Smart Control of Water Heaters Coupled to Thermal Energy Storage

June 13th, 2025

Julian Vries S4964055

Bachelor Integration Project

Faculty of Science and Engineering

University of Groningen

The Netherlands

Supervisors: prof. dr. Michele Cucuzzella, Edoardo Vacchini, PhD Sebastian Trip

Abstract

This thesis explores the implementation of a control strategy that controls an electric boiler. The controller needs to react to dynamic electricity prices and imbalance prices. Historical data from the year 2024 is used in a simulation to see the of cost-savings profits that can be made by using such control strategy.

The increasing share of variable renewable electricity in Europe requires electricity consumers to be more flexible. This study evaluates whether a 100 kW electric boiler and 10 m³ thermal-storage tank on a dairy farm can be operated economically through price-responsive control. Two strategies are implemented in Python: a price-weighted linear–quadratic regulator (LQR) which uses dynamic electricity prices to determine the boiler power output. And a Model Predictive Controller (MPC) which uses both dynamic electricity prices and imbalance prices to reduce costs as much as possible. Both strategies are tested with 2024 Dutch day-ahead and imbalance-price data under realistic thermal constraints.

Findings indicate that predictive, price-driven control of electric boilers might be technically feasible and financially attractive for industrial applications, particularly when imbalance price signals are incorporated.

Contents

1 Introduction

The share of renewable energy in Europe is growing. As a result of commitments to the energy transition and climate change, this growth is expected to continue to meet the 2050 EU goals regarding CO_2 emissions [1], [2]. An increased share of renewable energy will introduce fluctuations in electricity supply and demand, due to the varying nature of the weather conditions [3]. Batteries, electric water heaters, and hot water tanks can be used to mitigate fluctuations in supply and demand by acting as intermediate buffers. By storing energy in case of high supply and deploying energy in case of high demand, the power grid will become more stable, while also increasing energy efficiency and cost-effectiveness [4]. However, existing strategies to mitigate grid instability require changes in behaviour patterns of users of the electricity grid [5]. This research aims to develop a control strategy for optimal coordination between batteries, water heaters, and thermal energy storage (TES). It aims to improve energy efficiency, reduce costs, and support grid stability.

Recent studies have explored control strategies for energy storage and demand-side management. Rapucha et al. (2024) explore smart heat pump control in residential energy optimization, while Clift et al. (2021) examine control strategies for photovoltaic (PV) powered energy storage systems [5] [6]. Zhang et al. (2025) propose an integrated approach to optimize the operation of the battery and heat pump with TES [4]. Despite these advancements, there is limited research on the operation of battery and heat pump systems in combination with TES to balance the electricity grid. Due to the performance reduction during the startup phase that heat pumps experience, FIRN energy wants to explore the use of e-boilers instead of heat pumps, given their faster response time [7]. By using dynamic electricity contracts and participating in the electricity imbalance market, grid stability could be improved, and profits can be made [8]–[10].

This research fills the gap in the current research by developing a control strategy that coordinates an electric boiler coupled with thermal energy storage as a flexible asset. Two different control strategies will be explored: a Linear Quadratic Regulator and a Model Predictive Controller. The controller has to schedule heating and discharging of the TES so that hot-water production is conducted in the lowest-cost hours in the day-ahead market, or in times of favourable imbalance prices. If the 15-min imbalance price is negative, the farm consumes more than scheduled, which means the boiler soaks heat, and the transmission system operator pays the negative price for every extra kWh. When the price is positive, the boiler will turn off, selling the kWh saved at the imbalance rate. Only this deviation is settled at the imbalance price, so the day-ahead contract remains unchanged [9], [10]. Fast boiler response and thermal storage make the strategy feasible without disrupting the fixed cleaning schedule.

The specific use case studied is a dairy farm that cleans their milking robots and the livestock stables. The robots are rinsed every twelve hours with hot water. Additionally, stable cleaning is carried out once a week on Monday morning, and a deep clean is performed on the first Saturday of each month. This use case provides a strong indication of potential cost reductions by comparing the boiler controlled by prices to an boiler that is solely controlled based on temperature.

The rest of this paper will be organized as follows: First, sections 2 and 3 present the problem statement and research questions. In section 4, the gaps in the current knowledge are identified to help indicate what exactly needs to be researched. Next, in section 6, all necessary materials are listed, as well as the methods and validation for conducting the research. The system description in section 5 explains the system that needs to be controlled and the corresponding equations. Next in section 7 the design choices and the simulated use case for the simulation model of the boiler controller are explained. Simulation setup is presented in section 8. The results from the simulations are explained in section 9. Finally, the discussion and conclusion are formulated in sections 10 and 11.

2 Problem Statement

Heat Pumps / E-boilers, and Thermal Energy Storage (TES) must be effectively coordinated to improve energy efficiency, lower costs by at least 30%, and improve grid stability by balancing electricity supply and demand in response to fluctuations in renewable energy production and dynamic pricing.

3 Research Questions

Central Question
What is the most optimal way to control an e-boiler or heat pump in combination with a Thermal Energy Storage (TES) to reduce costs, improve energy efficiency and support grid stability in the BeNeLux climate?

Knowledge Questions

1. What are the main challenges for the integration of E-boilers / Heat Pumps, BESS and TES for energy management?
2. What control strategies have been explored in past research?
3. Which type of water heater is most suitable for a combined system, a heat pump, or an e-boiler?

Design Questions

1. Which decision-making algorithms are the most suitable for controlling combined systems?
2. What are the requirements for integrating the control strategy with FIRN Energy's current software and hardware?

Validation Questions

1. To what extent does the control technique enhance energy efficiency, cost savings, and grid stability under real-world conditions compared to existing solutions?
2. Does the Python script meet the requirements and operational constraints of FIRN Energy's existing software and hardware?

4 Literature Review

Integrating electric boilers with thermal energy storage (TES) offers a promising solution for improving flexibility, efficiency and improving grid stability. This literature review focuses specifically on the coordination of these two components. The review is structured as follows: it starts with an overview of the current integration strategies for boilers and TES, followed by recent developments in forecasting methods relevant to system optimization. Next it discusses how grid services and market policies affect boiler-TES operation. A critical comparison of control strategies is then presented, concluding with an assessment of gaps in the current research.

The share of renewable energy in Europe is growing. As a result of commitments to the energy and the energy transition, this growth is expected to continue until the 2050 EU emissions goals are satisfied [1], [2]. Due to the varying nature of weather conditions, more fluctuations will be introduced into the energy supply and demand [3]. These fluctuations can introduce more instability in the electricity grid.

The integration of electric boilers and TES is critical to making energy systems as efficient as possible while also contributing to a more stable grid. Coordinating these components remains challenging. Heat pumps are known to be highly efficient for heating and cooling. While electric boilers convert electric energy into heat, heat pumps transfer energy rather than converting it [7]. The heat pump extracts energy from a source such as surrounding air, and then amplifies and moves it to the place where it is needed [7]. A major disadvantage that comes with using a heat pump is the significant reduction in performance during its start up phase [11]. Therefore, heat pumps are not suitable for use cases where they need to be turned on and off repeatedly. Additionally, there is minimal research on how to optimize the use of heat pumps in combination with energy storage devices [5]. TES allows for flexibility by storing thermal energy during low-demand periods for later use, as was indicated by Zhang et al. (2025)[4]. This allows users to use this thermal energy in times of high demand, or in periods of high energy prices.

Effective coordination of water heaters and thermal energy storage could help for improving grid stability and reaching net-zero emissions, however there is still a gap in the research on integrated system optimization. Rapucha et al. (2024) emphasized the relevance of smart control strategies for managing electrical loads with heat pumps, but the integration with storage systems was not thoroughly investigated [5]. Laugs et al. (2019) highlight the need for better coordination between renewable energy sources, storage, and grid interactions. Clift and Suehrcke (2021) optimized control systems for PV-powered storage and heat pump water heaters, but did not test a system that utilizes the heat pump or boiler to minimize cost [6].

On the Dutch energy market, there are multiple types of energy contracts. One of these contract types is a dynamic energy contract. In this type of contract, the prices of electricity are determined one day ahead for 60 minute intervals. Prices can fluctuate based on supply and demand, which means they sometimes become negative [12]. Dynamic electricity contracts stimulate demand-side management. By strategically planning electricity consumption based on the day-ahead prices, costs can be minimized, and in some cases get paid for using electricity. Another benefit of using the dynamic prices as a trigger is that a consumer will contribute to grid stability by consuming or not consuming electricity based on the electricity price. By consuming in times of high production and not consuming during times of shortage, grid stability will be improved [8].

Strategically consuming electricity, and participating in the electricity imbalance market, allows for further cost reduction. Once day-ahead positions are fixed, the transmission system operator publishes a 15-minute imbalance price that rewards deviations from those positions that support the system balance. When there is excess production, this price often turns negative. By temporarily increasing electricity consumption, a consumer is paid the negative tariff for every extra kilowatt-hour they consume. Conversely, during shortage intervals the price rises well above the day-ahead level. Reducing consumption below the scheduled baseline yields a reward at this higher rate. Because only the deviation is settled at the imbalance price, the underlying day-ahead contract is unaffected [9], [10]. However, at this point in time there is a high entry barrier for participation in the imbalance market. The potential of improving grid stability would be significantly higher, if the regulations regarding entry on the imbalance market would be less restrictive [13].

Imbalance prices are highly dependent on factors such as weather conditions and electricity consumption. Therefore it is highly important to make a forecast for the imbalance prices if the objective is to make as much profit as possible. Deng et al. (2024) highlights the importance of creating accurate forecasts for imbalance prices in the UK. The only difference of the UK imbalance market compared to the Dutch imbalance market is that the UK imbalance market has 30 minute settlement periods and the Dutch market has 15 minute settlement periods [14], [15].

In the past Linear Quadratic Regulators have been used to control the induction heating of steel billets [16]. It was used to regulate the core-temperature of the billets from 1000 °C to 1200 °C under different inputs and weight, as well as different disturbances [16]. The dynamics of the system were transferred to state-space to be able to control them using LQR. A similar type of LQR could be applied to control the temperature of a Thermal Energy Storage that is heated by an electric boiler. Another type of controller that has been explored to control a boiler is Model Predictive control. Zlatkovikj et al. (2022) designed a feed forward MPC that controlled a bio-mass boiler. They found that an MPC controller was well suited for controlling this type of boiler due to the ability to find the optimal trajectory over a future time horizon [17]. This means that the boiler can look ahead and see when it needs to heat the water. However, Zlatkovikj et al. (2022) did not test the MPC controller on an electric boiler.

Comparing current research reveals trade-offs. Studies like Clift and Suehrcke (2021) adopt rule based controls for simplicity but lack responsiveness to dynamic price signals [6]. Such methods usually neglect real-time data inputs and fail to optimize over time horizons. In contrast, model predictive controllers as discussed by Zlatkovikj et al. (2022) offer better performance, but require more computational power and accurate forecasts [17]. Malecki et al. (2022) focusses on achieving detailed modelling, but does not take into account responses to electricity prices or grid interactions [11]. Furthermore, there is limited research on operating electric boilers in combination with TES while responding to dynamic pricing and imbalance market signals.

By addressing gaps in previous research and implementing a strategy that tries to minimize cost through strategic consumption, this research will investigate whether such an approach will lead to a reduction in cost and contribute to a more stable electricity grid.

5 Use Case and System Description

The use case is based on a dairy farm that has a sustainable energy system with different components that are controlled the FIRN energy controller. The farm has a PV installation, a Battery Energy Storage System, a water heater and a Thermal Energy Storage. This research focusses on optimizing the usage of the water heater and thermal energy storage of this farm. The rest of the components of the farms energy system were not modelled to reduce model complexity.

The farm requires large volumes of hot water for cleaning purposes, such as sanitizing milking robots and stables. The farm has a 10 m^3 hot water storage tank, that is being heated by a 100 kW water heater. It was assumed that all the water that is drawn from the tank is immediately replaced with new water. Therefore, the water level in the tank will be exactly the same at all times. The inflowing water was assumed to be ground water with a temperature of 10 degrees celsius [18].

A cleaning schedule was designed to simulate that hot water was being drawn from the tank. Every day at 7:00 and at 19:00 the milking area would be cleaned with a water flow rate of 20 L/min for 30 minutes. Every week on Saturday, the stable is cleaned for 120 minutes with a flow rate of 30 L/min. Finally, once a month the stable is deep cleaned for 180 minutes at a flow rate of 40 L/min. This cleaning schedule with the different flow rates and durations of cleaning causes fluctuations in the temperature of the water tank. The control system is tasked with maintaining a desired water temperature while minimizing costs by utilizing electricity pricing signals from both the day-ahead market and the imbalance market.

Figure 1 (Appendix A) displays the relations of all components in the system. In addition, the input and output signals and energy flows are connected to the corresponding component. To model each component, equations are needed to simulate the behaviour of these components. Heat pumps have a significantly longer start-up time compared to E-boilers [11]. Being able to quickly ramp up to maximum power is a key characteristic when needing to react to price signals from both the spot market and the imbalance market. Therefore, an E-boiler was chosen to evaluate whether the smart control system can reduce cost by participating on the dynamic and imbalance market. The focus on the E-boiler and TES means that only the equations corresponding to these components will be modelled.

E-Boiler

The electric boiler is modelled by multiplying the input power with the efficiency coefficient:

$$Q_{\text{boiler}}(t) = \eta_{\text{boiler}} P_{\text{boiler}}(t) \tag{1}$$

Variable Explanations:

- $Q_{\text{boiler}}(t)$ **[W]**: Thermal power output of the electric boiler.
- η_{boiler} **[–]**: Efficiency coefficient (usually almost 1).
- $P_{\text{boiler}}(t)$ **[W]**: Electrical power input to the boiler.

Thermal Energy Storage (TES)

For a well-mixed storage (e.g., water tank), the energy balance is modelled by adjusting the energy balance equation with the input from the boiler, output and loss coefficients [19]:

$$m_{\text{TES}}\, c_p \frac{dT_{\text{TES}}(t)}{dt} = Q_{\text{in,TES}}(t) - Q_{\text{use}}(t) - hA\left(T_{\text{TES}}(t) - T_{\text{env}}\right) \tag{2}$$

Variable Explanations:

– $T_{\text{TES}}(t)$ **[°C]**: Temperature of the thermal storage.

– m_{TES} **[kg]**: Mass of the storage medium (e.g., water).

– c_p **[J/(kg·K)]**: Specific heat capacity.

– $Q_{\text{in,TES}}(t)$ **[W]**: Thermal power input to the TES (from the heat pump or boiler).

– $Q_{\text{use}}(t)$ **[W]**: Thermal power extracted for use.

– hA **[W/K]**: Overall heat transfer coefficient that represents losses.

