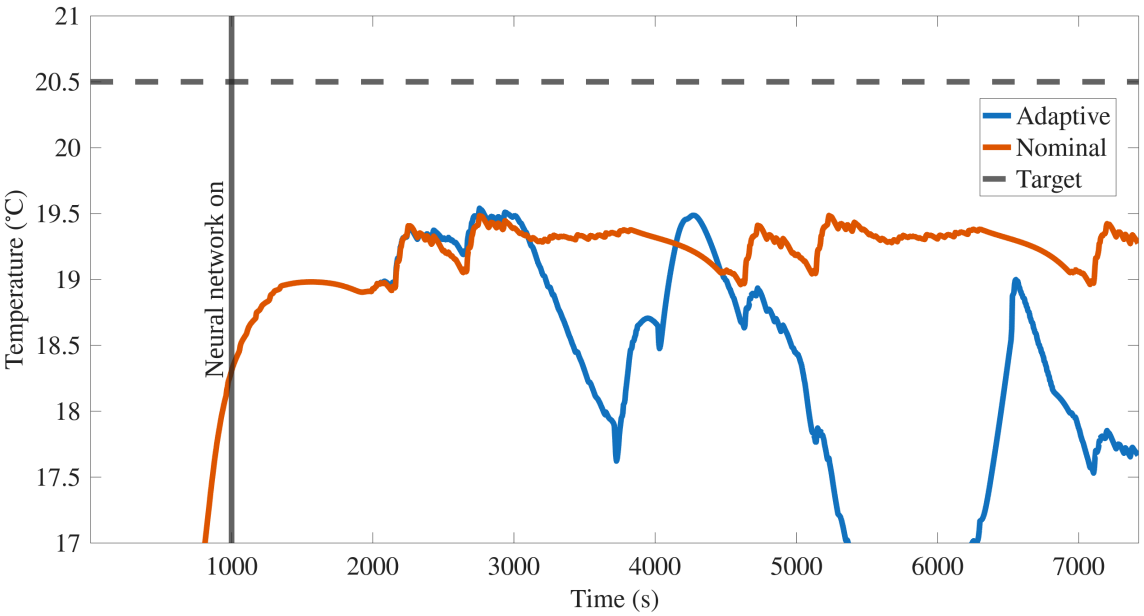


Figure 4.20: Adaptive, trained on derivative loss, and nominal controller tracking behavior for mismatched P-E model in a heating scenario. The nominal controller has a steady-state offset while the adaptive controller not only has this offset but even dips below it at 3700 s and again at 6000 s.

Neither controller reaches the reference in this scenario, so the RMSE is calculated starting from 1000 s. The adaptive controller has a higher RMSE of 2.36 °C while the nominal controller has an RMSE of 1.31 °C. Next the temperature trajectories are shown for a matched cooling scenario in Figure 4.21.

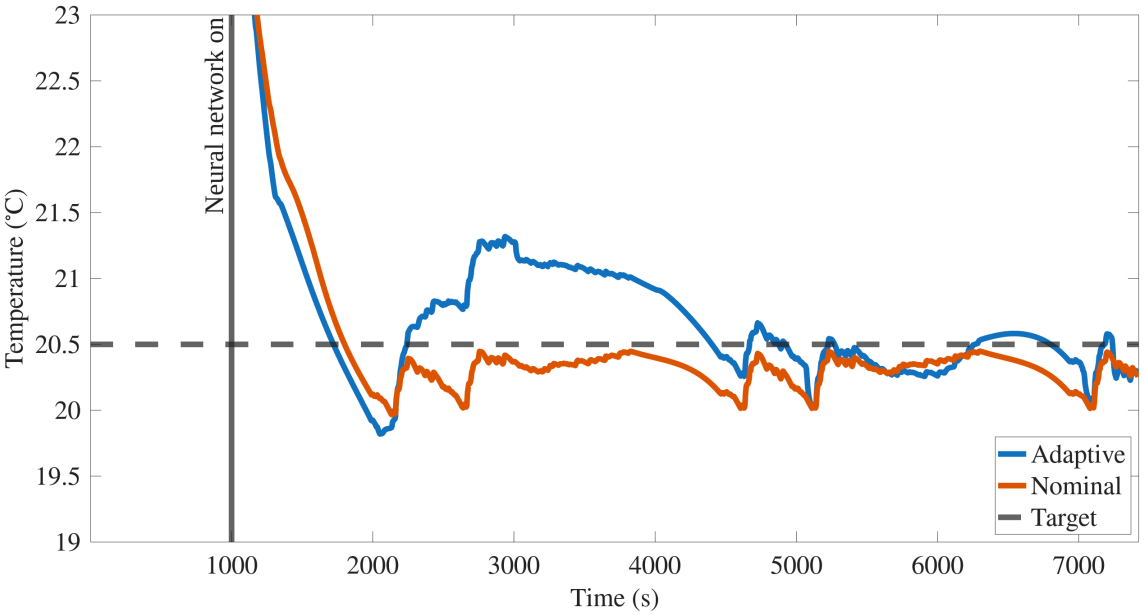


Figure 4.21: Adaptive, trained on derivative loss, and nominal controller tracking behavior for a matched model in a cooling scenario. The adaptive controller reaches the reference slightly faster but then undershoots it and subsequently overshoots it. The nominal controller has a steady-state offset below the reference.

The RMSE for the nominal controller is 0.23 °C while for the adaptive it is higher at 0.36 °C.

The case of cooling under a P-E mismatched model is shown in Figure 4.22.

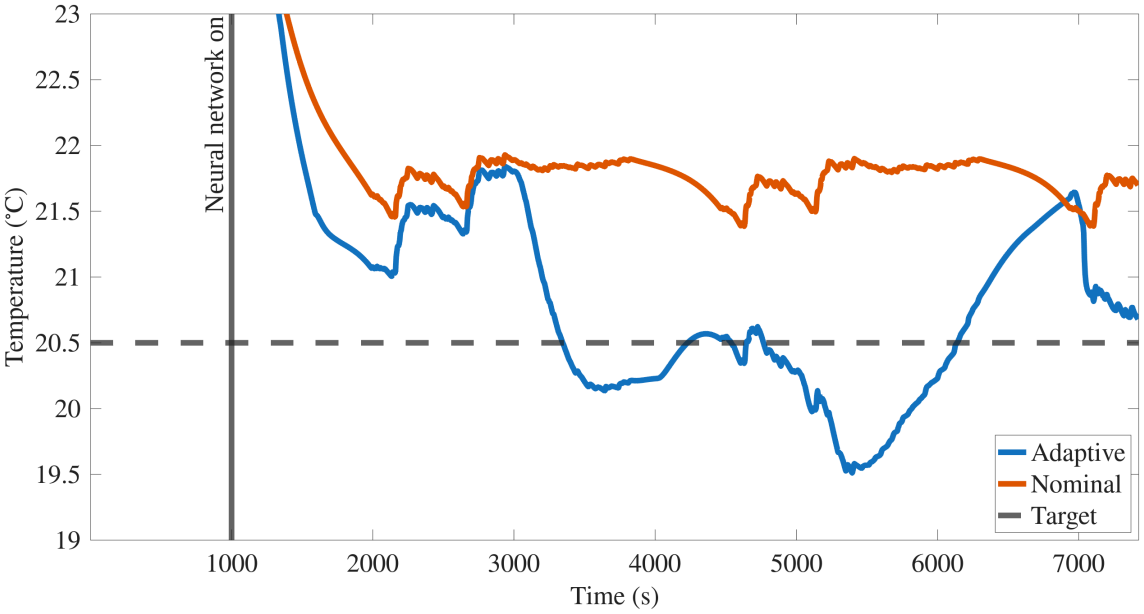


Figure 4.22: Adaptive, trained on derivative loss, and nominal controller tracking behavior for a P-E mismatched model in a cooling scenario. The adaptive controller reaches the reference while the nominal stays above the reference.

Here the adaptive controller reaches the reference, while the nominal controller maintains a steady state offset above the reference. Although the adaptive controller both undershoots and then again overshoots the reference its RMSE is lower at 0.52 °C compared to the nominal controller RMSE of 1.26 °C.

