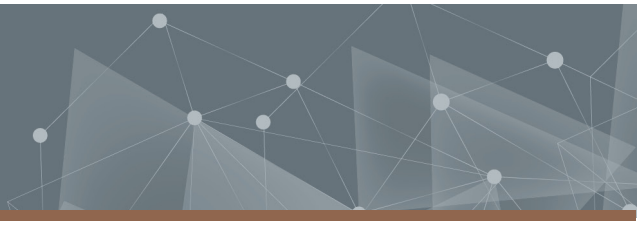
CHALMERS
UNIVERSITY OF TECHNOLOGY



Embedded Nonlinear Optimization Solver for Vehicle Energy Management

Master's thesis in Systems, Control and Mechatronics

DHRUVKUMAR PATEL
ANANTHAKRISHNAN KAIMAL

DEPARTMENT OF ELECTRICAL ENGINEERING

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

Master's thesis 2026

Embedded Nonlinear Optimization Solver for Vehicle Energy Management

DHRUVKUMAR PATEL
ANANTHAKRISHNAN KAIMAL

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2026

An Informative Headline describing the Content of the Report
DHRUVKUMAR PATEL
ANANTHAKRISHNAN KAIMAL

Supervisor:
Karthik Prasad, Geely Technology Europe
Niklas Legnedahl, Geely Technology Europe
Lorenzo Montalto, Chalmers University of Technology

Examiner:
Nikolce Murgovski, Department of Electrical Engineering

Master's Thesis 2026
Department of Electrical Engineering
Division of 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

Embedded Nonlinear Optimization Solver for Vehicle Energy Management
DHRUVKUMAR PATEL
ANANTHAKRISHNAN KAIMAL
Department of Electrical Engineering
Chalmers University of Technology

Abstract

This thesis investigates real-time embedded solution methodologies for intelligent charge- and trip-planning in electric vehicles, focusing on three core subproblems: optimal driving and charging, eco-driving, and battery thermal tracking. Formulated as constrained nonlinear optimal control problems, these tasks present significant computational challenges for hardware deployment. To balance numerical performance with modeling flexibility, this work explores and contrasts two distinct paradigms: a highly specialized approach based on Pontryagin's Maximum Principle and a versatile, solver-based Sequential Quadratic Programming framework using `acados`. Both methods are deployed and validated on a dSPACE MicroAutoBox II platform. The results demonstrate their real-time feasibility and outline the trade-offs between analytical customization and general optimization frameworks for embedded electric mobility applications.

Keywords: Electric vehicles, Intelligent charge- and trip-planning, Eco-driving, Battery Thermal Management, Nonlinear optimal control, Embedded systems, Pontryagin's Maximum Principle, acados, dSPACE MicroAutoBox II.

Acknowledgements

First and foremost, we extend our sincere thanks to our supervisors, Lorenzo Montalto, Karthik Prasad, and Niklas Legnedahl. Their invaluable guidance in the fields of optimization and electromobility, along with their constant availability to help us navigate complex challenges, was fundamental to the success of this project. We are also immensely grateful to Prashant Lokur for his extraordinary support. His assistance in providing problem formulations, his technical guidance in optimization, and his encouragement to explore diverse research directions truly helped shape the final trajectory of this work. Our sincere thanks go to our examiner, Nikolce Murgovski, for his expert advice in optimization and for providing excellent guidance throughout the evaluation of this thesis. Finally, we would like to thank everyone at Geely Technology Europe for providing a remarkably supportive workplace and the essential resources needed to conduct this research. We especially want to thank our fellow thesis students; the time spent together and the countless fruitful conversations we shared made this journey a truly enjoyable and memorable experience.

Dhruvkumar Patel & Ananthakrishnan Kaimal, Gothenburg, June 2026

List of Acronyms

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

EV	Electric Vehicle
BEV	Battery Electric Vehicle
OCP	Optimal Control Problem
PMP	Pontryagin's Maximum Principle
SQP	Sequential Quadratic Programming
QP	Quadratic Programming
MABXII	Microautobox II
MPC	Model Predictive Control
NMPC	Nonlinear Model Predictive Control
HVCH	High-Voltage Coolant Heater
HP	Heat Pump
HVAC	Heating, Ventilation, and Air Conditioning
SoC	State of Charge
RK4	Fourth-Order Runge-Kutta Method
BVP	Boundary Value Problem
ECU	Electronic Control Unit
RTI	Real-Time Iteration
EDU	Electric Drive Unit
RHC	Receding Horizon Control
CAN	Controller Area Network

Nomenclature

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

Indices

i, j	Generic indices
k	Discrete-time / prediction-horizon index
n	Discretization node / shooting interval index

Sets

$\mathcal{X}$	Admissible state set
$\mathcal{U}$	Admissible input set
$\mathcal{X}_f$	Terminal admissible state set

Parameters

N	Prediction horizon length / number of discretization intervals
n_x	Number of states
n_u	Number of control inputs
Δt	Time discretization step
Δs	Spatial discretization step
$\Delta \tau$	Normalized discretization step
t_f	Final time
$T_{\max}$	Maximum admissible final time
Q	State weighting matrix in MPC cost
R	Input weighting matrix in MPC cost

Q_e	Terminal state weighting matrix
V_f	Terminal cost matrix
c_b	Battery specific heat capacity
m_b	Battery mass
C_b	Battery capacity
U_{oc}	Battery open-circuit voltage
R_b	Battery internal resistance
$P_{chg}^{\max}$	Maximum charger power
P_{aux}	Auxiliary electrical power
w_t	Time-cost weight in charging objective
w_e	Energy-cost weight in charging objective
$\tilde{w}_t$	Modified time-cost weight
η_{HVCH}	Efficiency of high-voltage coolant heater
η_{QHVCH}	Thermal efficiency / conversion factor used in charging model
α_{HVCH}	Quadratic regularization coefficient for HVCH power
α_{HP}	Quadratic regularization coefficient for heat-pump power
$Q_{HP,0}, Q_{HP,1}$	Heat-pump model coefficients
$\sigma_{HP}(T_b)$	Temperature-dependent heat-pump coefficient
m	Vehicle mass
m_i	Equivalent rotational inertia mass
R_r	Wheel rolling radius
g	Gravitational acceleration
ρ	Air density
C_d	Aerodynamic drag coefficient
A	Vehicle frontal area
C_r	Rolling resistance coefficient
$a_1, \ldots, a_5$	EDU loss-model coefficients
$b_1, \ldots, b_6$	Maximum torque polynomial coefficients
$c_1, \ldots, c_6$	Minimum torque polynomial coefficients
λ_1	Weight on travel time in eco-driving objective
λ_2	Weight on energy consumption in eco-driving objective
v_{nom}	Nominal speed used for normalization
P_{nom}	Nominal power used for normalization
α_l	Lower speed-tolerance factor

α_u	Upper speed-tolerance factor
Z_L	Quadratic penalty weight for lower velocity slack
Z_U	Quadratic penalty weight for upper velocity slack
T_{env}	Ambient temperature
m_{bat}	Battery mass in thermal tracking model
c_{bat}	Battery specific heat capacity in thermal tracking model
c_{cool}	Coolant specific heat capacity
ρ_{cool}	Coolant density
V_{pump}	Pump displacement
R_{bat}	Battery resistance in thermal tracking model
C_{bat}	Battery capacity in thermal tracking model
hA_{bat}	Effective battery heat-transfer coefficient times area
$\alpha_0, \alpha_1, \alpha_2, \alpha_3$	Identified thermal model coefficients
γ	Ambient thermal coupling coefficient
s_ω	Pump-speed normalization factor
s_Q	Heating-power normalization factor
$I_{b,N}$	Nominal battery current used in steady-state calculation

Variables

t	Continuous time
τ	Normalized time
$x(t)$	State vector
$u(t)$	Control input vector
J	Objective / cost function
$S(x, u)$	Stage cost
$\Psi(x(t_f))$	Terminal cost
$H(x, u, \lambda)$	Hamiltonian
λ	Costate vector
w	Stacked optimization variable vector
d	SQP search direction
s	Slack variable vector
T_b	Battery temperature
SoC	Battery state of charge

τ	Driving time state
s	Vehicle position / traveled distance
E_k	Vehicle kinetic energy
P_b	Battery power
P_{HVCH}	High-voltage coolant heater power
P_{HP}	Heat-pump power
P_{HVAC}	HVAC power
P_{grid}	Grid power during charging
P_{HVCH}^{cab}	HVCH power used for cabin heating
P_{HVCH}^{b}	HVCH power used for battery heating
P_c	Cabin heating demand
$P_{b,\min}$	Vehicle-side minimum admissible battery power
$P_{b,\min}^{chg}$	Charger-side minimum admissible battery power
Q_{joule}	Joule heat generated in the battery
Q_{HVCH}	Heat supplied by the HVCH
Q_{HP}	Heat removed by the heat pump
Q_{amb}	Heat exchanged with the ambient environment
λ_{T_b}	Costate associated with battery temperature
λ_{SoC}	Costate associated with state of charge
v	Vehicle longitudinal speed
T	Electric drive unit torque
$\theta(s)$	Road gradient profile
$P_{loss}(v,T)$	Electric drive unit power loss
$T_{\max}(v)$	Maximum traction torque
$T_{\min}(v)$	Minimum / regenerative braking torque
$v_{\min}(s)$	Lower speed bound / speed-limit profile
$v_{\max}(s)$	Upper speed bound / speed-limit profile
ε_L	Lower slack variable for speed constraint
ε_U	Upper slack variable for speed constraint
T_{bat}	Battery temperature in thermal tracking model
SOC	State of charge in thermal tracking model
ω	Coolant pump speed
Q_{heat}	Heating / cooling actuator power
I_{bat}	Battery current

$\dot{m}_{cool}$	Coolant mass flow rate
Q_{cool}	Heat removed by coolant loop
T_{cool}^{in}	Coolant inlet temperature
T_{cool}^{out}	Coolant outlet temperature
NTU_{bat}	Number of transfer units for battery heat exchange
x_{ss}	Steady-state state vector
u_{ss}	Steady-state input vector
A	Linearized state Jacobian matrix
B	Linearized input Jacobian matrix

Contents

List of Figures

List of Tables

1
Introduction

To combat climate change and meet global greenhouse gas reduction targets, making the transport sector more sustainable through the promotion of electric vehicles (EVs) has become a primary objective [1, 2, 3]. However, widespread EV adoption remains significantly hindered by range anxiety, sparse charging infrastructure, and time-consuming recharging processes [4, 5, 6]. While increasing battery size provides a hardware-side solution to mitigate range anxiety, it introduces severe penalties regarding vehicle weight and production costs [7]. Alternatively, vehicle energy efficiency can be enhanced via advanced software strategies, such as battery thermal management to regulate operating temperatures and eco-driving to optimize driving profiles [8, 9]. Ultimately, integrating battery thermal management and eco-driving with the optimization of the charging process and station selection yields a comprehensive framework known as intelligent charge- and trip-planning. By maximizing operational efficiency and resolving range anxiety without hardware drawbacks, optimal trip-planning represents a critical paradigm for the viability of electric mobility, forming the central focus of this thesis.

Within this broader context, this thesis considers three problem classes related to optimal trip planning for electric vehicles. The first is the optimal driving and charging problem, where the goal is optimizing the energy consumption and the battery temperature of a vehicle driving to a charging station, and then subsequently optimizing the charging process itself. The second is eco-driving, where the vehicle speed is optimized to reduce energy consumption over a given route. The third is battery thermal tracking, where the battery is controlled so that its temperature remains in a favorable operating region in order to optimize battery performance, and prolong health and range. Although these problems differ in their distinct physical interpretations and localized dynamics, they all fall under the unifying umbrella of charge- and trip-planning for EVs. Conceptually, a comprehensive EV trip-planning framework represents an integrated system encompassing battery thermal management, charging infrastructure optimization, powertrain energy management, and velocity profile optimization i.e, eco-driving. In practice, rather than treating these domains in total isolation, individual sub-problems can be systematically viewed as isolating and controlling a targeted subset of these interdependent components. Crucially, despite variations in the specific subset of physical phenomena being addressed, all of these configurations share a fundamental mathematical feature: they can be formally transcribed and solved as constrained optimal control problems.

These problem classes are generally nonlinear due to the underlying vehicle, battery, charging, and thermal dynamics, as well as the structure of the associated objectives and constraints. For example, the relation between vehicle speed and energy con-

sumption is nonlinear, charging behavior is typically state dependent, and thermal behavior introduces additional nonlinear state evolution and coupling effects. In addition, practical limitations such as state and input bounds, terminal requirements, and path constraints must be enforced. As a result, the considered trip-planning tasks lead to nonlinear optimal control problems (OCP) which, after discretization, become nonlinear programming problems (NLP). For embedded applications, this creates a central challenge: the solution method must not only produce satisfactory control or planning decisions, but must do so with sufficiently low computational effort to be feasible for hardware deployment.

The main purpose of this thesis is therefore to investigate two different solution methodologies for such optimal trip planning problems. The first is based on Pontryagin's Maximum Principle (PMP), which provides a highly tailored and problem-specific approach. When a PMP-based formulation can be derived successfully, it offers insight into the structure of the optimal solution and may lead to computationally efficient implementations. However, this efficiency is typically achieved by exploiting the analytical structure of a specific problem, which makes the approach less flexible when the formulation changes. The second method is based on sequential quadratic programming (SQP) through the `acados` framework, which is used in this thesis as a more general nonlinear optimization framework for embedded optimal control. In contrast to PMP, `acados` is not tailored to one specific problem structure, but can accommodate a broader class of nonlinear optimal control problems after suitable discretization and transcription into nonlinear programming form.

This contrast between a specialized method and a more general embedded optimization framework forms the central methodological theme of the thesis, explored across different dimensions of the trip-planning problem. On the one hand, PMP represents a structured and highly problem-dependent route to fast embedded solutions, which is applied here specifically to the optimal charging and driving problem. On the other hand, `acados` represents a versatile, solver-based approach capable of handling a wider range of nonlinear optimal control formulations with minimal changes to the underlying numerical framework; this is leveraged for the eco-driving and battery thermal tracking problems. Rather than benchmarking both methods on a single task, evaluating these two distinct paradigms across different facets of the trip-planning ecosystem makes it possible to assess their respective trade-offs in flexibility, implementation effort, and computational performance in an embedded environment.

A further focus of the thesis is the feasibility of deploying these methods on embedded hardware. In particular, the developed solution approaches are evaluated with respect to implementation on the dSPACE MicroAutoBox II platform. The objective is therefore not only to formulate and solve the considered optimal trip-planning problems, but also to determine whether the chosen methodologies are sufficiently efficient and reliable for embedded execution. In this sense, the thesis is centered on the investigation of PMP-based and SQP-based optimal control methods for hardware deployment in the context of electric vehicle trip planning.

Accordingly, the thesis studies PMP for the optimal charging and driving problem, and `acados` for eco-driving and battery thermal tracking, with the overall aim

of evaluating their feasibility and performance on embedded hardware. The work therefore contributes to the broader question of how specialized and general nonlinear optimization methods can be used in practice for optimal trip planning in battery electric vehicles.

1.1 Literature Review

This section is structured around the three distinct problems investigated in this thesis: optimal charging, eco-driving, and battery thermal tracking. Although these problems focus on different physical scales and time horizons, they all fall under the unifying umbrella of charge- and trip-planning for EVs, where each specific problem considers a targeted subset of these highly interdependent components. Each subproblem is discussed sequentially below, establishing how optimization-based methods have been popularized to analyze and solve these problems [10, 11]. While the methods in literature do not necessarily prioritize real-time execution, they lay the groundwork for the subsequent discussion on advanced embedded optimization techniques, which are specifically designed to guarantee computational feasibility for real-time deployment across various applications and which subsequently can be applied to problems we are solving.
Optimal trip-planning for electric vehicles is commonly addressed using optimization-based methods, in which charging stops, charging decisions, and trip-level objectives are formulated explicitly and the problem is solved subject to vehicle and infrastructure constraints [12, 13, 14]. In parallel, data-driven methods such as deep reinforcement learning have also been explored for electric-vehicle trip planning, providing an alternative that can learn decision policies from data or simulation. These approaches are used as a way to avoid the disadvantages of model-based techniques, successfully bypassing the requirement of an explicit system model, which is sometimes very hard to obtain or borderline impossible. However, these data-driven alternatives suffer from the usual drawbacks of black-box methods, including limited interpretability and weaker guarantees on constraint satisfaction and stability [15, 16]. To address these limitations, model-based approaches offer a strong alternative that is not a black-box, allowing the control framework to take advantage of the physical knowledge we have on the specific problem. Within this category, Dynamic programming is often considered because it can handle discrete charging decisions and may provide globally optimal solutions for discretized formulations, but its computational complexity grows rapidly with problem size and discretization resolution, which limits scalability and embedded feasibility [12]. To avoid this computational burden, tailored optimal-control approaches based on Pontryagin's Maximum Principle have also been proposed, offering strong structural insight and very low online computational effort once the problem-specific solution has been derived, although this comes at the cost of reduced flexibility when the problem formulation changes [14]. Conversely, to retain flexibility while remaining suitable for embedded implementation, more general optimization-based formulations allow for richer physical effects and objectives, such as charging duration, battery degradation, vehicle mass variation, and wind influence, to be incorporated. While this increased modeling flexibility typically leads to more nonlinear and computation-

ally demanding problems [13], these direct formulations remain highly effective at balancing modeling richness with online tractability.

Eco-driving can be viewed as an important subproblem of overall electric-vehicle trip planning, since the selected velocity profile directly affects energy consumption, travel time, and consequently the charging decisions required over the trip. Similar to charge-planning, the eco-driving problem is commonly formulated as an optimal control or optimization problem, and the literature again shows the use of both global and approximate numerical methods. Dynamic programming has been used to compute globally optimal or near-optimal speed trajectories, especially when road slope and route-dependent effects are considered explicitly, although its computational burden remains a limiting factor for high-dimensional or finely discretized problems [17]. Sequential quadratic programming has also been proposed for eco-driving, showing that the problem can be treated within a structured nonlinear optimization framework and, under suitable conditions, solved to global optimality [18]. In addition, model predictive control has been employed to realize eco-driving strategies online, particularly for more detailed powertrain configurations, where repeated finite-horizon optimization offers improved flexibility but also increases the importance of efficient numerical solution [19].

Battery thermal tracking can likewise be viewed as an important component of electric-vehicle trip planning, since battery temperature directly affects performance, efficiency, health, and usable range. In the recent literature, this problem is predominantly addressed using model predictive control formulations. Nonlinear MPC has been used to jointly regulate battery temperature and thermal-management energy consumption within a coordinated optimization framework [8], while hierarchical MPC schemes have been proposed to manage battery and cabin thermal loads over multiple prediction horizons using traffic and speed preview information [20]. Adaptive MPC has also been investigated in order to account for changing battery thermal dynamics and improve temperature regulation under varying operating conditions [21]. Taken together, these works show that battery thermal management is increasingly treated as a predictive optimization problem, with MPC providing a flexible framework for handling coupled thermal and energy objectives.

Although these studies demonstrate the effectiveness of optimization-based methods for trip planning problems in electric vehicles, they primarily focus on solution quality and energy-saving potential, while the question of whether such methods can be deployed efficiently on embedded automotive hardware remains largely unaddressed. By contrast, a growing body of work has demonstrated that nonlinear optimization-based control can be deployed successfully on embedded hardware, although typically in application domains other than trip planning. Real-time NMPC with real-time iteration which is a specialized Newton-type algorithm designed to deliver fast feedback control by executing only one optimization iteration per time step [22], has been implemented for large multirotor aircraft and validated through hardware-in-the-loop tests on embedded computing platforms, showing that structure-exploiting formulations can satisfy stringent real-time constraints even for systems with many actuators [23]. In the automotive domain, embedded NMPC has also been validated for autonomous driving tasks such as trajectory tracking, collision avoidance, and lane change, with emphasis on SQP-based solvers, code gen-

eration, and hardware-in-the-loop development workflows [24]. Similarly, real-time MPC-based energy management has been demonstrated on an automotive-grade microcontroller for dual-motor battery electric vehicles, showing that optimization-based control can approach the performance of offline dynamic programming while remaining feasible for in-vehicle execution [25]. These works indicate that embedded nonlinear optimization is already sufficiently mature for real-time deployment in several control applications, but its use for optimal trip-planning problems in electric vehicles remains comparatively less explored. In this respect, the present thesis aims to bridge this gap by investigating whether methodologies that have proven effective for embedded implementation in other domains can be adapted to trip-planning-related problems such as charging, eco-driving, and battery thermal tracking.

1.2 Contributions

The primary contribution of this thesis lies in bridging the critical gap between theoretical optimal control design and the practical, real-time implementation of optimization-based solutions within the domain of electric vehicle trip planning. While advanced optimization algorithms are frequently validated in idealized simulation environments, their transition to physical targets often reveals significant computational and structural bottlenecks. This work addresses this disconnect by mapping out the complete deployment pipeline, from continuous-time abstractions to real-time execution. Thereby providing a reference for how optimization based energy and thermal management strategies can be reliably implemented on production-grade automotive hardware.

Methodologically, this thesis contributes with a comprehensive assessment of the fundamental trade-offs between highly specialized, problem-dependent control methodologies and versatile, general-purpose optimization frameworks. By analyzing PMP alongside SQP realized via the `acados` framework [26], this research highlights the engineering compromises inherent in solver selection. The insights gained delineate how a dedicated, analytical approach like PMP offers exceptional computational speed at the expense of modeling flexibility, whereas a direct, solver-based approach provides an adaptable framework that can accommodate evolving nonlinear system descriptions with minimal architectural redesign.

Finally, this work provides an investigation into the mathematical and structural modifications required to adapt continuous-time optimal control problems for resource-constrained, deterministic hardware platforms. Achieving embedded feasibility demands more than mere code compilation; it requires a deliberate reformulation of the underlying optimization problems. This thesis evaluates critical transcription techniques, such as time-normalization and robust discretization schemes, to ensure numerical stability and strict adherence to hardware-imposed timing constraints. Consequently, these architectural adjustments establish a clear blueprint for balancing mathematical fidelity with real-time computational tractability on embedded microcontrollers.

1.3 Limitations

While this thesis provides valuable insights into the embedded feasibility of specialized and general nonlinear optimization methods, several inherent limitations regarding problem scope, hardware compatibility, and the execution environment must be acknowledged. First, the empirical evaluation of the optimization methodologies is constrained to three specific automotive use cases: optimal charging, eco-driving, and battery thermal tracking. A defining characteristic of these applications is their relatively low temporal sensitivity. Because thermal transitions and macroscopic energy-planning profiles evolve over longer horizons, these problems can tolerate comparatively large sampling intervals. Consequently, the feasibility and timing results demonstrated herein may not automatically generalize to highly time-critical vehicle control systems—such as active chassis control or power electronics—which require sub-millisecond execution loops.
Furthermore, the physical deployment and validation of the optimization routines are bound to the dSPACE MicroAutoBox II rapid prototyping platform. Due to laboratory infrastructure constraints, the system interface is strictly limited to a legacy software environment, specifically MATLAB/Simulink 2017. This constraint introduces significant software-level restrictions, as any developed optimal control formulation must be completely translatable into Simulink-compatible structures, such as S-functions, before it can be cross-compiled into executable C-code. This legacy toolchain ultimately restricts the utilization of certain modern language features or contemporary optimization framework APIs such as L4CasADi [27] or L4acados [28] that are only supported in newer software releases.
Finally, the optimization problems evaluated on the dSPACE MicroAutoBox II are primarily executed without interfacing with an external physical environment or an independent, high-fidelity continuous-time plant model. Real-time hardware deployment demands strictly discrete-time computational representations. Because it is architecturally unfeasible to directly execute a continuous-time physical plant on the real-time target without explicit temporal discretization, the hardware validation focuses heavily on evaluating the solver's standalone profile-generation capabilities. It should be noted that for the battery thermal tracking application, a pseudo-plant behavior is introduced via a "white-box" approach, wherein the discrete nominal control equations are augmented with additive white noise to simulate basic process disturbances and model mismatch. However, this implementation remains highly idealized and lacks the complex non-idealities of an actual physical system, such as transport delays, unmodeled high-order continuous dynamics, or sensor latencies. Consequently, the evaluation of comprehensive closed-loop feedback mechanisms under authentic, continuous-time plant-model mismatches remains outside the scope of this thesis.

2

Mathematical Modeling

This chapter establishes the core mathematical frameworks required to translate the physical behaviors of EVs into actionable optimization problems. It begins by defining the continuous-time dynamical system models that capture the coupled electrical and thermal phenomena of the battery pack. To enable systematic optimization, these physics-based representations are embedded into a formal OCP framework. Because these continuous formulations are generally intractable analytically, the chapter details the numerical integration and transcription techniques used to discretize the trajectory. Ultimately, this process yields a structured, finite-dimensional discrete-time OCP that is computationally suitable for real-time numerical solvers.

2.1 Dynamical System

The system under consideration is modeled as a dynamical system, where the evolution of the states is described by a set of differential equations. In general form, the system dynamics can be written as

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

where $x(t) \in \mathbb{R}^{n_x}$ is the state vector, $u(t) \in \mathbb{R}^{n_u}$ is the control input vector, and $f(\cdot)$ represents the system dynamics.
The states and control inputs are subject to physical and operational constraints. These constraints define the feasible sets

$$x(t) \in \mathcal{X} \subseteq \mathbb{R}^{n_x}, \tag{2.2}$$
$$u(t) \in \mathcal{U} \subseteq \mathbb{R}^{n_u}, \tag{2.3}$$

where $\mathcal{X}$ and $\mathcal{U}$ denote the admissible state and input spaces, respectively.
For the electric vehicle energy management problem, a general state vector may include battery thermal, electrical, and vehicle motion states, expressed as

$$x(t) = \begin{bmatrix} T_b(t) & \mathrm{SoC}(t) & \tau(t) & s(t) & E_k(t) \end{bmatrix}^\top, \tag{2.4}$$

where T_b is the battery temperature, SoC is the battery state of charge, τ is the driving time, s is the vehicle position, and E_k is the kinetic energy.
The corresponding control input vector can be written as

$$u(t) = \begin{bmatrix} P_b(t) & P_{\mathrm{HVCH}}(t) & P_{\mathrm{HP}}(t) & P_{\mathrm{HVAC}}(t) & a(t) \end{bmatrix}^\top, \tag{2.5}$$

where P_b is the battery power, P_{HVCH} is the high-voltage coolant heater power, P_{HP} is the heat pump power, P_{HVAC} is the heating, ventilation, and air conditioning power, and a is the vehicle acceleration.
This formulation provides a general representation of the electric vehicle system. Depending on the specific optimization problem, only a subset of these states and inputs may be used.