– T_{env} **[°C]**: Ambient temperature.

Justification of Design Choices

- **10 m^3 tank:** Chosen to support multiple cleaning cycles and buffer heating during low electricity prices.
- **100 kW boiler:** Allows for fast recovery and cost-efficient operation in response to market signals.
- **No battery and PV:** Excluded for simplicity and FIRN energy prioritized the boiler-TES system.
- **Perfect mixing:** Assumed to simplify modelling of thermal dynamics.
- **Ramp rate:** Boiler is assumed to ramp instantaneously within 60-second time steps.
- **Time step:** $\Delta t = 60$ seconds selected for sufficient temporal resolution.

This setup provides the basis for evaluating the performance of smart control algorithms that optimize the operation of the E-boiler and TES in response to dynamic and imbalance electricity prices.

6 Materials and Methods

6.1 Materials

- Python (via PyCharm IDE)
- E-boiler
- Thermal Energy Storage
- Model of each component (E-boiler, TES), see section 5
- Dynamic electricity price data (day-ahead and imbalance)

- Ambient temperature data

Software environment

- Python 3.11
- packages: Numpy, CasADI, Matplotlib, SciPy
- LQR solver: Discrete-time Riccati solver using numpy
- MPC solver: CasADI with direct collocation and IPOPT

Hardware Specifications

- Intel Core i5-10600KF CPU
- 16GB 3600 MHz RAM
- 1 TB NVMe SSD
- Intel Arc B580 GPU
- Z490 motherboard

6.2 Methods and Validation

The research started with collecting data from various sources including dynamic electricity prices, Imbalance prices, and ambient temperature data. This data will be cleaned and prepared for use in models. Each component of the system (E-boiler, TES) will be modelled based on their real-world equivalent using Python. Key parameters such as thermal capacity, energy capacity and charging rates will be assigned to each of the components.

Each component was modelled in Python to reflect real-world behaviour. Parameters such as heat capacity, energy content, and charging and discharging dynamics were assigned based on physical standards and values found in literature.

Two control strategies were implemented: A Linear Quadratic Regulator (LQR) Discrete-time LQR controller designed using state-space representation. Which uses a Riccati equation solved via NumPy [20], [21]. And a Model Predictive Controller (MPC) which was implemented in CasADi with direct collocation, IPOPT solver, and rolling horizon framework [22].

Both controllers were designed to respond to day-ahead and imbalance price signals, and thermal demand. Simulation experiments tested performance under different ambient temperatures and pricing scenarios. The Key Performance Indicators used to assess the performance of the controllers were electricity consumption in kWh and cost savings in €.

Simulations were conducted under a baseline configurations of both controllers (flat rate) and compared with both LQR and MPC strategies which used dynamic and imbalance price signals. Performance was benchmarked over one-month periods with a set load schedule.

All data used in for this simulation is publicly available. No personal or sensitive information was processed.

7 Modelling

This section presents the detailed implementation of a control system designed to manage the heating operations of an E-boiler in combination with a Thermal Energy Storage (TES). By using the equations from section 5, the different components are modelled in python. Event-Driven Programming will be used to model the complete system. By using this type of programming, the different components of the system can react to certain triggers such as a signal telling the boiler to turn on or off [23].

For the LQR controller Ricatti was used to track the water temperature and adjust the boilers power [20]. Ricatti LQR updates every minute to adjust the boilers power. However, due to the lack of hard constraints the LQR controller may be less suited for this type of control. For the MPC contoller, a framework from CasADI was used. CasADi is a symbolic framework for automatic differentiation and numerical optimization[22]. It controls the boiler based on both spot prices and imbalance prices, with a reference temperature that changes based on those prices.

Parameter Table

Component	Parameter	Value	Unit
TES	Volume	10	m^3
TES	Mixing	Perfect	Assumption
TES	m_{TES}	10,000	kg
TES	c_p	4180	J/kg·K
TES	T_{env}	10	°C
TES	T_{init}	60	°C
TES	T_{set}	60	°C
TES	hA (loss coeff.)	152.2	W/K
TES	T_{in}	10	°C
Boiler	Efficiency (η)	1	–
Boiler	$P_{\max}$	100	kW
Time step	Δt	60	seconds

Table 1: Parameters for boiler and TES configuration.

Parameter Sources and Assumptions
All parameter values were selected based on industry practice or literature references. For example, the heat loss coefficient $hA = 79.5$ W/K was calculated by using the surface of the tank k average value for w/m2/k given by Allan et al. is 3.08 W/m2/K and multiply it by the tanks surface area of approximately 25.8m2 [24]. The tanks surface area was calculated as follows: $V = \pi r^2 h$, $h = 2r$ for 1:1 height-width ratio, which gives 10 = $2\pi r^3 = 10$, $r^3 = 10/2\pi = 1.59$, $r = \sqrt[3]{1.59} = 1.17m$, $h = 2r = 2.34m$, $A = 2\pi r^2 + 2\pi rh = 2\pi(1.17)^2 + 2\pi(1.17)(2.34) = 25.8m^2$. Boiler efficiency was assumed to be 1 for resistive electric heating. Inflowing water (T_{in}) was assumed to be 10 °C to reflect the temperature of groundwater in the Netherlands[18].

7.1 Price-Weighted Linear Quadratic Regulator Design

To allow the application of control methods, the model presented in Section 5 is rewritten in state-space form. With T as the state variable and $u = q_{in}$ as the input, the system can be expressed as:

$$\dot{T} = aT + bu + d(t), \tag{3}$$

where:

$$a = -\frac{hA}{mc_p},$$
$$b = \frac{1}{mc_p},$$
$$d(t) = \frac{hAT_{env}(t) - q_{use}(t)}{mc_p}.$$

The continuous model is discretized using the Forward Euler method with a timestep $\Delta t = 60$ seconds:

$$T_{k+1} = T_k + \Delta t(aT_k + bu_k + d_k). \tag{4}$$

The matrices used in the discrete model are:

$$A = 1 + -\frac{hA\Delta t}{mc_p}$$
$$B = \frac{\Delta t}{mc_p}.$$

LQR was chosen for its ability to produce a state-feedback law that balances temperature regulation against energy cost. A deviation variable is introduced to allow tracking of a dynamic temperature setpoint:

$$x_k = T_k - T_{sp}(k), \tag{5}$$

where $T_{sp}(k)$ is a time-varying set-point defined as:

$$T_{sp}(k) = \begin{cases} T_{soak}, & \text{if } p_k \leq p_{th}, \\ T_{ref}, & \text{otherwise}, \end{cases} \tag{6}$$

with T_{soak} set to 80°C and T_{ref} to 60°C. The deviation dynamics are:

$$x_{k+1} = Ax_k + Bu_k + w_k, \tag{7}$$

where $w_k = \Delta T_{sp}(k) + \Delta t d_k$ is treated as a known disturbance.

The LQR controller minimizes a quadratic cost function:

$$J = \sum_{k=0}^{N-1} \left(x_k^\top Q x_k + u_k^\top R_k u_k \right) + x_N^\top Q_f x_N, \tag{8}$$

where:

- Q penalizes temperature deviations,
- $R_k = R_0 + \alpha p_k$ penalizes energy use, scaled by the real-time electricity price p_k,
- $Q_f = Q$ for indefinite horizon approximation.

For a single-state system, the Discrete Algebraic Riccati Equation (DARE) simplifies to a scalar form. Using SciPy's `solve_discrete_are`, the optimal feedback gain K_k is computed as:

$$K_k = \frac{BAP}{R_k + B^2 P}, \tag{9}$$

with P being the solution of the DARE [21].

To respect physical boiler constraints, the control input is contrained to make sure it does not exceed the maximum power of the boiler:

$$u_k = \min\left(\max(0, -K_k x_k), P_{\max}\right), \tag{10}$$

and additionally zeroed out if $T_k \geq T_{sp}(k)$.

7.2 Model Predictive Controller Design

To efficiently control the boiler, a two-layer control approach was chosen. The first layer consists of a baseline MPC controller that determines a minimum-cost control sequence based solely on day-ahead prices and system constraints. The second layer decides whether the system should deviate from this baseline by introducing imbalance market considerations through a deviation-based MPC.

Both solvers share a common state-space model of the tank's thermal dynamics, discretized using the Forward Euler method. At each time step, the evolution of the water temperature is computed as:

$$T_{k+1} = T_k + \frac{\Delta t}{mc_p}\left[u_k - hA\left(T_k - T_{\text{amb}}\right) - Q_{\text{draw},k}\right] \tag{11}$$

where T_k is the current tank temperature, u_k is the boiler power, T_{amb} is the ambient temperature, and hA is the overall heat-loss coefficient.

Baseline Solver

The baseline controller computes a reference heating trajectory $\{u_{\text{ref},k}\}_{k=0}^{N-1}$ over a prediction horizon of $N = 15$ minutes by minimizing the cost:

$$J = \sum_{k=0}^{N-1} Q_{\text{wt}}\left(T_k - T_{\text{sp},k}\right)^2 + R_k\, u_k^2 \tag{12}$$

where $T_{\text{sp},k}$ is the dynamic set-point (either comfort or soak temperature), and R_k is a price-weighted regularization term that penalizes energy use depending on the electricity price:

$$R_k = R_0 + \alpha p_k$$

This allows the controller to encourage higher energy use during low or negative price periods (e.g., when prices fall below -0.01 EUR/kWh) and reduce consumption during expensive hours.

Imbalance Solver

The second MPC layer extends the baseline by explicitly modelling deviation penalties or rewards based on real-time imbalance prices. In this formulation, the cost function is augmented with a deviation term:

$$J = \sum_{k=0}^{N-1} \left[Q_{\text{wt}}\left(T_k - T_{\text{sp},k}\right)^2 + R_k\, u_k^2 - W\, p_{\text{imb},k}\left(u_k - u_{\text{ref},k}\right)\frac{\Delta t}{3600}\right] \tag{13}$$

Here, $p_{\text{imb},k}$ denotes the imbalance price at time k, $u_{\text{ref},k}$ is the reference control from the baseline, and W is a weight parameter governing sensitivity to imbalance deviations. The deviation term stimulates positive deviation when imbalance prices are negative which indicate a power surplus, and discourages unnecessary consumption during periods of shortage.

The weight values Q and R for both the LQR and MPC controllers were manually tuned using trial and error to identify combinations that maintained thermal comfort while responding to price volatility. The final weights were selected to minimize total cost over a time period of one month.

8 Simulation

The simulation for both types of controllers begins with the loading and pre-processing of time series data. These include hourly spot-market electricity prices, daily average ambient temperatures, and 15-minute imbalance prices. All datasets are interpolated and resampled to 1-minute resolution to match the simulation time-step ($\Delta t = 60$ s). Missing data is forward-filled to ensure a complete dataset.

Electricity prices are retrieved from a Dutch dynamic tariff dataset and subsequently scaled by a factor of 1.21 to account for value-added tax (VAT) [25]. Daily average ambient temperatures are interpolated to represent typical weather conditions at the site [26]. Imbalance prices are adjusted from units of EUR/MWh to EUR/kWh by dividing by 1000 and treated as external input signals for control refinement [27] (Not in LQR controller).

In parallel, a heating demand profile is constructed to simulate cleaning operations on the dairy farm. These cleaning activities include scheduled flushing of milking parlours, stables, and deep cleaning routines. The exact schedule with the corresponding flow rates can be found in Section 5 The heat extraction from the tank is computed as:

$$Q_{\text{draw}} = \dot{V} \cdot c_p \cdot \left(T_{\text{ref}} - T_{\text{inlet}}\right) \tag{14}$$

where $\dot{V}$ is the water flow rate in L/min, c_p is the specific heat capacity of water, and the temperature difference reflects the heating requirement. This part of the simulation is the same for both types of controllers. The water flow rate is determined by the cleaning schedule presented in Section 5. The initial temperature of the water tank is assumed to be 60 °C based on the reference temperature that needs to be tracked. The next subsection will explain the characteristics of both controllers separately.

8.1 LQR Simulation

The LQR controller loops over all valid time steps in the input dataset for a one-year period. Every minute the Euler integration is updated and a value for boiler power u_k is chosen to minimize the quadratic cost function. It makes a trade-off between the deviation from the reference temperature and the cost of energy. Instead of re-solving for every step, a single feedback gain K_k is computed at the start of every hour and kept constant for the next 60 minutes. Additionally, the reference temperature is based on the electricity price, and is also kept constant for these 60 minutes. Because the electricity price only changes once every hour there is no need to recompute every minute. This will allocate more heating to the hours with low prices and less in periods of high prices.

Finally, all relevant data are logged and accumulated to generate a table and a graph to have a numeric and visual representation of the system performance.

8.2 MPC Simulation

The MPC controller loops over all valid time steps in the input datasets for a one-month period. Every 15 minutes, the spot-only loop computes a new baseline control sequence, which is then used to initialize the imbalance optimization. In both loops, the reference temperature is determined based on whether the electricity and the imbalance price are negative or positive. In case of negative prices, the reference temperature is increased from 60 °C to 80 °C. The imbalance loop then looks at the current imbalance price and decides if and by how much to deviate from the baseline control sequence. The optimal boiler power u_k is applied, while adhering to the boiler's physical power limit ($P_{\max} = 100$ kW) and current reference temperature. The tank temperature is updated using the dynamics presented in section 5, and all relevant data are recorded. For the simulation the IPOPT solver was configured as follows: `{'ipopt':'print_level':0,'max_iter':30,'tol':1e-3,'print_time':0}`

For each step, the cost and electricity consumption are logged and accumulated. Spot-market cost, including the tiered energy tax and VAT is accumulated. Imbalance-market profit or penalty, depending on the deviation is added or subtracted from the spot market cost. The flat-rate benchmark cost, assuming a constant electricity price of €0.29/kWh is calculated exactly the same.

For a visual representation of the system performance, different plots are created. A Tank temperature plot over time, which shows how well the set-points are followed and how effective the utilization of low-price periods for energy storage is. Additionally, a boiler power profile is plotted, which captures how the boiler responds to both spot and imbalance pricing signals.

9 Simulation Results

This section presents the outcomes of the simulations that compare the performance of two control strategies applied to a diary farm equipped with an electric boiler coupled to thermal energy storage. A linear quadratic regulator was tested with flat and dynamic electricity prices, while a model predictive controller was tested using flat, dynamic and imbalance prices. All simulations use real-world data from the 2024 Dutch electricity market, including hourly day-ahead prices and 15-minute imbalance prices [25]–[27].

To have a baseline for comparison, both the LQR and MPC controllers were first operated under a fixed electricity price of €0.29 per kWh. In this configurations the controllers tried to maintain the reference temperature without looking at the time-varying price signals. This meant that the controllers only optimized for thermal comfort. This approach ensures that all comparisons between the strategies isolate the value added by market-aware behaviour rather than differences in algorithm complexity or system modelling.