The battery temperature for nonlinear saturation is shown in Figure 4.23.

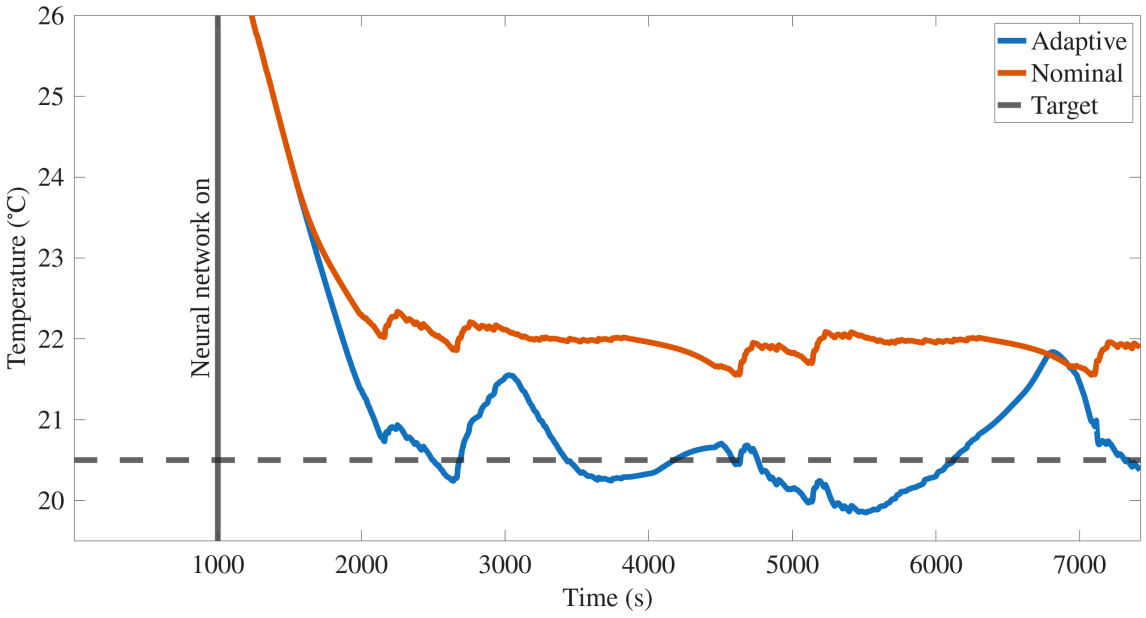


Figure 4.23: Adaptive, trained on derivative loss, and nominal controller tracking behavior for a mismatched model in a cooling scenario with nonlinear saturation of the heater power. The adaptive controller reaches the reference while the nominal stays above the reference.

Here the adaptive controller does not exhibit the same heating problem as occurred in the adaptive controller trained on trajectory loss in the nonlinear saturation scenario, see Figure 4.11. The RMSE of the adaptive controller is also smaller than of the nominal at 0.52 °C instead of 1.43 °C.

Table 4.4 summarizes the reference tracking results for the nominal controller and the adaptive derivative loss controller. Recall that Heating and Cooling correspond to matched models, whereas Heating, P-E, Cooling, P-E correspond to increased temperature losses together with a product-exponential function applied to the heater power, and Cooling, N-S corresponds to a model with increased temperature losses and a nonlinear saturation of the heater power.

Table 4.4: Results for battery temperature tracking using derivative loss.

Scenario	$\text{RMSE}_{\text{nom}}^{\text{deriv}}$	$\text{COST}_{\text{nom}}^{\text{deriv}}$	$\text{RMSE}_{\text{ada}}^{\text{deriv}}$	$\text{COST}_{\text{ada}}^{\text{deriv}}$	$\text{COST}_{\text{ada}}^{\text{deriv}}/\text{COST}_{\text{nom}}^{\text{deriv}}$
Heating	0.13 °C	1860	0.41 °C	9480	5.09
Heating, P-E	1.31 °C	36100	2.36 °C	105400	2.92
Cooling	0.23 °C	17800	0.36 °C	21400	1.20
Cooling P-E	1.26 °C	92400	0.52 °C	81100	0.88
Cooling, N-S	1.43 °C	118000	0.52 °C	138000	1.18

Heating scenarios seem especially difficult for the derivative loss controller, with costs that are 192 % to 409 % higher than for the nominal controller. In heating, both the heater and the current will contribute to an increase in temperature. This poor performance could be the result of the derivative loss controller not being able to distinguish between these two sources of heating.

4.2.3 Computation Times and Solver Errors

Model predictive control (MPC) requires solving an optimization problem online. For the control input to be applied at the correct time, the solver time must be less than the controller sampling time. In adaptive model predictive control, training of the neural network also takes time. However, this training can be done in parallel to ensure that the controller can still send input signals while training is taking place. This also increases robustness in cases where the neural network optimizer fails or stalls. Table 4.5 summarizes the computation times for the three controllers, nominal MPC, trajectory loss adaptive MPC, and derivative loss adaptive MPC. The controller sampling time is 5 s. Computations were performed on a ThinkPad T14 Gen 4 with a 13th Gen Intel i5-1335U CPU.

Table 4.5: Computation times for the nominal controller and the two adaptive controllers using different loss functions.

	Nominal	Adaptive, trajectory loss	Adaptive, derivative loss
Mean MPC solver time	35 ms	76 ms	87 ms
Max MPC solver time	97 ms	259 ms	236 ms
Mean training time	N/A	30060 ms	1300 ms
Max training time	N/A	36000 ms	1500 ms

All three controllers have mean and maximum MPC solver times below the sampling time, meaning that they can be used online. As expected, the adaptive controllers have longer solver times, since the learned model dynamics are more complex. However, the MPC solver times are still low enough for online use, which highlights the usefulness of a lightweight model architecture. Mean training time for the adaptive controller using trajectory loss is approximately 6 times the sampling time, meaning

that the updates to the dynamics would be delayed by about 6 MPC steps. The adaptive controller using derivative loss is much quicker, with training times that are less than the sampling time. The longer training time for trajectory loss was expected, since it requires simulating the trajectory multiple times using tensors to correctly formulate the graph for the loss function. The implementation of trajectory loss was not optimized for computation times, and decreases in computation times could be achieved, e.g., by vectorization. Additionally, computations could be performed on a GPU, which accelerates neural network training. In this thesis, the practical implementation of the adaptive controller on embedded hardware is not explored. The computation times reported here were obtained while simultaneously dedicating substantial computational resources to simulating the battery thermal management system.

Another issue with implementing model predictive control is that the optimization solver can fail. To mitigate this, multiple methods have been implemented, such as slack variables, small values added to divisions to avoid singularities and regularization of the neural networks so that they do not yield exploding gradients. No solver errors occurred for any of the controllers in the simulations presented in this thesis. If a solver error is encountered, a fail-safe mechanism switches the controller into bang-bang control, where maximum angular velocity together with either maximum heating or maximum cooling power depending on the battery temperature is applied. This fallback strategy ensures that the battery remains within safe temperature limits even in the event of an optimization failure.

4.3 Economic Model Predictive Control for Battery Thermal Management Using Nominal and Adaptive Controllers

Results of using the economic model predictive control (EMPC) formulation derived in Section 3.3 are presented in this section. First, the performance of the nominal controller under matched models is presented to validate that the economic model predictive control (EMPC) is performing as expected. Second, the performance of nominal and adaptive EMPC schemes is presented.

In Figure 4.24 the nominal economic controller is tested in a heating scenario where the environment temperature is 0 °C. To isolate potential impact from current two cases are compared, the case where current is applied as usual, the blue line, and the case where current is turned off in the simulator and in the controller set to zero for all time steps, the red line. The grey shaded area is the target region for the controllers.