2.2 Thermal Modeling

Battery temperature strongly affects electric vehicle charging performance, discharging performance, efficiency, and battery lifetime. Therefore, it must be maintained within a suitable operating range. Active regulation of the battery temperature before or during operation is commonly referred to as battery preconditioning.
The battery temperature is represented by the thermal state $T_b(t)$, whose dynamics are written in compact form as

$$\dot{T}_b(t) = f_{T_b}(x(t), u(t)), \tag{2.6}$$

where $f_{T_b}(\cdot)$ denotes the battery thermal model.
The temperature evolution depends on active thermal management inputs and passive heat flows. The active inputs include the high-voltage coolant heater power P_{HVCH}, heat pump power P_{HP}, and HVAC power P_{HVAC}. In addition, the battery is affected by Joule losses, heat generated by the electric driveline, and heat exchange with the ambient environment. These effects are collected in the function f_{T_b} for the optimal control formulation.

2.3 Electrical Modeling

The battery electrical behavior is represented using an equivalent circuit model with the state of charge, $\mathrm{SoC}(t)$, as the main electrical state. The state of charge describes the available energy level in the battery. Its dynamics are written in compact form as

$$\dot{\mathrm{SoC}}(t) = f_{\mathrm{SoC}}(x(t), u(t)), \tag{2.7}$$

where $f_{\mathrm{SoC}}(\cdot)$ denotes the battery electrical model.
The electrical model includes the battery internal resistance R_b and the open-circuit voltage U_{oc}. In general, R_b depends on the battery temperature T_b, while U_{oc} depends on the state of charge. These dependencies influence the battery current, losses, and available power.
A battery power balance constraint is also imposed to ensure that the requested power does not exceed the available battery power. The battery power is distributed among Joule losses, active thermal management loads, auxiliary loads, cabin heating, and electric driveline power.

2.4 Optimal Control Problem Formulation

The control objective is framed as an Optimal Control Problem (OCP), where the goal is to determine a control trajectory $u(t)$ and a state trajectory $x(t)$ that minimize a predefined performance index while satisfying system dynamics and operational constraints. In a general continuous-time setting, the cost function J is typically expressed in form:

$$J = \Psi(x(t_f)) + \int_0^{t_f} S(x(t), u(t))\, dt \tag{2.8}$$

where $S(x(t), u(t))$ represents the *stage cost* (the instantaneous cost rate), and $\Psi(x(t_f))$ denotes the *terminal cost* associated with the state at the final time t_f.

The general continuous-time OCP is defined as:

$$\begin{aligned}
\min_{x,u,b,t_f} \quad & J = \Psi(x(t_f)) + \int_0^{t_f} S(x(t), u(t))\, dt && \text{(2.9a)}\\
\text{s.t.:} \quad & \dot{x}(t) = f(x(t), u(t)) && \text{(2.9b)}\\
& x(t) \in \mathcal{X}, \quad u(t) \in \mathcal{U}, \quad t \in [0, t_f] && \text{(2.9c)}\\
& t_f \in [0, T_{\max}] && \text{(2.9d)}\\
& b \in \{0,1\}^{N_b} && \text{(2.9e)}\\
& h_b(x, u, b) \leq 0 && \text{(2.9f)}\\
& x(0) = \bar{x}_0, \quad x(t_f) \in \mathcal{X}_f && \text{(2.9g)}
\end{aligned}$$

where $f(x, u)$ represents the system dynamics, $\mathcal{X}$ and $\mathcal{U}$ define the feasible states and control inputs, b is a vector of binary decision variables of length N_b, and h_b constitutes the vector of inequality constraints, $\bar{x}_0$ is some known initial condition, and $\mathcal{X}_f$ is a known target set for the states.

As a continuous-time problem, (2.9) is infinite-dimensional, which might make it hard to solve on a digital computer. A discretization step is therefore necessary to transform the continuous-time OCP into a finite-dimensional NLP that can be efficiently handled by numerical optimization algorithms.

2.5 Discretization of an OCP

Different methods can be used to discretize an OCP. This section provides description of the methods used and the resulting discrete OCP formulation.

2.5.1 The Fourth-Order Runge-Kutta Method

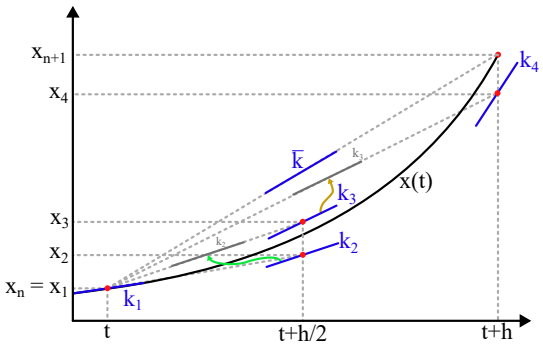


Figure 2.1: Illustration of the 4th order Runge-Kutta method.

To ensure numerical accuracy while maintaining computational efficiency, the fourth-order Runge-Kutta (RK4) method is utilized. The method is illustrated in Figure 2.1 which has been referenced from [29], where x is the function that needs to be discretized. By sampling the system derivatives at four locations within each interval h, the method achieves a higher order of accuracy than the standard Euler integration but this comes at a higher cost in computation. Assuming a Zero-Order Hold (ZOH) on the control inputs u_n over the discretization interval $[t, t+h]$, the discrete state transition is given by:

$$x_{n+1} = x_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{2.10}$$

where the intermediate slopes are defined as:

$$k_1 = f(x_n, u_n) \tag{2.11a}$$

$$k_2 = f\left(x_n + \frac{h}{2}k_1, u_n\right) \tag{2.11b}$$

$$k_3 = f\left(x_n + \frac{h}{2}k_2, u_n\right) \tag{2.11c}$$

$$k_4 = f(x_n + hk_3, u_n) \tag{2.11d}$$

2.5.2 Direct Multiple Shooting

While the RK4 method provides a numerical approximation of the system evolution over a single discretization interval, an additional transcription step is required in order to convert the continuous-time optimal control problem into a finite-dimensional optimization problem. In this thesis, this transcription is performed using direct multiple shooting, which is a standard approach in numerical optimal control [30]. The method is particularly well suited for embedded optimization, since it preserves the stage-wise structure of the optimal control problem and allows constraints to be imposed directly at the discretization nodes.

In direct multiple shooting, the state values at the grid points,

$$x_0, x_1, \ldots, x_N,$$

are introduced as optimization variables together with the control inputs

$$u_0, u_1, \ldots, u_{N-1},$$

and, in the present case, the final time t_f. In each interval, the control input is assumed to be constant (ZOH), and the continuous-time dynamics are integrated independently using the RK4 method described above. This yields a discrete transition map from one node to the next, which may be written as

$$x_{n+1} = f_d(x_n, u_n, t_f), \qquad n = 0, \ldots, N-1, \tag{2.12}$$

where f_d denotes the discretized dynamics obtained from one RK4 integration step over the normalized interval of fixed length $\Delta\tau = 1/N$ within the dimensionless time domain $\tau \in [0, 1]$ established by the time-normalization transformation. By operating on this normalized interval, the numerical integration step size remains invariant throughout the optimization process, while the true physical duration of each shooting stage is dynamically scaled via the t_f factor.
The essential idea of direct multiple shooting is that the state trajectory is not obtained by one single forward simulation over the entire horizon. Instead, each shooting interval is propagated separately, and continuity between neighboring intervals is enforced through equality constraints. The corresponding multiple-shooting constraints are therefore given by

$$x_{n+1} - f_d(x_n, u_n, t_f) = 0, \qquad n = 0, \ldots, N-1. \tag{2.13}$$

In contrast to single shooting, where only the control inputs are treated as optimization variables, direct multiple shooting includes the intermediate state variables explicitly in the decision vector. This has two important advantages. First, state and path constraints can be imposed directly at the discretization nodes. Second, the resulting optimization problem has a sparse and structured form, since each continuity constraint couples only two neighboring shooting nodes. This structure is highly beneficial for numerical methods intended for real-time and embedded implementation. Furthermore, direct multiple shooting offers superior numerical precision and stability compared to single shooting methodologies. While a single shooting approach integrates the entire state trajectory sequentially all at once and evaluates the boundary constraints only at the terminal node, a limitation shared by indirect solvers like the PMP framework, it is highly susceptible to severe error accumulation, where small integration errors stack up significantly over long horizons. In contrast, multiple shooting mitigates this numerical drift by partitioning the trajectory into localized intervals. This allows the optimizer to precisely enforce corrections along the path at intermediate shooting nodes, yielding a much higher degree of final accuracy for highly sensitive or nonlinear system dynamics.
With this transcription, the discretized optimal control problem can be expressed as a finite-dimensional problem in the variables x_n, u_n, and t_f.

2.5.3 Discrete-Time OCP Formulation

In scenarios where the final time t_f is a decision variable, the continuous optimal control problem must be reformulated to operate on a fixed discretization grid. We introduce the normalized time variable τ:

$$\tau = \frac{t}{t_f} \implies dt = t_f \, d\tau \tag{2.14}$$

From a practical implementation standpoint, this transformation is necessary because the continuous-time stage cost is evaluated via an integral bounded from 0 to t_f. Having an active optimization decision variable serve directly as the upper bound of an integral makes it impossible for numerical solvers to evaluate the integral expression. By normalizing the time domain, the varying final time t_f is extracted from the integration limits and becomes a standard algebraic multiplier instead, which is readily handled by numerical optimization frameworks. Applying the chain rule to the system dynamics yields the scaled differential equation $\frac{dx}{d\tau} = f(x,u) \cdot t_f$. The integral component of the cost function is similarly transformed into:

$$\int_0^{t_f} S(x,u)\, dt = t_f \int_0^1 S(x,u)\, d\tau \tag{2.15}$$

This normalization allows the optimal control problem to be transcribed into a finite-dimensional, discrete-time format suitable for numerical solvers. Because embedded optimization methods cannot operate directly on a continuous-time abstraction, casting the problem as a structured mathematical program is of central importance for the remainder of this thesis; it allows approximate local solutions to be computed efficiently on real-time hardware. Depending on the specific application, certain trip-planning tasks require discrete switching logic alongside continuous dynamics, whereas others rely purely on smooth continuous variables. To provide a unified, general framework, the comprehensive discrete-time optimization problem is formulated as a Mixed-Integer Nonlinear Program:

$$\begin{aligned}
\min_{x,u,b,t_f} \quad & J = \Psi(x_N) + t_f \sum_{n=0}^{N-1} S(x_n, u_n) && \text{(2.16a)}\\
\text{s.t.:} \quad & x_{n+1} = t_f \cdot f_d(x_n, u_n), \quad n = 0, \dots, N-1 && \text{(2.16b)}\\
& x_n \in \mathcal{X}, \quad u_n \in \mathcal{U}, \quad n = 0, \dots, N-1 && \text{(2.16c)}\\
& h(x_n, u_n) \le 0, \quad n = 0, \dots, N-1 && \text{(2.16d)}\\
& t_f \in [0, T_{\max}] && \text{(2.16e)}\\
& b_k \in \{0,1\}^{N_b}, \quad k \in \mathcal{B} && \text{(2.16f)}\\
& h_b(x_n, u_n, b_k) \le 0 && \text{(2.16g)}\\
& x_0 = \bar{x}_0, \quad x_N \in \mathcal{X}_f && \text{(2.16h)}
\end{aligned}$$

In this comprehensive representation, the objective function (2.16a) consists of the terminal cost $\Psi(x_N)$ and the accumulated discretized stage cost $S(x_n, u_n)$ over the horizon. The equality constraints in (2.16b) enforce the discretized system dynamics, where f_d denotes the transition map obtained via independent numerical integration

steps (such as the RK4 method) over the normalized interval. Vector boundaries on the state and control paths are enforced in (2.16c), general non-integer nonlinear path constraints are restricted in (2.16d), and the admissible interval for the free final time decision variable is bounded in (2.16e). To accommodate combinatorial decisions—such as gear selection or discrete charging state transitions—the formulation introduces binary decision variables b_k in (2.16f) which restrict corresponding mixed-integer path constraints in (2.16g). Finally, initial state constraints and terminal target sets are governed by (2.16h).
The structural configuration of (2.16) provides a broad mathematical formulation, yet this thesis focuses exclusively on continuous NLP formulations where binary decision variables b_k and their corresponding mixed-integer path constraints (2.16g) are omitted. While incorporating discrete combinatorial logic yields a highly generalized framework, solving Mixed-Integer NLPs introduces severe combinatorial complexity that scales poorly under real-time conditions. Within the context of resource-constrained hardware platforms like the dSPACE MicroAutoBox II, the non-deterministic execution times and heavy computational overhead associated with mixed-integer optimization strategies are fundamentally unfeasible for embedded deployment. Consequently, to guarantee strict execution predictability and meet real-time constraint requirements, all optimization problems addressed in this research are restricted to discrete NLP formulations.

3
Optimization Tools

This chapter outlines the different solution methodologies and algorithmic frameworks used to solve the formulated optimization problems. It details both direct and indirect methods, evaluating their respective strengths in managing the nonlinearities and constraints inherent to electric vehicle control. By reviewing these numerical approaches, the chapter provides the necessary context for the selection of the specific algorithms and solvers deployed in this work to achieve a reliable and computationally efficient solution.

3.1 Model Predictive Control

Model predictive control (MPC) is an optimization-based control strategy in which the control input is obtained by repeatedly solving a finite-horizon optimal control problem online [31]. Rather than relying on an explicit, offline control law design, MPC defines a dynamic state-feedback policy through real-time computation. At each sampling instant, the current state of the system is measured or estimated, a prediction model is used to forecast the future system evolution over a finite horizon, and an optimal input sequence is computed by minimizing a cost function subject to the system dynamics and constraints. Only the first element of the optimal input sequence is applied to the plant, after which the horizon is shifted forward and the optimization is repeated at the next sampling instant. This receding-horizon principle allows MPC to explicitly account for state and input constraints while continuously updating the control action based on new state information.
In this thesis, the primary focus is on Nonlinear Model Predictive Control (NMPC), which utilizes the discrete NLP structure previously transcribed in (2.16). By defining the current measured plant state as the initial condition boundary parameter $\bar{x}_0$ in (2.16h), the NMPC framework directly embeds the system's discrete-time dynamics (2.16b), physical boundary constraints (2.16c), general path inequalities (2.16d), and final time bounds (2.16e). The finite prediction horizon is dictated by the number of discretization nodes N. The receding-horizon control law is subsequently realized by solving the optimization problem (2.16) at each discrete sampling interval, extracting the resulting optimal solution sequence:

$$\mathbf{u}^\star = \{u_0^\star, u_1^\star, \ldots, u_{N-1}^\star\} \tag{3.1}$$

and applying only the first computed optimal control element to the physical actuators:

$$u_{\mathrm{MPC}}(\bar{x}_0) = u_0^\star \tag{3.2}$$

Once this first input is executed, the state is updated by the plant, the prediction horizon is shifted forward, and the entire numerical problem is reinstituted at the next sampling node. Because the underlying system models, state constraints, and path constraints involved in electric vehicle trip planning are non-convex and nonlinear, this repeated execution generates a sequence of complex NLPs that must be resolved online. Consequently, the practical viability of NMPC hinges on deploying embedded optimization methods that are fast and computationally efficient enough to reliably solve the formulation in (2.16) within the strict real-time sampling-time constraints of the targeted embedded hardware.

3.2 Boundary Value Problems

A boundary value problem (BVP) consists of a set of differential equations subject to boundary conditions specified at more than one point of the independent variable. In the context of optimal control, BVPs commonly arise after applying necessary optimality conditions, where both the system states and costates must be determined.
A general BVP can be written as

$$\dot{\xi}(t) = f_{\xi}(\xi(t), u(t)), \tag{3.3}$$
$$\xi(t_0) = \xi_0, \quad \xi(t_f) = \xi_f, \tag{3.4}$$

where $\xi(t)$ is the extended state vector. In optimal control problems, this vector typically contains both the state vector $x(t)$ and the costate vector $\lambda(t)$, defined as

$$\xi(t) = \begin{bmatrix} x(t) & \lambda(t) \end{bmatrix}^{\top}. \tag{3.5}$$

The objective is to determine the trajectory $\xi(t)$ over the interval $[t_0, t_f]$ such that the differential equations and the boundary conditions are satisfied simultaneously. In the context of this thesis, the application of Pontryagin's Maximum Principle converts the optimal control problem into a boundary value problem. Therefore, the optimization tools discussed in the following sections can be interpreted as numerical methods for solving, or approximating the solution of, the resulting BVP.

3.3 Interior Point Method

The interior point method is a numerical optimization approach commonly used for solving constrained optimization problems. In the context of optimal control, it can be applied after discretizing the continuous-time problem into a finite-dimensional nonlinear programming problem.
The main idea is to handle inequality constraints by adding barrier terms to the objective function. These barrier terms penalize solutions that approach the boundary of the feasible region. As a result, the optimizer searches for a solution from inside the feasible set, rather than moving along its boundary.

A constrained optimization problem of the form

$$\min_z \quad J(z) \tag{3.6}$$

$$\text{s.t.} \quad g(z) \leq 0 \tag{3.7}$$

can be reformulated using a barrier function as

$$\min_z \quad J(z) - \mu \sum_i \log(-g_i(z)), \tag{3.8}$$

where z is the vector of decision variables, $g_i(z)$ are the inequality constraints, and $\mu > 0$ is the barrier parameter. The logarithmic barrier tends to infinity as any constraint approaches its boundary, thereby keeping the iterates strictly inside the feasible region.
The optimization is solved through a sequence of barrier problems. At the beginning, a larger value of μ keeps the solution away from the constraint boundaries. As μ is gradually reduced, the solution is allowed to move closer to the boundary and converge toward the optimum of the original constrained problem. The resulting sequence of solutions is often referred to as the central path.
Interior point methods are powerful and widely used in nonlinear optimization solvers such as IPOPT. However, for embedded implementation, they may introduce challenges due to variable iteration counts, matrix factorizations, and memory requirements. These characteristics can make the worst-case execution time difficult to predict, which motivates the investigation of more structured approaches for real-time automotive control.

3.4 Pontryagin's Maximum Principle

Pontryagin's Maximum Principle (PMP) provides necessary conditions for optimality in continuous-time optimal control problems. It transforms the original problem of finding an optimal control trajectory into a set of state equations, costate equations, boundary conditions, and a pointwise optimization condition.
For an OCP with state dynamics

$$\dot{x}(t) = f(x(t), u(t)), \tag{3.9}$$

and cost functional

$$J = \Psi(x(t_f)) + \int_0^{t_f} S(x(t), u(t))\, dt, \tag{3.10}$$

the Hamiltonian is defined as

$$H(x, u, \lambda) = S(x, u) + \lambda^\top f(x, u), \tag{3.11}$$

where $\lambda(t)$ is the costate vector associated with the state vector $x(t)$.
The costate dynamics are given by

$$\dot{\lambda}(t) = -\frac{\partial H}{\partial x}. \tag{3.12}$$

the costate gives you an estimate of the dynamics of the future cost for a change in the current state .
For a minimization problem, the optimal control minimizes the Hamiltonian pointwise:

$$u^* = \arg\min_u H(x^*, u, \lambda^*). \tag{3.13}$$

Thus, rather than optimizing directly over all admissible control trajectories, PMP determines the optimal control through a pointwise minimization of the Hamiltonian along the state and costate trajectories. This structure is particularly useful for the charging problem, since it allows the control law to be evaluated systematically at each time instant.
The terminal conditions depend on the problem formulation. For a free final state with terminal cost ($L_f(\mathrm{x}(t_f))$), the terminal costate satisfies

$$\lambda(t_f) = \frac{\partial L_f(x(t_f))}{\partial x}. \tag{3.14}$$

If the final state is free and no terminal cost is included, the condition reduces to

$$\lambda(t_f) = 0. \tag{3.15}$$

When the final time is also free, an additional transversality condition must be imposed:

$$H(t_f) + \frac{\partial L_f(x(t_f), t_f)}{\partial t_f} = 0. \tag{3.16}$$

If the terminal cost has no explicit dependence on (t_f), this condition simplifies to

$$H(t_f) = 0. \tag{3.17}$$

In the considered charging problem, the final state of charge is prescribed, whereas the final battery temperature is free. Since no terminal cost is associated with the battery temperature, the corresponding transversality condition is

$$\lambda_{T_\mathrm{b}}(t_f) = 0. \tag{3.18}$$

Furthermore, because the charging duration is optimized, the final time is free and the corresponding Hamiltonian transversality condition is imposed at (t_f).

3.5 Sequential Quadratic Programming

The NLP in (2.16) is finite-dimensional, but in general nonlinear in both the objective function and the constraints. A widely used class of methods for solving such problems is sequential quadratic programming, which is regarded as one of the standard approaches for constrained nonlinear optimization [32]. The main idea of SQP is to solve the NLP iteratively by forming, at each iteration, a quadratic programming (QP) approximation of the original problem. In this way, the nonlinear optimization problem is replaced by a sequence of QP subproblems, each of which

is constructed from a local approximation of the objective and a linearization of the constraints.
For the derivation of the SQP method, it is convenient to collect the decision variables of (2.16), namely the state trajectory $x := \{x_0, \dots, x_N\}$, the input trajectory $u := \{u_0, \dots, u_{N-1}\}$, and the final time t_f, into a single vector

$$w = \begin{bmatrix} x_0^\top & u_0^\top & x_1^\top & u_1^\top & \cdots & x_{N-1}^\top & u_{N-1}^\top & x_N^\top & t_f \end{bmatrix}^\top. \tag{3.19}$$

Using this compact notation, the NLP in (2.16) can be written as

$$\begin{aligned} \min_{w} \quad & J(w) && \text{(3.20a)} \\ \text{s.t.} \quad & g(w) = 0 && \text{(3.20b)} \\ & h(w) \leq 0, && \text{(3.20c)} \end{aligned}$$

where $g(w)$ collects the equality constraints, in particular the discretized dynamics and boundary conditions, and $h(w)$ collects the inequality constraints, including state, input, path, and final-time constraints. This compact form is introduced here only for notational convenience in the SQP derivation, while the underlying variables remain those defined in the discrete-time OCP formulation.
The SQP method is based on the Lagrangian function associated with (3.20),

$$\mathcal{L}(w, \lambda, \mu) = J(w) + \lambda^\top g(w) + \mu^\top h(w), \tag{3.21}$$

where λ and μ denote the multipliers associated with the equality and inequality constraints, respectively. At iteration k, the nonlinear problem is approximated by a QP in the step variable d, obtained by taking a quadratic approximation of the Lagrangian and a first-order linearization of the constraints around the current iterate w_k [33]. The resulting QP subproblem is

$$\begin{aligned} \min_{d} \quad & \frac{1}{2} d^\top B_k d + \nabla J(w_k)^\top d && \text{(3.22a)} \\ \text{s.t.} \quad & g(w_k) + \nabla g(w_k) d = 0 && \text{(3.22b)} \\ & h(w_k) + \nabla h(w_k) d \leq 0, && \text{(3.22c)} \end{aligned}$$

where B_k is an approximation of the Hessian of the Lagrangian with respect to the decision variables,

$$B_k \approx \nabla^2_{ww} \mathcal{L}(w_k, \lambda_k, \mu_k). \tag{3.23}$$

After solving (3.22), the decision variables are updated according to:

$$w_{k+1} = w_k + \alpha_k d_k^*, \tag{3.24}$$

where d_k^* is the solution of the QP subproblem and $\alpha_k \in (0, 1]$ is a scalar step size parameter used to control the update magnitude. The process is repeated until the algorithm converges to a local minimum.
For discretized OCPs such as (2.16), the QP subproblem in (3.22) inherits the structure induced by the dynamic constraints. In particular, the equality constraints obtained from the system dynamics couple only neighboring stages of the horizon.

This stage-wise sparsity is highly relevant for embedded optimization, because it allows specialized QP solvers to exploit the underlying problem structure instead of treating the problem as a dense, unstructured NLP [31]. For this reason, SQP is particularly well suited to nonlinear optimal control problems that must be solved efficiently under limited computational resources.

3.5.1 ACADOS Framework

A software framework that is directly relevant in this context is `acados`, which is an open-source framework for fast embedded optimization of structured nonlinear optimal control problems [26]. In the setting of this thesis, `acados` is of interest because it provides a practical implementation of SQP-based methods for discretized nonlinear optimization problems of the type introduced in (2.16). More specifically, `acados` is designed for continuous-valued optimal control problems and estimation problems arising from dynamical systems, and therefore fits well with the present objective of investigating nonlinear optimization methods for embedded implementation. At the same time, it should be stated precisely that `acados` is not intended as a general solver for arbitrary nonlinear programs or mixed-integer formulations, but rather as a high-performance framework for structured problems in optimal control.
In `acados`, the user specifies the system model, the cost function, and the constraints of the optimal control problem. After transcription to a finite-dimensional NLP, typically by direct multiple shooting, the resulting problem is solved by SQP-based methods such as full SQP or the real-time iteration variant SQP-RTI [31]. At each nonlinear iteration, the dynamics and constraints are linearized and the objective is locally approximated, which yields a structured QP of the form (3.22). This QP can then be solved either in its structured optimal-control form or after condensing, depending on the selected solver configuration and problem dimensions.
A central component in this workflow is `HPIPM` a high-performance solver, used within `acados`, for the QP subproblems that arise from the SQP iterations [34]. The low-level linear algebra operations required in these computations are accelerated by `BLASFEO`, a library specifically developed for dense linear algebra in embedded optimization and tailored to the matrix sizes that commonly occur in optimal control applications [35]. The combination of SQP-based nonlinear optimization, `HPIPM` for QP solution, and `BLASFEO` for efficient linear algebra is one of the main reasons why `acados` is well suited to real-time and embedded use cases.
From the perspective of this thesis, an additional advantage of `acados` is that it does not only provide numerical algorithms, but also supports a deployment-oriented workflow. For a given optimal control problem, `acados` can generate tailored C code for the resulting solver, which is valuable when the aim is to move from algorithm design to repeatable simulation and embedded implementation. Through its interfaces, this code generation workflow can also be integrated with MATLAB/Simulink, including the generation of S-functions that are convenient when constructing closed-loop simulation setups. This makes `acados` particularly suitable for evaluating nonlinear optimization methods in a framework that connects mathematical formulation, numerical solution, and implementation aspects within a single toolchain.