Under the fixed price scenario the LQR controlled boiler consumed 8116 kWh in January and 8003 kWh in august, and incurred €2353.67 in January and €2320.76 in august. The MPC controller shows a reduced energy consumption when using the same flat rate which could be caused by the MPC's ability to schedule pre-heating before cleaning events, or by mistakes in the modelling of the controllers. Under the flat rate the MPC controlled boiler consumed 6913 kWh in January and 5894 kWh in august and incurred €2004.81 in January and € 1709.40 in August (see table 2).

When the LQR controller was applied with the dynamic day-ahead prices the electricity consumption was reduced by only 1.4% in January and actually increased by 0.08% in August, but the energy cost was reduced by 33.4% and 33.0% respectively. Due to participation on the imbalance market, the MPC controlled boiler achieved much better results. Even though it used more energy when reacting to price signals, profit on the imbalance market meant that costs were reduced much more. The MPC controlled boiler consumed 7285 kWh in January and 6797 kWh in august, and incurred €1053.99 and €952.85 respectively. This means that costs were reduced by 47.4% and 44.3% when responding to dynamic day-ahead and imbalance prices. (see table 3)

When looking at the graphs 2 - 9 we can see that the LQR is not good at tracking the temperature, and is constantly below the reference temperature of 60°C. In contrast, the MPC controlled boiler is performing significantly better at being at or above the reference temperature, with only small dips below the reference (see figures 14 - 17). In addition, the MPC controller consumes less energy in all scenarios, while tracking the reference temperature more accurately. This might suggest that there is a mistake in one of the two models.

LQR results

Metric	January		August	
	Flat Rate	Dynamic Rate	Flat Rate	Dynamic Rate
Energy [kWh]	8,116	8,101	8,003	8,010
Total Cost [€]	2,353.67	1,567.55	2,320.76	1,554.72

Table 2: Comparison of energy use and cost under flat and dynamic pricing for January and August.

MPC results

Metric	January		August	
	Flat Rate	Dyn. + Imbalance	Flat Rate	Dyn. + Imbalance
Energy [kWh]	6,913	7,285	5,894	6,797
Spot-Market Cost [€]	2,004.81	1,420.38	1,709.40	1,159.87
Imbalance P&L [€]	–	366.39	–	207.02
Net Cost [€]	**2,004.81**	**1,053.99**	**1,709.40**	**952.85**

Table 3: Comparison of energy and cost under flat rate and dynamic + imbalance control for January and August.

10 Discussion

The results that were obtained confirm that using smart control strategies can provide significant operational and economic benefits when controlling boilers coupled to thermal energy storages in dynamic electricity markets. Both the LQR and MPC managed to maintain the required thermal comfort while adapting to price signals to reduce cost. The LQR controller useful but it needs more work to be suitable for participation on the imbalance market. The MPC, with its ability to participate on the imbalance market achieved even greater cost and energy consumption reductions, particularly during months where imbalance prices were volatile.

The comparison between the controllers under fixed and dynamic pricing was central in this analysis. The results highlighted that using dynamic and imbalance pricing signals did not result in a reduced energy consumption, but did cause significant cost savings. The MPC's performance in January highlights the value of high-resolution predictive control when price signals are favourable. With the imbalance prices occasionally turning negative, the controller scheduled the heating operations in these intervals of negative pricing. The lower profit in August suggest that it might not always be possible to be profitable on the imbalance market.

The model assumes full availability of imbalance signals and perfect execution of control actions. Real-world systems may face latency, regulatory barriers, or actuation delays. Future work could incorporate uncertainty modelling, battery integration, or real-time implementation on hardware-in-the-loop systems. Additionally, the incorporation of imperfect or real-world forecasts of the imbalance price into the MPC formulation would be a promising direction for future work.

In addition to financial performance, the control strategies also impact the systems interaction with the electricity grid and its environmental footprint. The LQR controller managed to reduce energy consumption by only used a negligible amount. It is unlikely to have any impact on reducing stress on the electricity grid. The MPC controller absorbed surplus energy in periods of negative imbalance prices, which further contributes to a more stable grid. However, this did mean that the MPC controller consumed more energy when using dynamic pricing compared to flat rates. Therefore, it will not result in a reduction of CO_2 emissions when using grid electricity.

In the future, PV forecasts should be taken into account in order to make a more accurate consumption planning to benefit the most from negative electricity prices and imbalance prices. When combining the boiler and TES with Battery Energy Storage Systems and PV installations, more gains could be made. For example, by consuming excess solar energy instead of energy from the grid. In addition, forecasts for the imbalance prices should also be included instead of using historical data. Additionally, real-world experiments should be conducted to find out whether the control strategy works on FIRN energy's existing software and hardware. This would further validate the improvements in performance, when compared to only running a simulation.

Together, these results indicate that smart control of water heating systems offers a practical and impactful method for strategically consuming electricity, and increasing responsiveness to market dynamics. When combined with PV installations and BESS, energy efficiency can be improved.

11 Conclusion

This research has explored the usage of LQR and MPC algorithms to optimise the operation of an electric boiler and thermal energy storage system on a dairy farm. By using real 2024 electricity market data and realistic physical constraints, the simulations showed that both controllers can significantly reduce operational costs compared to traditional fixed-price, temperature optimized strategy.

The LQR controller managed to reduce the monthly electricity costs by 33.4% and 33.0% in the months January and August 2024 respectively. It did this by shifting energy consumption to hours with cheap electricity within the day-ahead market. However, thermal comfort and system constraints were never sufficiently respected. Most of the cost saving was achieved by the lower average price of the dynamic contracts. The issue with the LQR controller is that it is unable to accurately track the reference temperature. This could be caused by multiple things. Such as, mistakes in the modelling or programming, or due to a boiler that is not powerful enough.

The MPC controller, on the other hand achieved even higher reductions due to the participation in the imbalance market. It achieved a cost reduction of 47.4% in January and 44.3% in August 2024, highlighting how participation on the imbalance market can result in further cost reductions. Another benefit of participation in the imbalance market is that excess electricity is consumed, which means the grid will become more balanced. The MPC controller did manage to accurately track the reference temperature, with only minimal dips. In this particular case MPC would be the preferred algorithm.

However, the results also highlight the importance of context. In periods with lower volatility in imbalance prices, like August, the MPC did not achieve the same amount of profit. Which suggests that participation on the imbalance market may not always be profitable. In addition, the exact rules for the Dutch imbalance market were not incorporated in the code. Only the submission of a consumption schedule and deviation was included. If these rules had been included, the performance of the controller would have been to be worse. Therefore, the rules should be embedded more explicitly in future research.

The assumption of perfect price forecasts was necessary for isolating the performance of the control algorithms, but it remains a limitation. Future research should incorporate forecast uncertainty into the controller design and assess performance across multiple years and use cases. Additionally, integrating environmental and grid-side metrics more explicitly into the objective function could further align control outcomes with broader sustainability goals. Finally, this research was only using simulations, and not real-world experiments. Therefore, it could not be determined if the algorithms could work together with FIRN energy's existing software and hardware. This is also something that should be explored in future research.

In conclusion, this research has shown that model predictive control is a technically and economically viable strategy for managing flexible heat demand in agriculture. Due to the ability to look ahead and respond to price signals, MPC controllers are very well suited for controlling an electric boiler. Having access to the right data and infrastructure, these systems can contribute energy cost reduction, and improve grid stability in an increasingly volatile electricity market.

References

[1] L. G. A.H., B. R. M.J., and M. H. C., "Balancing responsibilities: Effects of growth of variable renewable energy, storage, and undue grid interaction," *Energy Policy*, vol. 136, 2019.

[2] E. Commission, "2050 long-term strategy," Tech. Rep., 2025.

[3] E. Szaruga, M. Frankowska, and K. Drela, "Spatial autocorrelation of power grid instability in the context of electricity production from renewable energy sources in polish regions," *Energy Reports*, vol. 8, 2022.

[4] L. Zhang, G. Feng, K. Huang, Y. Bi, S. Chang, and A. Li, "Design and optimization for photovoltaic heat pump system integrating thermal energy storage and battery energy storage," *Energy & Buildings*, vol. 329, 2025.

[5] A. Rapucha, R. Narayanan, and M. Jha, "Heat pumps with smart control in managing australian residential electrical load during transition to net zero emissions," *Energies*, vol. 17, no. 12, p. 2977, 2024.

[6] D. H. Clift and H. Suehrcke, "Control optimization of pv powered electric storage and heat pump water heaters," *Solar Energy*, vol. 226, pp. 489–500, 2021.

[7] International Energy Agency, "The future of heat pumps: How a heat pump works," International Energy Agency, IEA Special Report, 2022, CC BY 4.0.

[8] J. Freier and V. von Loessl, "Dynamic electricity tariffs: Designing reasonable pricing schemes for private households," *Energy Economics*, vol. 112, p. 106 146, 2022.

[9] J. Browell and C. Gilbert, "Predicting electricity imbalance prices and volumes: Capabilities and opportunities," *Energies*, vol. 15, no. 10, p. 3645, 2022.

[10] J. Lago, K. Poplavskaya, G. Suryanarayana, and B. D. Schutter, "A market framework for grid balancing support through imbalances trading," *Renewable and Sustainable Energy Reviews*, vol. 137, p. 110 467, 2021.

[11] R. Malecki, F. Bünning, D. Müller, T. Münch, and D. Müller, "Impact of startup and defrosting on the modeling of hybrid systems in district heating," *Energy Reports*, vol. 8, pp. 15 057–15 068, 2022.

[12] ANWB Energie. "Alles over dynamische energiecontracten." (2025), [Online]. Available: `https://www.anwb.nl/energie/wat-is-een-dynamisch-energiecontract`.

[13] I. Lampropoulos, J. Frunt, F. A. Nobel, A. Virag, P. P. J. van den Bosch, and W. L. Kling, "Analysis of the market-based service provision for operating reserves in the netherlands," in *Proceedings of the 9th International Conference on the European Energy Market (EEM)*, 2012, pp. 1–8.

[14] S. Deng, J. Inekwe, V. Smirnov, A. Wait, and C. Wang, "Seasonality in deep learning forecasts of electricity imbalance prices," *Energy Economics*, vol. 137, p. 107 770, 2024.

[15] TenneT, *Balanceringsmarkten*, Online; Dutch, Accessed 2025-06-12, 2025.

[16] S. Zarghoon, S. Emebu, R. Matušů, *et al.*, "Full-state feedback lqr with integral gain for control of induction heating of steel billet," *Engineering Science and Technology, an International Journal*, vol. 55, p. 101 721, 2024.

[17] M. Zlatkovikj, H. Li, V. Zaccaria, and I. Aslanidou, "Development of feed-forward model predictive control for applications in biomass bubbling fluidized bed boilers," *Journal of Process Control*, vol. 115, pp. 167–180, 2022.

[18] M. J. Linders, R. Segers, M. van Middelkoop, *et al.*, "Aardwarmte en bodemenergie," Centraal Bureau voor de Statistiek (CBS), Longread, hoofdstuk 6 in "Hernieuwbare Energie in Nederland 2020", Sep. 2021.

[19] J. Verrett, “Foundations of chemical and biological engineering i,” in BCcampus Open Publishing, 2020, ch. Introduction to Energy Balances.

[20] L. Magni and R. Scattolini, *Advanced and Multivariable Control*. Bologna, Italy: Pitagora Editrice Bologna, 2014.

[21] $Scipy.linalg.solve_discrete_are|scipyv1.15.2manual$, SciPy Community, 2025.

[22] 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*, pp. 1–31, 2018.

[23] G. C. Philip, “Software design guidelines for event-driven programming,” *Journal of Systems and Software*, vol. 41, no. 2, pp. 79–91, May 1998.

[24] J. Allan, L. Croce, R. Dott, G. Georges, and P. Heer, “Calculating the heat loss coefficients for performance modelling of seasonal ice thermal storage,” *Journal of Energy Storage*, vol. 52, p. 104 528, 2022.

[25] Jeroen.nl. “Data feeds – jeroen.nl.” (2025), [Online]. Available: `https://jeroen.nl/account/feeds`.

[26] Weerverleden.nl. “Etmaalgemiddelde temperatuur 2024.” (2025), [Online]. Available: `https://weerverleden.nl/download/2024-tempavg`.

[27] ENTSO-E. “Imbalance data view.” (2025), [Online]. Available: `https://transparency.entsoe.eu/balancing/r2/imbalance/show`.

Gen-AI usage: ChatGPT models o3, o4, and o4-mini-high were used in this report for brainstorming, language correction and assistance with coding in python.