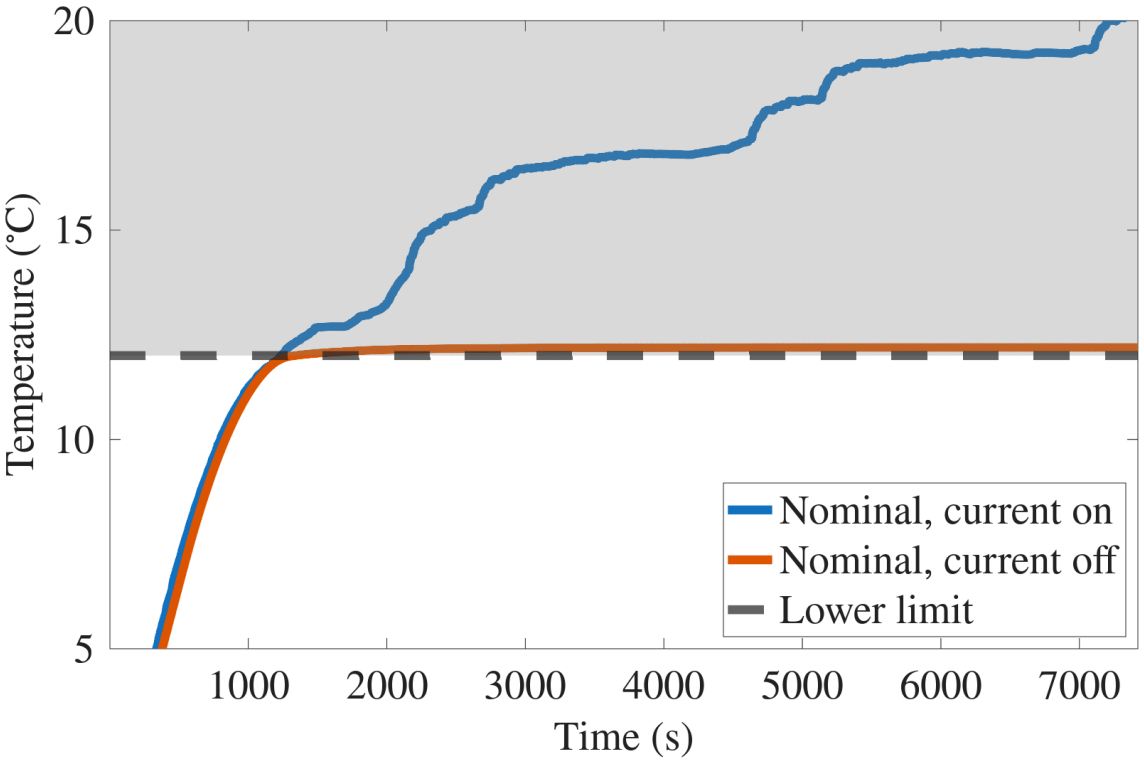


Figure 4.24: Economic model predictive control in a heating case for current turned on and off.

The economic controller behaves as expected. In the case of the current being turned off, the economic controller follows the lower limit of the target region without leaving it, meaning that the controller supplies just enough heat to stay in the target region. With the current turned on, a constant increase in temperature can be seen, which is attributable to the current heating the battery. For cooling from the initial temperature 40 °C, similar behavior is shown in Figure 4.25.

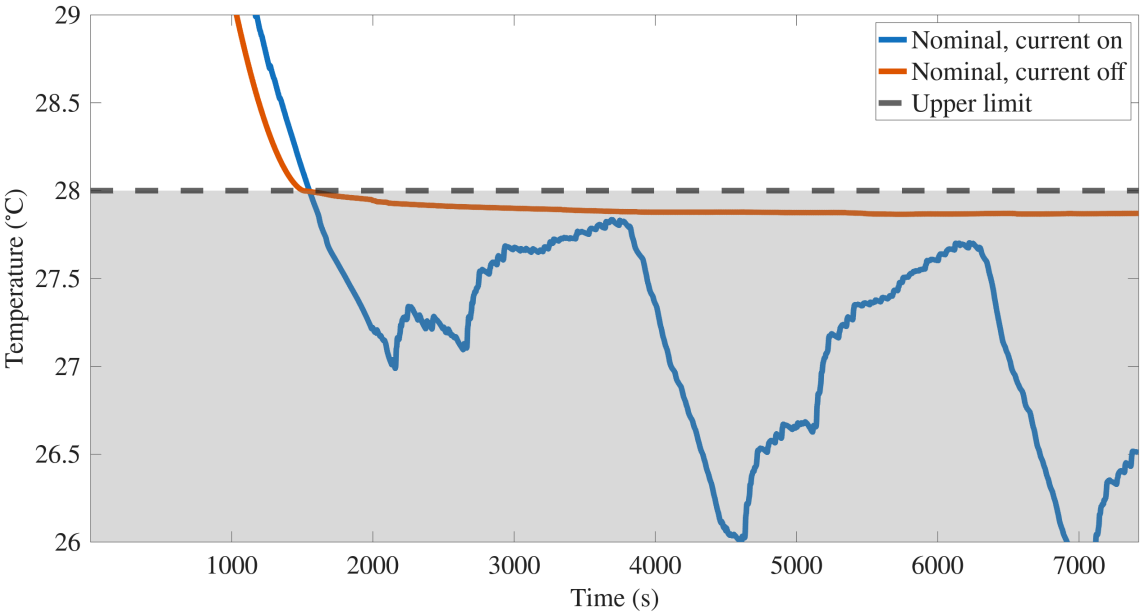


Figure 4.25: Economic model predictive control in a cooling case for current turned on and off.

For the case with the current off, the controller keeps the battery temperature just below the upper limit. In the case of current being turned on, it needs to cool the battery before large current spikes to ensure it stays in the target region. That the temperature of the battery at 3700 s and 6200 s is close to the upper limit indicates that the controller has chosen a strategy to keep the temperature in the target region while minimizing power. The cooling case is more interesting since it involves needing to cool to counteract the effect of heating from the current, whereas in heating the controller can stay idle for much of the simulation. For this reason, the offline training of the adaptive controller is performed for a cooling case, instead of heating, as was done for reference tracking.

The simulation scenarios for economic MPC are similar to those used for reference tracking. However, the environment temperatures are changed to be 12 °C away from the lower or upper limit. Model mismatch is introduced in the same way as for reference tracking. P-E is adopted as shorthand for multiplying the heater power with an exponential term, and N-S is adopted as shorthand for nonlinear saturation of the heater power. The simulation scenarios are presented in Table 4.6.

Table 4.6: Simulation parameters for economic model predictive control.

Scenario	Initial temperature	Heater function	Environmental losses
Heating	0 °C	None	20 W/(m^{2}K)
Heating, P-E	0 °C	P-E	600 W/(m^{2}K)
Cooling	40 °C	None	20 W/(m^{2}K)
Cooling, P-E	40 °C	P-E	600 W/(m^{2}K)
Cooling, N-S	40 °C	N-S	600 W/(m^{2}K)

Note, again, that even if the name of the scenario is differentiated by what function is applied to the heater, the environmental losses play a much bigger role in introducing model mismatch.

The economic cost for both controllers is evaluated using pump power P_{pump} in kilowatt, heating power $\bar{Q}$ in kilowatt, and slack on battery temperature $\bar{s}_{T_{\text{bat}}}$, incurred because the battery was outside of the target region. The cost is accumulated over the time steps for which the neural network is active, $t_{\text{nn,on}}$ until the ending time of the simulation $T_{\text{simul,end}}$. Denote these samples as $N_{\text{nn,on}} = t_{\text{nn,on}}/\Delta T$ and $N_{\text{simul,end}} = T_{\text{simul,end}}/\Delta T$ respectively. The cost is then calculated as

$$\begin{aligned} &\text{COST}^{\text{econ}} = \\ &= \sum_{k=N_{\text{nn,on}}}^{N_{\text{simul,end}}} \left(\begin{bmatrix} P_{\text{pump}}(k) & \bar{Q}(k) \end{bmatrix} \begin{bmatrix} 15 & 0 \\ 0 & 15 \end{bmatrix} \begin{bmatrix} P_{\text{pump}}(k) \\ \bar{Q}(k) \end{bmatrix} + 10\bar{s}_{T_{\text{bat}}}(k)^2 + 1000\bar{s}_{T_{\text{bat}}}(k) \right), \end{aligned} \tag{4.6}$$

which corresponds to the cost function in the economic controller, excluding slack on coolants. Battery thermal management should weigh the importance of staying in the target region against the power necessary to achieve this. The cost in (4.6) is used here to compare the two controllers under the assumption that it adequately captures this trade-off.