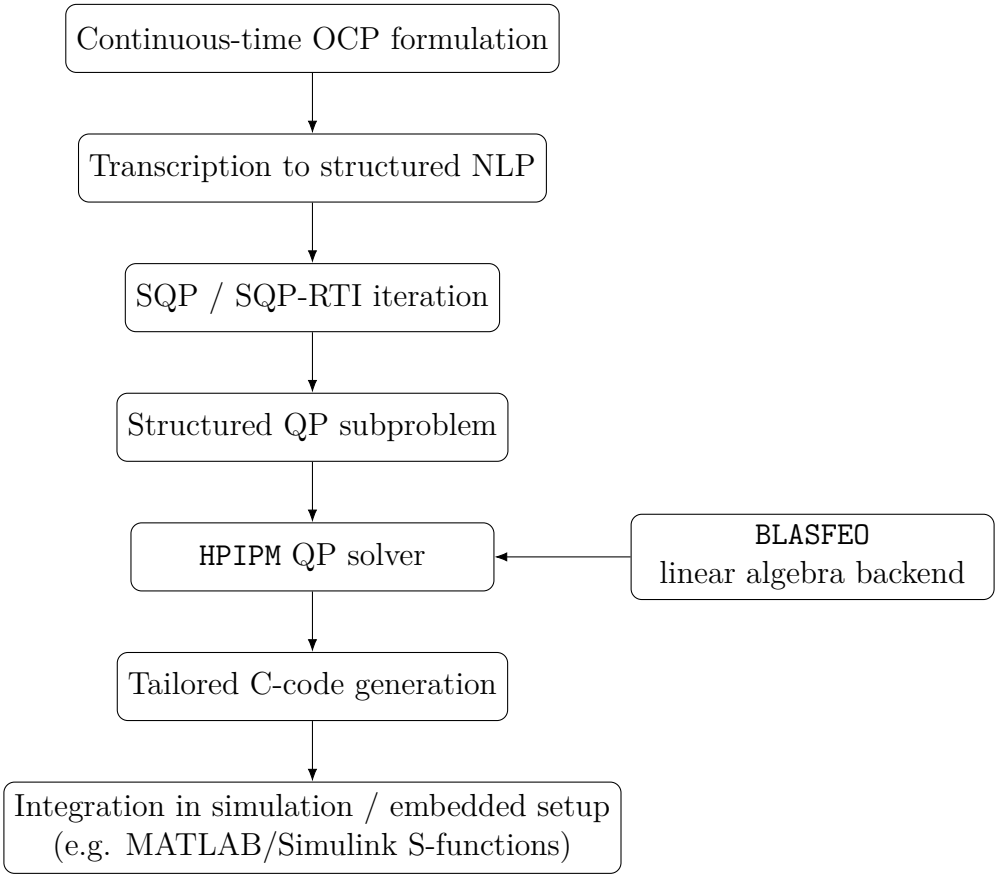


Figure 3.1: Schematic workflow of `acados` for structured nonlinear optimal control problems.

The workflow of `acados` illustrated in Fig. 3.1 can therefore be summarized as follows. Starting from the discretized optimal control problem, the model equations, cost terms, and constraints are transcribed into a structured NLP. An SQP-based method is then applied to this NLP, leading to a sequence of structured QP subproblems. These QP problems are solved efficiently using `HPIPM`, with the associated linear algebra handled by `BLASFEO`. Finally, the resulting solver can be exported as C code and embedded into the target simulation or implementation environment.

3.5.2 Solver Options in ACADOS

The numerical performance of `acados` depends not only on the problem formulation, but also on the choice of solver configuration. Since the objective of this thesis is to investigate nonlinear optimization methods for embedded implementation, the selection of solver options is of central importance. In particular, the trade-off between numerical robustness and computational complexity is strongly influenced by the nonlinear programming solver type, the QP solution strategy, the Hessian approximation, the regularization method, and the numerical integrator used for the system dynamics. The `acados` framework provides several options in each of these categories, allowing the solver to be adapted to the structure and timing requirements of the application [26].

Table 3.1: Main solver configurations considered in the present `acados`-based workflow.

Category	Option	Description
NLP solver	`SQP`	Multiple SQP iterations per sampling instant; higher NLP accuracy, higher cost.
NLP solver	`SQP_RTI`	One SQP step per sampling instant; lower cost, suited to real-time MPC.
QP solver	`FULL_HPIPM`	Condenses the OCP-QP to a dense QP; effective for smaller problems and short horizons.
QP solver	`PARTIAL_HPIPM`	Retains part of the stage structure; often a good compromise between runtime and memory.
Hessian Approx.	`EXACT`	Uses second-order derivatives of the Lagrangian; most accurate, but most expensive.
Hessian Approx.	`GAUSS_NEWTON`	Efficient approximation for nonlinear least-squares costs.
Hessian Approx.	`BFGS`	Quasi-Newton approximation used for general nonlinear costs when least-squares structure is absent.
Regularization	`PROJECT`	Projects small or negative eigenvalues; improves conditioning of the QP.
Regularization	`CONVEXIFY`	Modifies curvature information to obtain a more convex QP approximation.
Integrator	`ERK`	Explicit Runge–Kutta integration; efficient for non-stiff explicit dynamics.
Integrator	`IRK`	Implicit Runge–Kutta integration; more expensive, but useful for stiff systems.

Table 3.1 summarizes the main solver choices that are relevant to the present work. The most important distinction at the NLP level is between `SQP` and `SQP_RTI`. At the QP level, `HPIPM` can be used with either full or partial condensing. Further design choices concern the approximation of second-order information, the treatment of ill-conditioning through regularization, and the discretization method used inside the transcription.

The choice between `SQP` and `SQP_RTI` depends upon your execution strategy and requirements. Full `SQP` aims to solve the nonlinear program more completely at each control step by iterating until a convergence criterion is met. This may improve solution quality when the problem changes significantly from one sampling instant to the next, but it also increases the online computational burden. In contrast, `SQP_RTI` computes only one SQP step per sampling instant and relies on the receding-horizon structure, warm-starting, and the fact that successive NMPC problems are often similar [31]. For embedded applications with strict timing constraints, `SQP_RTI` is therefore frequently preferred.

At the QP level, the distinction between full and partial condensing is equally important. Full condensing eliminates the dynamic equality constraints and transforms the structured OCP-QP into a dense QP. This can be efficient for small-scale problems or short prediction horizons, but the dense condensed problem may become

costly as the horizon length or state dimension increases. Partial condensing reduces the problem dimension while preserving part of the original multiple-shooting structure, which is often advantageous in embedded applications. In combination with `HPIPM`, partial condensing is widely used because it offers a good balance between memory footprint and execution time for structured optimal control problems [34].

The treatment of second-order information also has a strong influence on solver behavior. When `EXACT` Hessians are used, `acados` incorporates second derivatives of the Lagrangian, including contributions from the nonlinear dynamics and constraints. This can improve local curvature information and may reduce the number of SQP iterations required for convergence. However, the cost of evaluating and factorizing these second-order terms is higher, and the resulting QP can become numerically more delicate when the Hessian is indefinite. By contrast, the `GAUSS_NEWTON` approximation avoids many of these costs by exploiting the structure of nonlinear least-squares type objectives. For many practical NMPC problems, this leads to a favorable trade-off between speed and robustness. In the broader SQP literature, quasi-Newton updates such as BFGS serve a similar purpose of reducing the cost of second-order information, but in the `acados` OCP setting the standard comparison is typically between `EXACT` and `GAUSS_NEWTON`.

Regularization is often required when the QP subproblems become ill-conditioned, for example due to strong nonlinearities, rapidly changing active sets, or near-singular curvature information. In such situations, the `PROJECT` option is commonly used to prevent numerical breakdown by modifying the Hessian approximation so that the resulting QP remains sufficiently well conditioned. This is especially relevant in embedded settings, where solver failure due to numerical instability may be unacceptable even if it occurs only occasionally. More aggressive regularization strategies such as `CONVEXIFY` may further improve robustness when exact second-order information leads to nonconvex QP approximations, although this can increase the computational load.

The numerical integrator used inside the transcription also affects the final solver behavior. For the class of problems considered in this thesis, explicit Runge–Kutta integration is attractive because it provides a good compromise between integration accuracy and computational cost. In particular, a classical RK4-type explicit scheme can capture the nonlinear system behavior sufficiently accurately while keeping the per-step complexity moderate. Implicit integrators remain relevant for stiff systems, but they are typically more expensive and are therefore less attractive when the primary emphasis is on embedded real-time execution.

A concise comparison of the main trade-offs is given in Table 3.2. The table emphasizes that the most computationally efficient settings are generally not those that provide the most complete second-order information, but rather those that exploit the problem structure and accept a controlled approximation in exchange for lower online effort.

Table 3.2: Qualitative comparison of selected `acados` solver options for embedded implementation.

Option	Cost	Effect	Remarks
`SQP`	High	Higher NLP accuracy	Multiple SQP iterations per sample.
`SQP_RTI`	Low	Lower per-step accuracy	One SQP step with warm-starting.
`FULL_HPIPM`	Med.	Neutral on optimum	Dense condensed QP.
`PARTIAL_HPIPM`	Low–Med.	Neutral on optimum	Retains part of the OCP structure.
`EXACT`	High	Best local curvature	Highest derivative cost.
`GAUSS_NEWTON`	Low	Good for least-squares costs	Efficient approximate curvature.
`BFGS`	Low–Med.	Good for general nonlinear costs	Quasi-Newton update with reduced second-order cost.
`PROJECT` regularization	Low–Med.	Small perturbation	Improves QP conditioning.
`ERK` integration	Low	Step-size dependent	Efficient for non-stiff dynamics.

For the problems considered in this thesis, the solver configuration is generally kept close to a low-complexity baseline in order to reduce online computation. In practice, this means favoring `SQP_RTI` when strict real-time execution is required, using `PARTIAL_CONDENSING_HPIPM` as the default QP solution strategy, and employing `PROJECT` regularization to avoid numerical failures in ill-conditioned situations. For the curvature approximation, `GAUSS_NEWTON` is preferred when the cost has nonlinear least-squares structure, whereas `BFGS` is used for more general nonlinear cost functions in order to avoid the cost of exact second derivatives. The system dynamics are discretized with an explicit Runge–Kutta integrator, which provides a suitable balance between numerical accuracy and computational effort. Overall, these settings are retained whenever possible because they reduce online computational load while maintaining adequate numerical performance for embedded implementation.

4

Optimal Charging

This chapter investigates the embedded implementation of nonlinear optimal charging for electric vehicles. The charging problem formulation and the PMP-based solution methodology presented here build on the framework introduced in [14]. In the present work, the emphasis is on studying how the method proposed in [14] can be realized on embedded automotive hardware. To that end, the chapter first presents the charging problem and its PMP-based solution structure, followed by an implementation-oriented evaluation of the same problem using both a specialized PMP solver and a direct SQP-based solver implemented through `acados`.

4.1 Challenges in Hardware Implementation

Implementing an optimal control algorithm on embedded hardware requires additional considerations beyond numerical performance. In a firmware implementation, the algorithm should exhibit deterministic execution behavior, limited memory usage, and predictable computational complexity. In particular, large variations in computation time between control cycles can complicate task scheduling and make it difficult to estimate the worst-case execution time.

These requirements create challenges for general-purpose nonlinear optimization methods. Many conventional solvers rely on a variable number of iterations, repeated derivative evaluations, matrix factorizations, and memory-intensive intermediate computations. Depending on the problem formulation and software implementation, this may lead to non-deterministic execution times and a memory footprint that is difficult to control. As a result, a direct deployment of generic nonlinear programming solvers on automotive embedded hardware can be difficult, especially when strict real-time constraints must be satisfied.

An additional challenge is robustness against solver failure, for example when no feasible or sufficiently accurate solution is found within the available computation time. In such cases, the control architecture must include fallback strategies to ensure safe and stable operation. These fallback strategies may include simplified rule-based controllers, linear feedback controllers, or reference-holding mechanisms that can be executed reliably on the target platform.

In this thesis, the charging problem is used as a case study for comparing two complementary embedded solution paradigms. The first is a specialized indirect method based on PMP, which exploits the analytical structure of the problem to obtain a compact and deterministic solver architecture. The second is a direct solver-based approach using SQP through the `acados` framework. Unlike generic large-scale

SQP implementations, `acados` is specifically designed for structured optimal control problems and embedded deployment, with tailored multiple-shooting formulations, specialized QP solvers, and code-generation support. The charging problem therefore provides a useful setting for comparing analytical structure exploitation against structured numerical optimization under realistic embedded implementation constraints.

4.1.1 Hardware Trends in the Automotive Industry

Automotive ECUs are increasingly required to execute complex control, estimation, and diagnostic algorithms under strict real-time constraints [36, 37]. Compared with desktop or server-grade systems, production ECUs provide limited memory and computational resources and must satisfy predictable execution requirements. These constraints directly affect the design of optimization-based control algorithms for embedded implementation.
Table 4.1 summarizes representative production-oriented automotive microcontroller families. Table 4.2 lists representative rapid-prototyping and test platforms. The reported specifications illustrate the order of magnitude of the available computational resources and do not define a universal hardware standard.

Table 4.1: Representative production-oriented automotive microcontroller families.

Platform	Representative specifications
Infineon AURIX TC3x	Up to 6 TriCore cores at 300 MHz, up to 16 MB flash memory, and more than 6 MB RAM.
NXP S32K3	Arm Cortex-M7 processor at up to 320 MHz, up to 12 MB flash memory, and up to 2304 kB SRAM.

Table 4.2: Representative automotive rapid-prototyping and test platforms.

Platform	Primary use
dSPACE MicroAutoBox III	In-vehicle rapid control prototyping and experimental validation.
Vector VN8900	Network simulation, testing, bypassing, and prototyping.

The tables show the difference between production ECUs and rapid-prototyping platforms. Production ECUs typically provide only a few megabytes of memory and must satisfy strict timing, scheduling, and safety requirements. Rapid-prototyping platforms provide greater computational capability and are therefore suitable for algorithm development and experimental validation. However, they are not representative of the resource constraints encountered in a production ECU.
These constraints motivate the use of structured optimal-control methods with limited memory requirements and predictable execution characteristics. In this work, PMP is used to derive a solver structure based on state propagation, costate propagation, and pointwise Hamiltonian minimization. This reduces the reliance on generic nonlinear programming solvers and supports embedded deployment. The experimental implementation presented in this thesis is evaluated on a dSPACE MicroAutoBox II rapid-prototyping platform.

4.1.2 Implications of Memory and Computation Limits

The hardware limits shown in Table 4.1 have direct implications for the implementation of optimization-based control algorithms. Production-oriented automotive microcontrollers typically provide memory in the order of a few megabytes. For example, Infineon AURIX TC3x devices provide up to 16 MB flash memory and more than 6 MB integrated RAM, with up to six TriCore cores operating at 300 MHz. Similarly, the NXP S32K3 family provides up to 12 MB program flash and up to 2304 KB SRAM, with Arm Cortex-M7 cores operating up to 320 MHz. In contrast, rapid-prototyping platforms such as the dSPACE MicroAutoBox III provide substantially larger memory and higher computational capability, for example 2 GB DDR4 RAM and a quad-core ARM processor in the 1.4 GHz class. These values show the large gap between prototyping hardware and production ECU hardware.
For an optimal control solver, the available RAM is particularly important because the solver must store discretized state trajectories, control trajectories, costate trajectories, model parameters, lookup tables, temporary variables, and possibly derivatives such as Jacobians or Hessians. Assuming double-precision variables, each scalar requires 8 bytes. Therefore, storing a trajectory with N grid points, n_x states, n_u inputs, and n_λ costates requires approximately

$$M_{\text{traj}} \approx 8N(n_x + n_u + n_\lambda) \text{ bytes.} \tag{4.1}$$

For example, with $N = 1000$, $n_x = 5$, $n_u = 5$, and $n_\lambda = 5$, the trajectory storage alone requires approximately 120 kB. This value is acceptable on most prototyping systems, but it becomes significant on production ECUs when additional buffers, lookup tables, communication stacks, operating-system overhead, and safety-related software are considered.
The following table gives an approximate comparison between the hardware memory and the practical memory budget that may be available for the optimal control software. The software budget is not a fixed hardware specification; it represents a conservative engineering estimate after accounting for other ECU tasks, communication, calibration data, diagnostics, and safety-related functions.

Table 4.3: Approximate memory implications for optimal control implementation.

Platform	RAM	Solver Budget	Implication
AURIX TC3x	$>$ 6 MB	0.5–2 MB	Fixed-size solver
NXP S32K3	2304 kB	100–500 kB	Careful allocation
MicroAutoBox II	16 MB	1–4 MB	RT prototyping
MicroAutoBox III	2 GB	$>$ 100 MB	Large models

The computation-time constraint depends on the controller sampling rate. Typical available computation windows are shown in Table 4.4. In practice, the optimization algorithm should use only a fraction of the sampling period, since sensing, communication, diagnostics, and actuation tasks must also be executed within the same cycle.

Table 4.4: Approximate computation-time implications for real-time control.

Controller Rate	Sampling Period	Suggested Solver Budget	Typical Use Case
1 Hz	1000 ms	100–500 ms	Slow thermal optimization
10 Hz	100 ms	10–50 ms	Energy management control
50 Hz	20 ms	2–10 ms	Vehicle supervisory control
100 Hz	10 ms	1–5 ms	Fast embedded control
1000 Hz	1 ms	< 0.5 ms	Low-level actuator control

For the charging problem considered in this work, the dominant battery thermal dynamics are slow. Therefore, the solver can be executed at a relatively low update rate, or the complete optimal trajectory can be computed once and stored in memory. However, the execution time must still be bounded and predictable, especially when the solver is deployed as part of a real-time embedded control architecture.

4.1.3 Target Hardware Requirements

To evaluate the embedded feasibility of both the PMP-based and SQP-based charging solvers, a target performance envelope is defined rather than focusing on a single production ECU. The selected range represents low- to medium-performance automotive control hardware, for which memory usage and execution time must remain limited and predictable [36, 37, 38].

For the present implementation study, the desired solver RAM usage is set to 0.5–4 MB. This budget includes state and control trajectories, model parameters, lookup data, solver variables, and temporary arrays. Since predictable execution is a central design objective, the implementation avoids dynamic memory allocation wherever possible and relies on fixed-size data structures.

The target execution time is selected based on the charging application. Because battery charging and thermal dynamics evolve more slowly than fast actuator-level vehicle dynamics [**?**], the solver does not need to operate at high-frequency control rates. In this work, an execution time below 100 ms is considered suitable for repeated supervisory updates. Shorter execution times in the range of 10–50 ms are preferred when repeated online updates are required, while longer runtimes may still be acceptable when the full charging trajectory is computed once at the beginning of the charging process.

Table 4.5: Target implementation requirements for the embedded charging solver.

Requirement	Target Range	Purpose
Solver RAM usage	0.5–4 MB	Fits low- to medium-range ECUs
Preferred RAM usage	< 2 MB	Leaves margin for other tasks
Solver execution time	< 100 ms	Supervisory charging control
Preferred execution time	10–50 ms	Repeated online updates
Memory allocation	Static	Predictable memory usage
Iteration count	Fixed or bounded	Predictable execution time

These targets are used as practical design guidelines for the runtime architecture. The aim is to keep the PMP solver compact, deterministic, and suitable for deployment on automotive embedded hardware.

4.2 Optimal Charging Problem Formulation

This section outlines the optimal control problem for the charging problem we aim to implement.

4.2.1 Main Formulation

In this work, the charging process is modeled using two states: battery temperature and state of charge. The state vector is defined as

$$x(t) = \begin{bmatrix} T_b(t) & \mathrm{SoC}(t) \end{bmatrix}^\top, \tag{4.2}$$

where T_b is the battery temperature and SoC is the battery state of charge.
The control input vector is defined as

$$u(t) = \begin{bmatrix} P_b(t) & P_{\mathrm{HVCH}}(t) & P_{\mathrm{HP}}(t) \end{bmatrix}^\top, \tag{4.3}$$

where P_b is the battery power, P_{HVCH} is the high-voltage coolant heater power, and P_{HP} is the heat-pump power.
The battery thermal dynamics are modeled as

$$\frac{dT_b}{dt} = \frac{1}{c_b m_b} \left(Q_{\mathrm{joule}} + Q_{\mathrm{HVCH}} - Q_{\mathrm{amb}} - Q_{\mathrm{HP}} \right), \tag{4.4}$$

where c_b is the battery specific heat capacity and m_b is the battery mass. The terms Q_{joule}, Q_{HVCH}, Q_{amb}, and Q_{HP} denote Joule heat generation, heat supplied by the HVCH, heat exchanged with the ambient environment, and heat removed by the heat pump, respectively.
The state-of-charge dynamics are given by

$$\frac{d\mathrm{SoC}}{dt} = -\frac{P_b}{C_b U_{\mathrm{oc}}}, \tag{4.5}$$

where C_b is the battery capacity and U_{oc} is the open-circuit voltage. During charging, P_b is negative according to the sign convention used in this work, which results in an increase in SoC.
The charging dynamics can therefore be written compactly as

$$\dot{x}(t) = f_x(x(t), u(t)), \tag{4.6}$$

4.2.2 Charging Power and Battery Power Limits

During charging, the battery power P_b is constrained by both vehicle-side and charger-side limits. In this work, the sign convention is defined such that $P_b < 0$ during charging, yielding $\dot{\mathrm{SoC}} > 0$. Consequently, the lower bound on P_b corresponds to the maximum admissible charging-power magnitude.

The first bound is a vehicle-side battery power limit, denoted as $P_{b,\min}(T_b, \mathrm{SoC})$. This bound depends on the battery temperature and state of charge, and represents the charging power that can be safely accepted by the battery.
The second bound is a charger-side limit, denoted as $P^{\mathrm{chg}}_{b,\min}$. This bound is derived from the maximum available charger power $P^{\max}_{\mathrm{chg}}$. Since both limits must be satisfied simultaneously, the effective lower bound on the battery power is given by

$$P_b \geq \max\left\{P_{b,\min},\, P^{\mathrm{chg}}_{b,\min}\right\}. \tag{4.7}$$

This expression means that the most restrictive lower bound is selected at each operating point.
The charger-side power constraint is obtained from the power balance

$$P_{\mathrm{chg}} = \frac{R_b}{U_{\mathrm{oc}}^2} P_b^2 - P_b + P^{\mathrm{cab}}_{\mathrm{HVCH}} + P_c + P_{\mathrm{HP}} + P^{b}_{\mathrm{HVCH}} + P_{\mathrm{aux}}, \tag{4.8}$$

where R_b is the battery internal resistance, U_{oc} is the open-circuit voltage, $P^{\mathrm{cab}}_{\mathrm{HVCH}}$ is the HVCH power used for cabin heating, P_c is the cabin heating demand, P_{HP} is the heat-pump power, P^{b}_{HVCH} is the HVCH power used for battery heating, and P_{aux} represents auxiliary electrical loads such as infotainment, air-conditioning support, and other vehicle subsystems not included in the main traction or charging power flow.
The charger-side constraint is written as

$$P_{\mathrm{chg}} - P^{\max}_{\mathrm{chg}} \leq 0. \tag{4.9}$$

Solving this inequality for P_b gives the charger-side lower bound

$$P^{\mathrm{chg}}_{b,\min} = \frac{U_{\mathrm{oc}}\left(U_{\mathrm{oc}} - \sqrt{U_{\mathrm{oc}}^2 - 4R_b\left(P^{\mathrm{cab}}_{\mathrm{HVCH}} + P_c + P_{\mathrm{HP}} + P^{b}_{\mathrm{HVCH}} + P_{\mathrm{aux}} - P^{\max}_{\mathrm{chg}}\right)}\right)}{2R_b}. \tag{4.10}$$

The negative-root solution is selected because P_b is negative during charging. This bound ensures that the total requested charging power does not exceed the available charger power.

4.2.3 Cost Function

During charging, the objective is to minimize both the charging time and the energy drawn from the grid. The original cost function is written as

$$J = w_t t_f + w_e \int_{t_0}^{t_f} P_{\mathrm{grid}}(t)\, dt, \tag{4.11}$$

where w_t is the time-cost weight, w_e is the energy-cost weight, t_f is the final charging time, and P_{grid} is the power drawn from the grid.
Using normalized time $\tau \in [0, 1]$, the cost can be expressed as

$$J = w_t t_f + w_e t_f \int_0^1 P_{\mathrm{grid}}(\tau)\, d\tau. \tag{4.12}$$

Time normalization is used to define the problem over a fixed interval, since the final time t_f is an optimization variable and cannot be used directly as the upper integration limit.
After simplification, the constant terms are collected into $\tilde{w}_t$, and the cost used in the PMP formulation becomes

$$\tilde{J} = \underbrace{t_f \tilde{w}_t}_{\text{terminal cost}} + \int_{t_0}^{t_f} \underbrace{w_e \left(\frac{R_b P_b^2}{U_{\text{oc}}^2} + P_{\text{HVCH}} + P_{\text{HP}} + P_{\text{HVAC}} - \frac{Q_{\text{HP}}}{\eta_{Q_{\text{HVCH}}}} \right)}_{\text{stage cost}} dt. \tag{4.13}$$

Here, $\tilde{w}_t$ is the modified time-cost weight after collecting constant power terms. The terminal cost has unit SEK, while the stage cost has unit SEK/s and becomes SEK after integration. Therefore, $\tilde{J}$ represents the total charging cost.

4.3 PMP Implementation

The first solution methodology investigated for the charging problem is an indirect optimal-control approach based on Pontryagin's Maximum Principle. The motivation for using PMP is that it allows the structure of the charging problem to be exploited explicitly, which can lead to a compact and computationally efficient solver architecture. This is particularly attractive in an embedded setting, where predictable execution time and low memory overhead are important design requirements.
Applying PMP transforms the original charging optimal control problem into a BVP involving state dynamics, costate dynamics, transversality conditions, and pointwise Hamiltonian minimization. In contrast to a generic nonlinear programming approach, this formulation does not rely on repeatedly solving large-scale optimization subproblems online. Instead, it reduces the problem to the propagation of state and costate trajectories together with the evaluation of candidate control laws. This makes the method leaner and more deterministic, although also more specialized and less flexible when the model or objective changes.
The boundary conditions depend on the problem formulation. If the final state is free, the terminal costate condition is

$$\lambda(t_f) = \frac{\partial \Psi(x(t_f))}{\partial x}. \tag{4.14}$$

In this work, PMP is used to derive structured optimality conditions for the charging problem. By exploiting the problem structure, the optimization is decomposed into state dynamics, costate dynamics, and pointwise Hamiltonian minimization. This results in a lean and largely deterministic solver with limited nested iteration, making the approach suitable for embedded implementation. However, this efficiency comes at the cost of flexibility. Changes to the system model, constraints, or objective function may require a manual re-derivation of the state equations, costate equations, and Hamiltonian expressions before the optimization procedure can be applied.