A Figures

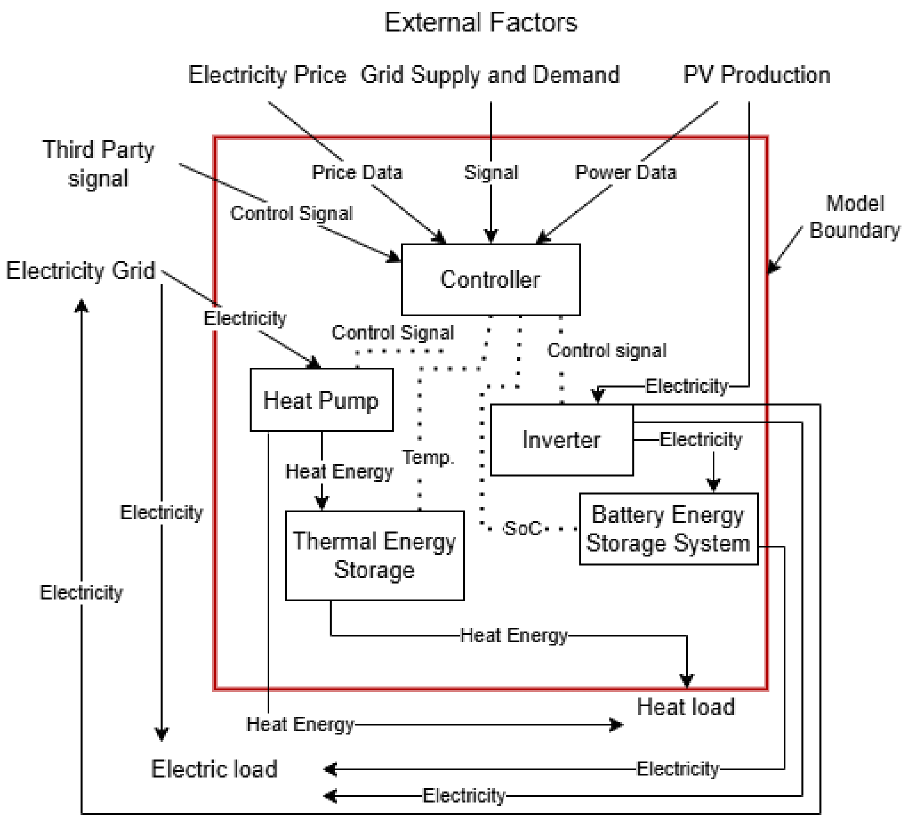

Figure 1: System Diagram

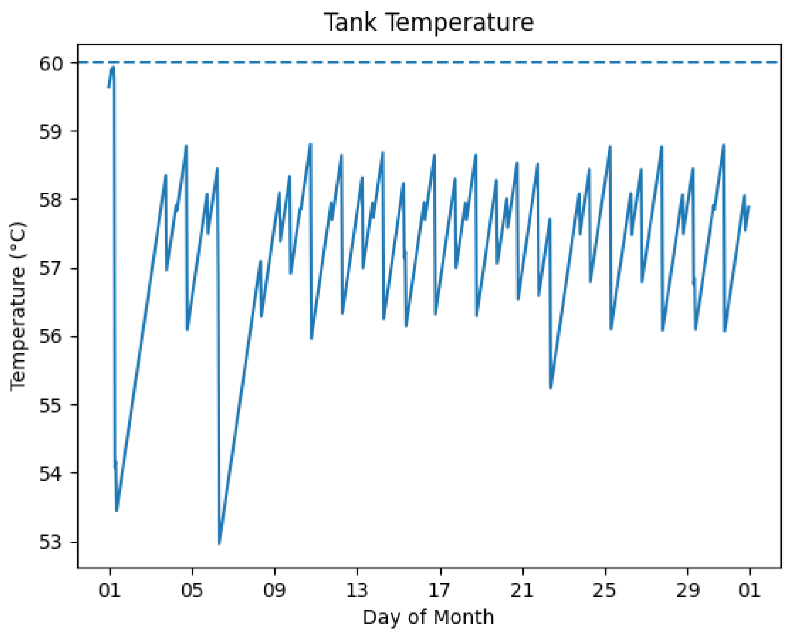

Figure 2: Tank temperature using LQR - flat rate for the month January

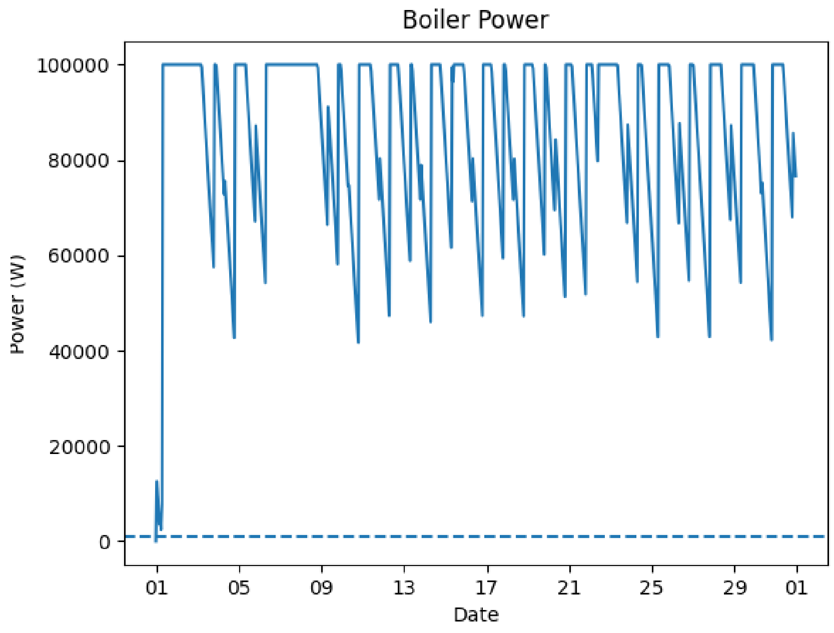

Figure 3: Boiler power using LQR - flat rate for the month January

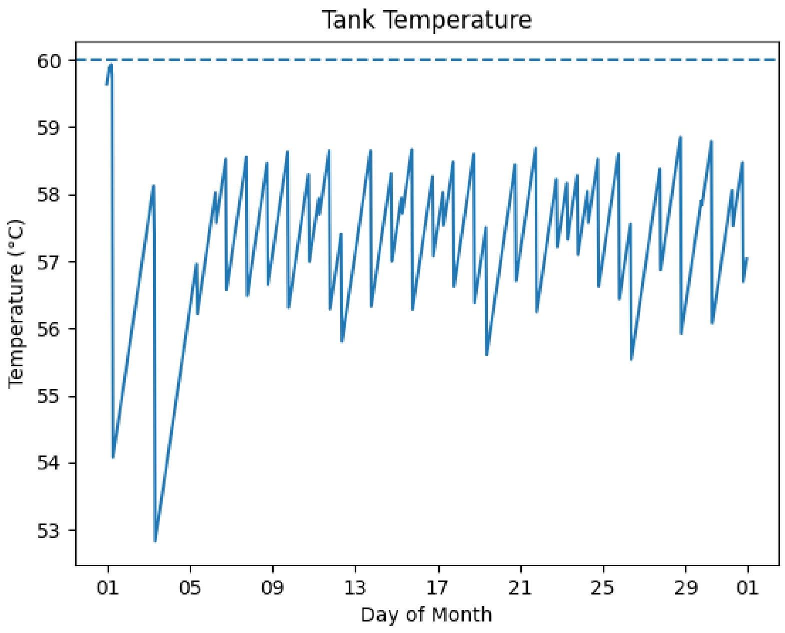

Figure 4: Tank temperature using LQR - flat rate for the month August

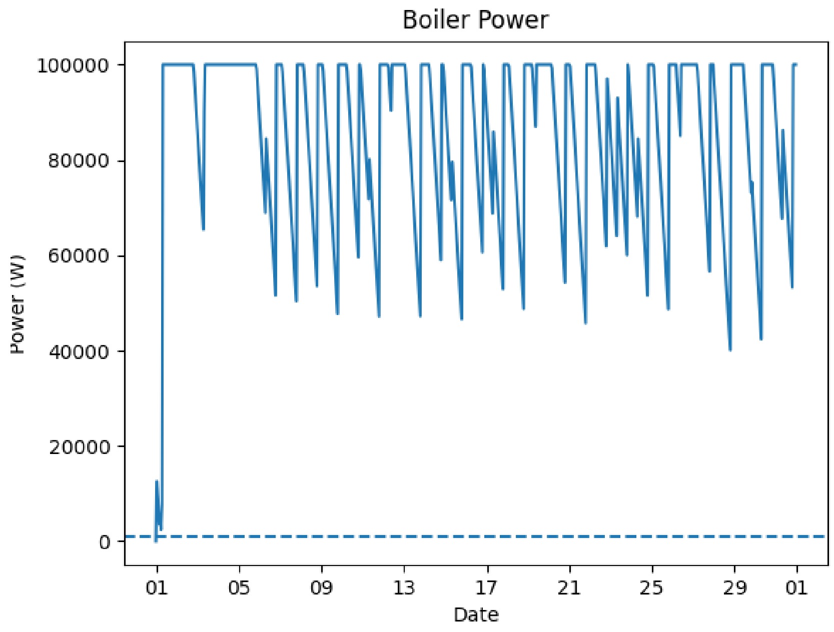

Figure 5: Boiler power using LQR - flat rate for the month August

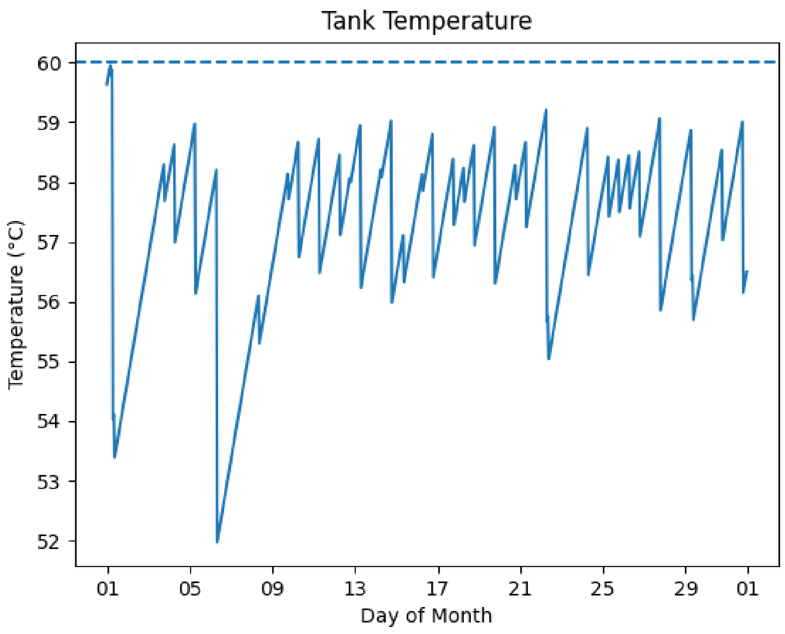

Figure 6: Tank temperature using LQR - dynamic rate for the month January

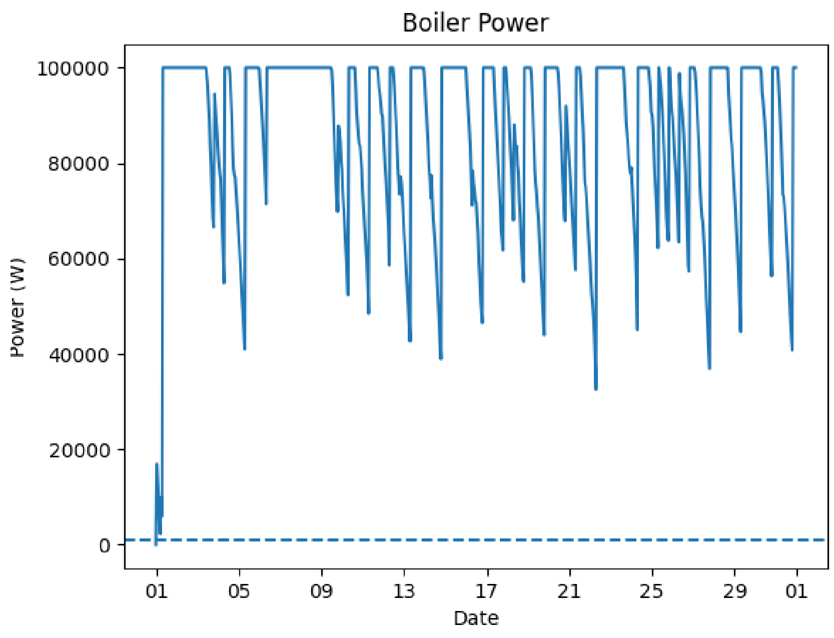

Figure 7: Boiler power using LQR - dynamic rate for the month January

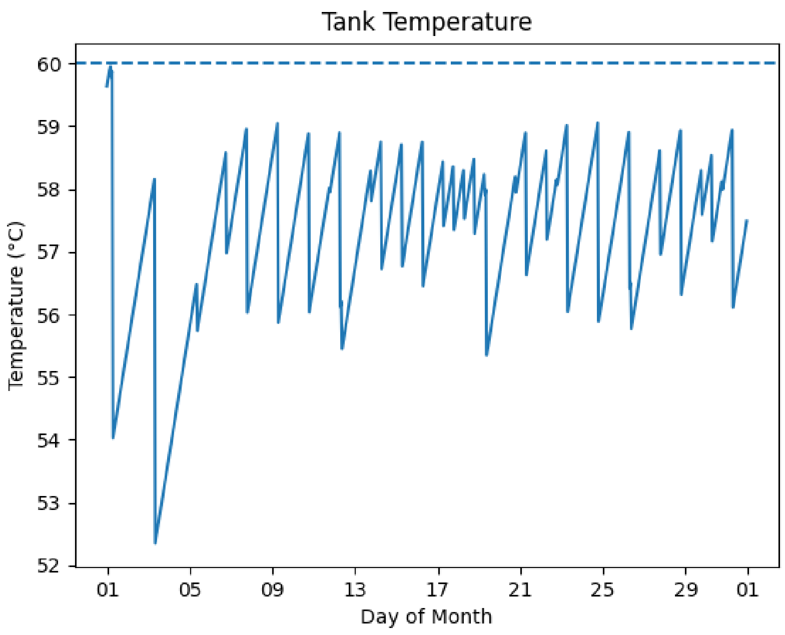

Figure 8: Tank temperature using LQR - dynamic rate for the month August

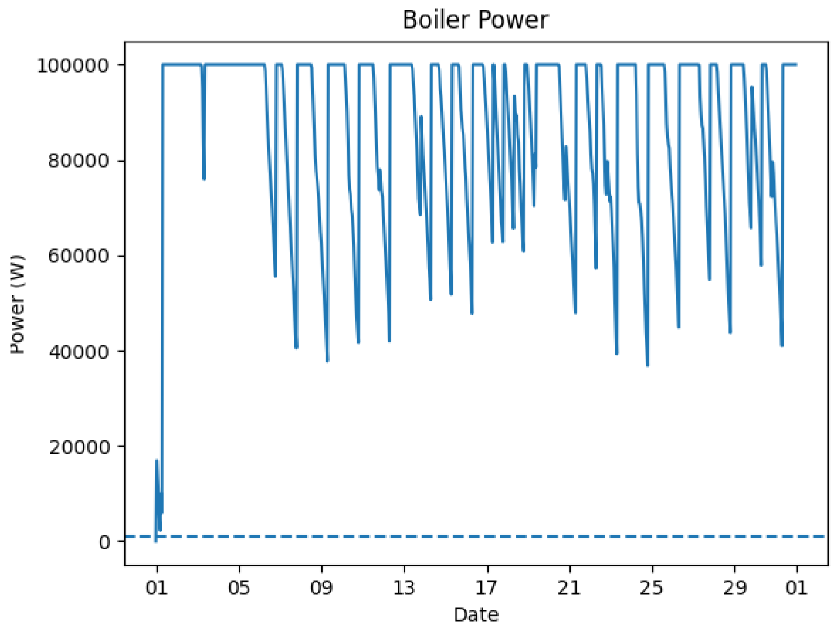

Figure 9: Boiler power using LQR - dynamic rate for the month August

