ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA ELEKTROTECHNICKÁ

Artificial Intelligence Based Intelligent Thermostat

Inteligentní termostat využívající metod umělé inteligence

DIPLOMOVÁ PRÁCE

2018 Martin Procházka

CTU
CZECH TECHNICAL UNIVERSITY IN PRAGUE

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Procházka Martin**	Personal ID number:	**420153**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Control Engineering**		
Study program:	**Cybernetics and Robotics**		
Branch of study:	**Cybernetics and Robotics**		

II. Master's thesis details

Master's thesis title in English:

Artifical intelligence based intelligent thermostat

Master's thesis title in Czech:

Inteligentní termostat využívající metod umělé inteligence

Guidelines:

1. Study the reinforcement learning method used in Artificial Intelligence, research the model-less optimal control design methods.
2. Validate the control performance of intelligent, self-learnining thermostats based on Q-learning approach using simulation model of a building with various properties (thermal innertia, insulation etc.), evaluate applicability of proposed methods for commercial thermostat applications,
3. Estimate the benefits resulting from improved energy efficiency of domestic heating due to improved controller tuning.

Bibliography / sources:

[1] R. S. Sutton a A. G. Barto, Reinforcement Learning, The MIT Press, 1998 (několik reedic).
[2] M. Wiering, M. van Otterlo (Eds)., Reinforcement Learning State-of-the-Art, Springer, 2012.
[3] C. E. Rasmussen, C. K. I. Williams, Gaussian Processes for Machine Learning, the MIT Press, 2006

Name and workplace of master's thesis supervisor:

prof. Ing. Vladimír Havlena, CSc., Department of Control Engineering, FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **27.03.2018** Deadline for master's thesis submission: **25.05.2018**

Assignment valid until: **30.09.2019**

prof. Ing. Vladimír Havlena, CSc.	prof. Ing. Michael Šebek, DrSc.	prof. Ing. Pavel Ripka, CSc.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.

Date of assignment receipt Student's signature

Declaration

Abstract

This thesis utilizes reinforcement learning, particularly Q-learning, to develop a self-learning algorithm for an intelligent thermostat. The algorithm is able to find an optimal controller solely based on provided data. The data available for learning include room temperature, reference temperature, outside temperature, and applied input (heat flow generated by the boiler). The research part of this work explains in depth the principle of Q-learning and shows on several examples the learning process itself. The advantage of using Q-learning is its capability to incorporate a priori known information. In our case, it can be shown that the Q-function learned by the algorithm will be quadratic and positive definite, which substantially reduces the required amount of training data. The Q-learning approach is also able to deal with constraints introduced by the boiler if merged with MPC. The final chapter includes several experiments verifying the three major developed techniques (controllers comparison, optimal controller generation and merging Q-learning and MPC) on a realistic thermal model.

Keywords:

Control theory; optimal control; adaptive control; MPC; reinforcement learning, Q-learning

Abstrakt

Tato práce používá zpětnovazební učení, konkrétně Q-learning, k vytvoření algoritmu pro inteligentní termostat. Algoritmus je schopen nalézt optimální regulátor pouze na základě poskytnutých dat. Data, která jsou k dispozici pro učení, zahrnují pokojovou teplotu, referenční teplotu, venkovní teplotu a aplikovaný vstup (tepelný tok generovaný kotlem). Výzkumná část této práce podrobně vysvětluje princip Q-learningu a ukazuje na několika příkladech samotný proces učení. Výhodou použití Q-learningu je jeho schopnost využít předem známé informace. V našem případě lze dokázat, že Q-funkce nalezená algoritmem bude kvadratická a pozitivně definitní, což podstatně snižuje požadované množství trénovacích dat. Q-learning je také schopen se vyrovnat omezeními způsobenými kotlem, pokud je sloučen s prediktivním řízením (MPC). Závěrečná kapitola obsahuje několik experimentů ověřujících tři hlavní vyvinuté techniky (porovnání reglátorů, generování optimálních regulátorů a sloučení Q-learningu a MPC) na realistickém modelu.

Klíčová slova:

Teorie řízení, optimální řízení, adaptivní řízení, prediktivní řízení (MPC), zpětnovazební učení, Q-learning

Acknowledgments

I would like to thank my supervisor prof. Ing. Vladimír Havlena, CSc. for his guidance, support and constructive criticism during consultations. I would also like to thank all the members of staff at Honeywell Prague Laboratory and everyone who has supported me during my work on this thesis.

Contents

1 Introduction

Modeling dynamical systems using the first principles for later controller design is long and tedious work. First, a set of differential and algebraic equations have to be created with respect to physical laws describing the dynamical system in question. Those equations include loads of constants that need to be estimated based on data measured on the real system. This is where the main complication comes. Because of transportation delays and long time constants of some systems, it is difficult to obtain sufficient amount of data in a reasonable amount of time to accurately estimate constants in the model. This problem combined with many others resulted in the search for an alternative approach.

The first significant breakthrough was achieved by using adaptive controllers, which could be described as dynamic systems with on-line parameter estimation. Adaptive control was first used in the 1950's to design autopilots for high-performance aircraft. Since then, the theory was found useful in lots of other fields. Self-Tuning Controllers (STC) as a part of the adaptive control theory were found to be a suitable substitution for the modeling of dynamic systems. STCs simultaneously identify parameters and design controllers accordingly. [3, 4]

Parameter estimation usually utilizes the black box approach, where model parameters are estimated based on input and output data. The model structure can take several forms. Auto regressive models (ARX – Auto Regressive model with eXternal input, ARMAX – Auto Regressive Moving Average model with eXternal input, OE – Output Error model, ...) are one of the simplest, yet often sufficient model structures used in system identification. Those models have a simple transfer function form and work under the assumption that the current output value depends on finite number of previous outputs, inputs and some stochastic term $e(t)$. [3]