Due to the poor performance of the derivative loss adaptive controller in the tracking case, only the trajectory loss adaptive controller will be used here. A comparison of the adaptive and nominal economic controllers in a matched heating case is presented in Figure 4.26.

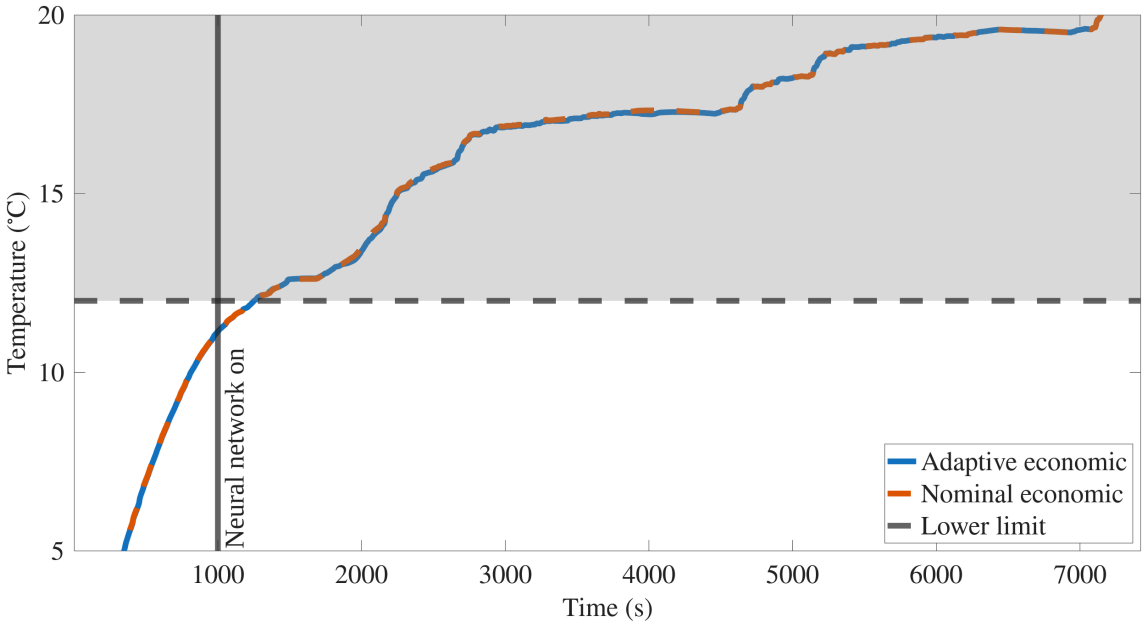


Figure 4.26: Closed-loop temperature trajectories obtained using nominal and adaptive controllers for an economic formulation in a matched heating scenario. The performance of the nominal and adaptive controllers is nearly identical. The trajectory of the nominal controller is dashed for clarity.

The matched heating case is a simple control problem, however it is good to validate that the adaptive controller does not degrade performance significantly. Overall, the adaptive controller has a slightly larger cost, with an increase of 0.2 %, showcasing that it does not discover a more efficient strategy for maintaining the battery in the target region. For a P-E mismatched heating case results become more interesting, as seen in Figure 4.27.

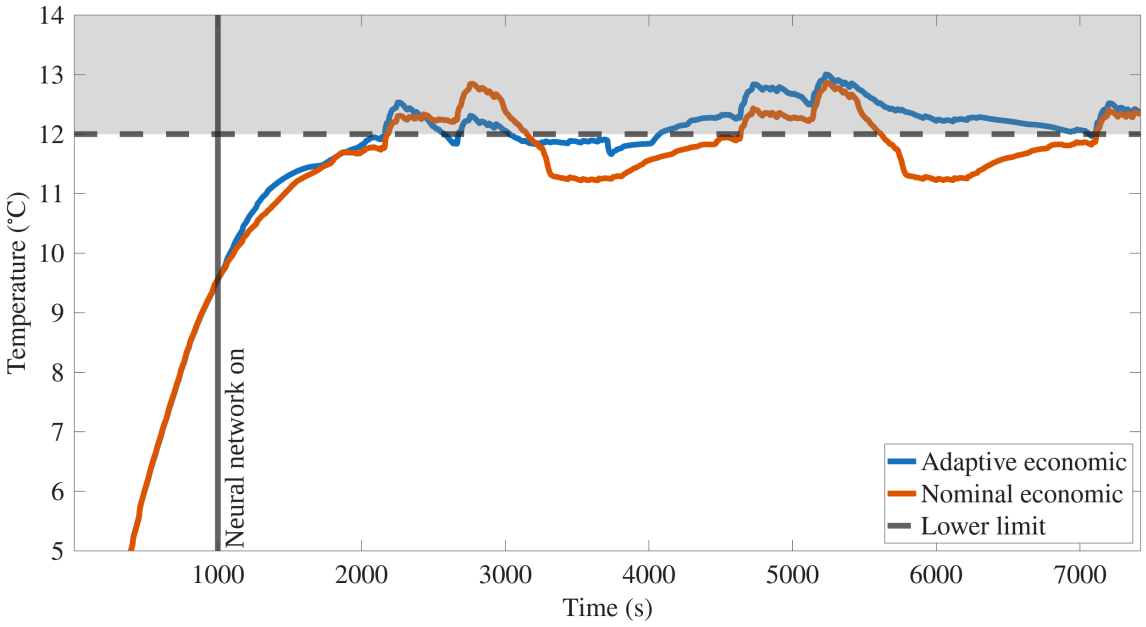


Figure 4.27: Closed-loop temperature trajectories obtained using nominal and adaptive controllers for an economic formulation in a P-E mismatched heating scenario. The nominal controller stays outside of the target region for much of the simulation, while also overcompensating and shooting further into the target region than the adaptive controller.

The nominal controller undershoots the target region on multiple occasions, while the adaptive also does this but with smaller undershooting. This reduces the violation of the target region for the adaptive controller and leads to the adaptive controller performing roughly 23 % better measured in the cost (4.6). Next the matched cooling case is presented in Figure 4.28. The adaptive controller reaches the target region quicker but then maintains a lower temperature than the nominal in the target region. This is not desirable in economic model predictive control, where staying closer to the upper limit is the objective, thereby reducing cooling effort. This calls into question the adaptive controller's ability to make small corrections to achieve lower power use.

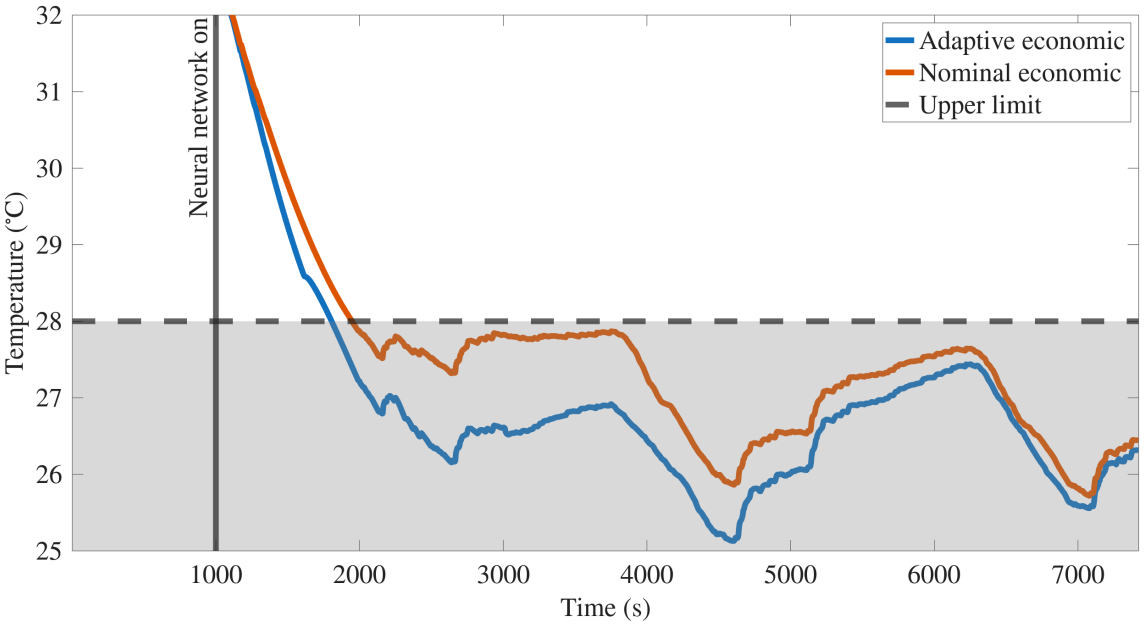


Figure 4.28: Closed-loop temperature trajectories obtained using nominal and adaptive controllers for an economic formulation in a matched cooling scenario. As opposed to heating the two controllers have noticeably different trajectories, highlighting that this is a more involved control problem.