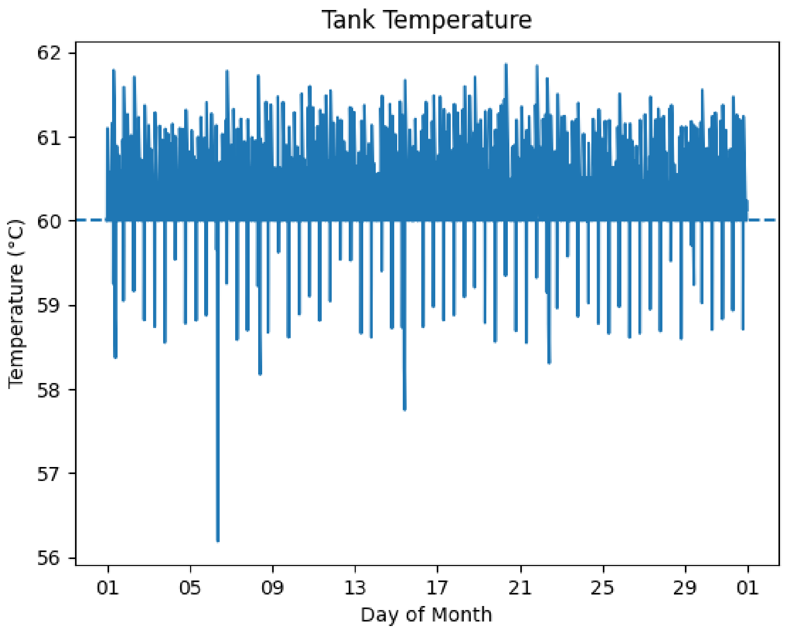

Figure 10: Tank temperature using MPC - flat rate for the month January

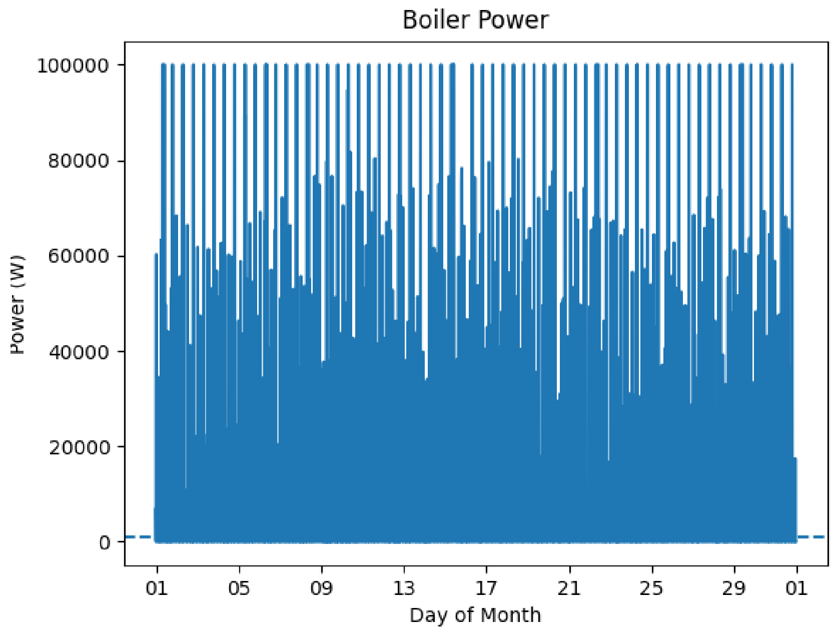

Figure 11: Boiler power using MPC - flat rate for the month January

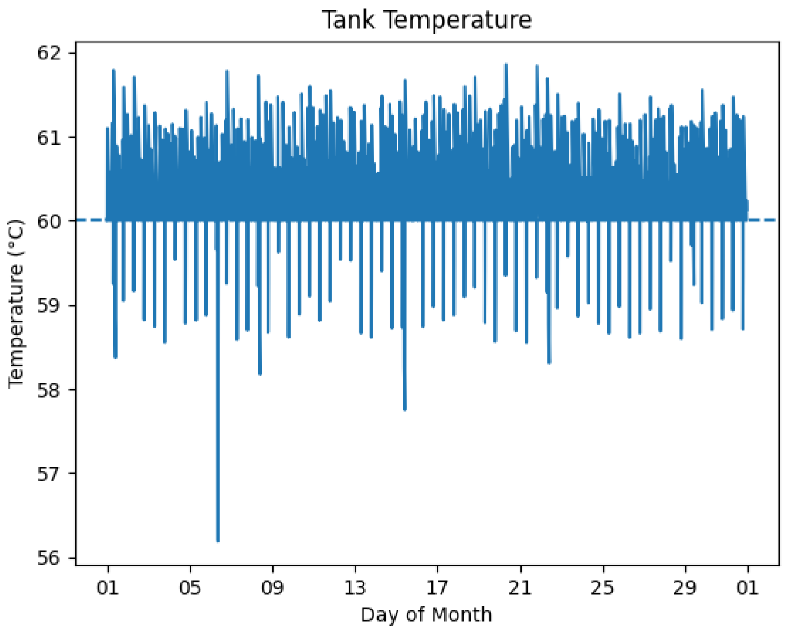

Figure 12: Tank temperature using MPC - flat rate for the month August

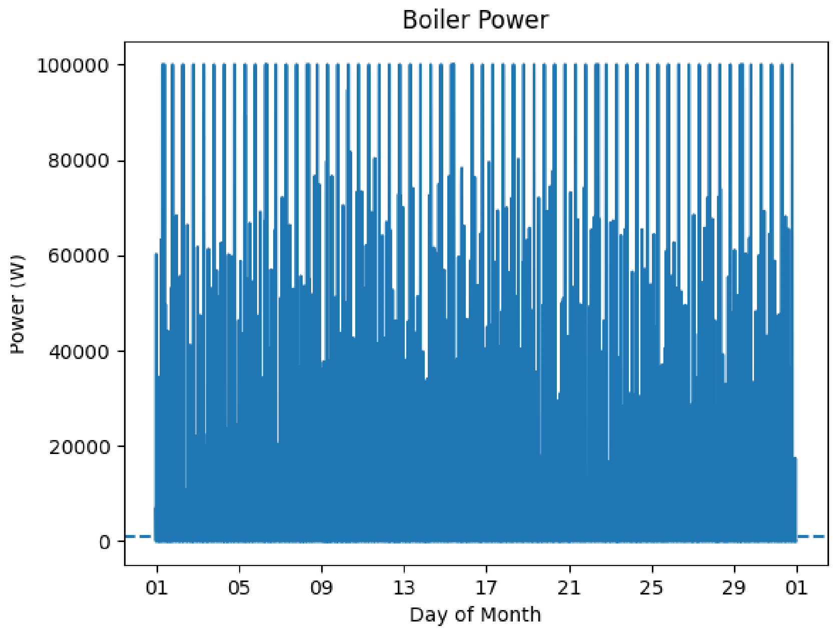

Figure 13: Boiler power using MPC - flat rate for the month August

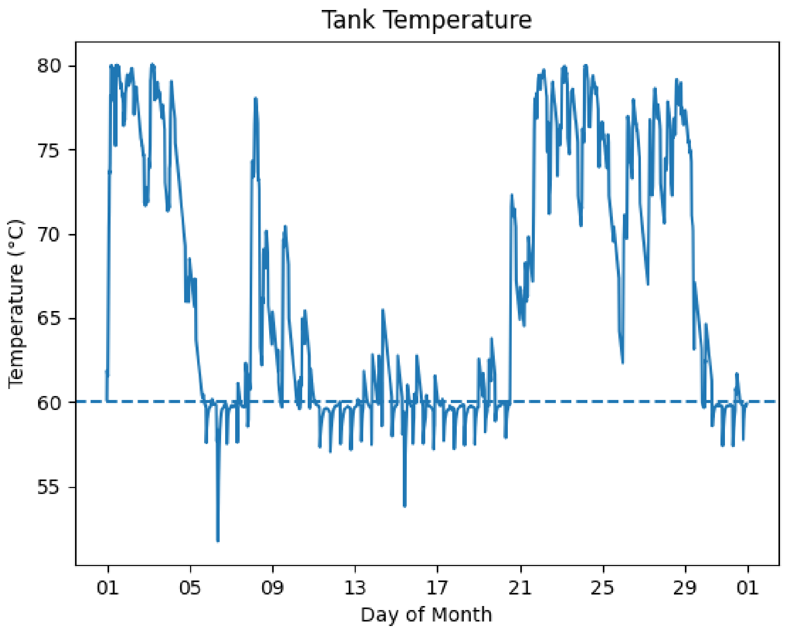

Figure 14: Tank temperature using MPC - dynamic and imbalance rate for the month January

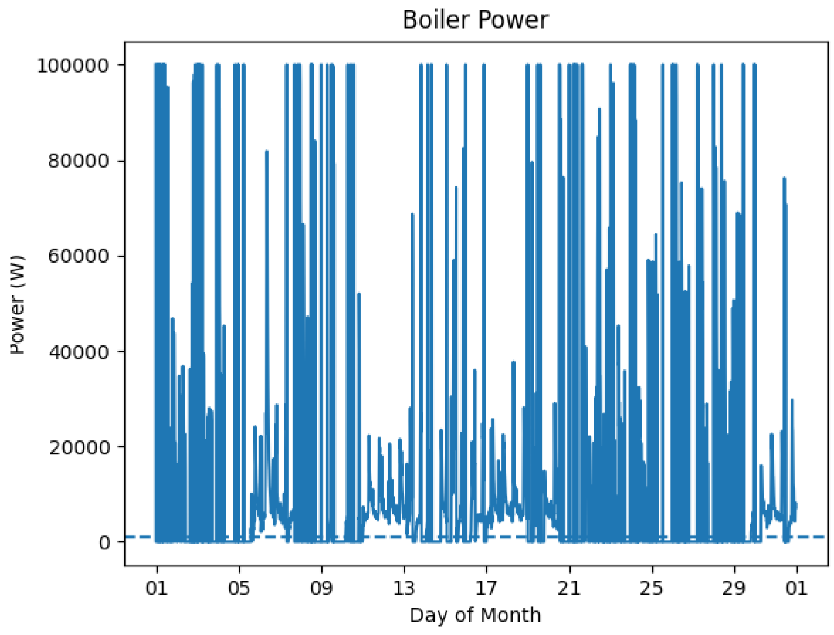

Figure 15: Boiler power using MPC - dynamic and imbalance rate for the month January

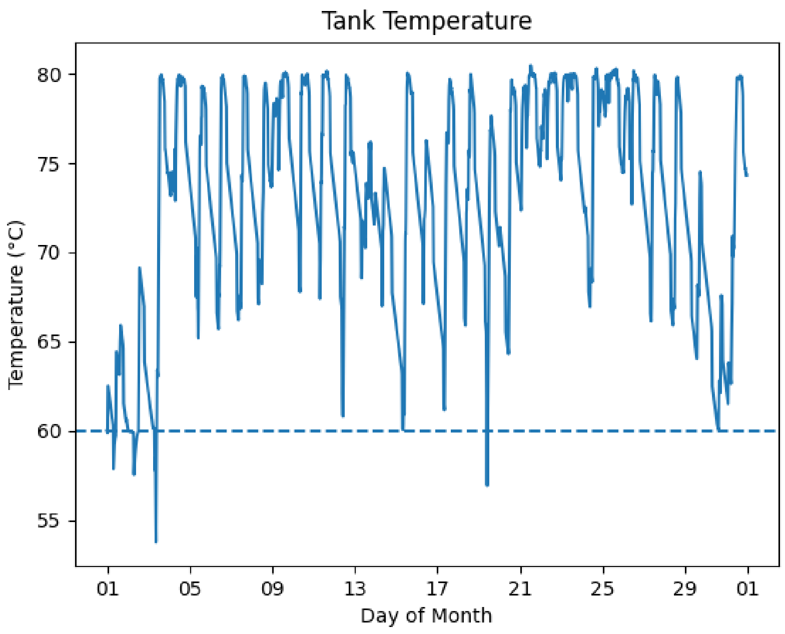

Figure 16: Tank temperature using MPC - dynamic and imbalance rate for the month August

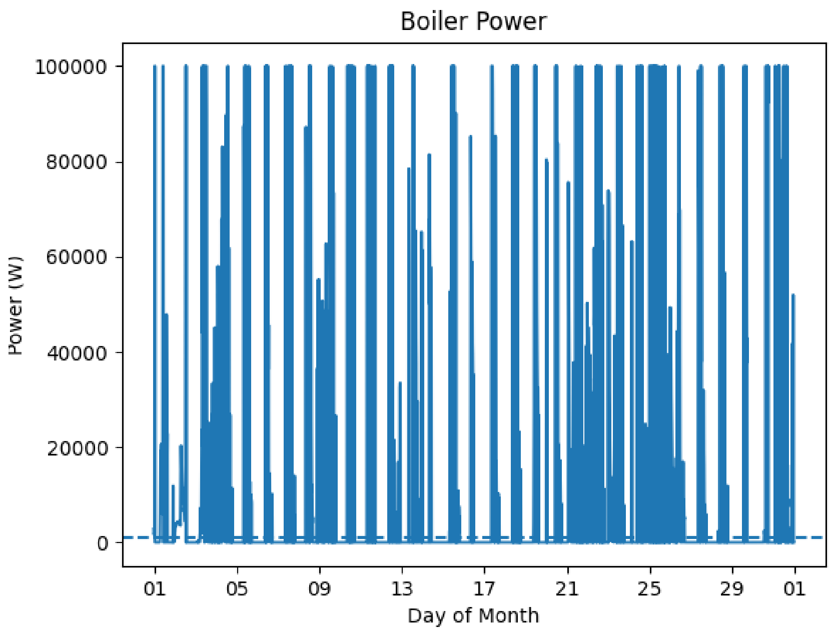

Figure 17: Boiler power using MPC - dynamic and imbalance rate for the month August