The role of classical adaptive control was later taken over by artificial intelligence, especially by Neural Networks (NN). Various neural network architectures with differing transfer function types are used to learn the system dynamics and even to generate appropriate control action. Application of NNs in this manner can be seen for example in *P. Henaff, M. Milgram, J. Rabit (1994)*, *C. Chen, Z. Liu, K. Xie, Y. Zhang, C. L. P. Chen (2017)* or *H. Leeghim, D. Kim (2015)*. Although the neural network approach undoubtedly works, it has several properties that could be seen as problematic. One of those properties is the black-box approach, which does not allow us to see how the training and prediction itself works. Another problem of NNs is a large amount of data required for training. [4]

The Q-learning method used in this thesis does not suffer from those problems. The research part of this work explains in depth the principle of Q-learning and shows on several examples the learning process itself. Q-learning is also capable of incorporating a priori known information. In our case, it can be shown that the Q-function learned by the algorithm is quadratic and positive definite, which substantially reduces the required amount of training data.

The goal of this thesis is to develop a self-learning algorithm for an intelligent thermostat. The algorithm should be able to find an optimal controller based on the provided data. The data available for learning include room temperature, reference temperature, outside temperature, and applied input (heat flow generated by the boiler).

The first chapter explores and describes the already known theory of optimal control and Q-learning, which was found useful for later use. It mostly includes a description of Q-learning, machine learning in general and Model Predictive Control (MPC). Then, the problem of developing the self-learning thermostat with all limitations and constraints is introduced. This chapter also contains a short general description of the controlled system. The next section introduces the realistic thermal model, which is later used to evaluate and test all major results of this thesis. The following chapter is related to the first one as it shows how were the already known theoretical results modified, combined and applied to the problem to derive the three main techniques of this thesis: Controllers comparison, optimal controller generation and merging Q-learning and MPC. This section also includes a solution to a series of partial problems that had to be addressed. The chapter ends with several tests proving the concept, justifying the use of Q-learning, showing its advantages and exposing few complications. The final chapter includes several experiments verifying the three major developed techniques on the thermal model. The evaluation of achieved results is summarized in conclusion.

2 Research

This chapter describes the ideas, techniques, and methods used later in the text as a solution to given problems. Those solutions are heavily based on Q-learning and reinforcement learning in general. Therefore, those techniques will be described more in detail. The chapter ends with the description of the Model Predictive Control (MPC).

2.1 Reinforcement Learning

Reinforcement learning is a type of machine learning inspired by learning in nature. Animals are given some rewards or punishments from the environment and modify their behavioral strategies accordingly. From the control engineering point of view, it can be classified as both optimal and adaptive control, since given sufficient amount of exploration data, the reinforcement learning methods converge to an optimal control law without knowing the system dynamics beforehand [2].

2.1.1 Markov Decision Processes

Markov decision processes (MDP) are used when a decision-making situation has to be modeled in a way that in a state x, the decision maker has several inputs[1] (actions) u to choose from, and after applying one of them, the system randomly moves to a state x'. The Markov decision process is defined as a (X, U, P, R), where X is the set of all states and U is the set of all inputs. P or $P^u_{x,x'}$ are the transition probabilities for every state $x \in X$ and input $u \in U$ of transitioning from x to $x' \in X$. R or $R^u_{x,x'}$ is the reward gained after transitioning from $x \in X$ to $x' \in X$ using the input $u \in U$. If sets X and U are finite, then it is called a finite Markov decision process. Most of the reinforcement learning theory was developed for the finite Markov decision processes framework. Therefore, using this theory for continuous systems with an infinite state and input space may (and will) cause some problems, which will be described as they come later in the text [1, 2].

2.1.2 Bellman Optimality Equation

To explain the reinforcement learning in the most understandable way, it will be first explained on a simple example, and generalizations will come later.

Suppose first that we have a deterministic discrete time-invariant model, which can be described by a set of linear difference equations written in a simple form as

$$x_{t+1} = f(x_t, u_t), \tag{1}$$

where x is the system state, and u is the input to the system. The model can be seen as a special case of the MDP, where the transition probabilities are replaced by a transition law (1).

[1]The terminology and labeling used for the values to be applied to a system differ from field to field. Control engineers use “inputs u”, but “actions a” are used in machine learning and artificial intelligence in general. Since this thesis utilizes machine learning methods for control purposes, the “inputs u” notation will be used from now on.

Now a cost[2] $r(x_t, u_t)$ can be defined as a function of x and u at time t, which is the expected immediate cost paid after a transition from x_t to x_{t+1} using the input u_t. How exactly does r depend on x_t and u_t will be defined later.

Next, a deterministic policy has to be defined. Suppose X is the set of all states and U is the set of all inputs. Policy $u = \pi(x)$ then defines which u from U should be applied when the system is in state x for every $x \in X$.

The value of a policy π as a function of x_t is defined as a sum of all future costs when starting in state x_t and using the input u given by policy π from now on

$$V^{\pi}(x_t) = \sum_{k=0}^{\infty} \gamma^k r(x_{t+k}, u^{\pi}_{t+k}). \tag{2}$$

Notice that the future costs are reduced by the discount factor γ ($0 \leq \gamma < 1$). The optimal policy π^* is defined as

$$\pi^*(x_t) = \arg\min_{\pi} V^{\pi}(x_t) \tag{3}$$

and its corresponding optimal value is defined as

$$V^*(x_t) = \min_{\pi} V^{\pi}(x_t). \tag{4}$$

Since the value of a policy π is defined as a sum, equation (2) can also be written as

$$V^{\pi}(x_t) = r(x_t, u^{\pi}_t) + \gamma \sum_{k=1}^{\infty} \gamma^k r(x_{t+k}, u^{\pi}_{t+k}), \tag{5}$$

$$V^{\pi}(x_t) = r(x_t, u^{\pi}_t) + \gamma V^{\pi}(x_{t+1}). \tag{6}$$

Equation (6) is called the Bellman equation. Using (3),(4) and (6), the Bellman optimality equations can be derived

$$V^*(x_t) = \min_{u_t} \left[r(x_t, u_t) + \gamma V^*(x_{t+1}) \right], \tag{7}$$

$$\pi^*(x_t) = \arg\min_{u_t} \left[r(x_t, u_t) + \gamma V^*(x_{t+1}) \right]. \tag{8}$$