4.3.1 Hamiltonian Formulation

For the PMP formulation, the Hamiltonian is constructed using the stage cost and the system dynamics. For the charging problem, it is defined as

$$H = S(x,u) + \lambda_{T_b}\frac{dT_b}{dt} + \lambda_{\mathrm{SoC}}\frac{d\mathrm{SoC}}{dt}, \tag{4.15}$$

where $S(x,u)$ is the stage cost, and λ_{T_b} and λ_{SoC} are the costates associated with the battery temperature and state of charge, respectively.
Equivalently, using vector notation,

$$H(x,u,\lambda) = S(x,u) + \lambda^\top f_x(x,u), \tag{4.16}$$

where $\lambda = \begin{bmatrix}\lambda_{T_b} & \lambda_{\mathrm{SoC}}\end{bmatrix}^\top$ and $f_x(x,u)$ represents the battery thermal and electrical dynamics.

4.3.2 Unconstrained PMP Control Laws

The unconstrained control candidates are obtained by applying the stationarity condition of the Hamiltonian with respect to each control input. For the battery power, the unconstrained candidate (P_b,uc) is determined by solving

$$\frac{\partial H}{\partial P_b} = 0. \tag{4.17}$$

The resulting expression is then evaluated against the admissible battery-power bounds to determine the optimal control input.
gives the unconstrained optimal battery power

$$P^*_{b,\mathrm{uc}}(x,\lambda) = \frac{c_b m_b \lambda_{\mathrm{SoC}} U_{\mathrm{oc}}}{2C_b\left(\lambda_{T_b} + c_b m_b w_e\right) R_b}. \tag{4.18}$$

Here, λ_{SoC} denotes the costate associated with SoC and represents the sensitivity of the optimal cost to variations in SoC. Under the charging convention adopted in this work, $P_b < 0$. Therefore, the unconstrained candidate P^*_b must lie within the admissible negative battery-power range. In addition, the denominator of the analytical expression for $P^*_{b,\mathrm{uc}}$ must satisfy

$$\lambda_{T_b} + c_b m_b w_e > 0. \tag{4.19}$$

This condition prevents division by zero and ensures that the resulting candidate is consistent with the battery-power bounds.
Similarly, the unconstrained HVCH power is obtained from

$$\frac{\partial H}{\partial P_{\mathrm{HVCH}}} = 0. \tag{4.20}$$

Since P_{HVCH} enters the thermal dynamics quadratically, the resulting unconstrained solution is

$$P^*_{\mathrm{HVCH,uc}} = \frac{\eta_{\mathrm{HVCH}}\lambda_{T_b} + c_b m_b w_e}{2\alpha^2_{\mathrm{HVCH}}\lambda_{T_b}}. \tag{4.21}$$

This expression is used only when it is feasible and satisfies the corresponding optimality condition. The expression becomes singular when $\lambda_{T_b} = 0$, which is relevant near the final time because the terminal battery temperature is free.
For the heat-pump power, the stationarity condition is

$$\frac{\partial H}{\partial P_{\mathrm{HP}}} = 0. \tag{4.22}$$

Solving this condition gives the unconstrained heat-pump power

$$P^*_{\mathrm{HP,uc}} = \frac{w_e \left(\frac{\sigma_{\mathrm{HP}}(T_b)(Q_{\mathrm{HP},0}+Q_{\mathrm{HP},1}T_b)}{\eta_{Q_{\mathrm{HVCH}}}} - 1 \right) + \frac{\lambda_{T_b}\sigma_{\mathrm{HP}}(T_b)(Q_{\mathrm{HP},0}+Q_{\mathrm{HP},1}T_b)}{c_b m_b}}{\frac{2\alpha^2_{\mathrm{HP}} w_e \sigma_{\mathrm{HP}}(T_b)}{\eta_{Q_{\mathrm{HVCH}}}} + \frac{2\alpha^2_{\mathrm{HP}}\lambda_{T_b}\sigma_{\mathrm{HP}}(T_b)}{c_b m_b}}. \tag{4.23}$$

In the implementation, the unconstrained solutions are used only when they satisfy their input bounds. If an unconstrained value is outside the admissible range, the corresponding lower or upper bound is used instead. This gives a finite set of feasible Hamiltonian candidates, and the control input applied at each grid point is selected as the candidate that gives the minimum Hamiltonian.

4.3.3 IPOPT Benchmark Solution

For benchmarking purposes, a reference solution is generated using IPOPT through the CasADi optimization framework. The same optimal control problem is formulated as a nonlinear programming problem and solved using IPOPT. The resulting solution is used as a baseline for evaluating the accuracy and performance of the proposed PMP-based solver.

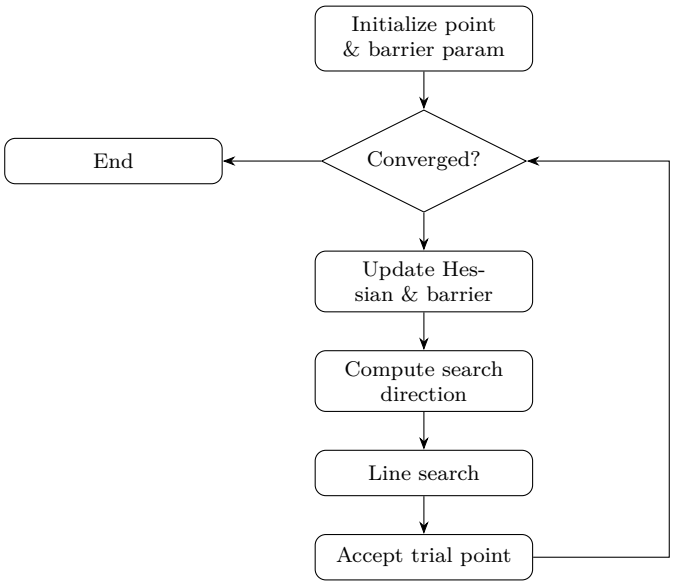


Figure 4.1: General architecture of the IPOPT-based benchmark solver for charging problem.

Each block in Fig. 4.1 represents a runtime computation module with its own computational and memory requirements. Operations such as derivative evaluation, linear system solution, and convergence checks can lead to variable execution time and memory usage.
In this work, IPOPT is not intended for direct embedded implementation. Instead, it is used as a high-accuracy benchmark solver executed on a PC. The IPOPT solution provides a reference trajectory, objective value, and computation time against which the proposed PMP-based solver and `acados`-based solver can be compared.

4.3.3.1 IPOPT Results

The IPOPT benchmark solver converged successfully to an optimal solution. The solver required 22 objective and constraint evaluations and 21 Lagrangian Hessian evaluations. The total wall-clock time reported by CasADi [39] was approximately 293.79 ms, while the total time reported by IPOPT was approximately 0.290 s. The detailed timing breakdown is reported in Table 4.6.
The IPOPT benchmark was executed on a PC with an 13th Gen Intel i7, *10* cores, *64* GB RAM, running *Windows 11* and MATLAB *2017b*. Therefore, the reported computation time is hardware-dependent and is used only as a PC-based reference benchmark, not as an embedded ECU execution-time estimate.

Table 4.6: Detailed timing of IPOPT function evaluations for Charging Problem.

Solver Component	Total Wall Time	Average Wall Time
Objective function, `nlp_f`	2.38 ms	108.20 μs
Constraint function, `nlp_g`	2.52 ms	114.59 μs
Objective gradient, `nlp_grad_f`	10.11 ms	219.76 μs
Lagrangian Hessian, `nlp_hess_l`	44.31 ms	2.11 ms
Constraint Jacobian, `nlp_jac_g`	8.34 ms	362.80 μs
Total solver time	293.79 ms	293.79 ms

4.3.4 Desktop PMP Implementation

This section describes the architecture of the PMP implementation on a desktop computer. The overall workflow is shown in Fig. 4.2. The implementation is divided into two main parts: an offline function-generation stage and an online solution stage.
In the offline stage, the vehicle, battery, and charging parameters are loaded from locally stored data files. These parameters define the battery thermal model, electrical model, power limits, cost function, and control bounds used in the charging optimization problem. A trained neural network is also loaded during this stage. The network provides an initial estimate for the unknown variables required to solve the PMP boundary value problem.
The neural network takes the initial state of charge, desired final state of charge, ambient temperature, and normalized cost weighting as inputs. Its outputs are the initial costates associated with battery temperature and state of charge, together

with an initial estimate of the final charging time. These outputs form the initial guess for the Newton-based solution of the boundary value problem.

The analytical expressions required by the PMP solver are generated using the MATLAB symbolic toolbox. These include the state dynamics, costate dynamics, unconstrained control laws, and Hamiltonian expressions. The symbolic expressions are then converted into executable MATLAB functions. Since the control inputs may operate either at their bounds or at their unconstrained PMP values, multiple Hamiltonian candidates are generated for the admissible input configurations.

In the online solution stage, the initial guess from the neural network is used to start the Newton iteration. For each Newton iteration, the charging process is simulated forward over the normalized time grid using Algorithm 1. At every grid point, the unconstrained control values are first evaluated and checked against their bounds. The feasible Hamiltonian candidates are then evaluated, and the control combination corresponding to the lowest Hamiltonian value is selected. Using this selected input, the state and costate trajectories are propagated forward.

After the forward simulation, a terminal residual is computed from the boundary conditions. The residual includes the final state-of-charge error, the terminal costate condition for the free battery temperature, and the transversality condition associated with the free final time. If the residual norm is below the selected tolerance, the solver terminates and returns the optimal final time, states, costates, controls, and Hamiltonian trajectory.

If the residual is above the tolerance, the unknown vector containing the initial costates and final time is updated using a Newton step. The residual Jacobian is approximated using finite differences by perturbing each component of the unknown vector by a small value Δw. The updated guess is then used for the next Newton iteration. This process is repeated until convergence or until the maximum number of iterations is reached.

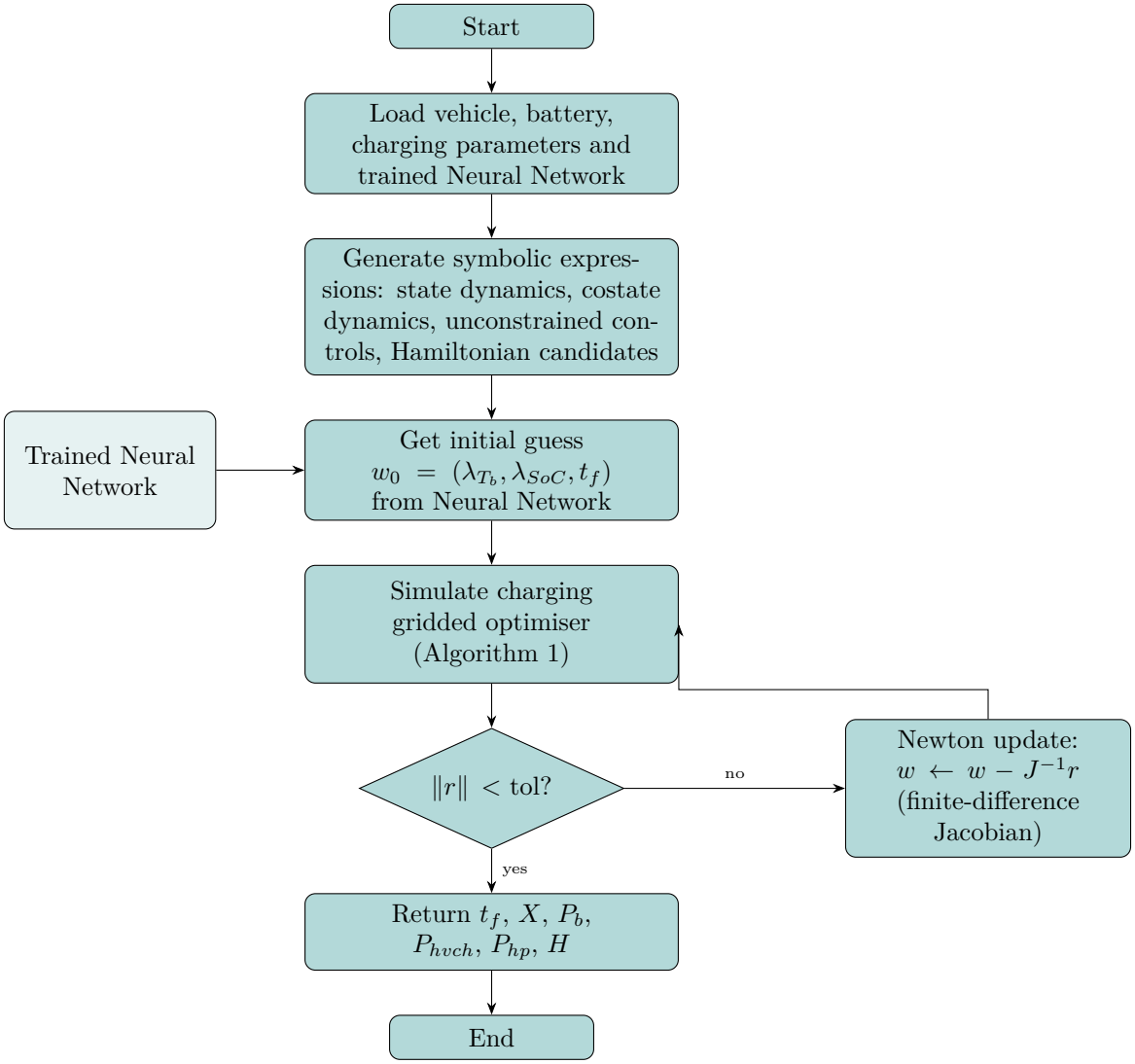


Figure 4.2: Architecture of the PMP-based desktop solver for the charging problem.

4.3.4.1 Neural Network Initialization

The neural network is trained using simulation data generated from charging cycles and is developed using the MATLAB Deep Learning Toolbox. It is used to provide the initial guess required by the PMP solver. The network inputs are the initial state of charge, desired final state of charge, ambient temperature, and time-cost weight:

$$[\mathrm{SoC}_0,\ \mathrm{SoC}_{\mathrm{des}},\ T_{\mathrm{amb}},\ C_t]\,.$$

The outputs are the initial costates associated with battery temperature and state of charge, and the final charging time estimate:

$$[\lambda_{T_b}(0),\ \lambda_{\mathrm{SoC}}(0),\ t_f]\,.$$

The network consists of two layers, with 10 neurons in the input-side hidden layer

Algorithm 1 Simulate Charging Gridded Optimiser (PMP)

Require: w, $x0$, SoC_des, wt_t, $Phvch\_max$, P_hpmax, $Tamb$, we
Ensure: r, X, $Lambda$, Pb, $Phvch$, Php, H
$N \leftarrow 50, \quad d\tau \leftarrow 1/N$
$X(:,1) \leftarrow x0, \quad Lambda(:,1) \leftarrow w(1{:}2), \quad tf \leftarrow w(3)$
for $k = 1$ to N **do**
    $Tb, SoC \leftarrow X(:,k); \quad lambdaTb, lambdaSoC \leftarrow Lambda(:,k)$
    Compute Pb_uc, $Phvch\_uc$, Php_uc from costates
    Check feasibility of Pb_uc, $Phvch\_uc$, Php_uc against bounds
    for each mode $m \in \{1, \ldots, 27\}$ **do**
        if mode m is feasible **then**
            Evaluate H_m
            if $H_m < H(k)$ **then**
                $H(k) \leftarrow H_m$
                $Pb(k), Phvch(k), Php(k) \leftarrow$ controls for mode m
                $lam\_dot \leftarrow$ costate derivative for mode m
            end if
        end if
    end for
    $X(:,k+1) \leftarrow X(:,k) + tf \cdot f(Tb, SoC, Pb(k), Phvch(k), Php(k), Tamb) \cdot d\tau$
    $Lambda(:,k+1) \leftarrow Lambda(:,k) + tf \cdot lam\_dot \cdot d\tau$
end for
$r(1) \leftarrow X(2, N{+}1) - SoC\_des$
$r(2) \leftarrow Lambda(1, N{+}1)$
$r(3) \leftarrow H(N) + wt\_t$
return r, X, $Lambda$, Pb, $Phvch$, Php, H

and 4 neurons in the following hidden layer. These outputs form the initial guess for the Newton-based solution of the boundary value problem.

Since the network is trained from simulation data, it is model-dependent. Therefore, if the vehicle hardware, battery model, or any major subsystem model is changed, the training data must be regenerated and the neural network must be retrained for the updated system.

4.3.5 Validation of the PMP Algorithm

The PMP-based solver is validated by comparing its solution with a direct numerical optimization solution obtained using IPOPT through the CasADi framework. Both methods are evaluated using the same initial condition and objective weights. The initial state of charge is set to $\mathrm{SoC}_0 = 0.4$, the desired final state of charge is set to $\mathrm{SoC}_{\mathrm{des}} = 0.9$, and the energy and time weights are selected as $w_e = 0.8$ and $w_t = 0.2$, respectively.

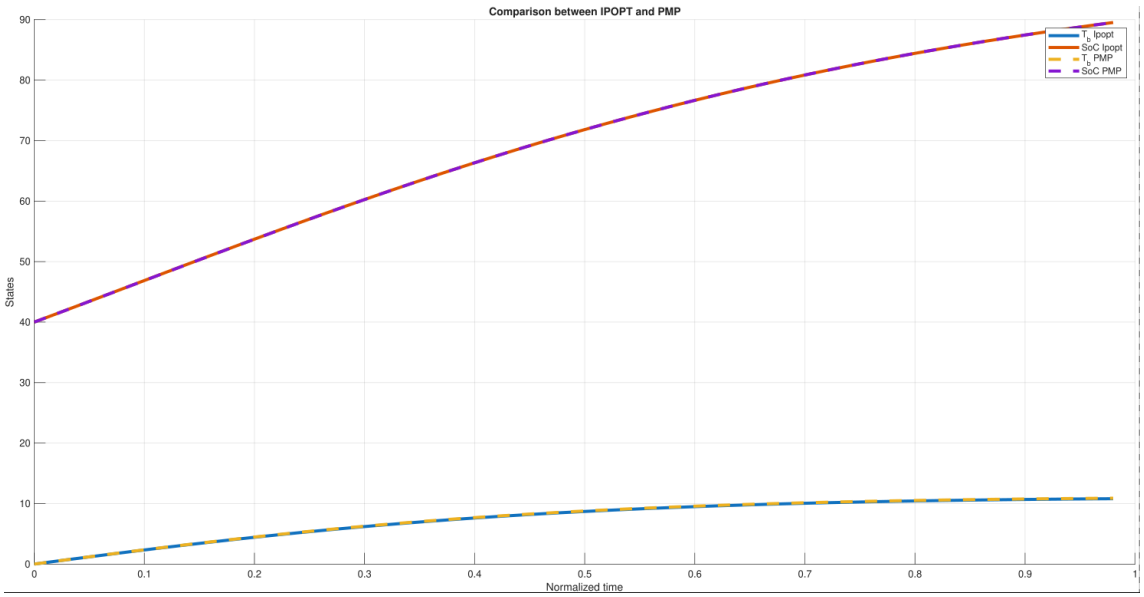


Figure 4.3: Comparison of the PMP-based solution and the IPOPT solution for the charging problem.

Fig. 4.3 compares the state trajectories obtained from the PMP-based solver and the IPOPT-based solution. The battery-temperature and state-of-charge trajectories closely overlap for both methods. This indicates that the PMP formulation produces a solution consistent with the direct nonlinear programming benchmark.

The final charging times obtained from the two methods are summarized in Table 4.7. The PMP solver gives a final charging time of 46.90 min, while the IPOPT solution gives 47.03 min. The difference between the two solutions is therefore 0.13 min, which corresponds to approximately 7.8 s.

Table 4.7: Comparison of final charging time obtained using IPOPT and PMP.

Method	Final Charging Time	Difference from IPOPT
IPOPT	47.03 min	–
PMP	46.90 min	0.13 min
Absolute difference	–	≈ 7.8 s

4.3.6 Runtime Implementation

The runtime implementation is designed considering the main embedded constraints discussed earlier: flash memory, runtime memory, and computation time. The desktop implementation shown previously is not directly suitable for deployment because it contains operations that should not be executed during runtime.

The first issue is the repeated construction of symbolic functions. In the desktop implementation, functions such as the state dynamics, costate dynamics, control laws, and Hamiltonian expressions are generated using the symbolic toolbox. This is unnecessary for runtime execution. Therefore, these expressions are generated once on the desktop, exported as standalone procedural functions, and stored in flash memory. During runtime, the controller only evaluates these pre-generated functions for the current inputs.

The second issue is the neural network evaluation. Direct execution of the neural network on the embedded target can be computationally expensive and may introduce implementation overhead. To avoid this, the trained network is converted into an equivalent procedural function, denoted as `NN2man`. This function evaluates the network using fixed mathematical operations and provides the initial guess for the PMP solver. This conversion is discussed in Section 4.3.6.1.

In the runtime architecture, the blocks marked in yellow in Fig. 4.4 are stored in flash memory together with the generated code. Since these blocks are fixed functions with predefined inputs and outputs, they do not require symbolic computation or dynamic construction during execution. The worst-case execution time of the runtime solver is therefore mainly determined by the maximum number of Newton iterations, since each iteration performs a fixed forward simulation and a finite-difference Jacobian evaluation.

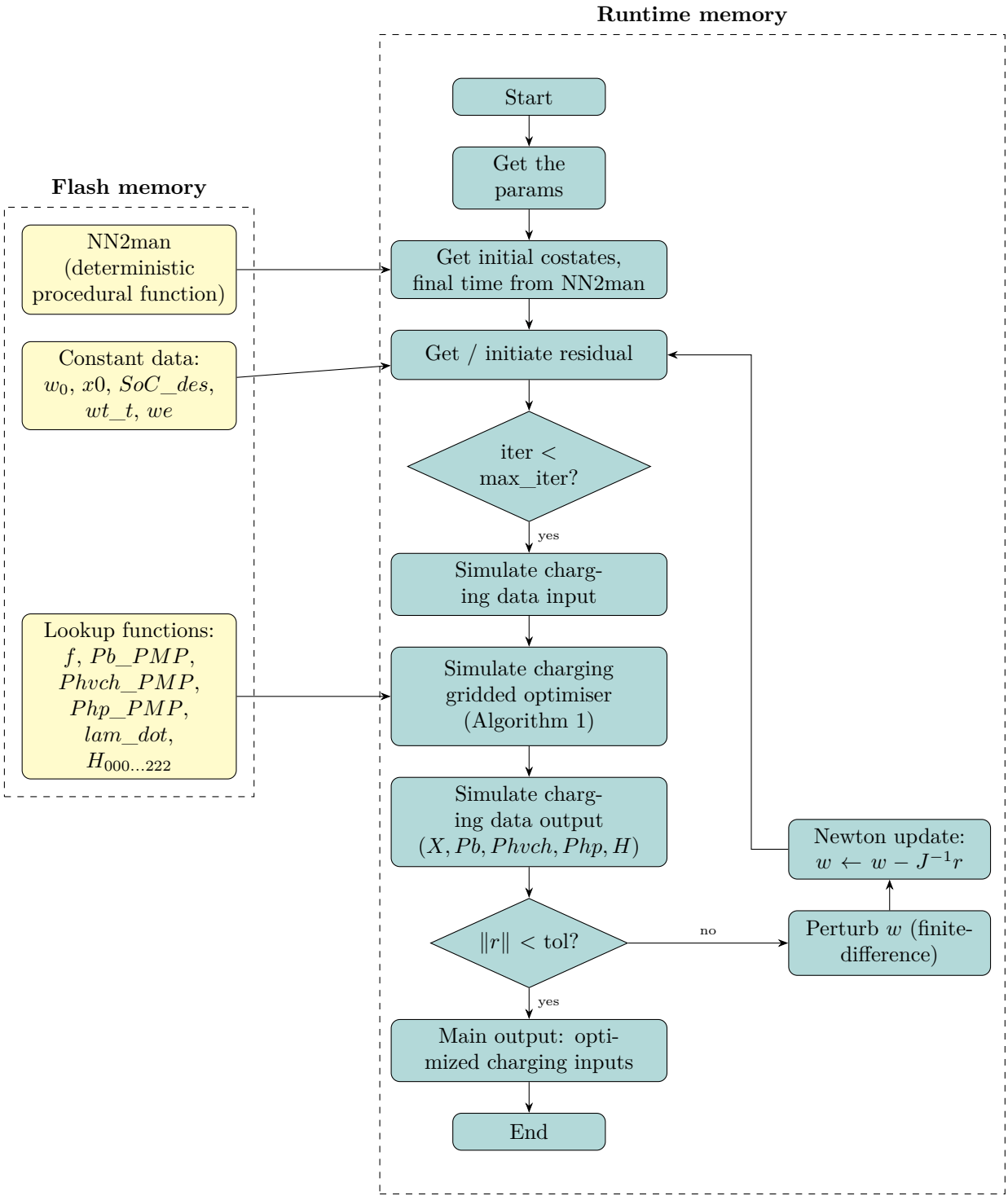


Figure 4.4: Embedded implementation of the PMP-based solver for the charging problem.

4.3.6.1 Adapting Neural Network to RTI Implementation

As discussed earlier, the neural network output is required to initialize the PMP solver, since it provides the initial costates and the final charging time estimate. The network is trained for a specific vehicle and battery model. Since this thesis considers

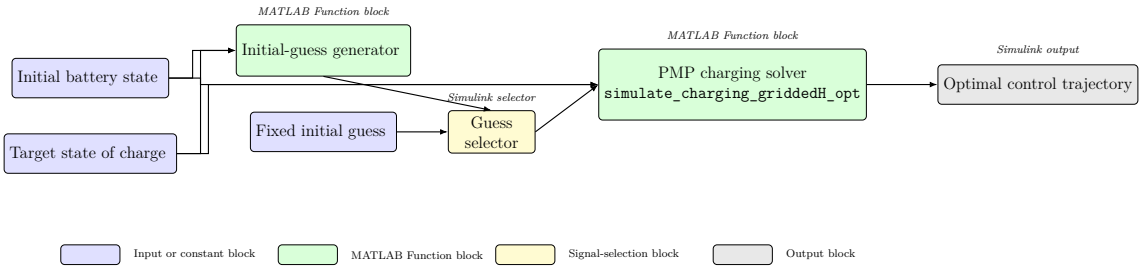


Figure 4.5: Simplified block diagram of the Simulink model used for embedded PMP solver development for the charging problem.

implementation on a fixed target vehicle, the vehicle parameters are assumed to remain unchanged after deployment. Therefore, the trained network parameters are fixed before flashing the controller to the hardware.

To make the network suitable for RTI implementation, it is converted into a procedural function using the same principle applied to the symbolic PMP functions. The weights and biases are extracted from the trained MATLAB neural network and written explicitly into a standalone function. This avoids loading or executing the full neural network object during runtime. The resulting function, denoted as `NN2man`, computes the initial guess using fixed matrix operations and activation functions, making the implementation more suitable for embedded execution.