Because of the more aggressive cooling of the adaptive controller it incurs a cost 11 % higher than the nominal controller. The P-E mismatched cooling case is presented in Figure 4.29.

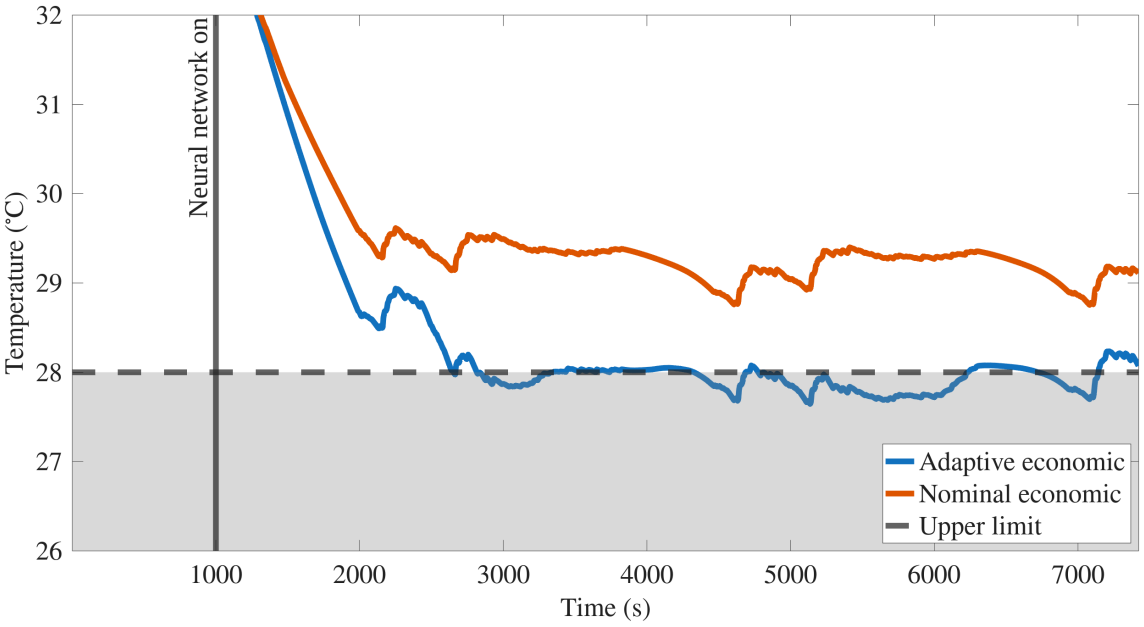


Figure 4.29: Closed-loop temperature trajectories obtained using nominal and adaptive controllers for an economic formulation in a P-E mismatched cooling scenario. The nominal controller stays outside of the target region for much of the simulation, similar to the mismatched heating case, while the adaptive controller reaches the target region.

Once again the nominal controller stays outside of the target region and thus incurs a much higher cost than the adaptive controller. The adaptive controller, except for a few overshoots, stays in the target region effectively and achieves a 32 % lower cost.

The performance under nonlinear saturation for economic MPC is presented in Figure 4.30.

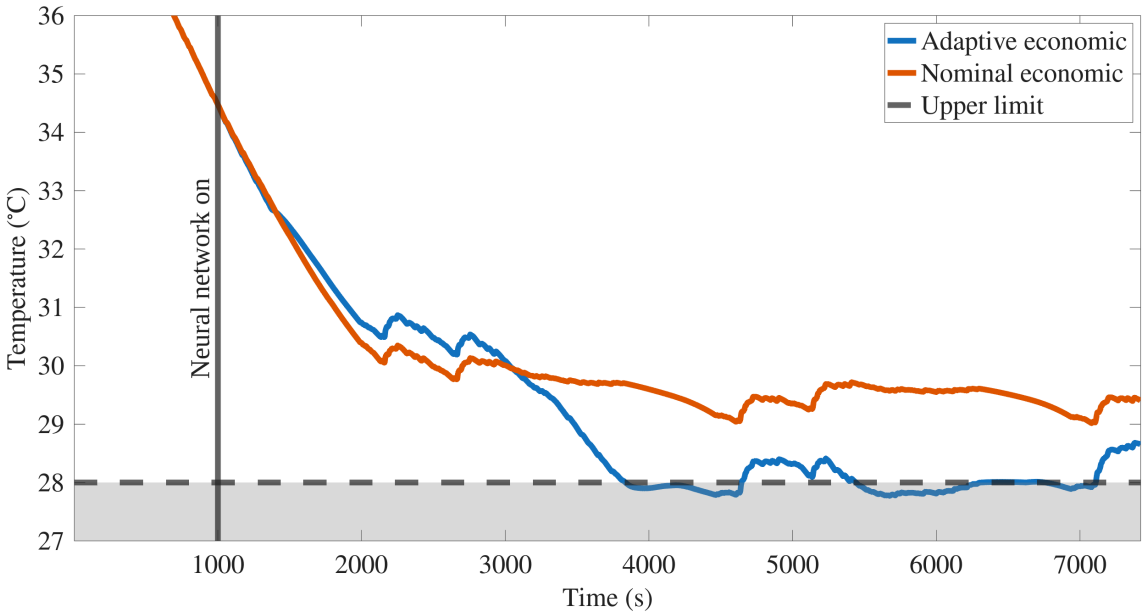


Figure 4.30: Closed-loop temperature trajectories obtained using nominal and adaptive controllers for an economic formulation in a N-S mismatched cooling scenario. The nominal controller stays outside of the target region for much of the simulation, similar to the mismatched heating case, while the adaptive controller reaches the target region.

Here, the adaptive controller does not exhibit the same problematic behavior seen in the reference tracking case with nonlinear saturation, where it applied heating power inappropriately, and instead achieves a 35 % lower cost than the nominal controller.

For an overview of the performance of the controllers in an economic model predictive formulation the Table 4.7 can be consulted. Recall that Heating and Cooling correspond to matched models, whereas Heating, P-E, Cooling, P-E correspond to increased temperature losses together with a product-exponential function applied to the heater power, and Cooling, N-S corresponds to a model with increased temperature losses and a nonlinear saturation of the heater power.

Table 4.7: Costs for economic controllers.

Scenario	$\text{COST}^{\text{econ}}_{\text{nom}}$	$\text{COST}^{\text{econ}}_{\text{ada}}$	$\text{COST}^{\text{econ}}_{\text{ada}}/\text{COST}^{\text{econ}}_{\text{nom}}$
Heating	2890	2900	1.002
Heating, P-E	24100	18500	0.77
Cooling	24100	26100	1.11
Cooling, P-E	96500	66400	0.68
Cooling, N-S	118000	75800	0.65

It can be seen that the adaptive controller outperforms the nominal on mismatched

models while having a decreased performance on the matched cases. This aligns with the formulation of the economic model predictive control, where using a target region instead of a reference gives the controller more freedom and encourages explores of more inputs. The cost for the matched heating case is almost an order of magnitude smaller than the matched cooling case, showcasing that this scenario is an easy control task where the controllers can allow the current to contribute to much of the heating. If the simulations were extended, active cooling might be necessary to avoid the battery temperature leaving the target region, especially in the case of the environmental loss not being large enough to counteract this heating.