If the finite MDP framework is considered, the function V can be represented as an n-dimensional vector (since the state and input spaces are finite and discrete), where n is the dimension of the vector x_t. n can also be called the order of the system. On the other hand, if the state and input spaces were to be infinite and continuous, the V function would then be represented by a n-dimensional continuous function. This generalization makes the identification of such function way more complex, and some reinforcement learning methods cannot be used at all.

[2]Typical problems solved by artificial intelligence are defined so that the goal is to maximize a reward (labeled as r). On the contrary, typical control engineering problem is designed in a way that the goal is to minimize certain cost. Since those concepts are to some extent similar, the labeling from the artificial intelligence persists even in this context.

2.1.3 Dynamic Programming

Dynamic programming is closely related to the Bellman equation and Bellman optimality equation. It is a backward-in-time offline method for finding an optimal policy for given problem. This approach requires knowledge of the system dynamics and expected costs.

Since the method runs offline backward in time, some finite horizon h has to be set. Therefore, the optimization will be performed on a finite set of time steps between times t and $t+h$. Since the time $t+h$ is "last" for the optimization, a reward $r_h(x_{t+h})$ depending only on the state x_{t+h} at time $t+h$ has to be defined.

As stated before, dynamic programming runs backward in time and uses the Bellman equation, therefore, the value has to be computed at time $t+h$ first. Since there is no "$t+1$" time, the equation (7) for time $t+h$ can be written as

$$V(x_{t+h}) = r_h(x_{t+h}). \tag{9}$$

The value is computed as some function of x_{t+h} for every $x \in X$. Then the algorithm runs backward in time computing optimal input u^* and value $V(x_{t+k})$ for each state $x \in X$ according to the Bellman optimality equation (10) and equation (11) at every time step k between times t and $t+h$.

$$V^*(x_k) = \min_{u_k} \left[r(x_k, u_k) + \gamma V^*(x_{k+1}) \right] \tag{10}$$

$$u^* = \arg\min_{u_k} \left[r(x_k, u_k) + \gamma V^*(x_{k+1}) \right] \tag{11}$$

Example of a simple problem solved by dynamic programming is shown in Figure 1.

2.1.4 Policy Evaluation

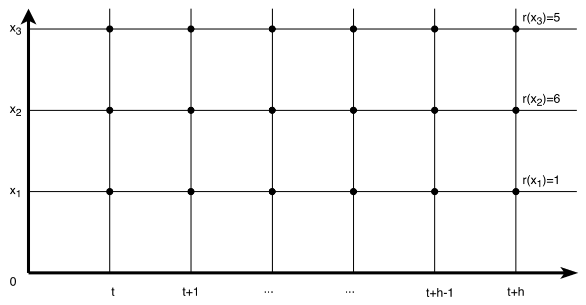
Considering the MDP framework, a way to compute the value function V^π for given policy π has to be found. This process is called a policy evaluation, and it utilizes the dynamic programming approach. The Bellman equation (6) can be understood in a way that its left side is at iteration $k+1$ and the right side is at iteration k

$$V^\pi_{k+1}(x_t) = r(x_t, u^\pi_t) + \gamma V^\pi_k(x_{t+1}). \tag{12}$$

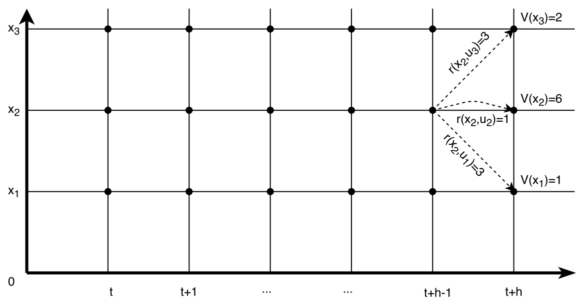
Indices k and $k+1$ are related to iterations of the policy evaluation, indices t and $t+1$ relate to time. The initial approximation of V^π can be chosen arbitrarily. If it is then updated using equation (12), it can be shown that as $k \to \infty$, the sequence of V^π approximations converges to the true V^π, if the steady-state V^π exists. This iterative process is usually stopped once the difference between V^π_k and V^π_{k+1} is lower than some threshold.

The update from V^π_k to V^π_{k+1} can be performed in at least two ways. Either the V^π_k is kept intact as it is and it is used to separately compute V^π_{k+1} for all $x \in X$, or V^π_k and V^π_{k+1} merge as V^π_k is updated for all $x \in X$ and the old value is immediately rewritten by the new one. Both of these two approaches converge to V^π as $k \to \infty$. [1]

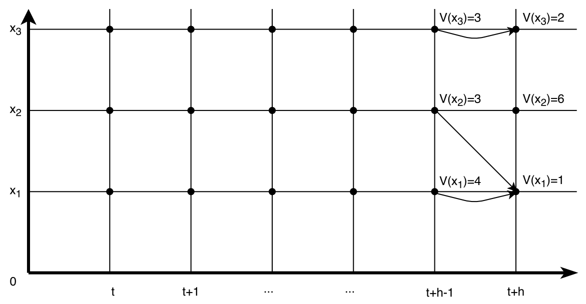
This value function computation approach can be used without the knowledge of transition probabilities between states (system dynamics) only if, as so far considered, the system is



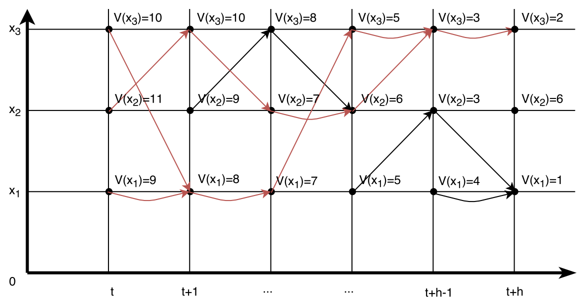
(a) The reward $r(x_{t+h})$ is computed for every $x \in X$.