4.3.7 Simulink and Hardware Deployment

This subsection describes how the desktop PMP solver is translated into an embedded implementation suitable for deployment on the dSPACE MicroAutoBox II platform. The discussion covers the Simulink-based code generation workflow, numerical representation issues such as floating-point precision, the characteristics of the target hardware, and the final real-time validation results. The purpose is to show how the analytically derived PMP solution can be converted into a practical embedded runtime implementation while preserving predictable execution behavior and acceptable memory usage.

4.3.7.1 Runtime Code Build Implementation

The runtime PMP solver is implemented in Simulink using a MATLAB Function block. This structure is suitable for the dSPACE workflow, since the same Simulink model can later be used for code generation and deployment on the MicroAutoBox II platform. The development model is shown in Fig. 4.5.

The execution time of the same runtime code was also measured in the desktop Simulink environment, as shown in Fig. 4.6. The measured elapsed time was approximately 0.08 s.

```
>> SCGOv0731_outerloop
Elapsed time is 0.081869 seconds.
>>
```

Figure 4.6: Execution-time measurement of the runtime PMP code in the desktop Simulink environment.

This desktop execution time is used only as a functional check during development. It is not used as the final performance measure, since the actual runtime performance must be evaluated on the target hardware.
The desktop memory usage of the runtime variables was estimated using the MATLAB `whos` command. The results are summarized in Table 4.8. The listed variables occupy approximately 54.99 kB in the MATLAB workspace. This value is not the flash memory usage of the generated embedded code, but it provides a useful estimate of the variables stored during execution.

Table 4.8: MATLAB workspace memory usage of PMP algorithm runtime variables.

Variable Group	Variables	Memory Usage
Model parameters	`coeffs`	52.41 kB
State and costate trajectories	`X_PMP`, `Lambda_PMP`	1.64 kB
Control trajectories	`Pb_PMP`, `Phvch_PMP`, `Php_PMP`	1.20 kB
Hamiltonian trajectory	`H_PMP`	0.40 kB
Solver variables and scalars	`w`, `r`, `dw`, `iter`, `ct_flag`, etc.	0.14 kB
Total	–	$\approx$ 54.99 kB

This section describes the implementation of the runtime PMP solver on the MicroAutoBox II rapid-prototyping platform.

4.3.7.2 Floating-Point Precision Considerations

One practical issue in embedded implementation is floating-point precision. Many vehicle ECUs either do not support double-precision arithmetic or execute it with a high computational cost. This is important for the `NN2man` function, since the neural-network weights and biases contain decimal values that must be represented accurately.
If these values are rounded excessively, the neural-network output can change significantly. This may result in an inaccurate initial guess for the costates and final time, which can affect the convergence of the PMP solver.
If double precision is not available on the target hardware, the weights, biases, and intermediate variables should be scaled and implemented using fixed-point or integer arithmetic. In that case, the scaling factors must be selected carefully to preserve numerical accuracy while satisfying the hardware limitations.
In this work, the main hardware platform used for development is the MicroAutoBox II rapid-prototyping system. Since the platform supports floating-point arithmetic, the solver variables can be implemented using floating-point data types. However,

the data type of each parameter must still be specified explicitly before loading it into the MATLAB workspace and generating code.
The following MATLAB function is used to create Simulink parameters with a defined data type and storage class.

```
function simpar = make_param_f(value, description)
    simpar = Simulink.Parameter;
    simpar.Value = single(value);
    simpar.DataType = 'single';
    simpar.StorageClass = 'Auto';
    simpar.Description = description;
end
```

In this function, the keyword `single` specifies single-precision floating-point representation. This can be changed to `double` if double-precision arithmetic is required and supported by the target hardware. The field `StorageClass` defines how the parameter appears in the generated C code. When `StorageClass` is set to `Auto`, Simulink decides how the variable is represented during code generation. If the parameter must be accessible outside the generated code module, the storage class can be changed to `ExportedGlobal`.

4.3.7.3 MicroAutoBox II Hardware Specification

The target embedded platform used for hardware implementation is the dSPACE MicroAutoBox II. It is a compact real-time prototyping system intended for in-vehicle control development and embedded algorithm testing. The main hardware specifications are summarized in Table 4.9.

Table 4.9: Hardware specification of dSPACE MicroAutoBox II.

Parameter	Specification
Platform	dSPACE MicroAutoBox II
Processor	IBM PowerPC 750GL / 750GX class processor
Clock frequency	900 MHz
Local RAM	16 MB
Flash memory	16 MB non-volatile flash
Communication memory	6 MB

As per standard assumptions for a practical solver RAM and flash memory budget, the following allocation is considered to ensure a safe and jitter-free implementation.
Runtime memory

- RTOS, I/O, drivers, and communication: ∼4–6 MB
- Other vehicle/control tasks: ∼3–4 MB
- Safety margin and fragmentation: ∼2–3 MB
- **Available for optimal control task:** ≈3–5 MB

Flash memory

- RTOS, drivers, and middleware: ∼3–4 MB
- Other application tasks: ∼3–4 MB

- Safety margin and future growth: ∼2–3 MB
- **Available for optimal control task (code + constant tables):** ≈4–6 MB

4.3.7.4 Simulink Implementation

The MicroAutoBox implementation uses a Simulink-based development workflow. The PMP algorithm is first written in MATLAB and then integrated into Simulink using a MATLAB Function block. The required inputs, parameters, and outputs are defined in the Simulink model before code generation.
TargetLink is used in the workflow to generate production-oriented C code from the Simulink model. It supports safer code generation, explicit data typing, and integration with real-time task scheduling. This is important for embedded implementation, where memory usage, execution order, and timing behavior must be controlled.
The dSPACE toolchain is then used to build the real-time application for the MicroAutoBox II. During this process, the generated code is compiled and the required interface files are produced. In particular, the generated `.sdf` file is used by dSPACE ControlDesk to configure, monitor, and interact with the application running on the MicroAutoBox.

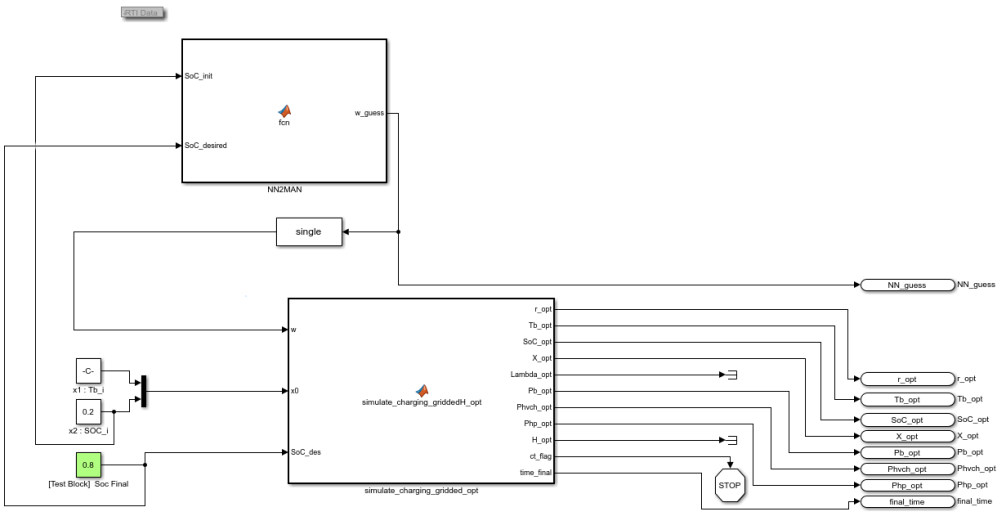


Figure 4.7: Final Simulink model used for MicroAutoBox II implementation of the charging problem.

Fig. 4.7 shows the Simulink model used for the final MicroAutoBox II implementation. The PMP solver is implemented as a MATLAB Function block and connected to the required input and output signals. The stop block at the output is used to terminate the execution once the optimization is completed.
This stop condition is required because the MicroAutoBox application continues to run until an interrupt or stop condition is triggered. In the implemented architecture, the loop is kept active until the optimization result is available. After

termination, the computed values can be recorded and used for post-processing, result comparison, and plotting.

4.3.7.5 MicroAutoBox II Results

After the runtime application is built and flashed to the MicroAutoBox II, the PMP solver can be executed on the target hardware. The relevant solver outputs are connected to Simulink output blocks, which allows the computed data to be retrieved after execution through the dSPACE workflow.

The measured output data include the optimal state trajectory, control trajectory, costate trajectory, Hamiltonian values, residuals, convergence flag, and solver execution time. The execution time represents the time taken by the solver from the start of the optimization routine until convergence or termination. This value is used to evaluate whether the implementation satisfies the real-time computation requirements of the target platform.

Once the solver execution is completed, the recorded output signals can be exported and used for post-processing, comparison with the desktop implementation, and plotting of the final results.

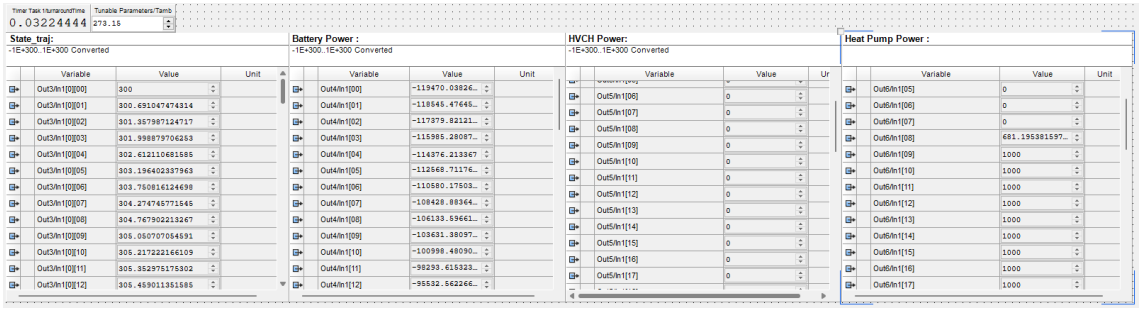


Figure 4.8: Recorded PMP solver output for the charging problem in dSPACE ControlDesk.

Fig. 4.8 shows a recorded output snapshot from dSPACE ControlDesk. The displayed signals are exported from the MicroAutoBox II runtime application. The value shown as turnaround time in the top-right corner is treated as the effective solver execution time in this implementation. In this test, the measured execution time is approximately 0.03 s. This value is used to evaluate the runtime performance of the PMP solver on the target hardware.

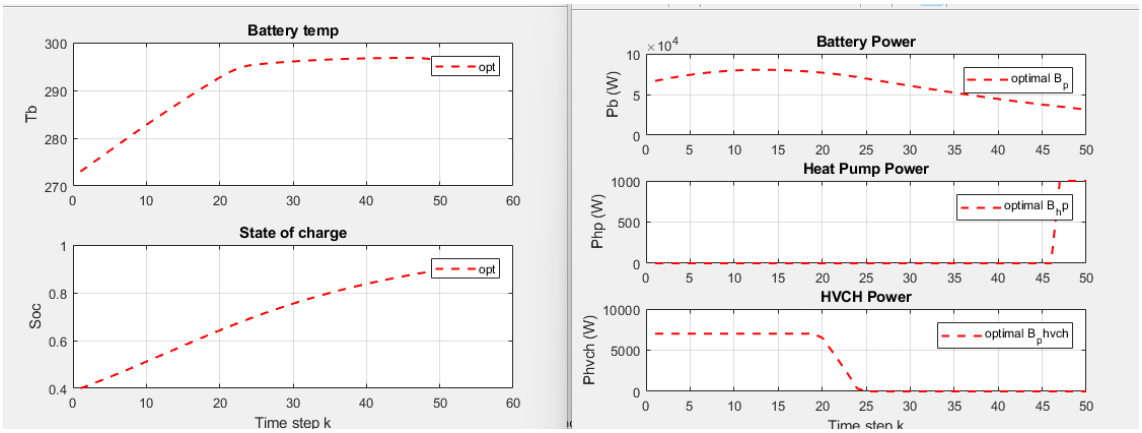


Figure 4.9: Charging result extracted from the MicroAutoBox II implementation.

Fig. 4.9 shows the charging result extracted from the MicroAutoBox II implementation. The recorded trajectory matches the MATLAB desktop implementation, which confirms that the runtime code was executed correctly on the target hardware.
The memory footprint of the MicroAutoBox II implementation was estimated from the generated linker `.map` file. The extracted values are summarized in Table 4.10. The runtime RAM estimate is approximately 4.47 MB, which is slightly above the initial solver memory target of 4 MB. Therefore, the current implementation demonstrates successful deployment, but further optimization is required to meet the intended memory budget.

Table 4.10: Memory usage estimated from the generated linker map file for PMP solver implementation.

Memory Category	Estimated Usage
Flash memory: code and read-only data	$\approx$ 2.37 MB
Flash memory: including initialized data image	$\approx$ 6.61 MB
Runtime RAM: data, stack, and heap	$\approx$ 4.47 MB

The result shows that the current runtime RAM usage is close to the target range but still exceeds the desired limit. Future work will focus on reducing the memory footprint by simplifying stored data structures, removing unused generated functions, and further optimizing the runtime architecture.

Table 4.11: Summary of MicroAutoBox II implementation results for Charging Problem.

Metric	Result	Evaluation
Solver execution time	30 ms	Within target
Target execution time	10–50 ms	–
Runtime RAM usage	$\approx$ 4.47 MB	Slightly above target
Target RAM usage	0.5–4 MB	–
Memory allocation	Fixed-size implementation	Suitable for embedded execution

The MicroAutoBox II results show that the PMP-based charging solver is suitable for embedded deployment in its present application context. The measured execution time of approximately 30 ms lies within the desired target range for supervisory charging control, and the fixed-size implementation provides deterministic runtime behavior. The main remaining limitation is memory usage, which at approximately 4.47 MB is slightly above the desired upper bound of the target budget. Nevertheless, the implementation demonstrates that a structured PMP-based charging solver can be deployed successfully on embedded automotive prototyping hardware with only moderate further optimization required.

4.4 ACADOS Implementation

Having established a structured charging solver using PMP, the same charging problem is next evaluated using a direct embedded optimization framework based on SQP in `acados`. The purpose of this second implementation is to assess how the charging problem behaves when handled by a more general numerical solver architecture rather than by an analytically specialized method. This enables a comparison between solver flexibility, numerical accuracy, and embedded suitability for the same underlying application.

4.4.1 Formulation Modifications and Soft Constraints

The primary modification required to transition from the analytical PMP formulation to the `acados` setup is the introduction of slack variables, converting hard path constraints into soft constraints. In an embedded environment, unexpected disturbances, unmodeled physical dynamics, or discretization errors can cause the system states to temporarily drift outside their strict bounds. Under rigid constraints, this causes the underlying Quadratic Programming (QP) solver to instantly become infeasible, leading to a catastrophic controller failure.
The charger-side power constraint $P_{\text{chg}} - P_{\text{chg}}^{\max} \leq 0$ and the vehicle-side dynamic battery power limit $P_{b,\min}(T_b, \text{SoC}) \leq P_b(t)$ are softened. The original path constraint vector $h(x, u) \leq u_h$ is modified to:

$$lh \leq h(x, u) + s \leq uh \tag{4.24}$$

where $s \geq 0$ is the optimization slack variable vector.
To ensure that the slack variables are only activated when absolutely necessary to prevent infeasibility, they are heavily penalized in the augmented objective function. In this implementation, a purely quadratic penalty structure is selected by setting the linear penalty vectors z_l and z_u to zero. The total objective function minimized by *acados* over the horizon is expressed as:

$$\min_{x,u,s} \quad J_{\text{original}}(x, u) + \sum_{k=0}^{N-1} \left(\frac{1}{2} s_k^\top Z_l s_k + \frac{1}{2} s_k^\top Z_u s_k \right) \tag{4.25}$$

The baseline objective function, J_{original}, matches the cost function derived for the charging problem in (4.13). Z_l and Z_u represent the quadratic slack penalty matrices.

Setting these weights to a high value ($\sim 10^5$) ensures that the slack variables remain strictly zero during normal operation, preserving the exact physical constraints unless numerical failure is imminent, while avoiding the numerical non-smoothness sometimes introduced by linear penalty lines.

Additionally, to optimize computational efficiency on the embedded target, all states, controls, and parameters are fully normalized between 0 and 1 prior to compilation. This standardizes the numerical conditioning of the Hessian and constraint Jacobians within the solver.

4.4.2 Solver Settings and Configuration

The transcribed NLP is solved using a direct multiple shooting method over a prediction horizon tailored to the charging duration. The solver configurations specified within the `setup_acados.m` script are selected to balance execution speed with numerical accuracy on resource-constrained hardware. These fixed algorithmic parameters are summarized in Table 4.12.

Table 4.12: `acados` Solver Configuration Parameters for Charging Problem

Parameter	Value/Type
NLP Solver Type	`SQP` / `SQP_RTI`
QP Solver	`PARTIAL_CONDENSING_HPIPM`
Max QP Iterations	15
Hessian Approximation	`BFGS`
Regularization Method	`PROJECT`
Integrator Type	`ERK`
Integrator Stages	4
Integrator Steps	1
Cost Type	`EXTERNAL`

These solver settings are strictly chosen to meet the limitations of the embedded environment. Utilizing an explicit Runge-Kutta fourth-order integrator (`ERK`) alongside a `BFGS` Hessian approximation eliminates the heavy computational burden of evaluating explicit system Jacobians and second derivatives. To handle the resulting mathematical program, the partial condensing `HPIPM` solver reduces the size of the QP problem over the horizon, while limiting the inner loops to a maximum of 15 iterations to ensure a deterministic runtime.

Crucially, to find the best compromise between solution accuracy and real-time computation, this work evaluates both the standard `SQP` solver (which iterates until convergence at each time step) and the highly efficient Real-Time Iteration (`SQP_RTI`) scheme, which executes exactly one SQP iteration per sample step to minimize computational delay on hardware.

The compiled solver architecture is fully exported as a custom Simulink S-Function block, allowing it to be deployed on dSPACE MicroAutoBox II.

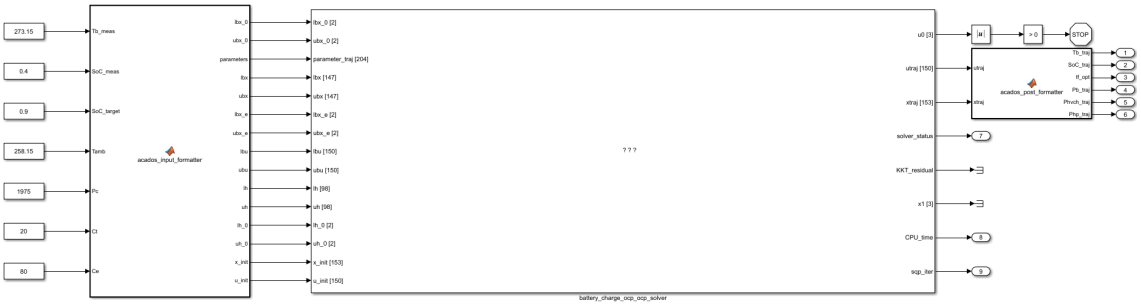


Figure 4.10: Simulink integration of the `acados`-based charging solver

We feed in all the required initial parameters and final target values to the controller through the `acados_input_formatter` block which converts them to appropriate dimensions required by the solver block. Then we export the optimal state and input trajectories through the `acados_post_formatter` block which separates the different values from the flattened arrays received from the solver block. As this block only needs to run once to generate the optimal trajectories, we have a stop block which stops the simulation as soon as we receive an output from the solver block.

4.4.3 Results and Solver Comparison

To validate the accuracy and performance of the compiled `acados` solver, a two-phase comparative analysis is conducted. First, the standard `acados` SQP setup is verified against an offline, high-precision benchmark using the IPOPT solver via CasADi. Second, the performance and trajectory tracking of the full `SQP` loop are compared against the Real-Time Iteration (`SQP_RTI`) variant to evaluate the computational trade-offs within the embedded framework.

4.4.3.1 Verification Against IPOPT Benchmark

To establish the numerical fidelity of the `acados` implementation, both solvers were simulated using identical initial conditions and target SoC profiles. Because `IPOPT` is an offline, full-space barrier method that iterates until tight convergence criteria are met, it serves as an ideal baseline for optimal solution quality.

The state and control trajectory comparisons between `acados` SQP and CasADi IPOPT are illustrated in Figure 4.11. The quantitative deviations between the two solvers are summarized in Table 4.13.

Table 4.13: Solver Metric Comparison Summary for Charging Problem (SQP vs IPOPT)

Metric	Value
Final Charge Time (`acados` SQP)	47.554 min
Final Charge Time (CasADi IPOPT)	47.035 min
Difference in Final Time (Δt_f)	31.14 s
Max. Absolute Temperature Error	0.9715 K
Max. Absolute SoC Error	0.1324 %
Max. Absolute Battery Power Error	0.76 kW

The results demonstrate that the `acados` `SQP` framework tracks the benchmark solution with exceptional accuracy. The discrepancy in final charging time is negligible at approximately 31 seconds ($\Delta t_f = 0.519$ min) over a roughly 47-minute charging horizon. Furthermore, the maximum absolute errors remain small throughout the entire sequence, with temperature deviations staying below 1 K and SoC deviations restricted to 0.13%. The minor power discrepancy (≤ 0.76 kW) is attributed to the local tracking nature of the sequential quadratic approximations and the regularized `BFGS` Hessian updates. These results firmly establish that the `acados` solver yields a high-quality, physically accurate solution.

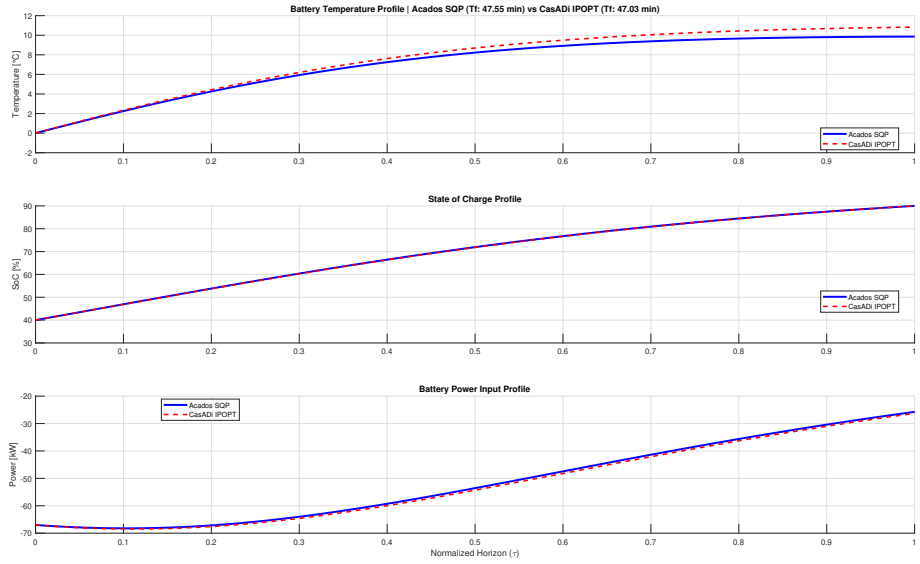


Figure 4.11: Trajectory comparison of battery temperature, state of charge, and input power between `acados` SQP and the CasADi IPOPT benchmark.

4.4.3.2 Comparison between SQP and SQP_RTI

Next, the execution modes within the embedded framework are evaluated by contrasting the full `SQP` loop against the single-iteration `SQP_RTI` scheme. The comparative trajectories for both modes are depicted in Figure 4.12, and their precise numerical deviations are detailed in Table 4.14.

Table 4.14: Solver Metric Comparison Summary for Charging Problem (SQP vs SQP_RTI)

Metric	Value
Final Charge Time (SQP)	47.554 min
Final Charge Time (SQP_RTI)	40.792 min
Difference in Final Time (Δt_f)	6.762 min
Max. Absolute Temperature Error (ΔT_b)	8.8282 K
Max. Absolute SoC Error (ΔSoC)	2.8372 %
Max. Absolute Battery Power Error (ΔP_b)	9.6282 kW

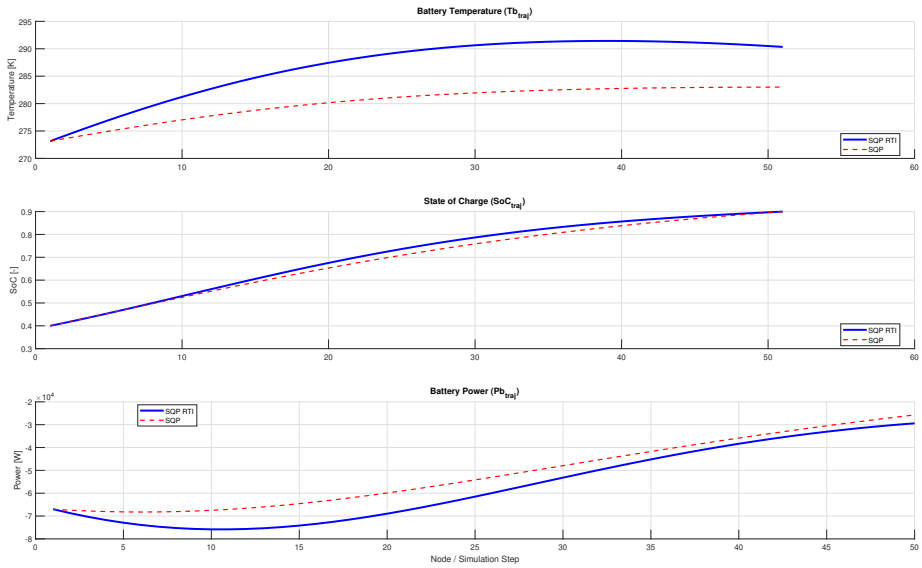


Figure 4.12: Trajectory comparison of battery temperature, state of charge, and input power between SQP and SQP_RTI.

The simulation reveals a profound divergence in the behavior of the two algorithms. As shown in Table 4.14, the `SQP_RTI` scheme achieves a final charging time of 40.792 minutes, dropping the process duration by an aggressive 6.762 minutes compared to the full `SQP` baseline. However, this speed comes at a severe physical cost. The maximum absolute tracking errors skyrocket between the two modes, yielding a massive temperature deviation of 8.8282 K, an SoC mismatch of 2.84%, and a sharp battery power error peak of 9.6282 kW.
Because the RTI scheme restricts the solver to exactly one QP iteration per sampling interval, it lacks the numerical freedom to fully resolve the strong thermal-electrical nonlinearities and state-dependent bounds before updating the plant. This single-iteration truncation introduces a tracking lag, causing the solver to severely overshoot optimal thermal thresholds and demand excess energy.
Crucially, this aggressive reduction in final time directly contradicts our optimization priorities. In this implementation, the cost weights are scaled as $w_t = 20$ and $w_e = 80$, preserving the same prioritization of energy over charging time.