5
Discussion

In this chapter a summary of the adaptive controller results is presented, with analysis on why the adaptive controller often degraded performance. This analysis is continued by evaluating the method and proposing directions for future work.

5.1 Summary of Adaptive Controller Results

Adaptive controllers were tested on both a cascaded tank benchmark and on a battery thermal management system. For the cascaded tank benchmark, both the derivative loss and trajectory loss adaptive controllers outperformed a nominal controller on a mismatched model. For matched models the adaptive controllers performed similarly to the nominal controller. The benchmark is a nonlinear model predictive control (MPC) task and thus the method was verified for such an application.

For battery temperature reference tracking, comparisons between the nominal and adaptive controllers yielded mixed results. Table 4.4 shows that the adaptive controller with derivative loss performed worse on four out of five scenarios, in terms of higher cost. Table 4.2 shows that the adaptive controller with trajectory loss achieved better performance, i.e. lower RMSE and cost, for product-exponential (P-E) mismatch, but exhibited worse performance under other conditions. The worse performance under nonlinear saturation deserves more analysis. Nonlinear saturation masks the effect of negative and positive heater powers. A recurring issue for all cases of adaptive control is the need to sufficiently explore the input space before training the neural network. In this work, no explicit check was performed to verify that the input space had been sufficiently explored before initializing training. It would be possible instead of training the neural network every 1000 s to wait until a sufficient portion of the input space has been explored.

A comparison of solver times in Table 4.5 shows that the MPC solver times for both nominal and adaptive controllers are of the same order of magnitude and well below the controller sampling time. A comparison of the training times for the adaptive controller with trajectory and derivative loss showed that the trajectory loss formulation had much longer training times. This is because the trajectory loss network simulates a trajectory multiple times to formulate the loss function. Various improvements, such as training on a GPU, were suggested. However, this thesis aims to provide a proof of concept rather than a ready-to-be-deployed architecture. How

adaptive model predictive control is deployed in practical automotive systems is left for future work.

The results for economic MPC proved more promising. Table 4.7 illustrates that, for matched models, the adaptive controller performs worse than the nominal, but under model mismatch it achieves significant improvements. This was due to the adaptive controller having significantly less constraint violations, i.e. remaining more within the target temperature region. If the target temperature region is chosen to correspond to the most efficient region for operating the battery, then this also means that the adaptive controller can lead to increased energy efficiency.

5.2 Evaluation of Adaptive Controller Method and Future Work

The adaptive controller was found to have good performance in the cascaded tank benchmark. However, the type of mismatch was simple. Only the case of mismatched parameters and a saturation on the inputs was included, with this mismatch being time-invariant. A strength of online adaptive control is its ability to account for time-varying parameters and this was not investigated. Another case that could have been tested is one where the mismatch varies along the prediction horizon. In contrast, in battery thermal systems, the dynamics will most likely behave differently at the end of the prediction horizon, which is 1000 s later than the current time. Including time-variant mismatch in the benchmark would have made the benchmark align better with the battery thermal management case. However, it still serves as a validation of the adaptive method.

For the battery thermal management (BTM) system the results were mixed. A possible explanation for the poor performance in BTM, is that thermal dynamics are slow dynamics in the states but fast in their derivatives, i.e. a stiff system. Stiff systems exhibit complicated dynamic characteristics, as discussed in [44]. Typically regions with steep gradients are resolved by using a larger number of samples, but for the network proposed in this work no special treatment for this problem was presented. This is a major complication that is worthy of future exploration. Additionally, there is a lag between when inputs are supplied and when they manifest, due to how heating and cooling gradually spread in the BTM. The adaptive controller instead tries to explain the residual in the dynamics at the current time step only with inputs applied in the previous time step, but by this time they might not have had their full effect. However, this is not something that can be definitely concluded from the results and would require further analysis.

A comparison of derivative and trajectory losses in reference tracking showed that the trajectory loss controller scheme performed consistently better. A complication resulting from this is that the network training times are roughly 30 times longer compared to the derivative loss control scheme. This poses a problem for the tra-

jectory loss scheme as this exceeds the controller's sampling time by a factor of 6, meaning that the model parameters would have a delay in being updated. The present work is not concerned with computational efficiency but this would be a topic worthy of research.

The hyperparameter tuning methodology presented in Section 3.4.4 is fairly manual and heuristic. Doing hyperparameter tuning manually is time consuming and there is a risk of missing the optimal hyperparameters. An automatic method that sweeps over hyperparameters and compares the results would be more suitable. In this way it would also be possible to explore more simulation scenarios. Currently, the adaptive controllers only perform better in certain scenarios. To properly evaluate their usefulness it would be necessary to build a library of simulation scenarios, that explore all reasonable sources of mismatch together with different drive cycles. In this thesis only a single drive cycle was used that models city driving. City driving is only one use case for electric vehicles, and to properly evaluate the performance other drive cycles should also be explored. It is noteworthy, however, that city driving is uniquely challenging, because of the sharp accelerations, and it is thus possible that if a drive cycle representing highway driving was used the results would be better.

A major improvement to the method would be to estimate the confidence parameter c, introduced in Section 3.4, from data. This confidence parameter determines how much the residual network impacts the nominal model. For cases of matched models this could be set to 0 and then nearly identical performance would be obtained between nominal and adaptive model predictive control. In cases of severe mismatch the confidence parameter could instead be set much higher. However, this touches upon a more major fundamental issue in the adaptation. True dynamics could not be queried from the model and were thus required to be estimated through filtering. Filtering however introduces artifacts into the dynamics. Consequently, determining whether the model is matched or mismatched is difficult.

The simulation performance for the trajectory loss controller, even using an aggressive neural network with high confidence, was found to be improved using the adaptive model, as seen in Figure 4.14. Similarly, the neural network in the derivative loss controller is only turned on if it passes a check on performing better on test data. Therefore, it would seem, that both neural networks should result in the adaptive control model being closer to the plant model. However, the closed-loop control performance was in many cases degraded. An interesting research direction would be, instead of trying to increase control performance by improving the control model, to instead focus on closed-loop performance. One approach would be to adaptively change the cost weighting used in the controller. As shown in Figure 4.13, having a relatively higher weighting on temperature led to improvements in reference tracking for a nominal controller.

In Figure 4.11 and Figure 4.12 it was shown that the adaptive controller applies heating even when the battery temperature is above the reference, and this in turn yields worse tracking performance. A way of remedying this would be to incorpo-

rate a physics-informed loss that penalizes the network if it learns dynamics that go against physics. This could be done by penalizing cases where the residual has a different sign than the heating power. However, this is not without issue. It is possible that the residual should be positive, even when the heating power is negative, e.g. when the cooling power is small and there is a large current applied. However, it still seems fruitful to explore ways in which this could be done. Another improvement would be to revise how the scaling was done. Inputs, states and disturbances were scaled in a way that they were all within the same order of magnitude. However, for neural networks, it is often good to have scaling between -1 and 1, or between 0 and 1. This also applies to the hyperbolic tangent activation functions that were used, which saturate for large input values.

6
Conclusion

In this thesis, an adaptive control framework was developed and applied to a cascaded tanks system benchmark and to a battery thermal management (BTM) system. A reference tracking model predictive controller (MPC) was tested on both systems, and additionally an economic model predictive control formulation was tested on the battery thermal management system. Nominal control models for both systems were developed based on physical laws, and the tracking and economic model predictive control formulations were tuned to obtain good baseline performance. The adaptive control framework was based on using neural networks to approximate the residual between the true plant dynamics and the nominal control model dynamics. Two different training losses were formulated, one using derivative loss which tries to minimize the error in the dynamics directly, and one using trajectory loss that tries to minimize error in the dynamics indirectly by comparing simulation performance. These different training losses in turn lead to two differently trained neural network models for the residual dynamics. The residual neural networks were trained on data collected while running a model predictive controller. Once trained the neural network residual became part of the control model dynamics, thus creating an adaptive controller. By updating the dynamics the adaptive controller started operating differently than a controller using nominal dynamics.