(b) Reward for all $u \in U$ is computed at each state $x \in X$.



(c) u^* (from all $u \in U$) and value $V(x_{t+k})$ is found for each state $x \in X$.



(d) Once the value is computed for all $x \in X$ at every time step k between times t and $t+h$, the optimal policy for each $x \in X$ at time t (red) can be shown.

Figure 1: Dynamic programming example

deterministic. However, it cannot be used without known system dynamics, if the system has a stochastic part (see 2.2.6). Therefore, some other way to obtain the value function has to be found.

2.2 Q-Learning

Q-learning is a type of reinforcement learning. It is a data-driven approach, where the system dynamics is unknown. The Bellman equation (6) can be modified and used in Q-learning. First, a new function has to be defined using the Bellman function (6)

$$Q^\pi(x_t, u_t) = r(x_t, u_t) + \gamma V^\pi(x_{t+1}). \tag{13}$$

This new Q-function (also called “action-value function”) has a similar meaning as the V-function defined by (2), but it is a function of x_t and u_t as opposed to only being function of x_t in the case of the V-function. The Q-function tells us how would the value of strategy π change if the input u_t was applied at time t and then the policy π would be used from time $t+1$ onward. Since

$$V^\pi(x_t) \equiv Q^\pi(x_t, u_t^\pi), \tag{14}$$

the equation (13) can be written as the Bellman equation

$$Q^\pi(x_t, u_t) = r(x_t, u_t) + \gamma Q^\pi(x_{t+1}, u_{t+1}^\pi). \tag{15}$$

The Bellman optimality equation as a Q-function equivalent of (7) is

$$Q^*(x_t, u_t) = r(x_t, u_t) + \gamma \min_{u_{t+1}} Q^*(x_{t+1}, u_{t+1}). \tag{16}$$

2.2.1 Q-Function Identification – Learning

Considering a Q-learning problem, the only information given by the system after applying the input u_t, is the state x_t, reward $r(x_t, u_t)$, and state x_{t+1}. Therefore, the Q-function has to be identified using only these values $\left\{ \ u_t, \ \ x_t, \ \ x_{t+1}, \ \ r(x_t, u_t) \ \right\}$. The equation (15) can be rewritten to fit a temporal difference form

$$r(x_t, u_t) = Q^\pi(x_t, u_t) - \gamma Q^\pi(x_{t+1}, u^\pi_{t+1}). \tag{17}$$

This equation only holds for a deterministic system and already identified Q-function. Generally, there is a temporal difference error e_t defined as

$$e_t = r(x_t, u_t) - Q^\pi(x_t, u_t) + \gamma Q^\pi(x_{t+1}, u^\pi_{t+1}). \tag{18}$$

During the learning period, such e_t is obtained at each time step t. The goal now is to find such Q that the e_t at each time t is minimized, which can be achieved for instance by the Recursive Least Squares (RLS) method.

On-Policy Learning

If the equation (17) uses u_t chosen by the policy π as in (19), then the Q-function identification method is described as on-policy learning. Therefore, the state x_{t+1} is a result of π.

$$r(x_t, u^\pi_t) = Q^\pi(x_t, u^\pi_t) - \gamma Q^\pi(x_{t+1}, u^\pi_{t+1}) \tag{19}$$

Off-Policy Learning

On the other hand, the Q-function of a policy π_1 can be evaluated while inputs given by some other policy π_0 are applied. This method is called off-policy learning and is described by equation (20). In this case, the state x_{t+1} is a result of using policy π_0 even though policy π_1 is being evaluated.

$$r(x_t, u^{\pi_0}_t) = Q^{\pi_1}(x_t, u^{\pi_0}_t) - \gamma Q^{\pi_1}(x_{t+1}, u^{\pi_1}_{t+1}) \tag{20}$$

2.2.2 Policy Iteration

Policy iteration and value iteration are dynamic programming based methods used to obtain the optimal value function V^* or optimal Q-function Q^*, and their corresponding optimal policy π^*. As the name of those methods suggests, the process of finding such optimal function and policy is iterative. Let us start with the policy iteration method.

Suppose we have some policy π_0. Its corresponding value function V^{π_0} (or Q-function Q^{π_0} since $V^{\pi_0}(x_t) = Q^{\pi_0}(x_t, u^{\pi_0}_t)$) can be computed either by the policy evaluation method (2.1.4), which can only be performed "on-policy" or on/off-policy learning (2.2.1). The policy improvement theorem then states that

$$\left(Q^{\pi_0}(x_t, u_t^{\pi_1}) \leq Q^{\pi_0}(x_t, u_t^{\pi_0})\right) \Rightarrow \left(Q^{\pi_1}(x_t, u_t^{\pi_1}) \leq Q^{\pi_0}(x_t, u_t^{\pi_0})\right) \quad \forall x_t \in X. \tag{21}$$

In other words, if the Q-function corresponding to the policy π_0 has lower or equal value for all states $x_t \in X$ after the input given by the policy π_1 was applied, than if the input given by the policy π_0 was applied, then the policy π_1 is as good as, or better than policy π_0. The proof of this theorem is trivial, and it is given in [1]. This shows that given some policies π_0 and π_1, it can be easily evaluated if the policy π_1 lowers the value at some state x_t.