B Python Code

Listing 1: Python LQR code - dynamic rate

```python
import numpy as np
import pandas as pd
from datetime import datetime, timedelta
import sys
from scipy.linalg import solve_discrete_are
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

# Constants
M = 10_000.0  # kg water
CP = 4_186.0  # J/(kg K) heat capacity
BOIL_MAX = 100_000.0  # W element
hA = 79.5  #  W K   heat loss
T_IN = 10.0  #  C  inlet
T_SP = 60.0  #  C  setpoint for storage
Q_SPACE = 1_000.0  # W base load
T_MIN = 50.0  #  C  minimum temperature
T_MAX = 85.0  #  C  maximum temperature
DT = 60.0

# Cleaning constants
FLOW_PARLOR = 20.0  # L/min for parlor cleaning
FLOW_STABLE = 30.0  # L/min for stable cleaning
FLOW_DEEP = 40.0
TIME_PARLOR = 30  # minutes for parlor cleaning
TIME_STABLE = 120  # minutes for full stable cleaning
TIME_DEEP = 180  # minutes for monthly deep cleaning

# Tariff settings
FLAT_RATE = 0.29  #   /kWh

# Read dynamic electricity prices
try:
    prices_df = pd.read_csv("C:/Users/julia/Downloads/jeroen_punt_nl_dynamische_stroomprijzen_jaar_2024.csv",
                            sep=";", decimal=",")

    # Flexible column detection
    col_date = next(c for c in prices_df.columns if "datum" in c.lower() or "time" in c.lower())
    col_price = next(c for c in prices_df.columns if "prijs" in c.lower() or "eur" in c.lower())

    # Convert date and sort
    prices_df[col_date] = pd.to_datetime(prices_df[col_date])
    prices_df = prices_df.rename(columns={col_date: 'datum', col_price: 'prijs_excl_belastingen'})
    prices_df = prices_df.sort_values('datum')

except FileNotFoundError:
    sys.exit("Dynamic price file not found.")
except StopIteration:
    print("Available columns:", prices_df.columns.tolist())
    sys.exit("Could not find date/price columns in the price file.")
```

```python

# Read temperature data
temp_df = pd.read_csv("C:/Users/julia/Downloads/2024-tempavg-weerverleden (1).csv")
temp_df['DATUM'] = pd.to_datetime(temp_df['DATUM'])
temp_df = temp_df.sort_values('DATUM')

def is_cleaning_time(h, d, m):
    """
    Determines if cleaning should occur based on hour, day, and month
    """
    parlor_cleaning = (h == 7 or h == 19)
    stable_cleaning = (d == 0 and h == 9)   # Monday at 9:00
    deep_cleaning = (d == 5 and h == 8 and m <= 7)   # First Saturday at 8:00

    return {
        'parlor': parlor_cleaning,
        'stable': stable_cleaning,
        'deep': deep_cleaning
    }

def water_demand(h, d, m):
    """Calculate water demand in L/min based on cleaning schedule"""
    cleaning = is_cleaning_time(h, d, m)

    if cleaning['deep']:
        return FLOW_DEEP
    elif cleaning['stable']:
        return FLOW_STABLE
    elif cleaning['parlor']:
        return FLOW_PARLOR
    else:
        return 0.0

# Model matrices and weights
A = 1 - hA*DT/(M*CP)
B = DT/(M*CP)
Q = 5                      # temperature weight
BASE_R = 1e-9               # baseline effort weight
ALPHA = 1e-8                # price     effort weight (   /kWh)

# Control parameters
SOAK_PRICE = -0.01      #    /kWh threshold to start soaking
SOAK_LIMIT = 80.0       #   C  maximum soak temperature
T_REF = 60.0            #   C  reference temperature

# Riccati matrices
A_mat = np.array([[A]])
B_mat = np.array([[B]])
Q_mat = np.array([[Q]])

def riccati_gain(r_now: float) -> float:
    """Return optimal  s t a t e feedback  gain K for current R=r_now."""
    R_mat = np.array([[max(r_now, 1e-15)]] )  # force R>0
    P = solve_discrete_are(A_mat, B_mat, Q_mat, R_mat)
    K = float(np.linalg.inv(R_mat + B_mat.T @ P @ B_mat) @ (B_mat.T @ P @ A_mat))
    return K

def get_ambient_temp(hour_index):
    """Get ambient temperature for the given hour"""
    day_index = hour_index // 24
    return float(temp_df.iloc[day_index]['ETMAALGEMIDDELDE TEMPERATUUR'])

def get_price(hour_index, cumulative_usage_kwh, tariff_type='dynamic'):
    if tariff_type == 'dynamic':
        base_price = float(prices_df.iloc[hour_index]['
            prijs_excl_belastingen'])
        return calculate_total_price(base_price, cumulative_usage_kwh,
            tariff_type)
    else:
        return FLAT_RATE

def lqr_control_dynamic(T, price, hour_index, hour, weekday, month_day):
    """LQR control strategy optimized for dynamic pricing"""
    if price <= 0:
        r_now = BASE_R + ALPHA * price * 2  # Double the effect of negative
             prices
    else:
        r_now = BASE_R + ALPHA * price

    K = riccati_gain(r_now)
    T_target = SOAK_LIMIT if price <= SOAK_PRICE else T_REF
    err = T - T_target
    u = np.clip(-K*err, 0.0, BOIL_MAX)
    if T >= T_target:
        u = 0.0
    return u

def lqr_control_flat(T, hour_index, hour, weekday, month_day):
    """LQR control strategy optimized for flat rate pricing"""
    r_now = BASE_R + ALPHA * FLAT_RATE
    K = riccati_gain(r_now)
    err = T - T_REF  # Always use T_REF as target for flat rate
    u = np.clip(-K*err, 0.0, BOIL_MAX)
    if T >= T_REF:
        u = 0.0
    return u

def simulate_year(control_type='lqr', tariff_type='dynamic'):
    start_date = datetime(2024, 8, 1)
    days = 31
    hours_per_day = 24
    total_hours = days * hours_per_day

    energy_used = 0.0
    cost_total = 0.0
    T = T_SP
    T_log = np.zeros(total_hours)
    U_log = np.zeros(total_hours)

    for hour_index in range(total_hours):
        current_date = start_date + timedelta(hours=hour_index)
        hour = current_date.hour
        weekday = current_date.weekday()
        month_day = current_date.day

        # Get price based on tariff type and current usage
        price = get_price(hour_index, energy_used, tariff_type)

        # Use actual ambient temperature (daily average)
        T_AMB = get_ambient_temp(hour_index)

        flow = water_demand(hour, weekday, month_day)

        if control_type == 'lqr':
            if tariff_type == 'dynamic':
                control = lqr_control_dynamic(T, price, hour_index, hour, weekday, month_day)
            else:
                control = lqr_control_flat(T, hour_index, hour, weekday, month_day)

        Q_heat = flow * CP * (T_SP - T_IN) / 60.0 if flow > 0 else 0
        Q_loss = hA * (T - T_AMB)
        Q_total = Q_heat + Q_loss + Q_SPACE

        Q_applied = min(control, Q_total)

        dT = (Q_applied - Q_loss - Q_heat) / (M * CP) * 3600
        T += dT

        energy_hour = Q_applied * 1.0 / 1000.0  # Convert to kWh
        energy_used += energy_hour
        cost_total += energy_hour * price
        T_log[hour_index] = T
        U_log[hour_index] = control  # Convert normalized control to actual power (W)

    return energy_used, cost_total, T_log, U_log


def calculate_total_price(base_price, cumulative_usage_kwh, tariff_type='dynamic'):
    if tariff_type == 'flat':
        return FLAT_RATE  # Flat rate already includes everything

    # For dynamic pricing, add energy tax based on usage tier
    if cumulative_usage_kwh <= 2900:
        energy_tax = 0.10880
    elif cumulative_usage_kwh <= 10000:
        energy_tax = 0.10880
    elif cumulative_usage_kwh <= 50000:
        energy_tax = 0.09037
    else:
        energy_tax = 0.03943

    # Total price = base + energy tax (no VAT for dynamic)
    total = base_price + energy_tax

    return total

# Run simulations for all combinations
```

```python
results = {}
strategies = [('lqr', 'dynamic')]

for control, tariff in strategies:
    energy, cost, T_log, U_log = simulate_year(control, tariff)
    results[(control, tariff)] = (energy, cost)

# Print results
print("\n===== YEAR 2024 CLEANING SCHEDULE =====")
print("Daily: Parlor cleaning 2x (7:00 and 19:00)")
print("Weekly: Full stable cleaning (Mondays 9:00)")
print("Monthly: Deep cleaning (First Saturday 8:00)")

print("\n===== SIMULATION RESULTS =====")
for (control, tariff), (energy, cost) in results.items():
    print(f"\n{control.upper()} control with {tariff} tariff:")
    print(f"Energy Used: {energy:,.0f} kWh")
    print(f"Total Cost:    {cost:,.2f}")
    print(f"Average Daily Cost:    {(cost/31):,.2f}")

# Build time axis
start_date = datetime(2024, 1, 1)
dates = [start_date + timedelta(hours=i) for i in range(len(T_log))]

# Plot temperature
plt.figure()
plt.plot(dates, T_log, label="Tank Temperature")
plt.axhline(T_REF, linestyle='--', label="Setpoint (T_REF)")
plt.title("Tank Temperature")
plt.ylabel("Temperature ( C )")
plt.xlabel("Day of Month")
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%d'))
plt.show()

# Plot heater power
plt.figure()
plt.plot(dates, U_log, label="Heater Power")
plt.title("Boiler Power")
plt.ylabel("Power (W)")
plt.xlabel("Date")
plt.axhline(Q_SPACE, linestyle='--', label="Base Load")
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%d'))
plt.show()
```

Listing 2: Python LQR code - flat rate

```python
import numpy as np
import pandas as pd
from datetime import datetime, timedelta
import sys
from scipy.linalg import solve_discrete_are
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

# Constants
M = 10_000.0   # kg water
CP = 4_186.0   # J/(kg K) heat capacity
BOIL_MAX = 100_000.0  # W element
hA = 79.5  #   W K    heat loss
```

```python
T_IN = 10.0  #  C  inlet
T_SP = 60.0  #  C  setpoint for storage
Q_SPACE = 1_000.0  # W base load
T_MIN = 50.0  #  C  minimum temperature
T_MAX = 85.0  #  C  maximum temperature
DT = 60.0

# Cleaning constants
FLOW_PARLOR = 20.0  # L/min for parlor cleaning
FLOW_STABLE = 30.0  # L/min for stable cleaning
FLOW_DEEP = 40.0
TIME_PARLOR = 30  # minutes for parlor cleaning
TIME_STABLE = 120  # minutes for full stable cleaning
TIME_DEEP = 180  # minutes for monthly deep cleaning

# Tariff settings
FLAT_RATE = 0.29  #    /kWh

# Read dynamic electricity prices
try:
    prices_df = pd.read_csv("C:/Users/julia/Downloads/jeroen_punt_nl_dynamische_stroomprijzen_jaar_2024.csv",
                            sep=";", decimal=",")

    # Flexible column detection
    col_date = next(c for c in prices_df.columns if "datum" in c.lower() or "time" in c.lower())
    col_price = next(c for c in prices_df.columns if "prijs" in c.lower() or "eur" in c.lower())

    # Convert date and sort
    prices_df[col_date] = pd.to_datetime(prices_df[col_date])
    prices_df = prices_df.rename(columns={col_date: ’datum’, col_price: ’prijs_excl_belastingen’})
    prices_df = prices_df.sort_values(’datum’)

except FileNotFoundError:
    sys.exit("Dynamic price file not found.")
except StopIteration:
    print("Available columns:", prices_df.columns.tolist())
    sys.exit("Could not find date/price columns in the price file.")

# Read temperature data
temp_df = pd.read_csv("C:/Users/julia/Downloads/2024-tempavg-weerverleden (1).csv")
temp_df[’DATUM’] = pd.to_datetime(temp_df[’DATUM’])
temp_df = temp_df.sort_values(’DATUM’)


def is_cleaning_time(h, d, m):
    """
    Determines if cleaning should occur based on hour, day, and month
    """
    parlor_cleaning = (h == 7 or h == 19)
    stable_cleaning = (d == 0 and h == 9)  # Monday at 9:00
    deep_cleaning = (d == 5 and h == 8 and m <= 7)  # First Saturday at 8:00

    return {
        'parlor': parlor_cleaning,
        'stable': stable_cleaning,
        'deep': deep_cleaning
    }


def water_demand(h, d, m):
    """Calculate water demand in L/min based on cleaning schedule"""
    cleaning = is_cleaning_time(h, d, m)

    if cleaning['deep']:
        return FLOW_DEEP
    elif cleaning['stable']:
        return FLOW_STABLE
    elif cleaning['parlor']:
        return FLOW_PARLOR
    else:
        return 0.0


# Model matrices and weights
A = 1 - hA * DT / (M * CP)
B = DT / (M * CP)
Q = 5  # temperature weight
BASE_R = 1e-9  # baseline effort weight
ALPHA = 1e-8  # price     effort weight (   /kWh)

# Control parameters
SOAK_PRICE = -0.5  #    /kWh threshold to start soaking
SOAK_LIMIT = 80.0  #  C  maximum soak temperature
T_REF = 60.0  #  C  reference temperature

# Riccati matrices
A_mat = np.array([[A]])
B_mat = np.array([[B]])
Q_mat = np.array([[Q]])


def riccati_gain(r_now: float) -> float:
    """Return optimal s t a t e feedback gain K for current R=r_now."""
    R_mat = np.array([[max(r_now, 1e-15)]])  # force R>0
    P = solve_discrete_are(A_mat, B_mat, Q_mat, R_mat)
    K = float(np.linalg.inv(R_mat + B_mat.T @ P @ B_mat) @ (B_mat.T @ P @
       A_mat))
    return K


def get_ambient_temp(hour_index):
    """Get ambient temperature for the given hour"""
    day_index = hour_index // 24
    return float(temp_df.iloc[day_index]['ETMAALGEMIDDELDE TEMPERATUUR'])


def get_price(hour_index, cumulative_usage_kwh, tariff_type='dynamic'):
    if tariff_type == 'dynamic':
        base_price = float(prices_df.iloc[hour_index]['
           prijs_excl_belastingen'])
        return calculate_total_price(base_price, cumulative_usage_kwh, tariff_type)
    else:
        return FLAT_RATE


def lqr_control_dynamic(T, price, hour_index, hour, weekday, month_day):
    """LQR control strategy optimized for dynamic pricing"""
    if price <= 0:
        r_now = BASE_R + ALPHA * FLAT_RATE * 2  # Double the effect of negative prices
    else:
        r_now = BASE_R + ALPHA * FLAT_RATE

    K = riccati_gain(r_now)
    T_target = SOAK_LIMIT if price <= SOAK_PRICE else T_REF
    err = T - T_target
    u = np.clip(-K * err, 0.0, BOIL_MAX)
    if T >= T_target:
        u = 0.0
    return u


def lqr_control_flat(T, hour_index, hour, weekday, month_day):
    """LQR control strategy optimized for flat rate pricing"""
    r_now = BASE_R + ALPHA * FLAT_RATE
    K = riccati_gain(r_now)
    err = T - T_REF  # Always use T_REF as target for flat rate
    u = np.clip(-K * err, 0.0, BOIL_MAX)
    if T >= T_REF:
        u = 0.0
    return u


def simulate_year(control_type='lqr', tariff_type='dynamic'):
    start_date = datetime(2024, 8, 1)
    days = 31
    hours_per_day = 24
    total_hours = days * hours_per_day

    energy_used = 0.0
    cost_total = 0.0
    T = T_SP
    T_log = np.zeros(total_hours)
    U_log = np.zeros(total_hours)

    for hour_index in range(total_hours):
        current_date = start_date + timedelta(hours=hour_index)
        hour = current_date.hour
        weekday = current_date.weekday()
        month_day = current_date.day

        # Get price based on tariff type and current usage
        price = get_price(hour_index, energy_used, tariff_type)

        # Use actual ambient temperature (daily average)
        T_AMB = get_ambient_temp(hour_index)

        flow = water_demand(hour, weekday, month_day)

        if control_type == ’lqr’:
            if tariff_type == ’dynamic’:
                control = lqr_control_dynamic(T, price, hour_index, hour, weekday, month_day)
            else:
                control = lqr_control_flat(T, hour_index, hour, weekday, month_day)

        Q_heat = flow * CP * (T_SP - T_IN) / 60.0 if flow > 0 else 0
        Q_loss = hA * (T - T_AMB)
        Q_total = Q_heat + Q_loss + Q_SPACE

        Q_applied = min(control, Q_total)

        dT = (Q_applied - Q_loss - Q_heat) / (M * CP) * 3600
        T += dT

        energy_hour = Q_applied * 1.0 / 1000.0  # Convert to kWh
        energy_used += energy_hour
        cost_total += energy_hour * price
        T_log[hour_index] = T
        U_log[hour_index] = control  # Convert normalized control to actual power (W)

    return energy_used, cost_total, T_log, U_log


def calculate_total_price(base_price, cumulative_usage_kwh, tariff_type=’dynamic’):
    if tariff_type == ’flat’:
        return FLAT_RATE  # Flat rate already includes everything

    # For dynamic pricing, add energy tax based on usage tier
    if cumulative_usage_kwh <= 2900:
        energy_tax = 0.10880
    elif cumulative_usage_kwh <= 10000:
        energy_tax = 0.10880
    elif cumulative_usage_kwh <= 50000:
        energy_tax = 0.09037
    else:
        energy_tax = 0.03943

    # Total price = base + energy tax (no VAT for dynamic)
    total = base_price + energy_tax

    return total


# Run simulations for all combinations
results = {}
strategies = [(’lqr’, ’flat’)]

for control, tariff in strategies:
    energy, cost, T_log, U_log = simulate_year(control, tariff)
    results[(control, tariff)] = (energy, cost)