The adaptive MPC controller on the BTM system performed relatively poorly for reference tracking, particularly when trained with derivative loss, as indicated by RMSE and cost. Compared to the nominal MPC controller the adaptive scheme exhibited consistently worse performance for matched models. For matched model cases the nominal MPC controller exhibited 17 % to 60 % decrease in cost compared to the best adaptive framework, as well as a lower RMSE value. The trajectory loss adaptive controller achieved better results for some mismatched models, with a 25 % to 50 % reduction in cost with an RMSE over 1 °C lower than that of the nominal MPC controller. However, for another case of model mismatch, namely non-linear output saturation, the nominal scheme still performed better.

The adaptive controller for economic MPC showed a similar trend, though the cost for matched model was much closer to that of the nominal MPC controller than it was for reference tracking, with only a 0.2 % to 11 % increase in cost. For mismatched models the adaptive controller saw a reduction of cost of 23 % to 35 %, and decrease in cost was exhibited in all mismatch scenarios. These results demonstrate that the adaptive controller still needs further development for the economic MPC formulation but also indicate that the approach has potential in economic MPC

settings.

Using neural networks trained on trajectory loss to approximate residuals was found to improve simulation performance, suggesting that the residual neural networks help to close part of the gap between the nominal physics model and the true dynamics. In this way a more accurate control model is obtained. However, the adaptive controller did not generally improve control performance in the battery thermal management system. This may be due to the controller starting to operate in regions of the state-input space that were not well represented in the residual's training data. Additionally, adding a neural network residual to a control model changes how the model predictive controller decides on inputs. The nominal model in turn has a simple operating scheme, which yields control inputs with consistent and predictable effects on the plant. With adaptive controllers the operating scheme becomes more complicated and, as shown by the decreased performance, does not necessarily lead to better control. Alternative methods to improve control performance should thus be explored, not only improvements in the control model.

The adaptive control framework proposed here showed clear improvements on the cascaded tanks benchmark, but mixed results on the battery thermal management system. An important finding is therefore that model adaptation via neural network residuals is not automatically beneficial. The objective was to assess whether a learning-enhanced model predictive controller could improve control performance. The results show that this is indeed possible in specific scenarios, but that the approach is sensitive to loss design, hyperparameters, and the training data. The framework does improve closed-loop controller performance in some scenarios, and making it more generalizable warrants further investigation.

Bibliography

[1] Directorate-General for Mobility European Commission and Transport. *COMMUNICATION FROM THE COMMISSION TO THE EUROPEAN PARLIAMENT, THE COUNCIL, THE EUROPEAN ECONOMIC AND SOCIAL COMMITTEE AND THE COMMITTEE OF THE REGIONS Sustainable and Smart Mobility Strategy – putting European transport on track for the future*. COM/2020/789. Brussels, Belgium, Dec. 2020. [Online]. Accessible: https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX:52020DC0789 (visited on 11/21/2025).

[2] Virender Singh, Vedant Singh, and S. Vaibhav. "A review and simple meta-analysis of factors influencing adoption of electric vehicles". In: *Transportation Research Part D: Transport and Environment* vol. 86 (2020), p. 102436. ISSN: 1361-9209. DOI: https://doi.org/10.1016/j.trd.2020.102436. Accessible: https://www.sciencedirect.com/science/article/pii/S1361920920306234.

[3] Jiangong Zhu, Ranjun Huang, and Haifeng Dai. "Low-Temperature Performance of Lithium-Ion Batteries for Electric Vehicles". In: *Alternating Current (AC) Heating for Lithium-Ion Batteries in Electric Vehicles: Heating Principles, Modeling, and Implementation*. Singapore: Springer Nature Singapore, 2025, pp. 1–29. DOI: 10.1007/978-981-96-9071-8_1. [Online]. Accessible: https://doi.org/10.1007/978-981-96-9071-8_1.

[4] Giorgio Previati, Giampiero Mastinu, and Massimiliano Gobbi. "Thermal Management of Electrified Vehicles—A Review". In: *Energies* vol. 15.4 (2022). ISSN: 1996-1073. DOI: 10.3390/en15041326. Accessible: https://www.mdpi.com/1996-1073/15/4/1326.

[5] Qianhao Yue et al. "Review of Battery Thermal Management Techniques for Electric Vehicles in Cold Environments". In: *2024 6th International Academic Exchange Conference on Science and Technology Innovation (IAECST)*. 2024, pp. 1601–1605. DOI: 10.1109/IAECST64597.2024.11118071.

[6] James Blake Rawlings, David Q. Mayne, and Moritz Diehl. *Model predictive control: theory, computation and design*. eng. 2nd edition. Santa Barbara: Nob Hill Publishing, LLC, 2020. ISBN: 9780975937754.

[7] Mohammad Reza Amini et al. "Cabin and Battery Thermal Management of Connected and Automated HEVs for Improved Energy Efficiency Using Hierarchical Model Predictive Control". In: *IEEE Transactions on Control Systems Technology* vol. 28.5 (2020), pp. 1711–1726. DOI: 10.1109/TCST.2019.2923792.

[8] Jie Zhang et al. “Nonlinear Model Predictive Control for Coordinated Temperature and Energy Optimization in Battery Thermal Management System”. In: *2025 IEEE 26th China Conference on System Simulation Technology and its Applications (CCSSTA)*. 2025, pp. 1–8. DOI: 10.1109/IEEECONF65522.2025.11137334.

[9] Weichao Zhuang et al. “A thermal management method for lithium-ion battery based on fuzzy model predictive control”. In: *2019 Chinese Automation Congress (CAC)*. 2019, pp. 2706–2711. DOI: 10.1109/CAC48633.2019.8996265.

[10] Changfu Zou, Chris Manzie, and Dragan Nešić. “A Framework for Simplification of PDE-Based Lithium-Ion Battery Models”. In: *IEEE Transactions on Control Systems Technology* vol. 24.5 (2016), pp. 1594–1609. DOI: 10.1109/TCST.2015.2502899.

[11] Lennart Ljung. *System identification: theory for the user*. eng. 2. ed., 14. printing. Prentice-Hall information and system sciences series. Upper Saddle River, NJ: Prentice Hall, 2012. ISBN: 9780136566953.

[12] Gianluigi Pillonetto et al. “Deep networks for system identification: A survey”. In: *Automatica* vol. 171 (2025), p. 111907. ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2024.111907. Accessible: https://www.sciencedirect.com/science/article/pii/S0005109824004011.

[13] Ioan Doré Landau et al. *Adaptive Control: Algorithms, Analysis and Applications*. en. Communications and Control Engineering. London: Springer London, 2011. ISBN: 9780857296634. DOI: 10.1007/978-0-85729-664-1. [Online]. Accessible: https://link.springer.com/10.1007/978-0-85729-664-1 (visited on 05/27/2026).

[14] Lukas Hewing et al. “Learning-Based Model Predictive Control: Toward Safe Learning in Control”. en. In: *Annual Review of Control, Robotics, and Autonomous Systems* vol. 3.1 (May 2020), pp. 269–296. ISSN: 2573-5144, 2573-5144. DOI: 10.1146/annurev-control-090419-075625. Accessible: https://www.annualreviews.org/doi/10.1146/annurev-control-090419-075625 (visited on 11/14/2025).

[15] Yu Mei et al. “Fast Online Adaptive Neural MPC via Meta-Learning”. In: *IFAC-PapersOnLine* vol. 59.30 (2025), pp. 377–382. DOI: 10.1016/j.ifacol.2025.12.266.

[16] Tim Salzmann et al. “Real-time Neural-MPC: Deep Learning Model Predictive Control for Quadrotors and Agile Robotic Platforms”. In: *IEEE Robotics and Automation Letters* (2023). DOI: 10.1109/LRA.2023.3246839.

[17] Tim Salzmann et al. “Learning for CasADi: Data-driven Models in Numerical Optimization”. In: *Learning for Dynamics and Control Conference (L4DC)*. 2024.

[18] Saket Adhau, Sebastien Gros, and Sigurd Skogestad. “Reinforcement learning based MPC with neural dynamical models”. In: *European Journal of Control* vol. 80 (2024). 2024 European Control Conference Special Issue, p. 101048. DOI: https://doi.org/10.1016/j.ejcon.2024.101048. Accessible: https://www.sciencedirect.com/science/article/pii/S0947358024001080.