Now, a way has to be found to find a policy which lowers (or at least not increases) the value at all states. This is achieved by

$$\pi_1(x_t) = \arg\min_{u_t} Q^{\pi_0}(x_t, u_t). \tag{22}$$

Equation (22) has to be computed for all states $x \in X$, and each computation is done over all $u \in U$. This way a new policy π_1 is obtained. Simply put, this new policy does what is best at one step look-ahead. As proved by the policy improvement theorem, the newly obtained policy π_1 is as good as, or better than the original policy π_0. This process is called a policy improvement.

Once a policy improvement has been applied to obtain a policy π_1, a value of such policy can be evaluated and then improved again to get even better policy π_2. This method, where policy improvement alternates with policy evaluation is called a policy iteration. Since each policy π_{i+1} is better (or as good as) π_i, inevitably a point, where the optimal π^* is found, has to be reached. This occurs when $V^{\pi_{i+1}} = V^{\pi_i} = V^*$. This convergence is conditioned by the finite input and state spaces of the MDP framework, which results in a finite number of policies. Using the infinite continuous input and state spaces, resulting in an infinite number of policies, no such convergence can be guaranteed in a finite number of steps. This topic will be discussed more in section 2.2.4.

The policy iteration theory will be supported by a simple example. The convergence will be tested on a simple, stable discrete second-order system, with known optimal (with respect to some criterion) policy in the form of state feedback controller $u = \pi(x) = -Kx$. The policy iteration will be used to find the same optimal policy. The gain K consists of five individual values, corresponding to the augmented state vector $x_{aug} = \begin{bmatrix} 1 & x_t & s_t & u_{t-1} \end{bmatrix}$, where x_t is the state of the second-order system (therefore, two values), s_t is the set-point at time t and u_{t-1} is the input at time $t-1$. The iteration starts from a $K_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \end{bmatrix}$, which is stabilizing since the system itself is stable. The algorithm is stopped if

$$\|K_i - K_{i-1}\|_2 \leq 1 \times 10^{-10}, \tag{23}$$

where i denotes the iteration step.

Figure 2 shows, how the values of the optimized policy converged to those of the known optimal gain. The stopping condition (23) was met at the seventh iteration. Since the policy was evaluated using the least squares method and the policy evaluation method as described in 2.1.4 was not used, no "inner iterations" are present.

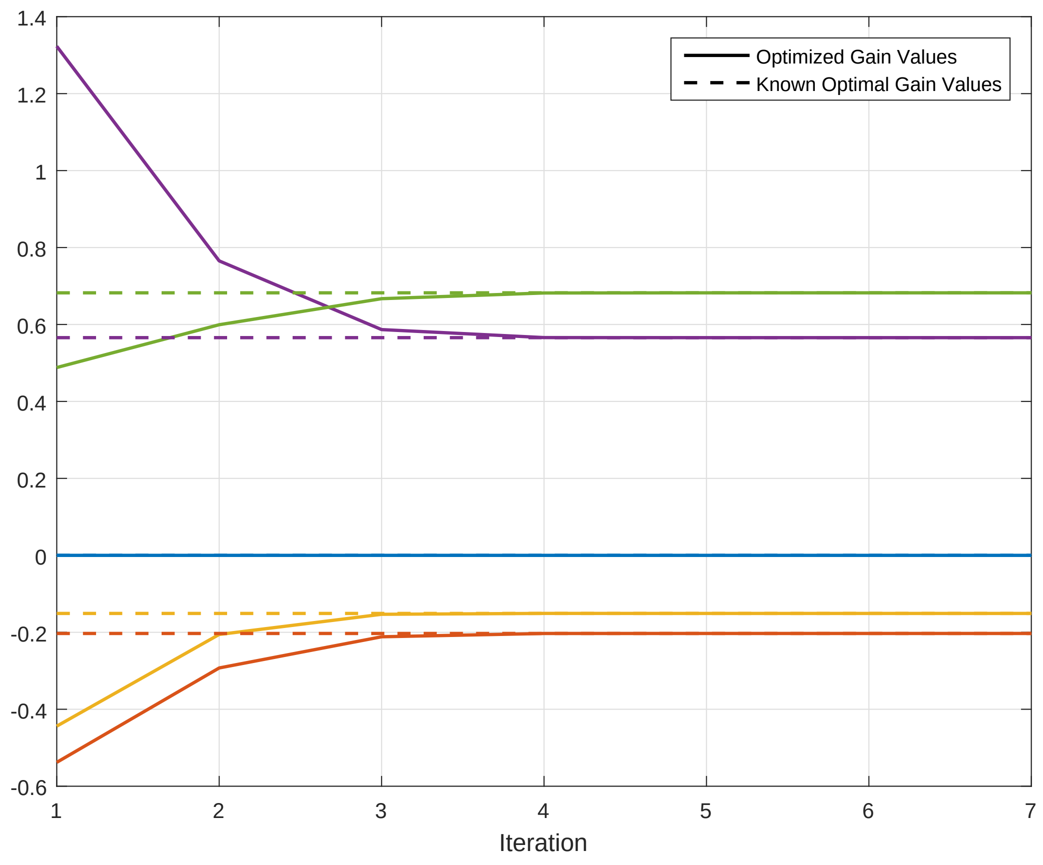


Figure 2: Policy Iteration Example

2.2.3 Value Iteration

The value iteration algorithm is similar to the policy iteration. The policy evaluation step (2.1.4) of the policy iteration algorithm is actually an iterative process itself. The value of a policy V^{π} is in theory found in an infinite number of steps. The idea behind value iteration is stopping this policy evaluation only after one step and combining it with the policy improvement. The idea comes from the Bellman optimality equation (7), except that it is seen as an update of some value V, similarly as in policy evaluation (2.1.4).

$$V^*_{k+1}(x_t) = \min_{u_t} \left[r(x_t, u_t) + \gamma V^*_k(x_{t+1}) \right]. \tag{24}$$

The equation (24) has to be yet again computed for all states $x \in X$, and each computation is done over all $u \in U$. Considering the LQ problem, the value iteration turns into the Riccati differential equation. This will be shown in a section 2.2.5.