While the SQP_RTI approach achieves vastly superior computational efficiency and lower calculation times per step, the optimal charging problem is inherently less time-sensitive than high-frequency dynamics such as vehicle stability or powertrain control. The slow time constants of thermal systems permit sufficient computational headroom to run a solver to tight convergence. Therefore, given the severe quality degradation, massive thermal overshoots ($\sim 8.8\,\mathrm{K}$), and high power errors observed in the RTI sequence, we conclude that the full SQP loop is the superior choice for this application, rejecting the fast calculation times of SQP_RTI to safeguard the high-priority energy and thermal requirements of the system.

4.5 Discussion

The charging problem illustrates clearly the trade-off between analytical specialization and numerical flexibility in embedded optimal control. The PMP-based method achieves excellent execution speed and deterministic behavior by exploiting the structure of the charging formulation directly. This makes it highly attractive for embedded deployment when the problem structure is fixed and well understood. In contrast, the acados-based SQP method offers a more general optimization workflow that can represent the same charging problem accurately with relatively limited manual derivation effort, but at the cost of higher algorithmic complexity and a stronger sensitivity to solver configuration.
The comparison between full SQP and SQP_RTI further shows that the fastest solver variant is not necessarily the best choice for every application. In the present charging problem, the slower thermal dynamics permit the use of full SQP, and the resulting solution quality is significantly better than that of SQP_RTI. This observation is important for embedded energy-management applications more broadly, since it shows that solver selection should be matched to the physical timescale and control objective of the problem rather than being based solely on raw runtime considerations.

5

Eco-Driving

This chapter investigates embedded optimization strategies for eco-driving in battery electric vehicles, with the objective of generating energy-efficient velocity trajectories over known routes. The problem formulation adopted here is based on the framework presented in [19], where route topography, speed limits, and vehicle powertrain characteristics are incorporated into a spatial-domain optimal control problem. In the present work, the emphasis is on implementing and evaluating this formulation using the `acados` framework under embedded-oriented constraints, with particular focus on comparing one-shot and receding-horizon execution strategies as well as the performance trade-off between `SQP` and `SQP_RTI`.

5.1 Eco-Driving Problem Formulation

The primary objective of the eco-speed planning problem is to determine an optimal velocity trajectory $v(s)$ that minimizes a trade-off between energy consumption and total travel time over a known route. By leveraging data such as road topography (elevation/grade) and legal speed limits, the vehicle management system can proactively adjust the torque command to utilize the vehicle's energy consumption more efficiently.

This formulation focuses exclusively on the longitudinal dynamics of a Battery Electric Vehicle (BEV). The balance of traction and resistive forces determines the acceleration of the vehicle. In the time domain, the traction torque demand is expressed as:

$$\frac{T}{R_r} = mg\sin(\theta) + \frac{1}{2}\rho C_d A v^2 + C_r mg\cos(\theta) + (m + m_i)\frac{dv}{dt} \tag{5.1}$$

where T is the traction torque, R_r is the tire rolling radius, m is the vehicle mass, g is the gravitational acceleration, θ is the road gradient, ρ is the air density, C_d is the aerodynamic drag coefficient, A is the vehicle frontal area, v is the longitudinal velocity, C_r is the tire rolling resistance coefficient, and m_i is the equivalent rotational inertia mass of the driveline.

Because road information (e.g., gradient θ and reference speeds) is intrinsically location-based, the system dynamics are reformulated into the spatial domain (s). By applying the chain rule,

$$\frac{dv}{dt} = \frac{dv}{ds}\frac{ds}{dt} = \frac{dv}{ds}v \tag{5.2}$$

Substituting this yields the continuous spatial dynamics:

$$\frac{dv}{ds} = \frac{1}{(m+m_i)v}\left(\frac{T}{R_r} - mg\sin(\theta(s)) - \frac{1}{2}\rho C_d A v^2 - C_r mg\cos(\theta(s))\right) \tag{5.3}$$

To optimize powertrain efficiency, the internal losses and physical limitations of the Electric Drive Unit (EDU) must be explicitly defined. The EDU power loss P_{loss} is represented as a polynomial surface fitted to the motor's efficiency map:

$$P_{loss}(v,T) = a_1 v + a_2 v^2 + a_3 T^2 + a_4 v T^2 + a_5 v^2 T^2 \tag{5.4}$$

where a_{1-5} are fitting coefficients.
Furthermore, the maximum traction and regenerative braking capabilities of the motor are not static; they are highly dependent on the current operating speed. The speed-dependent torque envelope data is modeled using 5th-order polynomial functions:

$$T_{max}(v) = b_1 v^5 + b_2 v^4 + b_3 v^3 + b_4 v^2 + b_5 v + b_6 \tag{5.5}$$

$$T_{min}(v) = c_1 v^5 + c_2 v^4 + c_3 v^3 + c_4 v^2 + c_5 v + c_6 \tag{5.6}$$

where b_{1-6} and c_{1-6} are the fitting coefficients.
The objective of the eco-driving controller is to minimize a combination of trip time and powertrain energy loss over the prediction horizon. In the spatial domain, the time and energy are accumulated according to the following integrals:

$$t = \int \frac{1}{v}\, ds, \quad E = \int \frac{P_{\text{batt}}}{v}\, ds \tag{5.7}$$

To ensure numerical stability within the optimization solver, these variables are normalized against nominal highway cruising values (v_{nom} and P_{nom}). The normalized running cost $L(v,T)$ is formulated as:

$$L(v,T) = \lambda_1 \underbrace{\left(\frac{v_{nom}}{v}\right)}_{\text{Normalized Time}} + \lambda_2 \underbrace{\left(\frac{P_{loss}(v,T)/v}{P_{nom}/v_{nom}}\right)}_{\text{Normalized Energy}} \tag{5.8}$$

where λ_1 and λ_2 are weighting factors dictating the trade-off between aggressive and economical driving.
The optimization problem is constrained by the dynamic EDU capabilities and legal speed boundaries. The torque demand is bounded continuously by the envelope defined previously:

$$T_{min}(v(s)) \leq T(s) \leq T_{max}(v(s)) \tag{5.9}$$

To track the route's varying speed limits (v_{min}, v_{max}) without causing solver infeasibility, the velocity state constraints are formulated as soft constraints using exact penalty slack variables (ϵ_L, ϵ_U):

$$(1-\alpha_l)v_{min}(s) - \epsilon_L(s) \leq v(s) \leq (1+\alpha_u)v_{max}(s) + \epsilon_U(s) \tag{5.10}$$

where $\alpha_l \in [0, 1)$ and $\alpha_u \geq 0$ are design parameters defining the permissible velocity deviation envelope. The slack variables $\epsilon_L, \epsilon_U \geq 0$ measure any violation outside this expanded region, penalized by the following exact penalty function:

$$L_{slack}(\epsilon_L, \epsilon_U) = \frac{1}{2} Z_L \epsilon_L^2 + \frac{1}{2} Z_U \epsilon_U^2 \tag{5.11}$$

where Z_L and Z_U are the quadratic penalty weights for slack violation.
To formulate the problem for a digital MPC scheme, the continuous spatial dynamics are discretized over a step distance Δs. Applying the Euler forward method, the state transition equation for velocity from spatial step k to $k+1$ is derived as:

$$v_{k+1} = v_k + \frac{\Delta s}{(m + m_i) v_k} \left(\frac{T_k}{R_r} - mg \sin(\theta_k) - \frac{1}{2} \rho C_d A v_k^2 - C_r mg \cos(\theta_k) \right) \tag{5.12}$$

The complete spatially discretized Nonlinear Program (NLP) over a prediction horizon of N steps is defined as:

$$\min_{\mathbf{v}, \mathbf{T}, \boldsymbol{\epsilon}} \sum_{k=0}^{N-1} \left[L(v_k, T_k) + L_{slack}(\epsilon_{L,k}, \epsilon_{U,k}) \right] \Delta s \tag{5.13}$$

Subject to:

$$v_0 = v_{current} \tag{5.14}$$

$$v_{k+1} = F_{discrete}(v_k, T_k, \theta_k, \Delta s) \tag{5.15}$$

$$T_{min}(v_k) \leq T_k \leq T_{max}(v_k) \tag{5.16}$$

$$(1 - \alpha_l) v_{min,k} - \epsilon_{L,k} \leq v_k \leq (1 + \alpha_u) v_{max,k} + \epsilon_{U,k} \tag{5.17}$$

$$\epsilon_{L,k}, \epsilon_{U,k} \geq 0 \tag{5.18}$$

The physical parameters, EDU coefficients, and tuning weights used in this formulation are summarized in Table 5.1.

Table 5.1: Vehicle and Constraint Parameters for Eco-Driving Problem

Symbol	Description	Value	Unit
	Vehicle Parameters		
m	Vehicle mass	2640	kg
m_i	Equivalent rotational inertia mass	132	kg
R_r	Tire rolling radius	0.377	m
ρ	Air density	1.225	kg/m^3
C_d	Aerodynamic drag coefficient	0.372	-
A	Vehicle frontal area	2.8	m^2
C_r	Tire rolling resistance coefficient	0.01	-
g	Gravitational acceleration	9.81	m/s^2
	Cost & Constraint Weights		
α_l, α_u	Lower/Upper velocity tolerance	0.5, 0.05	-
Z_L, Z_U	Slack quadratic penalty weights	10^4, 10^8	-

5.2 ACADOS Implementation

The eco-driving problem was implemented in `acados` using a common baseline solver configuration. Since the purpose of this part of the work is to compare different execution strategies while keeping the numerical backend as consistent as possible, the QP solver, Hessian approximation, regularization, and integration settings were fixed across all experiments. The only solver setting varied deliberately was the NLP solver type, where both `SQP` and `SQP_RTI` were evaluated in order to compare the trade-off between solution quality and computational effort. The retained `acados` settings are summarized in Table 5.2.

Table 5.2: `acados` solver settings used for the Eco-Driving Problem.

Setting	**Value**
QP solver	`PARTIAL_CONDENSING_HPIPM`
QP solver iteration limit	15
NLP solver type	`SQP`, `SQP_RTI`
Hessian approximation	`BFGS`
Regularization method	`PROJECT`
Integrator type	`ERK`
Integration stages	4
Integration steps	1

The selected configuration was chosen to maintain a computationally efficient baseline while still preserving sufficient numerical robustness. `PARTIAL_CONDENSING_HPIPM` was retained as the QP solver because it offers a suitable compromise between preserving optimal-control structure and reducing QP size. A BFGS Hessian approximation was used since the eco-driving formulation does not rely on a least-squares cost structure, making a quasi-Newton approximation more appropriate than Gauss–Newton while still avoiding the cost of exact second derivatives. The `PROJECT` regularization method was included to improve numerical conditioning of the QP subproblems, especially in operating points where the curvature information becomes ill-conditioned. Finally, the system dynamics were discretized using an explicit Runge–Kutta integrator with four stages and one step per interval, which provides adequate numerical accuracy without introducing unnecessary computational overhead. Within this common setup, the comparison between `SQP` and `SQP_RTI` makes it possible to assess how much solution quality is gained by performing multiple SQP iterations relative to the reduced runtime of a real-time iteration scheme.
To evaluate the algorithm, two distinct execution strategies were developed and compared: a One-Shot optimization approach and a Receding Horizon Control (RHC) approach.

5.2.1 Strategy 1: One-Shot Optimization

The One-Shot approach is designed to compute the absolute global optimal trajectory for the entire route in a single calculation. In this configuration, the prediction horizon (N) is set equal to the total number of spatial data points in the route.

This approach stays true to the fundamental ideal of eco-speed planning: by having absolute "infinite preview" of all future topographical changes and speed limits, the solver can extract maximum efficiency (e.g., accelerating miles in advance of a massive hill to utilize momentum).

Because the problem dimension spans the entire route, the solver must navigate a massive variable space. Therefore, the maximum QP iterations per SQP step (`qp_solver_iter_max`) is set to 50 to guarantee convergence. This is the standard approach for eco-driving problems as we ideally want to optimize over the complete route information. While mathematically optimal, the One-Shot approach is computationally heavy, suffers from non-deterministic execution times, and requires a dynamic memory footprint that makes it unsuitable for real-time embedded controllers.

5.2.2 Strategy 2: Receding Horizon Control

To transition the algorithm from an offline benchmark to a real-time embedded application, a Receding Horizon Control (RHC) strategy was implemented. Instead of attempting to solve the entire route simultaneously, the solver looks ahead over a fixed prediction window.

In this implementation, the horizon is fixed to a physical distance of 10 km. The vehicle follows the generated optimal velocity profile, and the optimization problem is re-solved and updated every 2 km using the latest state feedback and preview data.

Because the NLP horizon is strictly fixed and significantly shorter, the memory footprint becomes static and manageable. Furthermore, because the solution from the previous 2 km update can be used to “warm-start” the current optimization problem, the solver requires far less effort to converge. Consequently, the `acados` configuration retains the `SQP` structure, but the `qp_solver_iter_max` is aggressively reduced to 15. This bounds the maximum execution time, ensuring the controller is lightweight, deterministic, and safe for ECU execution.

The ultimate goal of this framework is to validate the RHC algorithm on automotive-grade hardware. The deployment pipeline leverages `acados`'s automated code generation capabilities.

Once the mathematical model and solver options are defined via the MATLAB interface, `acados` generates the raw C-code and compiles it into an S-Function block (`get_acados_simulink_opts`). This standalone Simulink block encapsulates the entire optimization engine.

The Simulink model, housing the `acados` block alongside state-machine and signal routing logic, is then compiled using the dSPACE toolchain. The generated executable is flashed directly onto a dSPACE MicroAutoBox II (MABX II), a rapid prototyping unit widely used in the automotive industry.

5.3 Experimental Setup and Route Discretization

To rigorously evaluate the performance of both the One-Shot and RHC optimization strategies, simulations were conducted using a real-world route profile provided by Geely Technology Europe. The selected test route spans 30.84 km between Kungshamn and Munkedal in Sweden. The data comprises topographical road elevation (converted to gradient) and varying legal speed limits along the entire journey.

The raw route data extracted from the geographical database contains 546 fixed spatial nodes, resulting in a variable (dynamic) distance step size averaging $ds \approx 56.48$ meters between consecutive points.

Because the fundamental architectures of the One-Shot and RHC approaches differ, the raw route data must be handled differently for each strategy to ensure optimal solver performance.

For the One-Shot approach, the solver leverages the raw data directly. The prediction horizon length (N) is set to the total number of route nodes (546), and the variable spatial step size (ds) is passed to the solver at each node. By contrast, the RHC algorithm must maintain a fixed execution time and strict spatial continuity for embedded deployment. Therefore, the raw topographical and speed limit data is pre-conditioned through linear interpolation to enforce a strict, uniform step size of 50 meters. The RHC horizon is fixed at 10 km ($N = 200$ nodes), and the problem updates continuously as the vehicle progresses.

These fundamental configuration differences are summarized in Table 5.3.

Table 5.3: Horizon and discretization configurations for optimization strategies in eco-driving implementation.

Parameter	**One-Shot**	**RHC**
Route Data Handling	Unaltered	Interpolated
Spatial Step Size (ds)	Variable (Avg. $\sim$ 56.48 m)	Strictly Fixed (50 m)
Prediction Horizon (N)	Total Route Nodes (546)	Fixed Window (200 nodes)
Look-Ahead Distance	30.84 km (Entire Route)	10.0 km

To execute the generated `acados` C-code, a simulation environment was developed in Simulink. For One Shot approach we feed the velocity limits and road gradient profiles for the entire journey to the acados block which calculates the optimal trajectories for the entire route.

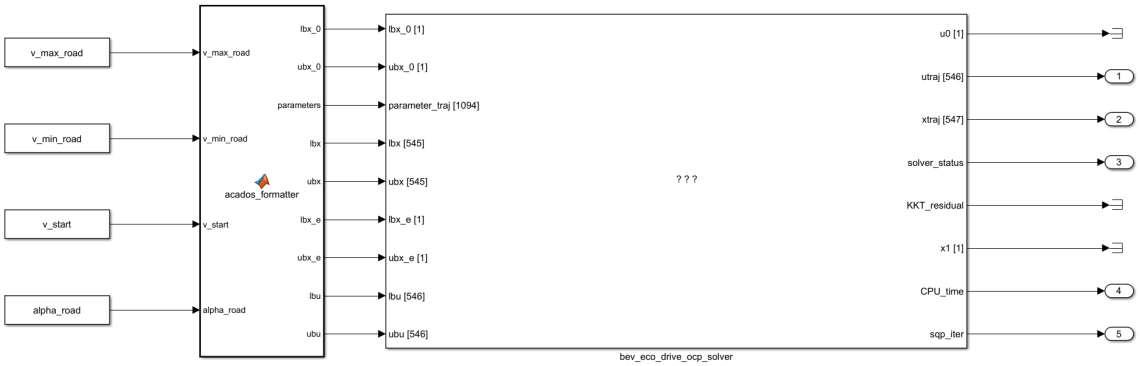


Figure 5.1: Simulink model architecture for one shot approach in eco-driving

The `acados_formatter` block ensures that the bounds and parameters being fed into the solver block match the required dimensions.

To implement the RHC strategy within Simulink, a distance-triggered "sliding window" algorithm manages the continuous flow of look-ahead data. Every 2 km, the system extracts a localized 10 km segment (200 nodes) from the global route arrays to update the road gradient $\theta(s)$ and permissible speed boundaries (v_{min}, v_{max}). These dynamically shifting parameters, alongside the vehicle's current velocity feedback (x_0), are fed directly into the `acados` S-Function block.

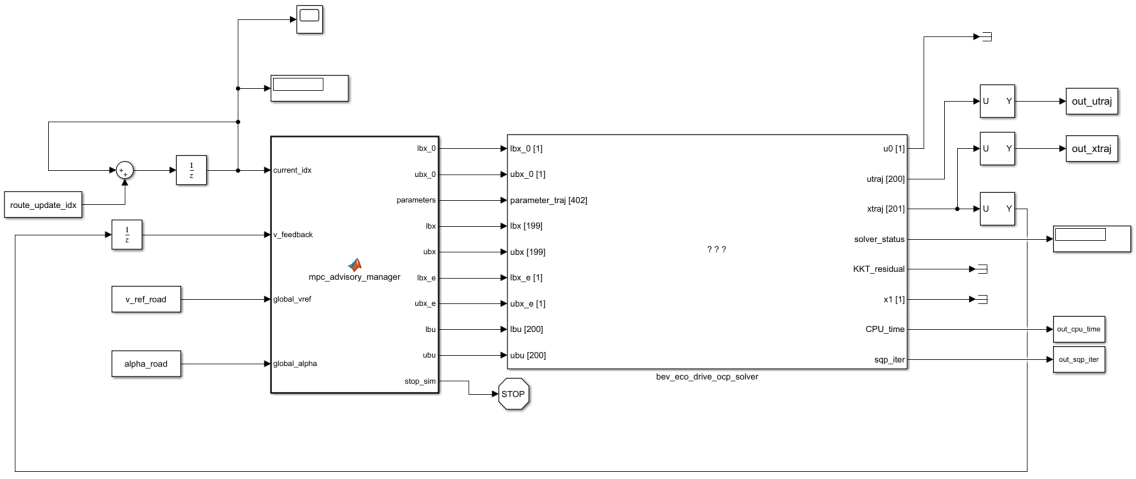


Figure 5.2: Simulink model architecture block for RHC approach in eco-driving

Here the `mpc_advisory_manager` function block manages the sliding window approach and appropriately provides the acados block with required parameters for the horizon, we shift the references by 2 km through the loop that feeds into `current_idx` input of the function block. A stopping criteria is added which stops the simulation when we reach the last index of the reference parameters.

5.4 Simulation Results

5.4.1 Comparison between One Shot and RHC

The optimal velocity and torque trajectories were computed for the 30.84 km test route under a specific calibration of objective weights ($\lambda_1 = 0.3, \lambda_2 = 0.7$). Figure 5.3 visually compares the velocity profiles and utilized motor torque between the global One-Shot optimum and the real-time RHC approach.

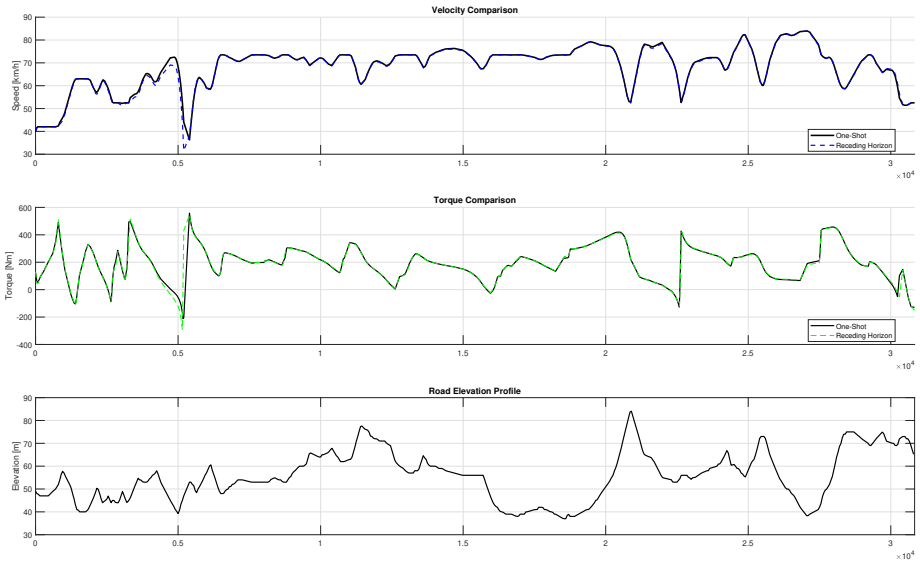


Figure 5.3: Comparison of optimal velocity profiles and EDU torque outputs across the Kungshamn-Munkedal route between one-shot and RHC approach.

To quantify the efficiency and feasibility of transitioning to a truncated horizon, key journey metrics were extracted and calculated. The comparative performance is detailed in Table 5.4.
The solve times mentioned in the time correspond to one iteration/run of the solver. Analyzing the numerical outcomes presented in Table 5.4, it is evident that the values obtained by the Receding Horizon Control approach are marginally, if at all, different from the absolute theoretical global minimum found by the One-Shot strategy. The One Shot approach fails to load on MABXII due to possibly the horizon length and memory being too large for the embedded hardware to process.
Because the 10 km preview window provided by the RHC contains sufficient topological variation to plan highly efficient energy recovery and momentum utilization strategies, extending the horizon further yields diminishing returns. Given that the RHC enforces a strictly bounded matrix size, inherently limits worst-case computational time via fewer maximum QP iterations, and achieves near-identical energy efficiency (Wh/km), it can be concluded that the RHC architecture is highly suitable and robust for real-time embedded eco-speed planning on hardware such as the dSPACE MicroAutoBox II.

Table 5.4: Journey results comparing One-Shot global optimization and RHC execution.

Metric	One-Shot Optimization	RHC
Total Distance	30835.0 m	30835.0 m
Total Time (seconds)	1658.61 s	1664.14 s
Total Time (minutes)	27.64 min	27.74 min
Average Speed	66.93 km/h	66.71 km/h
Total Energy Consumed	5.1673 kWh	5.1564 kWh
Efficiency	167.58 Wh/km	167.22 Wh/km
Solve Time (Simulink)	0.0490 s	0.0109 s (avg)
Solve Time (MABXII)	NaN s	0.3 s (avg)

5.4.2 Comparison between SQP and SQP_RTI

Next, we investigate the results obtained from the two different solver types available in `acados`, namely `SQP` (fully converged) and `SQP_RTI` (Real-Time Iteration), keeping all other solver and problem parameters identical.

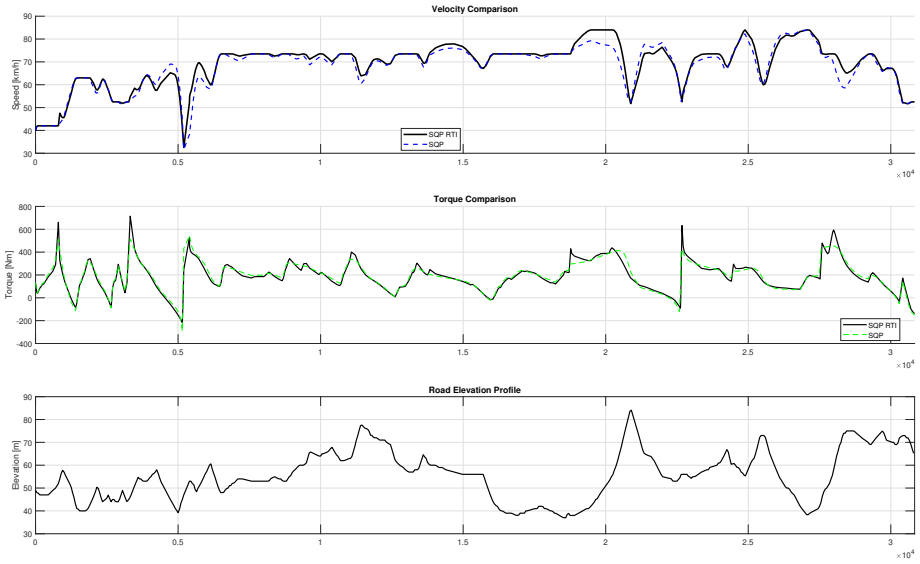


Figure 5.4: Comparison of optimal velocity profiles and EDU torque outputs across the Kungshamn-Munkedal route between SQP and SQP_RTI solvers.

Table 5.5: Performance Comparison between SQP and SQP_RTI Solvers on the Kungshamn-Munkedal Route

Metric	**SQP**	**SQP_RTI**
Total Distance (m)	30835	30835
Total Time (s)	1658.61	1642.43
Average Speed (km/h)	66.93	67.59
Total Energy Consumed (kWh)	5.1673	5.0691
Efficiency (Wh/km)	167.58	164.39
Solve Time (Simulink) (s)	0.0109 (avg)	0.0061 (avg)
Solve Time (MABXII) (s)	0.4 (avg)	0.07 (avg)

An analysis of the numerical outcomes presented in Table 5.5 and the trajectories in Figure 5.4 reveals a compelling divergence from classical optimization expectations. While a fully converged `SQP` solver is mathematically expected to yield superior tracking optimality, `SQP_RTI` achieves a lower specific energy consumption while completing the route 16.18 s faster.