```

```
# Print results
print("\n===== YEAR 2024 CLEANING SCHEDULE =====")
print("Daily: Parlor cleaning 2x (7:00 and 19:00)")
print("Weekly: Full stable cleaning (Mondays 9:00)")
print("Monthly: Deep cleaning (First Saturday 8:00)")

print("\n===== SIMULATION RESULTS =====")
for (control, tariff), (energy, cost) in results.items():
    print(f"\n{control.upper()} control with {tariff} tariff:")
    print(f"Energy Used: {energy:,.0f} kWh")
    print(f"Total Cost:    {cost:,.2f}")
    print(f"Average Daily Cost:    {(cost / 31):,.2f}")


# Build time axis
start_date = datetime(2024, 8, 1)
dates = [start_date + timedelta(hours=i) for i in range(len(T_log))]

# Plot temperature
plt.figure()
plt.plot(dates, T_log, label="Tank Temperature")
plt.axhline(T_REF, linestyle='--', label="Setpoint (T_REF)")
plt.title("Tank Temperature")
plt.ylabel("Temperature ( C )")
plt.xlabel("Day of Month")
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%d'))
plt.show()

# Plot heater power
plt.figure()
plt.plot(dates, U_log, label="Heater Power")
plt.title("Boiler Power")
plt.ylabel("Power (W)")
plt.xlabel("Date")
plt.axhline(Q_SPACE, linestyle='--', label="Base Load")
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%d'))
plt.show()
```

Listing 3: Python MPC code - dynamic rate and imbalance

```
import sys
from datetime import datetime, timedelta

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

from casadi import SX, vertcat, integrator, nlpsol
import casadi as ca

# File paths & global params
PRICE_CSV     = "C:/Users/julia/Downloads/
    jeroen_punt_nl_dynamische_stroomprijzen_jaar_2024.csv"
TEMP_CSV      = "C:/Users/julia/Downloads/2024-tempavg-weerverleden (1).csv
    "
IMBALANCE_CSV = "C:/Users/julia/Downloads/Imbalance_202401010000
    -202501010000.csv"

```

```
START      = datetime(2024,8,1)
DT         = 60                    # seconds per step (1 min)
HORIZON    = 15                    # prediction horizon (15 min)
STEPS      = 31*24*60               # total minutes in period

# Tank & boiler properties
M              = 10000.0          # kg water (10M3)
CP             = 4186.0           # J/(kg K)
hA             = 79.5              # W/K heat loss
BOIL_MAX       = 100000.0         # W max input
T_REF          = 60.0             #  C  comfort setpoint
T_SOAK         = 80.0             #  C  soak limit
T_MIN, T_MAX   = 50.0, 85.0       #  C  hard bounds
T_INLET        = 10.0             #  C  inlet water
default_load   = 1000.0           # W base space heating load

# Cleaning flows (L/min)
FLOW_PARLOR = 20.0
FLOW_STABLE = 30.0
FLOW_DEEP   = 40.0

Q_WT        = 0.5
PRICE_SOAK = -0.01
FLAT_RT     = 0.29
TX1, TR1    = 10e3, 0.10880
TX2, TR2    = 50e3, 0.09037
TR3         = 0.03943
IMB_WEIGHT = 5.0

# Data loading

def load_price():
    df = pd.read_csv(PRICE_CSV, sep=';', decimal=',')
    tcol = next(c for c in df if 'datum' in c.lower() or 'time' in c.lower
        ())
    pcol = next(c for c in df if 'prijs' in c.lower() or 'eur' in c.lower()
        )
    df[tcol] = pd.to_datetime(df[tcol])
    hourly = df.set_index(tcol)[pcol].astype(float)
    rng = pd.date_range(START, START+timedelta(days=366)-timedelta(hours=1)
        , freq='1h')
    hourly = hourly.reindex(rng).ffill()
    return hourly.resample('1min').ffill().to_numpy()


def load_ambient():
    df = pd.read_csv(TEMP_CSV, sep=',', decimal='.')
    tcol = next(c for c in df if 'datum' in c.lower() or 'date' in c.lower
        ())
    tmpc = next(c for c in df if 'gem' in c.lower() or 'temp' in c.lower())
    df[tcol] = pd.to_datetime(df[tcol])
    daily = df.set_index(tcol)[tmpc].astype(float)
    rng = pd.date_range(START, START+timedelta(days=366)-timedelta(days=1),
        freq='D')
    daily = daily.reindex(rng).ffill()
    return daily.resample('1min').ffill().to_numpy()


```

```python
def load_imbalance():
    df = pd.read_csv(IMBALANCE_CSV, sep=',', decimal='.')
    tcol   = next(c for c in df if 'imbalance settlement period' in c.lower
        () or 'date' in c.lower())
    imbcol = next(c for c in df if 'imbalance price' in c.lower())
    df[tcol] = df[tcol].str.split(' - ').str[0]
    df[tcol] = pd.to_datetime(df[tcol], format='%d.%m.%Y %H:%M')
    # convert EUR/MWh to EUR/kWh and build series
    qh = df.set_index(tcol)[imbcol].astype(float) / 1000
    # drop duplicate timestamps to avoid reindex errors
    qh = qh[~qh.index.duplicated(keep='first')]
    # create full 15-min index and forward-fill
    rng = pd.date_range(START, START + timedelta(days=366) - timedelta(
       minutes=15), freq='15min')
    qh = qh.reindex(rng).ffill()
    # up-sample to 1-min resolution by forward-fill
    return qh.resample('1min').ffill().to_numpy()

price_min  = load_price() * 1.21        # Add VAT to the electricity price
T_amb_min  = load_ambient()
imb_min    = load_imbalance()

def r_wrt_price(price):
    return np.maximum(0, 1e-8 * price)

# Precompute cleaning load as heat draw (W)
Q_draw = np.zeros(STEPS)
for i in range(STEPS):
    now = START + timedelta(minutes=i)
    h, wd, day = now.hour, now.weekday(), now.day
    flow = 0.0
    if h in (7,19) and now.minute < 30:         flow = FLOW_PARLOR
    elif wd==0 and h==9  and now.minute<120:    flow = FLOW_STABLE
    elif wd==5 and day<=7 and h==8 and now.minute<180: flow = FLOW_DEEP
    Q_draw[i] = flow * CP * (T_REF - T_INLET)/60 + default_load

# -- Build two MPC solvers once --#
# 1) Spot-only baseline solver
delta = SX.sym('delta')  # placeholder to avoid unused warning
U_bas    = SX.sym('U_bas', HORIZON)
X0       = SX.sym('X0')
Q_seq    = SX.sym('Q_seq',    HORIZON)
Tamb_seq = SX.sym('Tamb_seq', HORIZON)
R_seq    = SX.sym('R_seq',    HORIZON)
Spot_seq = SX.sym('Spot_seq', HORIZON)

J_bas = 0; g_bas = []
Xk = X0
for k in range(HORIZON):
    dT = (U_bas[k] - hA*(Xk-Tamb_seq[k]) - Q_seq[k])/(M*CP)
    Xk = Xk + dT*DT
    tgt = ca.if_else(Spot_seq[k] <= PRICE_SOAK, T_SOAK, T_REF)
    J_bas += Q_WT*(Xk-tgt)**2 + R_seq[k]*U_bas[k]**2
    g_bas += [Xk - T_MIN, T_MAX - Xk]

nlp_bas = {'x':U_bas, 'p':vertcat(X0, Q_seq, Tamb_seq, R_seq, Spot_seq),
           'f':J_bas, 'g':vertcat(*g_bas)}
solver_bas = nlpsol('solver_bas','ipopt', nlp_bas,
                    {'ipopt':{'print_level':0,'max_iter':30,'tol':1e-3},'
                       print_time':0})

# 2) Imbalance deviation solver (reuses Q_seq, Tamb_seq, R_seq, Spot_seq)
U_seq     = SX.sym('U_seq',     HORIZON)
Imb_seq   = SX.sym('Imb_seq',   HORIZON)
U_ref     = SX.sym('U_ref',     HORIZON)
W_sym     = SX.sym('W')    # imbalance weight parameter

J_imb = 0; g_imb = []
Xk = X0
for k in range(HORIZON):
    dT = (U_seq[k] - hA*(Xk-Tamb_seq[k]) - Q_seq[k])/(M*CP)
    Xk = Xk + dT*DT
    # dynamic target: soak when imbalance price is negative, otherwise use
       spot-soak logic
    tgt = ca.if_else(
        Imb_seq[k] < 0,
        T_SOAK,
        ca.if_else(Spot_seq[k] <= PRICE_SOAK, T_SOAK, T_REF)
    )
    J_imb += Q_WT*(Xk-tgt)**2 + R_seq[k]*U_seq[k]**2
    # deviation term: reward/penalty for imbalance deviations
    dev = U_seq[k] - U_ref[k]
    # weighted imbalance revenue
    J_imb += -W_sym * Imb_seq[k] * dev * (DT/3_600_000)
    g_imb += [Xk - T_MIN, T_MAX - Xk]

# pack parameters: include weight at end
nlp_imb = {
    'x': U_seq,
    'p': vertcat(
        X0,
        Q_seq, Tamb_seq, R_seq,
        Spot_seq, Imb_seq, U_ref,
        W_sym
    ),
    'f': J_imb,
    'g': vertcat(*g_imb)
}
solver_imb = nlpsol('solver_imb','ipopt', nlp_imb,
                    {'ipopt':{'print_level':0,'max_iter':30,'tol':1e-3},'
                       print_time':0})

# MPC loop
T_now      = T_REF
kwh_tot    = 0
cost_spot = 0     # cumulative spot-market cost
cost_imb   = 0     # cumulative imbalance-market profit/penalty
cost_tot   = 0     # net cost (spot + imbalance)
flat_cost = 0     # cost when using flat rate
T_log      = np.zeros(STEPS)
U_log      = np.zeros(STEPS)

valid_steps = min(STEPS-HORIZON, len(T_amb_min), len(Q_draw), len(imb_min))

# --- Initial baseline compute at i=0 ---
Q0    = Q_draw[0:HORIZON]
```

```
Tamb0= T_amb_min[0:HORIZON]
R0   = r_wrt_price(price_min[0: HORIZON])
Spot0= price_min   [0: HORIZON]
p0   = np.concatenate([[T_now], Q0, Tamb0, R0, Spot0])
# bounds
lbx0 = [0.0]*HORIZON; ubx0 = [BOIL_MAX]*HORIZON
n_constr0 = 2*HORIZON
lbg0 = [0.0]*n_constr0; ubg0 = [ca.inf]*n_constr0
sol0 = solver_bas(x0=np.zeros(HORIZON), p=p0, lbx=lbx0, ubx=ubx0, lbg=lbg0,
     ubg=ubg0)
u_ref_seq = np.array(sol0['x']).flatten()
# Predefine deviation bounds and constraint bounds for imbalance solver
lbx_dev  = [0.0] * HORIZON
ubx_dev  = [BOIL_MAX] * HORIZON
n_constr = 2 * HORIZON
lbg_imb  = [0.0] * n_constr
ubg_imb  = [ca.inf] * n_constr

for i in range(valid_steps):
    # Recompute baseline at market gate (every HORIZON steps)
    if i % HORIZON == 0 and i != 0:
        # new baseline for next block
        Qb    = Q_draw[i:i+HORIZON]
        Tambb= T_amb_min[i:i+HORIZON]
        Rb    = r_wrt_price(price_min[i:i+HORIZON])
        Spotb= price_min   [i:i+HORIZON]
        pb    = np.concatenate([[T_now], Qb, Tambb, Rb, Spotb])
        solb = solver_bas(x0=np.zeros(HORIZON), p=pb, lbx=lbx0, ubx=ubx0,
            lbg=lbg0, ubg=ubg0)
        u_ref_seq = np.array(solb['x']).flatten()

    # -- deviation solve --
    Q_h    = Q_draw[i:i+HORIZON]
    Tamb   = T_amb_min[i:i+HORIZON]
    R_h    = r_wrt_price(price_min[i:i+HORIZON])
    Spot   = price_min[i:i+HORIZON]
    Imb    = imb_min[i:i+HORIZON]
    p_imb = np.concatenate([
        [T_now], Q_h, Tamb, R_h,
        Spot, Imb, u_ref_seq,
        [IMB_WEIGHT]  # pass weight parameter W_sym
    ])
    # enforce control bounds and constraints (predefined)
    sol_imb = solver_imb(
        x0=u_ref_seq, p=p_imb,
        lbx=lbx_dev, ubx=ubx_dev,
        lbg=lbg_imb, ubg=ubg_imb
    )
    u_seq = np.array(sol_imb['x']).flatten()
    u_cmd = u_seq[0]

    # step & log
    T_now += (u_cmd - hA*(T_now-T_amb_min[i]) - Q_draw[i])/(M*CP)*DT
    T_log[i] = T_now
    U_log[i] = u_cmd

    # cost accounting
    kwh       = u_cmd * DT / 3_600_000
```

```python
    kwh_tot += kwh
    # spot-market cost
    tax         = TR1 if kwh_tot<=TX1 else TR2 if kwh_tot<=TX2 else TR3
    spot_cost   = kwh * (price_min[i] + tax)
    cost_spot += spot_cost
    # imbalance-market profit/penalty
    imb_cost    = imb_min[i] * (u_cmd - u_ref_seq[0]) * (DT/3_600_000)
    cost_imb   += imb_cost
    # total
    cost_tot += spot_cost - imb_cost


# Plotting & results
print(f"Energy: {kwh_tot:,.0f} kWh | Cost:    {cost_tot:,.2f}")
print(f"Total spot-market cost:    {cost_spot:.2f}")
print(f"Total imbalance-market P&L:    {cost_imb:.2f}")
print(f"Net cost:    {cost_tot:.2f}")
idx   = np.arange(0, valid_steps, 15)
dates = [START+timedelta(minutes=int(i)) for i in idx]

# Plot tank temperature
plt.plot(dates, T_log[idx])
plt.axhline(T_REF, ls='--')
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%d'))
plt.title("Tank Temperature")
plt.ylabel("Temperature ( C )")
plt.xlabel("Day of Month")
plt.show()

# Plot boiler power
plt.plot(dates, U_log[idx])
plt.axhline(1000, ls='--')
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%d'))
plt.title("Boiler Power")
plt.ylabel("Power (W)")
plt.xlabel("Day of Month")
plt.show()
```