Same as policy iteration, value iteration theory, too, will be supported by a simple example. The setup stays the same, except this time, the policy $u = \pi(x) = -Kx$ will be found using the value iteration.

Figure 3, again, shows the convergence of the value iteration algorithm. Using value iteration, the stopping condition (23) was met at the 62nd iteration.

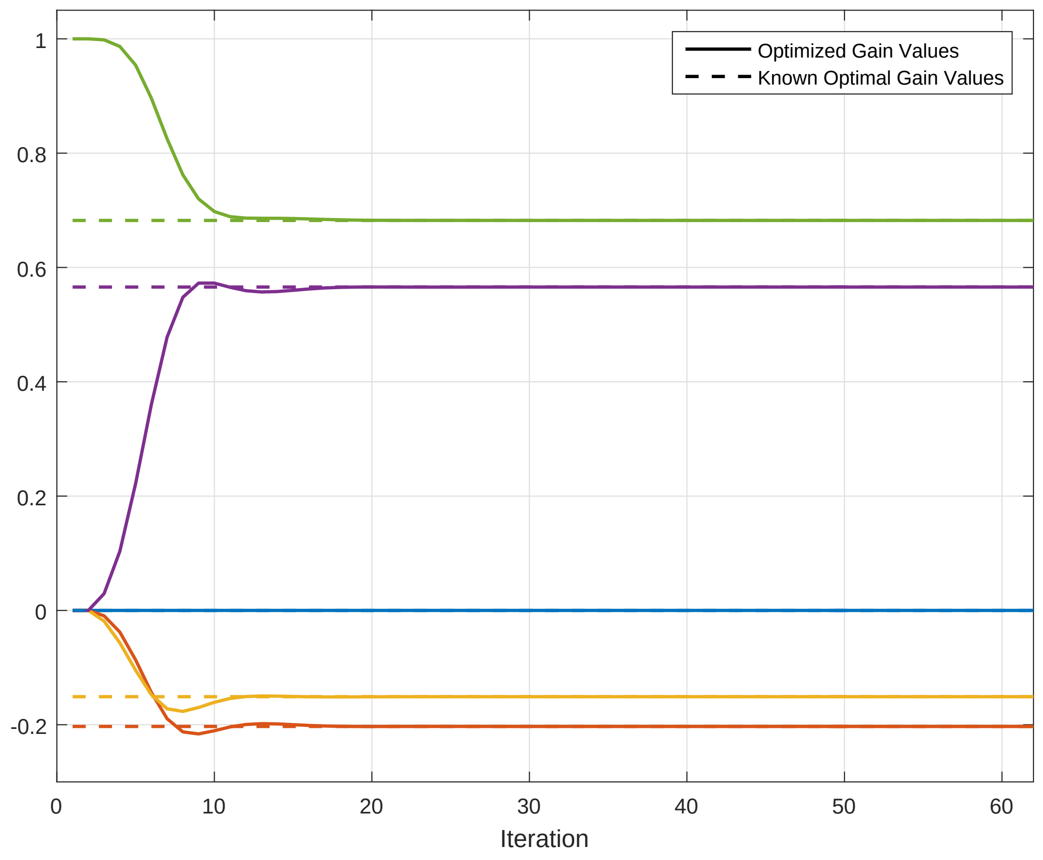


Figure 3: Value Iteration Example

2.2.4 Problems with Policy and Value Iteration

The first problem arises when the policy iteration is applied to the infinite continuous state and input spaces framework. The value function cannot be simply evaluated by the policy evaluation method (2.1.4). It has to be evaluated in the form of a Q-function by the on/off-policy learning (2.2.1).

As said before, the Bellman equation can be viewed either as an update rule (25) or as an equation (26) or set of equations, which can be solved to obtain some value V^π.

$$V^\pi_{k+1}(x_t) = r(x_t, u_t) + \gamma V^\pi_k(x_{t+1}) \tag{25}$$

$$V^\pi(x_t) = r(x_t, u_t) + \gamma V^\pi(x_{t+1}) \tag{26}$$

The Bellman equation (25) can be also seen as a functional equation, where $V^\pi_k(x_{t+1})$ is mapped into $V^\pi_{k+1}(x_t)$. Then for $0 \leq \gamma < 1$ and V^π finite, it can be shown that such mapping is so-called contraction mapping with a unique fixed point [11]. This means that given some two initial guesses on V^π, let us denote them V^π_{0a} and V^π_{0b}, their values get closer and closer (contraction) as the iteration converges. The fact that the contraction mapping has a unique fixed point means that if the iteration converges, it always converges to the same final $V^\pi = V^\pi_{ka} = V^\pi_{kb}$, no matter what the initial V^π_0 was.

The same V^π can be obtained by solving the equation or set of equations given by (26). Considering the MDP framework, where the state and input spaces are finite, and also the rewards are finite, this holds for any initial policy π_0 as long as $0 \leq \gamma < 1$. If the state and input spaces are not finite, and the initial policy is destabilizing, the iterative way to obtain the value function will not generally converge. It may diverge to the infinity even if the discount factor gamma is bounded as before. Using the second approach (26), some value function will be identified, but it will be wrong, and it will not have any physical meaning. Therefore, policy iteration requires the initial policy to be stabilizing to ensure convergence to the optimal value function.