[19] Surya Venkatesh Pandiyan, Sebastien Gros, and Jayaprakash Rajasekharan. "Physics informed neural network based multi-zone electric water heater modeling for demand response". In: *Applied Energy* vol. 380 (2025), p. 125037. ISSN: 0306-2619. DOI: https://doi.org/10.1016/j.apenergy.2024.125037. Accessible: https://www.sciencedirect.com/science/article/pii/S0306261924024218.

[20] S. Gros and M. Zanon. "Data-Driven Economic NMPC Using Reinforcement Learning". In: *IEEE Transactions on Automatic Control* vol. 65.2 (Feb. 2020), pp. 636–648. DOI: 10.1109/TAC.2019.2913768.

[21] Andreas B. Martinsen, Anastasios M. Lekkas, and Sébastien Gros. "Reinforcement learning-based NMPC for tracking control of ASVs: Theory and experiments". In: *Control Engineering Practice* vol. 120 (2022), p. 105024. ISSN: 0967-0661. DOI: https://doi.org/10.1016/j.conengprac.2021.105024. Accessible: https://www.sciencedirect.com/science/article/pii/S0967066121002823.

[22] Jorge Espin, Yuichi Kajiura, and Dong Zhang. "Safety-Driven Battery Charging: A Fisher Information-guided Adaptive MPC with Real-time Parameter Identification". In: *IFAC-PapersOnLine* vol. 58.28 (2024), pp. 186–191. DOI: 10.1016/j.ifacol.2024.12.032.

[23] Maximilian Degner et al. "Adaptive Economic Model Predictive Control: Performance Guarantees for Nonlinear Systems". In: *IEEE Transactions on Automatic Control* (2026), pp. 1–16. DOI: 10.1109/TAC.2026.3655254.

[24] Pozzoli S, Gallieri M, and Scattolini R. "Tustin neural networks: a class of recurrent nets for adaptive MPC of mechanical systems". In: *IFAC-PapersOnLine* vol. 53.2 (2020), pp. 5171–5176. DOI: 10.1016/j.ifacol.2020.12.1183.

[25] Shahin Razani, Mahsan Tavakoli-Kakhki, and Ahmad Kalhor. "Robust data-driven feedback linearization using neural network based sparse identification of nonlinear dynamics". In: *ISA Transactions* (2026). ISSN: 0019-0578. DOI: https://doi.org/10.1016/j.isatra.2026.05.007. Accessible: https://www.sciencedirect.com/science/article/pii/S0019057826002454.

[26] Chaoqi Wei, Jiapeng Liu, and Dongxiao Liu. "Adaptive neural network temperature control for thermoelectric refrigeration systems using online self-learning mechanism". In: *International Journal of Refridgeration* vol. 183 (2026), pp. 379–390. DOI: 10.1016/j.ijrefrig.2026.01.015.

[27] Dylan Wald et al. "A neural-network-enhanced parameter-varying framework for multi-objective model predictive control applied to buildings". In: *Applied Energy* vol. 21.100566 (2025). DOI: https://doi.org/10.1016/j.egyai.2025.100566.

[28] Mathworks. *EV Battery Thermal Management System*. [Online]. Accessible: https://se.mathworks.com/help/hydro/ug/ev-battery-cooling.html (visited on 06/12/2026).

[29] Francisco J. Aragón et al. "Constrained Optimization". en. In: *Nonlinear Optimization*. Ed. by Francisco J. Aragón et al. Cham: Springer International Publishing, 2019, pp. 253–309. ISBN: 9783030111847. DOI: 10.1007/978-3-030-11184-7_6. [Online]. Accessible: https://doi.org/10.1007/978-3-030-11184-7_6 (visited on 05/11/2026).

[30] Gianluca Frison and Moritz Diehl. "HPIPM: a high-performance quadratic programming framework for model predictive control". In: (2020). arXiv: 2003.02547 [math.OC]. Accessible: https://arxiv.org/abs/2003.02547.

[31] Jonathan Frey, Armin Nurkanović, and Moritz Diehl. "Advanced-Step Real-Time Iterations With Four Levels – New Error Bounds and Fast Implementation in acados". In: *IEEE Control Systems Letters* vol. 8 (2024), pp. 1703–1708. DOI: 10.1109/LCSYS.2024.3412007.

[32] Ilya Loshchilov and Frank Hutter. "Decoupled Weight Decay Regularization". In: *International Conference on Learning Representations (ICLR)* (2019).

[33] Enrico Terzi et al. "Robust predictive control with data-based multi-step prediction models". In: *2018 European Control Conference (ECC)*. 2018, pp. 1710–1715. DOI: 10.23919/ECC.2018.8550537.

[34] Marco Forgione and Dario Piga. "Model Structures and Fitting Criteria for System Identification with Neural Networks". In: *2020 IEEE 14th International Conference on Application of Information and Communication Technologies (AICT)*. 2020, pp. 1–6. DOI: 10.1109/AICT50176.2020.9368834.

[35] Kang Tian et al. "Physics-Informed Neural Network Model for Trajectory Tracking Control of Unmanned Surface Vehicles". In: *IEEE Internet of Things Journal* vol. 13.8 (2026), pp. 16924–16938. DOI: 10.1109/JIOT.2026.3660870.

[36] Yuki Tsuchiya et al. "Online Adaptation of Learned Vehicle Dynamics Model with Meta-Learning Approach". In: *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2024, pp. 802–809. DOI: 10.1109/IROS58592.2024.10801427.

[37] OAR US EPA. *Dynamometer Drive Schedules*. en. Data and Tools. Sept. 2015. [Online]. Accessible: https://www.epa.gov/vehicle-and-fuel-emissions-testing/dynamometer-drive-schedules (visited on 05/15/2026).

[38] Frank P. Incropera et al., eds. *Fundamentals of heat and mass transfer*. eng. 6. ed. Hoboken, NJ: Wiley, 2007. ISBN: 9780471457282.

[39] Daniela Lupu and Ion Necoara. "Exact representation and efficient approximations of linear model predictive control laws via HardTanh type deep neural networks". In: *Systems & Control Letters* vol. 186 (2024), p. 105742. ISSN: 0167-6911. DOI: https://doi.org/10.1016/j.sysconle.2024.105742. Accessible: https://www.sciencedirect.com/science/article/pii/S0167691124000306.

[40] Sophocles J. Orfanidis. "What Is a Savitzky-Golay Filter? [Lecture Notes]". In: *IEEE Signal Processing Magazine* vol. 28.4 (2011), pp. 111–117. DOI: 10.1109/MSP.2011.941097.

[41] Wallace Gian Yion Tan and Zhe Wu. "Robust machine learning modeling for predictive control using Lipschitz-Constrained Neural Networks". In: *Computers & Chemical Engineering* vol. 180 (2024), p. 108466. ISSN: 0098-1354. DOI: https://doi.org/10.1016/j.compchemeng.2023.108466. Accessible: https://www.sciencedirect.com/science/article/pii/S0098135423003368.

[42] Tim Salzmann. *Tim-Salzmann/l4casadi*. original-date: 2023-06-19T14:55:52Z. May 2026. [Online]. Accessible: https://github.com/Tim-Salzmann/l4casadi (visited on 05/21/2026).

[43] M. Schoukens and J.P. Noël. "Three Benchmarks Addressing Open Challenges in Nonlinear System Identification". en. In: *IFAC-PapersOnLine* vol. 50.1 (July 2017), pp. 446–451. ISSN: 24058963. DOI: 10.1016/j.ifacol.2017.08.071. Accessible: https://linkinghub.elsevier.com/retrieve/pii/S2405896317300915 (visited on 05/21/2026).

[44] Eunsuh Kim et al. "Stabilize physics-informed neural networks for stiff differential equations: Re-spacing layer". In: *Computers & Mathematics with Applications* vol. 200 (2025), pp. 167–179. ISSN: 0898-1221. DOI: https://doi.org/10.1016/j.camwa.2025.09.014. Accessible: https://www.sciencedirect.com/science/article/pii/S0898122125003955.

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

CHALMERS
UNIVERSITY OF TECHNOLOGY