Listing 4: Python MPC code - flat rate

```python
import sys
from datetime import datetime, timedelta

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

from casadi import SX, vertcat, integrator, nlpsol
import casadi as ca

# File paths & global params
PRICE_CSV     = "C:/Users/julia/Downloads/jeroen_punt_nl_dynamische_stroomprijzen_jaar_2024.csv"
TEMP_CSV      = "C:/Users/julia/Downloads/2024-tempavg-weerverleden (1).csv"
IMBALANCE_CSV = "C:/Users/julia/Downloads/Imbalance_202401010000-202501010000.csv"

```

```python
START       = datetime(2024,8,1)
DT          = 60                  # seconds per step (1 min)
HORIZON     = 15                  # prediction horizon (15 min)
STEPS       = 31*24*60             # total minutes in period

# Tank & boiler properties
M               = 10000.0          # kg water (10M3)
CP              = 4186.0           # J/(kg K)
hA              = 79.5             # W/K heat loss
BOIL_MAX        = 100000.0         # W max input
T_REF           = 60.0             #  C  comfort setpoint
T_SOAK          = 80.0             #  C  soak limit
T_MIN, T_MAX  = 60.0, 85.0         #  C  hard bounds
T_INLET         = 10.0             #  C  inlet water
default_load  = 1000.0             # W base space heating load

# Cleaning flows (L/min)
FLOW_PARLOR = 20.0
FLOW_STABLE = 30.0
FLOW_DEEP   = 40.0

Q_WT        = 5
PRICE_SOAK = -0.01
FLAT_RT     = 0.29
TX1, TR1    = 10e3, 0.10880
TX2, TR2    = 50e3, 0.09037
TR3         = 0.03943
IMB_WEIGHT = 5.0

# Data loading

def load_price():
    df = pd.read_csv(PRICE_CSV, sep=';', decimal=',')
    tcol = next(c for c in df if 'datum' in c.lower() or 'time' in c.lower
        ())
    pcol = next(c for c in df if 'prijs' in c.lower() or 'eur' in c.lower()
        )
    df[tcol] = pd.to_datetime(df[tcol])
    hourly = df.set_index(tcol)[pcol].astype(float)
    rng = pd.date_range(START, START+timedelta(days=366)-timedelta(hours=1)
        , freq='1h')
    hourly = hourly.reindex(rng).ffill()
    return hourly.resample('1min').ffill().to_numpy()


def load_ambient():
    df = pd.read_csv(TEMP_CSV, sep=',', decimal='.')
    tcol = next(c for c in df if 'datum' in c.lower() or 'date' in c.lower
        ())
    tmpc = next(c for c in df if 'gem' in c.lower() or 'temp' in c.lower())
    df[tcol] = pd.to_datetime(df[tcol])
    daily = df.set_index(tcol)[tmpc].astype(float)
    rng = pd.date_range(START, START+timedelta(days=366)-timedelta(days=1),
        freq='D')
    daily = daily.reindex(rng).ffill()
    return daily.resample('1min').ffill().to_numpy()


```

```python
def load_imbalance():
    df = pd.read_csv(IMBALANCE_CSV, sep=',', decimal='.')
    tcol   = next(c for c in df if 'imbalance settlement period' in c.lower() or 'date' in c.lower())
    imbcol = next(c for c in df if 'imbalance price' in c.lower())
    df[tcol] = df[tcol].str.split(' - ').str[0]
    df[tcol] = pd.to_datetime(df[tcol], format='%d.%m.%Y %H:%M')
    # convert EUR/MWh to EUR/kWh and build series
    qh = df.set_index(tcol)[imbcol].astype(float) / 1000
    # drop duplicate timestamps to avoid reindex errors
    qh = qh[~qh.index.duplicated(keep='first')]
    # create full 15-min index and forward-fill
    rng = pd.date_range(START, START + timedelta(days=366) - timedelta(minutes=15), freq='15min')
    qh = qh.reindex(rng).ffill()
    # up-sample to 1-min resolution by forward-fill
    return qh.resample('1min').ffill().to_numpy()

price_min  = np.full(STEPS, FLAT_RT)
T_amb_min  = load_ambient()
imb_min    = load_imbalance()

def r_wrt_price(price):
    return np.maximum(0, 1e-15 * price)

# Precompute cleaning load as heat draw (W)
Q_draw = np.zeros(STEPS)
for i in range(STEPS):
    now = START + timedelta(minutes=i)
    h, wd, day = now.hour, now.weekday(), now.day
    flow = 0.0
    if h in (7,19) and now.minute < 30:        flow = FLOW_PARLOR
    elif wd==0 and h==9  and now.minute<120:   flow = FLOW_STABLE
    elif wd==5 and day<=7 and h==8 and now.minute<180: flow = FLOW_DEEP
    Q_draw[i] = flow * CP * (T_REF - T_INLET)/60 + default_load

# -- Build two MPC solvers once --#
# 1) Spot-only baseline solver
delta = SX.sym('delta')  # placeholder to avoid unused warning
U_bas    = SX.sym('U_bas', HORIZON)
X0       = SX.sym('X0')
Q_seq    = SX.sym('Q_seq',    HORIZON)
Tamb_seq = SX.sym('Tamb_seq', HORIZON)
R_seq    = SX.sym('R_seq',    HORIZON)
Spot_seq = SX.sym('Spot_seq', HORIZON)

J_bas = 0; g_bas = []
Xk = X0
for k in range(HORIZON):
    dT = (U_bas[k] - hA*(Xk-Tamb_seq[k]) - Q_seq[k])/(M*CP)
    Xk = Xk + dT*DT
    tgt = T_REF
    dev = Xk-tgt
    J_bas += Q_WT*dev**2 + R_seq[k]*U_bas[k]**2
    g_bas += [Xk - T_MIN, T_MAX - Xk]

nlp_bas = {'x':U_bas, 'p':vertcat(X0, Q_seq, Tamb_seq, R_seq, Spot_seq),
           'f':J_bas, 'g':vertcat(*g_bas)}
```

```python
solver_bas = nlpsol('solver_bas','ipopt', nlp_bas,
                     {'ipopt':{'print_level':0,'max_iter':30,'tol':1e-3},'
                        print_time':0})

# 2) Imbalance deviation solver (reuses Q_seq, Tamb_seq, R_seq, Spot_seq)
#U_seq     = SX.sym('U_seq',     HORIZON)
#Imb_seq   = SX.sym('Imb_seq',   HORIZON)
#U_ref     = SX.sym('U_ref',     HORIZON)
#W_sym     = SX.sym('W')    # imbalance weight parameter

#J_imb = 0; g_imb = []
#Xk = X0
#for k in range(HORIZON):
 #    dT = (U_seq[k] - hA*(Xk-Tamb_seq[k]) - Q_seq[k])/(M*CP)
 #    Xk = Xk + dT*DT
 #    # dynamic target: soak when imbalance price is negative, otherwise use
      spot-soak logic
 #   tgt = ca.if_else(
 #        Imb_seq[k] < 0,
 #        T_SOAK,
  #       ca.if_else(Spot_seq[k] <= PRICE_SOAK, T_SOAK, T_REF)
  #   )
  #   J_imb += Q_WT*(Xk-tgt)**2 + R_seq[k]*U_seq[k]**2
  #   # deviation term: reward/penalty for imbalance deviations
  #   dev = U_seq[k] - U_ref[k]
    # weighted imbalance revenue
  #   J_imb += -W_sym * Imb_seq[k] * dev * (DT/3_600_000)
  #   g_imb += [Xk - T_MIN, T_MAX - Xk]

# pack parameters: include weight at end
#nlp_imb = {
#     'x': U_seq,
#     'p': vertcat(
#         X0,
#        Q_seq, Tamb_seq, R_seq,
 #        Spot_seq, Imb_seq, U_ref,
 #        W_sym
 #    ),
 #    'f': J_imb,
 #    'g': vertcat(*g_imb)
#}
#solver_imb = nlpsol('solver_imb','ipopt', nlp_imb,
              #       {'ipopt':{'print_level':0,'max_iter':30,'tol':1e-3},'
                 print_time':0})

# MPC loop
T_now      = T_REF
kwh_tot    = 0
cost_spot = 0     # cumulative spot-market cost
cost_imb   = 0     # cumulative imbalance-market profit/penalty
cost_tot   = 0     # net cost (spot + imbalance)
flat_cost = 0     # cost when using flat rate
T_log      = np.zeros(STEPS)
U_log      = np.zeros(STEPS)

valid_steps = min(STEPS-HORIZON, len(T_amb_min), len(Q_draw), len(imb_min))

# --- Initial baseline compute at i=0 ---
```

```python
Q0   = Q_draw[0:HORIZON]
Tamb0= T_amb_min[0:HORIZON]
R0   = r_wrt_price(price_min[0: HORIZON])
Spot0= price_min   [0: HORIZON]
p0   = np.concatenate([[T_now], Q0, Tamb0, R0, Spot0])
# bounds
lbx0 = [0.0]*HORIZON; ubx0 = [BOIL_MAX]*HORIZON
n_constr0 = 2*HORIZON
lbg0 = [0.0]*n_constr0; ubg0 = [ca.inf]*n_constr0
sol0 = solver_bas(x0=np.zeros(HORIZON), p=p0, lbx=lbx0, ubx=ubx0, lbg=lbg0, ubg=ubg0)
u_ref_seq = np.array(sol0['x']).flatten()
# Predefine deviation bounds and constraint bounds for imbalance solver
lbx_dev  = [0.0] * HORIZON
ubx_dev  = [BOIL_MAX] * HORIZON
n_constr = 2 * HORIZON
lbg_imb  = [0.0] * n_constr
ubg_imb  = [ca.inf] * n_constr

for i in range(valid_steps):
    # Recompute baseline at market gate (every HORIZON steps)
    if i % HORIZON == 0 and i != 0:
        # new baseline for next block
        Qb   = Q_draw[i:i+HORIZON]
        Tambb= T_amb_min[i:i+HORIZON]
        Rb   = r_wrt_price(price_min[i:i+HORIZON])
        Spotb= price_min   [i:i+HORIZON]
        pb   = np.concatenate([[T_now], Qb, Tambb, Rb, Spotb])
        solb = solver_bas(x0=np.zeros(HORIZON), p=pb, lbx=lbx0, ubx=ubx0, lbg=lbg0, ubg=ubg0)
        u_ref_seq = np.array(solb['x']).flatten()
    u_cmd = u_ref_seq[0]

    # -- deviation solve --
   # Q_h   = Q_draw[i:i+HORIZON]
   # Tamb  = T_amb_min[i:i+HORIZON]
   # R_h   = r_wrt_price(price_min[i:i+HORIZON])
   # Spot  = price_min[i:i+HORIZON]
   # Imb   = imb_min[i:i+HORIZON]
   # p_imb = np.concatenate([
    #    [T_now], Q_h, Tamb, R_h,
    #    Spot, Imb, u_ref_seq,
    #    [IMB_WEIGHT]  # pass weight parameter W_sym
   # ])
    # enforce control bounds and constraints (predefined)
   # sol_imb = solver_imb(
   #     x0=u_ref_seq, p=p_imb,
    #    lbx=lbx_dev, ubx=ubx_dev,
    #    lbg=lbg_imb, ubg=ubg_imb
    #)
    #u_seq = np.array(sol_imb['x']).flatten()
    #u_cmd = u_seq[0]

    # step & log
    T_now += (u_cmd - hA*(T_now-T_amb_min[i]) - Q_draw[i])/(M*CP)*DT
    T_log[i] = T_now
    U_log[i] = u_cmd

    # cost accounting
    kwh       = u_cmd * DT / 3_600_000
    kwh_tot += kwh
    # spot-market cost
    #tax        = TR1 if kwh_tot<=TX1 else TR2 if kwh_tot<=TX2 else TR3
    spot_cost  = kwh * (price_min[i]) #+ tax)
    cost_spot += spot_cost
    # imbalance-market profit/penalty
    #imb_cost   = imb_min[i] * (u_cmd - u_ref_seq[0]) * (DT/3_600_000)
    #cost_imb  += imb_cost
    # total
    cost_tot += spot_cost #- imb_cost
    #cost_flat = kwh * FLAT_RT
    #flat_cost += cost_flat

# Plotting & results
print(f"Energy: {kwh_tot:,.0f} kWh | Cost:    {cost_tot:,.2f}")
#print(f"Total spot-market cost:    {cost_spot:.2f}")
#print(f"Total imbalance-market P&L:    {cost_imb:.2f}")
#print(f"Net cost:    {cost_tot:.2f}")
#print(f"Cost when using flat rate (  0  .28/KWh):    {flat_cost:.2f}")
idx   = np.arange(0, valid_steps, 15)
dates = [START+timedelta(minutes=int(i)) for i in idx]

# Plot tank temperature
plt.plot(dates, T_log[idx])
plt.axhline(T_REF, ls='--')
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%d'))
plt.title("Tank Temperature")
plt.ylabel("Temperature ( C )")
plt.xlabel("Day of Month")
plt.show()

# Plot boiler power
plt.plot(dates, U_log[idx])
plt.axhline(1000, ls='--')
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%d'))
plt.title("Boiler Power")
plt.ylabel("Power (W)")
plt.xlabel("Day of Month")
plt.show()
```