On the other hand, value iteration does not require the initial policy to be stabilizing; however, its convergence is significantly slower compared to the policy iteration, as demonstrated by tests in 2.2.2 and 2.2.3.

An example, supporting the need for stabilizing initial policy for policy iteration and conversely that it is not needed while using the value iteration, will be given here. The setup remains identical to the one used in 2.2.2 and 2.2.3 with one exception. The system is now unstable, which in turn makes the initial policy $K_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \end{bmatrix}$ not stabilizing.

As expected, policy iteration did converge to some policy (Figure 4), but it was not the optimal one and maybe not even a stabilizing one. Value iteration did find the optimal policy. However, it took nearly 3000 iterations to converge (Figure 5).

Usually, this convergence speed versus need for stabilizing initial policy trade-off creates a dilemma: Which approach should be used? Fortunately, in our case, the system itself is stable. The room temperature will tend to decrease to the ambient temperature. Therefore, the initial policy $K_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \end{bmatrix}$ is undoubtedly stabilizing and the faster approach – policy iteration – can be used here without any concerns.

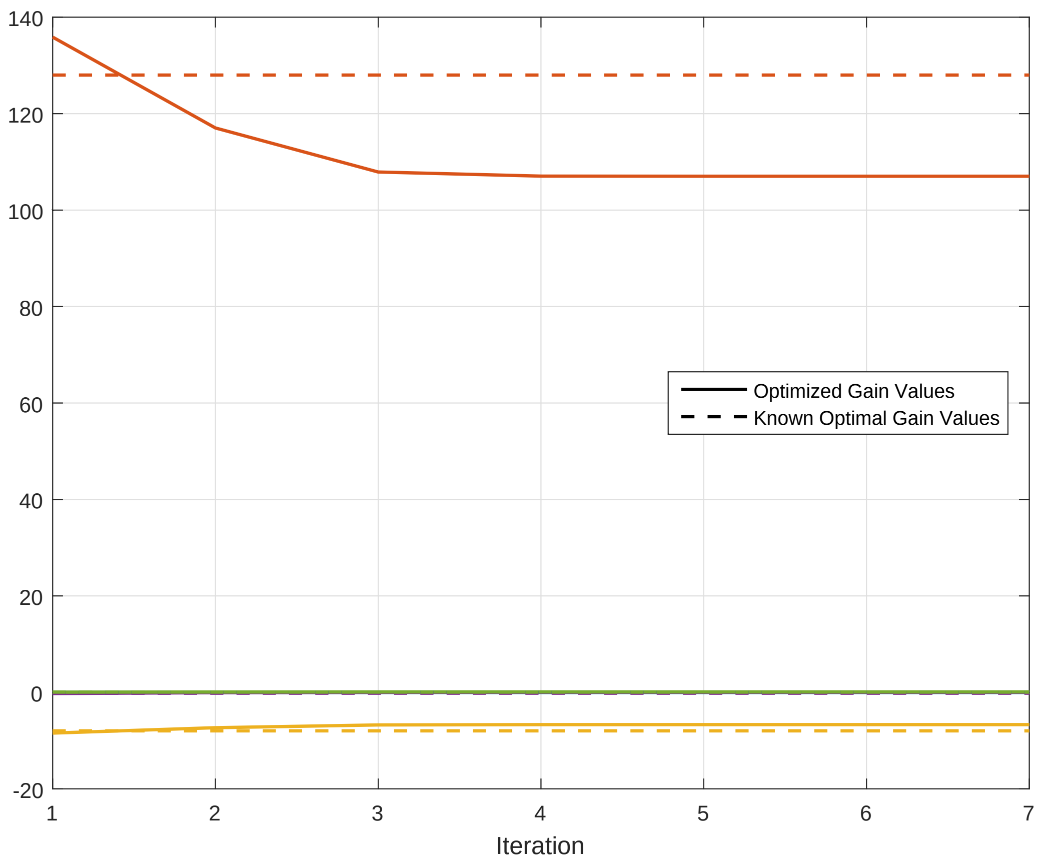


Figure 4: Policy Iteration Example with Not Stabilizing Initial Policy

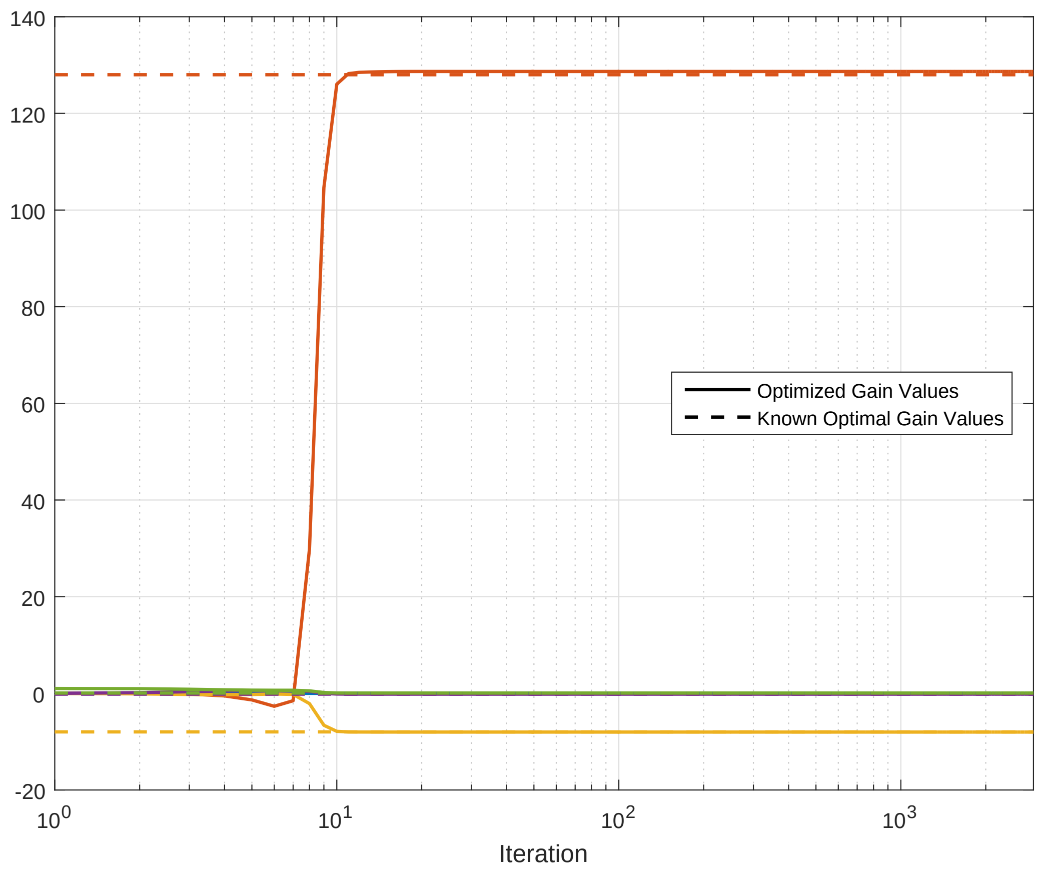


Figure 5: Value Iteration Example with Not Stabilizing Initial Policy

2.2.5 Relation with Algebraic Riccati Equation

Let us assume that the problem to be solved by the Q-learning methods will be given in an LQ (Linear Quadratic) form. Therefore, the system description is linear and given by

$$x_{t+1} = f(x_t, u_t) = Ax_t + Bu_t \tag{27}$$

and the quadratic criterion is given by

$$J_t = \frac{1}{2} \sum_{i=t}^{\infty} \left(x_i^T Q x_i + u_i^T R u_i + 2 x_i^T N u_i \right), \tag{28}$$

where Q, R and N are weight matrices. For our purposes the last term with the weight matrix N can be omitted. The reward will also take quadratic form

$$r(x_t, u_t) = \frac{1}{2} \left(x_t^T Q x_t + u_t^T R u_t \right). \tag{29}$$

Considering this new form of reward, the value function corresponding to a policy π given by $u_t = \pi(x_t)$ changes for $\gamma = 1$ from (2) to