This inversion stems from a known phenomenon in non-convex Receding Horizon Control (RHC) subjected to macro-step updates. Because the vehicle powertrain and aerodynamic models contain highly non-convex, high-order terms (such as quadratic drag and polynomial battery losses), the mathematical landscape features multiple localized valleys. When full `SQP` solves the truncated 10 km horizon to exact convergence at every 2 km update step, it aggressively over-optimizes for the artificial terminal boundary conditions of that isolated window. When these highly aggressive, localized trajectories are stitched together open-loop every 2 km, the resulting velocity profile falls into conservative local minima over sharp topographic transitions.

Conversely, `SQP_RTI` executes exactly one SQP iteration per spatial update. This structural limitation prevents the solver from over-fitting to the localized terminal quirks of a truncated horizon. By taking a single unit step ($\alpha = 1$) along the shifting initialization trajectory, `SQP_RTI` introduces an implicit numerical regularization that acts as a low-pass rolling filter. This effectively smooths out transient boundary mismatches, yielding a globally continuous and more energy-efficient velocity trajectory over short, highly dynamic routes.

Crucially, this algorithmic advantage translates to a massive reduction in computational latency. Within the desktop Simulink environment, the average execution time drops by 44.04%. The disparity becomes defining on the MABXII, where full `SQP` demands 0.4 s (400 ms) per run, whereas `SQP_RTI` requires just 0.07 s (70 ms)—an 82.50% computational saving. Although this eco-driving formulation is not strictly time-critical because the controller updates only every 2 km to plan over a 10 km horizon, a 400 ms latency still risks triggering watchdog timeouts on embedded platforms. Because `SQP_RTI` provides a bounded computational footprint while enhancing transient energy efficiency on this segment, it stands out as an exceptional candidate for practical deployment on shorter, topologically volatile tracks.

To verify whether this counter-intuitive superiority holds true under macro-scale conditions, the subsequent evaluation introduces a significantly longer route to test if the single-iteration framework maintains its edge when long-range steady-state dynamics begin to dominate.
Now we test these two solvers on a longer route that spans 575.1 km between Gothenburg and Idre in Sweden. The data comprises topographical road elevation (converted to gradient) and varying legal speed limits along the entire journey. This real-world route profile is provided by Geely Technology Europe.

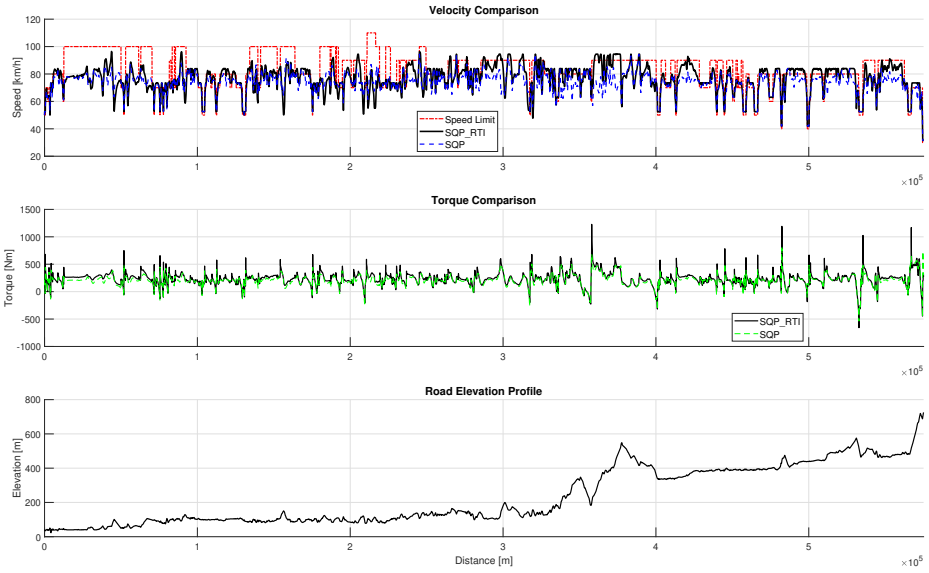


Figure 5.5: Comparison of optimal velocity profiles and EDU torque outputs across the Gothenburg-Idre route between SQP and SQP_RTI solvers.

Table 5.6: Performance Comparison between SQP and SQP_RTI Solvers on the Gothenburg-Idre Route

Metric	**SQP**	**SQP_RTI**
Total Distance (m)	575130	575130
Total Time (s)	28266.86	27412.89
Average Speed (km/h)	73.25	75.53
Total Energy Consumed (kWh)	105.4124	114.3607
Efficiency (Wh/km)	183.28	198.84
Solve Time (Simulink) (s)	0.0137 (avg)	0.0024 (avg)
Solve Time (MABXII) (s)	0.31 (avg)	0.07 (avg)

An analysis of the numerical outcomes presented in Table 5.6 and the trajectories in Figure 5.5 reveals that over this longer route from Gothenburg to Idre, the clas-

sical optimization paradigm is completely restored. Unlike the shorter Kungshamn-Munkedal segment, the fully converged SQP solver demonstrates a clear superiority in eco-driving optimality, yielding a specific energy consumption of 183.28 Wh/km compared to the 198.84 Wh/km required by SQP_RTI. This translates to an 8.49% energy savings for the full SQP scheme.
This macro-scale reversal highlights the structural trade-off between local boundary regularization and long-range model consistency. Over extended distances spanning hundreds of kilometers, short-range horizon boundary mismatches average out, and the accuracy of the nonlinear system constraints becomes the dominant factor in efficiency. Because SQP_RTI truncates its execution to a single iteration per step, it relies heavily on successive linearizations of the plant. Over a 575.1 km journey with continuous, compounding elevation changes, these unconverged linearizations accumulate significant linearization drift. This causes the RTI framework to structurally deviate from the true physical vehicle maps, missing highly efficient macro-scale energy recovery windows and driving the vehicle more aggressively than necessary (evidenced by the higher average speed of 75.53 km/h). Full SQP, by driving the system to complete mathematical convergence at every interval, successfully preserves the exact high-order physics of the powertrain over the entire macro-horizon.
From a computational standpoint, SQP_RTI maintains its massive latency advantage, reducing average execution times by 82.48% in Simulink (0.0024 s vs. 0.0137 s) and by 77.42% on the MABXII hardware target. However, within the context of this macro-level eco-speed planning layer, the strict execution limits of real-time iteration are no longer a definitive requirement. Because the controller evaluates a long-range 10 km preview horizon and updates its trajectory only once every 2 km, the embedded system has a large multi-second calculation window to finish its calculations. A 0.31 s (310 ms) average solve time is well within this generous execution window and presents no threat of hardware overruns or task slip. Consequently, given that the application layer is not strictly time-critical on a millisecond scale, the substantial 8.49% reduction in battery energy consumption achieved by full SQP easily justifies its higher computational cost. It can be concluded that for large-scale route profiles, accommodating the higher computational load of full SQP is the optimal choice to secure maximum driving efficiency.

6

Battery Thermal Tracking

This chapter investigates embedded battery thermal tracking for electric vehicles using a nonlinear model predictive control framework. The mathematical formulation adopted here is based on the battery thermal management problem presented in [40], where coupled thermal and electrical battery dynamics are used to regulate battery temperature subject to actuator and safety constraints. In the present work, the emphasis is on implementing and evaluating this formulation in an embedded setting using `acados`, with particular focus on real-time feasibility, plant approximation, and closed-loop tracking performance under disturbances.

6.1 Battery Thermal Tracking Problem Formulation

The battery thermal tracking objective is framed as a multi-variable MPC problem designed to regulate pack temperature while managing the system's energy storage state. In this formulation, the system state vector $\mathbf{x} \in \mathbb{R}^2$ concatenates the battery temperature T_{bat} and the state of charge SOC, while the control input vector $\mathbf{u} \in \mathbb{R}^2$ consists of the coolant pump speed ω and the actuator heating or cooling power Q_{heat}:

$$\mathbf{x} = \begin{bmatrix} T_{bat} \\ SOC \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} \omega \\ Q_{heat} \end{bmatrix} \tag{6.1}$$

The continuous-time system dynamics describe a tightly coupled electro-thermal network where the state evolution functions f_1 and f_2 dictate the temporal derivative of each state. The temperature rate of change $\dot{T}_{bat}$ is driven by internal Joule heating, modeled as a function of the exogenous battery current I_{bat} and internal resistance R_{bat}, as well as environmental convective losses and the heat transfer rate extracted by the cooling loop Q_{cool}:

$$\frac{dT_{bat}}{dt} = \frac{\alpha_0}{m_{bat} \cdot c_{bat}} \left(I_{bat}^2 \cdot R_{bat} - Q_{cool} + \gamma(T_{env} - T_{bat}) \right) \tag{6.2a}$$

$$\frac{dSOC}{dt} = \frac{-I_{bat}}{C_{bat}} \tag{6.2b}$$

The parameter α_0 represents an empirical thermal model scaling parameter, m_{bat} is the mass of the battery pack, c_{bat} corresponds to the specific heat capacity of the battery material, γ signifies the thermal transition coefficient for environmental losses, and T_{env} is the ambient environmental temperature. Conversely, thetime

derivative of the state of charge $S\dot{O}C$ depends strictly on the battery current I_{bat} and the total electrical capacity of the cells C_{bat}.
The fluid dynamics of the cooling loop dictate that the heat extraction rate Q_{cool} is an algebraic function of the coolant mass flow rate $\dot{m}_{cool}$, the specific heat capacity of the coolant fluid c_{cool}, and the temperature difference across the pack. Specifically, the coolant temperature at the inflow path $T_{cool,in}$ and the coolant temperature at the outflow path $T_{cool,out}$ are nonlinear algebraic equations dependent on the control inputs, the current battery temperature, and structural heat-exchange properties:

$$Q_{cool} = \dot{m}_{cool} c_{cool} (T_{cool,out} - T_{cool,in}) \tag{6.3a}$$

$$T_{cool,in} = T_{bat} + \alpha_1 \frac{1}{1 - \exp(-NTU_{bat})} \cdot \frac{Q_{heat}}{\dot{m}_{cool} c_{cool}} \tag{6.3b}$$

$$T_{cool,out} = (T_{cool,in} - T_{bat}) \cdot \alpha_2 \cdot \exp(-NTU_{bat}) + T_{bat} \tag{6.3c}$$

These structural parameters are defined by the dimensionless Number of Transfer Units (NTU_{bat}), which is a function of an empirical scaling coefficient α_3 and the product of the convective heat transfer coefficient and the effective surface area of the battery hA_{bat}, such that $NTU_{bat} = \alpha_3 \cdot \frac{hA_{bat}}{\dot{m}_{cool} c_{cool}}$. The algebraic relations are further scaled by empirical calibration coefficients α_1 and α_2 to accurately capture fluid-to-solid thermal transitions within the physical infrastructure.
To handle changing operational profiles and ensure recursive feasibility within a resource-constrained embedded environment, the control architecture utilizes a structured three-step optimization sequence. First, because the exogenous battery current I_{bat} continuously fluctuates and alters internal heat generation, a steady-state optimization problem is solved at each sampling interval to determine the nominal input vector $\mathbf{u}_{ss}$ and target state vector $\mathbf{x}_{ss}$ needed to maintain the temperature reference:

$$\min_{\mathbf{u}_{ss}} \quad \mathbf{u}_{ss}^T R \mathbf{u}_{ss} + (\mathbf{x} - \mathbf{x}_{ss})^T Q (\mathbf{x} - \mathbf{x}_{ss}) \tag{6.4a}$$

$$\text{s.t.:} \quad f(\mathbf{x}_{ss}, \mathbf{u}_{ss}, I_{b,N}) = 0 \tag{6.4b}$$

$$T_{cool,min}^{in} \leq T_{cool}^{in} \leq T_{cool,max}^{in} \tag{6.4c}$$

$$T_{cool,min}^{out} \leq T_{cool}^{out} \leq T_{cool,max}^{out} \tag{6.4d}$$

In this formulation, $\mathbf{u}_{\text{ss}}$ and $\mathbf{x}_{\text{ss}}$ represent the steady-state control input and state vectors, respectively, while $\mathbf{x}$ denotes the current state of the system. The matrices R and Q are positive-definite weighting matrices that penalize steady-state control effort and state deviations from the target. The continuous-time system dynamics are captured by the function $f(\cdot)$, which enforces the steady-state equilibrium condition under a normalized battery current baseline $I_{b,N}$. Finally, $T_{\text{cool}}^{\text{in}}$ and $T_{\text{cool}}^{\text{out}}$ represent the inlet and outlet coolant temperatures, which are bound by their physical operational limits.
This allocation step minimizes control effort while ensuring that the inputs are sufficient to prevent the optimizer from allowing the state to drift from its reference. As a second step, the nonlinear dynamics are locally linearized around these dynamically updated steady-state values to generate time-varying Jacobian matrices $A = \frac{\partial f}{\partial \mathbf{x}}|_{\mathbf{x}_{ss}, \mathbf{u}_{ss}, I_{b,N}}$ and $B = \frac{\partial f}{\partial \mathbf{u}}|_{\mathbf{x}_{ss}, \mathbf{u}_{ss}, I_{b,N}}$. These matrices parameterize a continuous Algebraic Riccati Equation (ARE), the numerical solution of which yields a

terminal cost-to-go matrix $V_f \leftarrow \mathrm{ARE}(A, B, Q, R)$ that serves as an infinite-horizon quadratic penalty at the end of the prediction horizon.
In the third and final step, the tracking task is evaluated over a finite prediction horizon N using a discrete time-step index k by solving the following constrained tracking MPC formulation:

$$\min_{\mathbf{x},\mathbf{u}} \quad \sum_{k=0}^{N-1} \Big((\mathbf{x}_k - \mathbf{x}_{ref})^T Q (\mathbf{x}_k - \mathbf{x}_{ref}) + (\mathbf{u}_k - \mathbf{u}_{ss})^T R (\mathbf{u}_k - \mathbf{u}_{ss}) \Big) + (\mathbf{x}_N - \mathbf{x}_{ss})^T V_f (\mathbf{x}_N - \mathbf{x}_{ss}) \tag{6.5a}$$

$$\text{s.t.:} \quad \dot{\mathbf{x}}_k = f(\mathbf{x}_k, \mathbf{u}_k, I_{b,k}) \tag{6.5b}$$

$$\mathbf{x}_0 = \mathbf{x}(0) \tag{6.5c}$$

$$\mathbf{x}_{min} \leq \mathbf{x}_k \leq \mathbf{x}_{max} \tag{6.5d}$$

$$\mathbf{u}_{min} \leq \mathbf{u}_k \leq \mathbf{u}_{max} \tag{6.5e}$$

$$T^{in}_{cool,min} \leq T^{in}_{cool,k} \leq T^{in}_{cool,max} \tag{6.5f}$$

$$T^{out}_{cool,min} \leq T^{out}_{cool,k} \leq T^{out}_{cool,max} \tag{6.5g}$$

The vector $\mathbf{x}_{\mathrm{ref}}$ is the target state reference trajectory, while $\mathbf{x}_0 = \mathbf{x}(0)$ serves as the initial state constraint, binding the optimization to the current measured or estimated state of the system. The terminal state error is penalized using the matrix V_f. The vectors $\mathbf{x}_{\mathrm{min}}$ and $\mathbf{x}_{\mathrm{max}}$ represent the lower and upper bounds enforcing physical and safety constraints on the system states (such as maximum allowable internal cell temperatures), while $\mathbf{u}_{\mathrm{min}}$ and $\mathbf{u}_{\mathrm{max}}$ define the hard limits on the control input actuators. The model parameters used in the implementation are summarized in Table 6.1

Table 6.1: Model parameters used for the battery thermal tracking problem.

Parameter	**Symbol**	**Value**
Environment temperature	T_{env}	293.15 K
Battery mass	m_{battery}	200 kg
Battery heat capacity	c_{battery}	795 J/(kg K)
Coolant heat capacity	c_{coolant}	3500 J/(kg K)
Coolant density	ρ_{coolant}	1050 kg/m^3
Pump displacement	V_{pump}	6.37×10^{-6} m^3/rad
Battery resistance	R_{battery}	1.024 Ω
Battery capacity	C_{battery}	100800 As
Battery heat-transfer term	hA_{bat}	2500
Identified coefficient	α_0	0.635039
Identified coefficient	α_1	0.915692
Identified coefficient	α_2	0.919681
Identified coefficient	α_3	1.47275
Ambient coupling coefficient	γ	7.38325
Pump-speed normalization	s_ω	100
Heating-power normalization	s_Q	1000
States	x	$[T_{\mathrm{bat}},\, SOC]^\top$
Inputs	u	$[\omega_{\mathrm{norm}},\, Q_{\mathrm{heat,norm}}]^\top$
Parameter	p	I_{bat}
Path constraints	$h(x,u)$	$[T_{\mathrm{cl,in}},\, T_{\mathrm{cl,out}}]^\top$

6.2 ACADOS Implementation

The battery thermal tracking problem was implemented in `acados` with a solver configuration chosen to remain lightweight enough for embedded execution. In contrast to fast mechanical systems, the thermal states evolve relatively slowly due to the large effective thermal mass of the battery and the coolant loop. This makes the problem well suited to a moderate sampling time and a computationally efficient NMPC formulation, provided that the dominant thermal dynamics and actuator limits are retained. The implemented model uses battery temperature and state of charge as states, normalized coolant-pump speed and heating/cooling power as inputs, and battery current as a time-varying parameter. The selected `acados` solver settings are listed in Table 6.2.

Table 6.2: `acados` solver settings for the battery thermal tracking problem.

Setting	**Value**
Prediction horizon	$N = 80$
Sampling time	$dt = 1\,\mathrm{s}$
Total horizon length	$t_f = 80\,\mathrm{s}$
State dimension	$n_x = 2$
Input dimension	$n_u = 2$
Parameter dimension	$n_p = 1$
Cost type	`LINEAR_LS`
Terminal cost type	`LINEAR_LS`
Stage state weight	$Q = \mathrm{diag}(10,\ 0.1)$
Stage input weight	$R = \mathrm{diag}(1,\ 1)$
Terminal weight	$Q_e = \mathrm{diag}(10,\ 0.1)$
NLP solver	`SQP_RTI`
QP solver	`FULL_CONDENSING_HPIPM`
Hessian approximation	`GAUSS_NEWTON`
Integrator type	`ERK`
Integration stages	4
Integration steps	1
Normalized pump-speed bounds	$\omega_{\mathrm{norm}} \in [0.157,\ 4.189]$
Normalized heating-power bounds	$Q_{\mathrm{heat,norm}} \in [-4,\ 4]$
Battery-temperature bounds	$T_{\mathrm{bat}} \in [253.15,\ 323.15]\,\mathrm{K}$
Path-constraint bounds	$T_{\mathrm{cl,in}},\ T_{\mathrm{cl,out}} \in [253.15,\ 323.15]\,\mathrm{K}$
Number of stage slacks	$n_s = 5$
Slack penalty	10^5

The selected solver settings were chosen primarily to reduce online computation. Since the cost is formulated in least-squares form, the `GAUSS_NEWTON` Hessian approximation is sufficient and avoids the overhead of exact second derivatives. The `SQP_RTI` scheme was used because it performs only one SQP step per sampling instant, which is well suited to slowly varying thermal dynamics and embedded execution. Because the state and input dimensions are small, `FULL_CONDENSING_HPIPM` remains computationally attractive for the present problem. Finally, the `ERK` integrator with four stages and one step provides adequate discretization accuracy

for the smooth thermal dynamics without introducing unnecessary computational cost. Soft constraints with a large slack penalty were retained to improve numerical robustness while still strongly discouraging constraint violation.

6.3 Experimental Setup and Embedded Plant Approximation

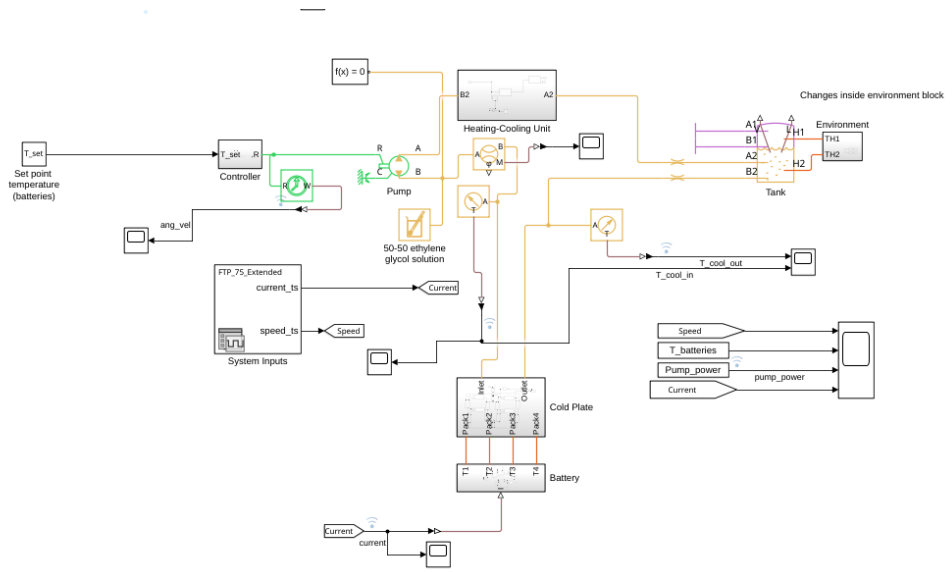


Figure 6.1: High-fidelity desktop Simulink/Simscape model used as the reference battery thermal plant during controller development.

Figure 6.1 shows the original desktop simulation setup of the battery thermal plant. This model is based on the MATLAB/Simscape battery thermal framework and serves as the high-fidelity reference environment during controller development. Because the plant is built using Simscape components, it is naturally simulated using a continuous-time variable-step solver in the desktop environment.
This creates a practical challenge for deployment on the dSPACE MicroAutoBox II. In an embedded real-time setting, both the controller and the plant approximation must run using a discrete fixed-step execution scheme. Such a formulation is required in order to ensure deterministic timing behavior, maintain synchronization between plant and controller execution, and satisfy the computation-time constraints imposed by the target hardware.
Two implementation paths were considered in order to address this mismatch between the desktop plant model and the embedded runtime environment. The first option was to retain the high-fidelity Simscape plant model on the desktop machine while deploying only the controller to the MicroAutoBox II. In that case, the plant states would be exchanged with the controller through CAN communication, and a sensor-emulation layer would provide sampled measurements to the controller at the controller update rate. This approach would resemble a hardware-in-the-loop

configuration without requiring a physical battery system. However, it would also introduce additional implementation complexity in the communication interface, and the measured execution time would include both solver time and communication overhead. Since this would significantly broaden the scope of the present thesis, this option was not pursued further.
The second option, which is the one adopted in this work, was to implement a simplified plant model together with the controller directly on the embedded target. The simplified plant is derived from the same governing equations used during the MPC design phase. The continuous-time dynamics are discretized using the forward Euler method with a sampling time of 1 s. This enables execution of the complete closed-loop setup on the MicroAutoBox II without relying on an external plant model or communication interface. The main disadvantage of this approach is that the measured runtime includes both controller execution and plant propagation. For this reason, the plant model was intentionally kept compact so that the timing results remain representative of a realistic embedded implementation.
To make the evaluation more demanding and to better reflect non-ideal operating conditions, disturbances were introduced into the simplified plant model. First, the battery-current input was treated as an external disturbance source. Since the available current profile has a finite duration of approximately 2400 s, the profile was repeated cyclically to allow continuous execution during ControlDesk measurements. Second, an additive disturbance term was introduced into the state update to emulate process uncertainty and model mismatch. This disturbance was not intended to represent a physically exact thermal uncertainty model; rather, it was used as a controlled stress-test mechanism for the embedded controller.
More specifically, the disturbance magnitude was varied between 0.1 and 0.4 times the battery-temperature signal in order to examine the robustness of the controller under increasingly challenging conditions. In addition to this disturbance sweep, both constant and time-varying temperature references were applied. This made it possible to assess not only nominal reference tracking, but also the sensitivity of the closed-loop system and the solver execution time under more aggressive tracking and disturbance scenarios.

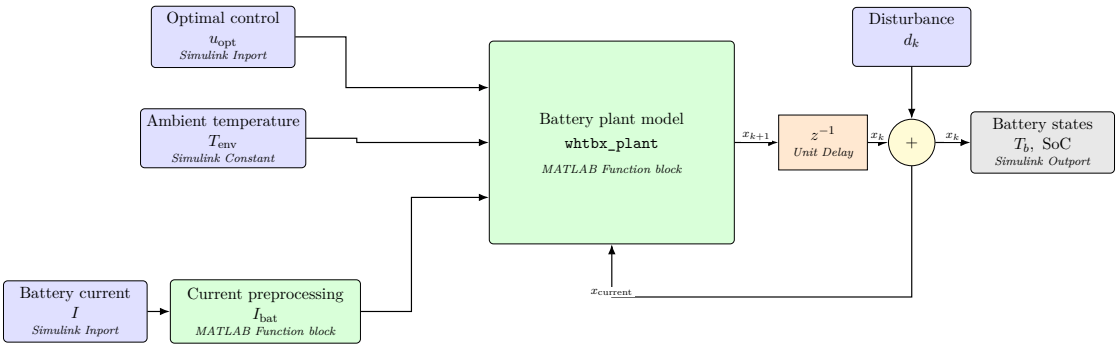


Figure 6.2: Discrete-time simplified battery thermal plant used for embedded closed-loop validation of thermal tracking.

Figure 6.2 shows the Simulink architecture of the simplified plant used for the embedded experiments. The controller output is applied to the plant together with the

ambient temperature and the processed battery-current input. The resulting state update is passed through a discrete delay block, an additive disturbance is injected, and the updated battery states are then fed back into the controller. This structure captures the essential feedback loop required for real-time MPC validation while remaining lightweight enough for execution on the target hardware.

A more accurate real-time plant approximation is left for future work. Ideally, the low-frequency thermal dynamics should continue to be represented using a compact first-principles model, while higher-frequency disturbance-driven effects could be captured using an identified model. In this context, an autoregressive model with exogenous input was considered, in which the current output is represented using past output values together with past input values. Such a formulation, combined with suitable filtering, is particularly attractive for capturing the fast variations induced by the battery-current profile. However, the identification process did not yet produce a sufficiently reliable model for embedded use. A validated identified plant model would therefore be a natural next step toward more realistic hardware-in-the-loop testing of the thermal MPC controller.

6.4 Simulation Results

The embedded battery thermal controller was evaluated in closed loop using the simplified discrete-time plant introduced in the previous section. The objective of the evaluation was twofold: first, to verify that the controller can regulate the battery temperature accurately under both constant and time-varying references; and second, to confirm that this performance can be achieved within the real-time timing constraints of the target hardware.

In all experiments, process disturbances were introduced through two mechanisms. The first was the battery-current input, which acts as an external disturbance source through the thermal model. The second was an additive noise disturbance applied directly in the plant update. This disturbance was used to stress-test the controller and assess its robustness to model mismatch and non-ideal operating conditions. For the results presented here, the disturbance magnitude was set to 0.1 relative to the battery-temperature signal.

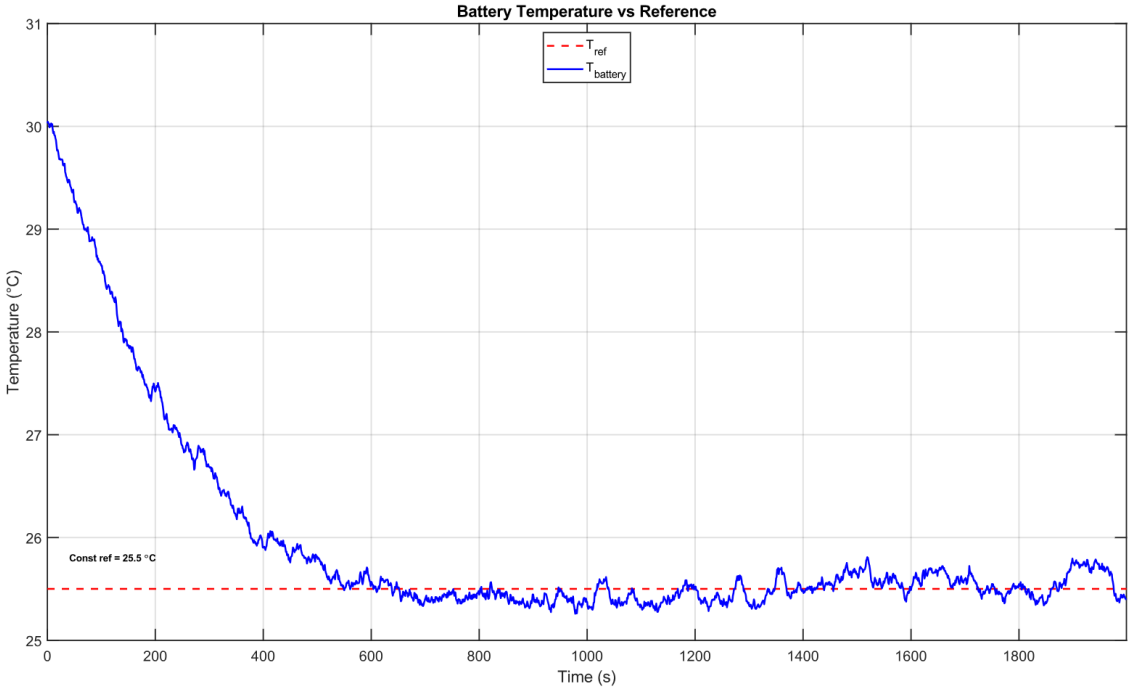


Figure 6.3: Battery-temperature tracking for a constant reference of 298.65 K (25.5°C) under additive disturbance

Fig. 6.3 shows the closed-loop battery-temperature response for a constant reference of 298.65 K, corresponding to 25.5°C. Since the reference is fixed during the experiment, this test primarily evaluates the disturbance-rejection capability of the controller. The controller maintains the battery temperature close to the desired set-point despite the imposed additive disturbance and the varying battery-current input. After the initial transient, the measured temperature remains centered around the reference with only small deviations. This indicates that the controller can reject persistent disturbances while maintaining stable thermal regulation.

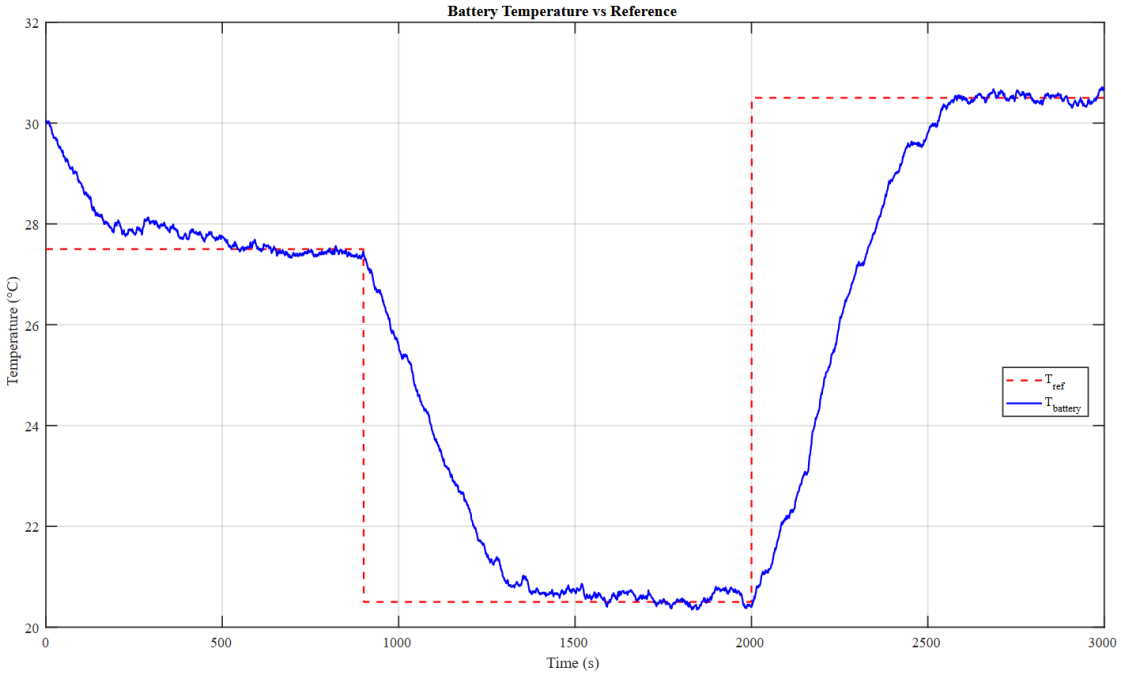


Figure 6.4: Battery-temperature tracking under a time-varying reference with additive disturbance

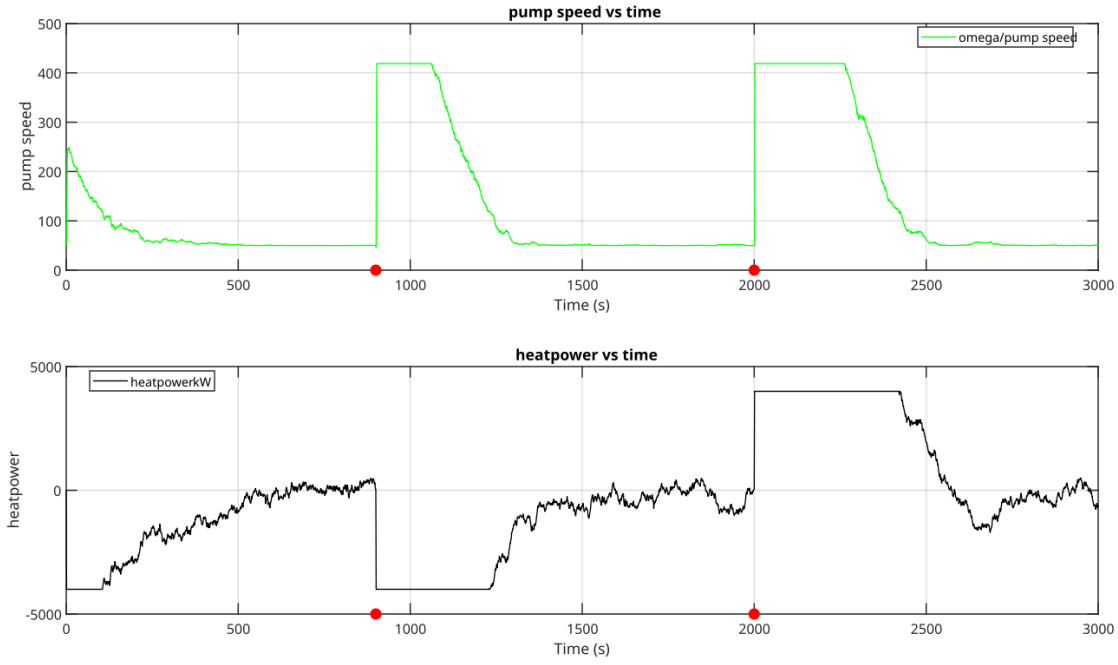


Figure 6.5: Control inputs under the time-varying temperature-reference test

Fig. 6.4 shows the controller response for a time-varying temperature-reference task. The reference temperature is initially set to 27.5°C, decreased to 20.5°C at $t = 900$ s, and then increased to 30.5°C at $t = 2000$ s. During the test, an additive disturbance with magnitude 0.1 is applied to the plant model.

The result shows that the controller tracks both reference transitions in a stable and well-damped manner. After each reference change, the battery temperature moves toward the new target without unstable oscillations or loss of regulation. The test therefore demonstrates that the controller is able to regulate the battery temperature both from a higher reference to a lower reference and from a lower reference to a higher reference, while rejecting the applied disturbance.
Fig. 6.5 shows the corresponding control inputs. When the temperature-tracking error is large, the inputs move toward their admissible limits. As the battery temperature approaches the reference, the inputs become smoother and more gradual. This behavior is consistent with the optimization-based control formulation, where the input action is adjusted according to the current tracking error and system constraints.
The input response also changes immediately after each reference transition. This fast response is a consequence of the simulation setup, since actuator and sensor dynamics are not explicitly included in the plant model. Therefore, the observed input response should be interpreted as the commanded control action rather than the delayed physical actuator response.
Together, the constant-reference and time-varying-reference tests show that the controller can provide both disturbance rejection and reference-tracking performance. The constant-reference case verifies regulation around a fixed operating point, while the time-varying-reference case verifies the ability to update the battery-temperature target during operation. These results support the suitability of the proposed control formulation for embedded battery thermal management. “‘
In addition to tracking performance, real-time feasibility was evaluated by monitoring the solver execution time on the MicroAutoBox II. For correct real-time operation, the solver execution time t_{solve} must satisfy

$$t_{\text{solve}} < T_s$$

at every update instant, where T_s is the controller sampling period. In the present implementation, the controller sampling period was set to $T_s = 1$ s.
During the closed-loop experiments, the average solver execution time was measured to be 0.42 s, while the maximum observed execution time remained below 0.60 s. This means that the controller consistently completed its optimization step within the available update interval, even during intervals with larger tracking error or stronger disturbance influence. The measured timing results therefore confirm that the thermal controller satisfies the computational requirements for real-time execution on the target hardware.

Table 6.3: Measured solver execution time during the battery thermal closed-loop experiment on the MicroAutoBox II.

Metric	Value
Controller update period, T_s	1.00 s
Average solver execution time	0.42 s
Maximum observed solver execution time	< 0.60 s
Maximum update-period utilization	$< 60\%$

Table 6.3 summarizes the observed runtime behavior. Since the maximum measured execution time remains well below the controller sampling period, the implementation preserves a meaningful timing margin for auxiliary software tasks such as I/O handling, communication, and supervisory logic. From an embedded deployment perspective, this is an important result: the controller not only performs the intended thermal tracking task, but does so within the timing limits of the MicroAutoBox II platform.

7
Conclusion

This thesis investigated embedded solution methodologies for nonlinear optimal control problems arising in electric vehicle energy management. The work focused on three application domains that represent important components of intelligent charge- and trip-planning: optimal charging, eco-driving, and battery thermal tracking. Although these problems differ in physical interpretation and timescale, they share a common mathematical structure in that they can be formulated as constrained nonlinear optimal control problems whose practical value depends strongly on whether they can be solved efficiently on embedded automotive hardware.

Two distinct methodological paradigms were studied. The first was a problem-specific approach based on Pontryagin's Maximum Principle (PMP), developed for the optimal charging problem. The second was a more general solver-based approach based on Sequential Quadratic Programming (SQP) implemented through the `acados` framework, which was applied to the charging, eco-driving, and battery thermal tracking problems. The central objective of the thesis was therefore to assess the feasibility of these methods under embedded implementation constraints, and to better understand the trade-off between analytical specialization and numerical flexibility in vehicle energy-management applications.

For the optimal charging problem, the PMP-based formulation showed that strong exploitation of the underlying problem structure can lead to a compact and computationally efficient runtime solver. By transforming the charging problem into a boundary value problem and combining the resulting structure with a neural-network-based initialization strategy, a deterministic runtime architecture was obtained. The solver was successfully deployed on the dSPACE MicroAutoBox II platform, where it achieved an execution time of approximately 30 ms. This result demonstrates that a carefully designed indirect method can satisfy real-time requirements for supervisory charging control and can be a highly attractive solution when the problem formulation is sufficiently fixed and analytically tractable.

The `acados`-based charging implementation showed that a direct SQP formulation can reproduce high-quality solutions with good numerical accuracy. When compared against the IPOPT benchmark, the full SQP configuration produced only minor deviations in battery temperature, state of charge, charging power, and final charging time. This confirms that the direct multiple-shooting formulation combined with embedded SQP can represent the charging problem accurately. At the same time, the comparison between full SQP and SQP_RTI showed that the real-time iteration scheme is not always the most appropriate choice. For the charging problem, SQP_RTI yielded more aggressive behavior and noticeably larger trajectory deviations, despite its lower computational cost. Since charging and battery thermal

dynamics evolve on relatively slow timescales, the additional computation required by full SQP is justified by the superior solution quality.

For the eco-driving problem, the `acados` framework enabled a comparison not only between solver types, but also between execution strategies. The results showed that a receding-horizon implementation with a fixed preview window can reproduce the performance of a one-shot full-route optimization with only marginal loss in energy efficiency and trip time, while being significantly more suitable for embedded deployment. This is an important outcome, since the one-shot strategy becomes increasingly difficult to realize as route length and problem dimension grow, whereas the receding-horizon formulation maintains bounded problem size and predictable computation. The solver comparison further showed that the relative merits of SQP and SQP_RTI depend strongly on the route characteristics. On shorter routes, SQP_RTI provided competitive and sometimes even favorable closed-loop behavior at substantially lower runtime. On longer routes, however, full SQP preserved the nonlinear model structure more accurately and yielded better energy efficiency. This demonstrates that solver selection in embedded optimal control must be made in relation to the application horizon, model structure, and performance objective, rather than according to a single universal criterion.

For the battery thermal tracking problem, the `acados`-based nonlinear model predictive controller successfully regulated battery temperature while respecting the coupled thermal and energy dynamics of the system. Even with a simplified discrete-time plant approximation and injected disturbances, the controller maintained the battery temperature close to its reference and completed its optimization steps within the real-time timing limits of the embedded platform. This shows that the proposed formulation is sufficiently robust for real-time closed-loop thermal management and supports the broader conclusion that embedded nonlinear optimization is not limited to offline planning problems, but can also be applied to dynamic thermal regulation tasks in automotive settings.

Taken together, the results of this thesis show that both investigated solver paradigms are viable for embedded implementation, but they serve different engineering purposes. The PMP-based method offers very high computational efficiency and deterministic behavior when the problem structure is sufficiently fixed and analytically exploitable. Its main limitation is reduced flexibility, since even moderate formulation changes may require substantial re-derivation and reimplementation. In contrast, the `acados`-based SQP framework provides a reusable and versatile workflow that can accommodate several nonlinear optimal control problems with relatively limited redesign effort. This flexibility comes at the cost of higher computational demand and a stronger dependence on solver tuning, transcription choices, and numerical conditioning. The main conclusion is therefore that there is no universally superior embedded solution methodology for vehicle energy management. Instead, the appropriate choice depends on the balance between computational constraints, required flexibility, model complexity, and the timescale of the application.

At the same time, the thesis also makes clear that the presented work should be viewed as a step toward a broader embedded optimization architecture rather than as a complete end solution. The three applications studied here were intentionally treated as separate subproblems in order to evaluate solver methodologies in

a controlled and interpretable manner. A natural continuation is therefore to integrate optimal charging, eco-driving, and battery thermal management into a unified predictive framework for charge- and trip-planning. Such an integrated formulation would more closely reflect the true system-level planning problem in battery electric vehicles, but it would also introduce stronger couplings, larger optimization problems, and more demanding embedded implementation challenges.

The present work also highlights the importance of improved plant-side validation. In particular, the battery thermal tracking study relied on a simplified discrete-time plant approximation in order to enable embedded closed-loop testing on the target hardware. While this was appropriate for demonstrating feasibility, future development should include higher-fidelity hardware-in-the-loop or vehicle-in-the-loop validation with more realistic plant models, communication interfaces, and actuator dynamics. Such extensions would make it possible to evaluate the robustness of the proposed methods under transport delays, sensor noise, model mismatch, and implementation effects that are closer to production conditions.

Another important continuation concerns the deployment target itself. The dSPACE MicroAutoBox II is highly useful as a rapid-prototyping platform, but production automotive ECUs impose much stricter limitations on memory, floating-point capability, scheduling, and software architecture. A transfer of the developed methods toward production-oriented hardware would therefore provide a more complete picture of their industrial applicability and would help identify which solver formulations remain feasible under tighter resource constraints. In parallel, further work is also warranted on systematic solver selection, particularly regarding the trade-off between full SQP and SQP_RTI across different applications, horizon lengths, and nonlinearities.

Finally, the thesis opens up interesting possibilities for combining model-based optimization with learning-based components. In the present work, learning was used only as an initialization mechanism for the PMP charging solver. However, future developments may incorporate learned model components, learned warm-starting strategies, or hybrid model-based and data-driven predictive-control formulations. Such directions are promising, but their value in an embedded automotive setting will depend on whether they can preserve the same level of numerical robustness, computational predictability, and constraint satisfaction that was central to this thesis.

In summary, this work has shown that embedded nonlinear optimization is a realistic and effective tool for electric vehicle energy management. By demonstrating successful implementation of both specialized and general-purpose optimal control methodologies on the dSPACE MicroAutoBox II platform, the thesis provides both a methodological comparison and a practical proof of feasibility. The results indicate that future intelligent charge- and trip-planning systems will benefit not from a single universal optimization strategy, but from a careful matching of solver architecture to problem structure, hardware limitations, and system-level objectives.

Bibliography

[1] European Commission, "European climate law." https://climate.ec.europa.eu/eu-action/european-climate-law_en, 2021. Accessed: May 2026.

[2] H. Ritchie, P. Rosado, and M. Roser, "Data page: CO2 emissions from transport." https://ourworldindata.org/grapher/co2-emissions-transport, 2023. Data adapted from Climate Watch. Accessed: May 2026.

[3] S. Sobczuk and A. Borucka, "Recent advances for the development of sustainable transport and their importance in case of global crises: A literature review," *Applied Sciences*, vol. 14, no. 22, p. 10653, 2024.

[4] N. Rauh, T. Franke, and J. F. Krems, "Understanding the impact of electric vehicle driving experience on range anxiety," *Human Factors*, vol. 57, no. 1, pp. 177–187, 2015.

[5] M. Ahmadi, N. Mithulananthan, and R. Sharma, "A review on topologies for fast charging stations for electric vehicles," in *2016 IEEE International Conference on Power System Technology (POWERCON)*, pp. 1–6, 2016.

[6] J. A. NeJame, "Examining the role of gender in electric vehicle interest and purchasing intention," Master's thesis, The Ohio State University, 2024.

[7] M. Redelbach, E. D. Özdemir, and H. E. Friedrich, "Optimizing battery sizes of plug-in hybrid and extended range electric vehicles for different user types," *Energy Policy*, vol. 73, pp. 158–168, 2014.

[8] J. Zhang, L. Yu, H. Zhang, X. Zhang, J. Zhu, D. Qian, A. Tu, and Z. Tang, "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)*, pp. 1–8, 2025.

[9] Y. Huang, E. C. Ng, J. L. Zhou, N. C. Surawski, E. F. Chan, and G. Hong, "Eco-driving technology for sustainable road transport: A review," *Renewable and Sustainable Energy Reviews*, vol. 93, pp. 596–609, 2018.

[10] W. Liljenström and M. Rothhämel, "Optimizing energy use in electric vehicles: A comparative study of optimization methods," in *rev2025–2nd Resource Efficient Vehicles, Graz, Austria, 23-25 September, 2025*, Universitätsbibliothek Graz, 2026.

[11] H. Tarout, H. Zaki, A. Chahbouni, E. Ennajih, and E. M. Louragli, "Optimizing energy consumption in electric vehicles: A systematic and bibliometric review of recent advances," *World Electric Vehicle Journal*, vol. 16, no. 10, 2025.

[12] P. K. Bhat, Z. Wu, and B. Chen, "Long trip charging planning of battery electric vehicle considering vehicle waiting time forecast at fast charging sta-

tions: A mixed-integer dynamic programming approach," *IEEE Access*, vol. 13, pp. 52100–52113, 2025.

[13] K. Maalej, S. Kelouwani, K. Agbossou, Y. Dubé, and N. Henao, "Long-trip optimal energy planning with online mass estimation for battery electric vehicles," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 11, pp. 4929–4941, 2015.

[14] L. Montalto, N. Murgovski, and J. Fredriksson, "Electrical vehicles charging: An optimal control approach via pontryagin's maximum principle," in *2025 IEEE 28th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 3198–3203, 2025.

[15] M. Roccotelli, G. Volpe, M. Fiore, M. Mongiello, A. M. Mangini, and M. A. del Cacho Estil-Les, "Optimizing trip planning of electric vehicles using deep reinforcement learning," *IEEE Transactions on Automation Science and Engineering*, vol. 23, pp. 3902–3915, 2026.

[16] N. Kumar, S. Batra, S. Vashistha, T. Rajesh, B. Nikitha, and A. G. Sai, "echarge: Deep reinforcement learning framework for multi-objective electric vehicle charging route planning," in *2025 IEEE 25th International Symposium on Cluster, Cloud and Internet Computing Workshops (CCGridW)*, pp. 34–41, 2025.

[17] Y. Wu, Y. Liu, J. Peng, Z. Chen, Y. Liu, and Y. Zhang, "Enhanced hierarchical eco-driving control for electric vehicles via global velocity optimization with road slope adaptation," *IEEE Transactions on Transportation Electrification*, vol. 11, no. 4, pp. 10322–10335, 2025.

[18] G. P. Padilla, S. Weiland, and M. C. F. Donkers, "A global optimal solution to the eco-driving problem," *IEEE Control Systems Letters*, vol. 2, no. 4, pp. 599–604, 2018.

[19] Y. Xu, P. Lokur, S. Klacar, S. George, A. Andersson, D. Sedarsky, and N. Murgovski, "Maximizing the energy-saving potential of declutchable bev powertrains via eco-driving," in *2024 IEEE International Conference on Electrical Systems for Aircraft, Railway, Ship Propulsion and Road Vehicles International Transportation Electrification Conference (ESARS-ITEC)*, pp. 1–6, 2024.

[20] M. R. Amini, H. Wang, X. Gong, D. Liao-McPherson, I. Kolmanovsky, and J. Sun, "Cabin and battery thermal management of connected and automated hevs for improved energy efficiency using hierarchical model predictive control," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 5, pp. 1711–1726, 2020.

[21] Z. Wang, K. Li, T. Dinh, and J. Chong, "Adaptive model predictive control for battery ther-mal management for electric vehicles," in *2023 26th International Conference on Mechatronics Technology (ICMT)*, pp. 1–4, 2023.

[22] M. Diehl, H. G. Bock, and J. P. Schlöder, "A real-time iteration scheme for nonlinear optimization in optimal feedback control," *SIAM Journal on control and optimization*, vol. 43, no. 5, pp. 1714–1736, 2005.

[23] J. Zwiener, J. Stephan, and C. Seiferth, "Real-time nonlinear model predictive control of large multirotor aircraft," *IEEE Transactions on Control Systems Technology*, vol. 34, no. 3, pp. 1665–1672, 2026.

[24] J. P. Allamaa, P. Listov, H. Van Der Auweraer, C. Jones, and T. D. Son, "Real-time nonlinear mpc strategy with full vehicle validation for autonomous driving," in *2022 American Control Conference (ACC)*, pp. 1982–1987, 2022.

[25] J.-W. Kim, M.-W. Gwon, J. Heo, and K.-K. K. Kim, "Real-time optimal energy management of dual-motor battery electric vehicles using mpc on an automotive-grade microcontroller," in *2025 IEEE 21st International Conference on Automation Science and Engineering (CASE)*, pp. 1915–1920, 2025.

[26] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, "acados – a modular open-source framework for fast embedded optimal control," *Mathematical Programming Computation*, 2021.

[27] T. Salzmann, J. Arrizabalaga, J. Andersson, M. Pavone, and M. Ryll, "Learning for casadi: Data-driven models in numerical optimization," 2023.

[28] A. Lahr, J. Näf, K. P. Wabersich, J. Frey, P. Siehl, A. Carron, M. Diehl, and M. N. Zeilinger, "L4acados: Learning-based models for acados, applied to gaussian process-based predictive control," 2025.

[29] L. Montalto, *Optimal control methods in charge-and trip-planning for electric vehicles*. PhD thesis, Chalmers University of Technology, 2026.

[30] H. G. Bock and K. J. Plitt, "A multiple shooting algorithm for direct solution of optimal control problems," *IFAC Proceedings Volumes*, vol. 17, no. 2, pp. 1603–1608, 1984.

[31] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2 ed., 2017.

[32] J. T. Betts, *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. SIAM, 2 ed., 2010.

[33] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer, 2 ed., 2006.

[34] G. Frison and M. Diehl, "Hpipm: a high-performance quadratic programming framework for model predictive control," 2020.

[35] G. Frison, D. Kouzoupis, T. Sartor, A. Zanelli, and M. Diehl, "Blasfeo: Basic linear algebra subroutines for embedded optimization," *ACM Transactions on Mathematical Software*, vol. 44, no. 4, p. 1–30, 2018.

[36] T. A. Johansen, "Toward dependable embedded model predictive control," *IEEE Systems Journal*, vol. 11, no. 2, pp. 1208–1219, 2017.

[37] S. D. Cairano and I. V. Kolmanovsky, "Real-time optimization and model predictive control for aerospace and automotive applications," in *Proc. American Control Conf.*, pp. 2392–2409, 2018.

[38] G. Cimini, D. Bernardini, S. Levijoki, and A. Bemporad, "Embedded model predictive control with certified real-time optimization for synchronous motors," *IEEE Transactions on Control Systems Technology*, vol. 29, pp. 893–900, Mar. 2021.

[39] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[40] L. Kula and D. McCauley, "Learning-enhanced non-linear model predictive control for battery thermal management systems," Master's thesis, Chalmers University of Technology, 2026.

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

CHALMERS
UNIVERSITY OF TECHNOLOGY