$$V^{\pi}(x_t) = \frac{1}{2} \sum_{i=t}^{\infty} \left(x_i^T Q x_i + u_i^T R u_i \right). \tag{30}$$

Analogically to (5) and (6) we then get

$$V^{\pi}(x_t) = \frac{1}{2}\left(x_t^T Q x_t + u_t^T R u_t\right) + \frac{1}{2}\sum_{i=t+1}^{\infty}\left(x_i^T Q x_i + u_i^T R u_i\right), \tag{31}$$

$$V^{\pi}(x_t) = \frac{1}{2}\left(x_t^T Q x_t + u_t^T R u_t\right) + V^{\pi}(x_{t+1}). \tag{32}$$

The result of an LQR problem is a controller in a form of

$$u_t = f(x_t) = -Kx_t. \tag{33}$$

Since $u_t = -Kx_t$, then according to (30), $V^{\pi}(x_t)$ is an infinite sum of quadratic functions depending on x_t therefore, the value will also have a quadratic form depending only on x_t with a symmetric kernel matrix P, such as

$$V^{\pi}(x_t) = \frac{1}{2}x_t^T P x_t, \tag{34}$$

which changes (32) to

$$\frac{1}{2}x_t^T P x_t = \frac{1}{2}\left(x_t^T Q x_t + u_t^T R u_t\right) + \frac{1}{2}x_{t+1}^T P x_{t+1}. \tag{35}$$

By substituting (27) and (33) into (35), we get

$$\frac{1}{2}x_t^T P x_t = \frac{1}{2}\left(x_t^T Q x_t + u_t^T R u_t\right) + \frac{1}{2}\left(Ax_t + Bu_t\right)^T P \left(Ax_t + Bu_t\right), \tag{36}$$

$$x_t^T P x_t = x_t^T Q x_t + \left(-Kx_t\right)^T R \left(-Kx_t\right) + \left(Ax_t + B\left(-Kx_t\right)\right)^T P \left(Ax_t + B\left(-Kx_t\right)\right), \tag{37}$$

$$x_t^T P x_t = x_t^T Q x_t + \left(Kx_t\right)^T RKx_t + \left(Ax_t - BKx_t\right)^T P \left(Ax_t - BKx_t\right), \tag{38}$$

$$x_t^T P x_t = x_t^T Q x_t + x_t^T K^T RKx_t + \left(x_t^T A^T - x_t^T K^T B^T\right) P \left(Ax_t - BKx_t\right), \tag{39}$$

$$x_t^T P x_t = x_t^T \left(Q + K^T RK\right) x_t + x_t^T \left(A^T - K^T B^T\right) P \left(A - BK\right) x_t, \tag{40}$$

$$P = Q + K^T RK + \left(A^T - K^T B^T\right) P \left(A - BK\right), \tag{41}$$

$$\left(A - BK\right)^T P \left(A - BK\right) - P = -\left(Q + K^T RK\right), \tag{42}$$

where (42) is a Lyapunov equation. Therefore, for a given K (or π), the Lyapunov equation is an LQR equivalent of the Bellman equation (6).

By taking (36) and subtracting $\frac{1}{2}x_t^T P x_t$ from the left side, we get

$$0 = x_t^T Q x_t + u_t^T R u_t + (Ax_t + Bu_t)^T P (Ax_t + Bu_t) - x_t^T P x_t, \tag{43}$$

which only holds for $u_t = -Kx_t$, with one concrete K. Generally, (43) has nonzero left side

$$H(x_t, u_t) = x_t^T Q x_t + u_t^T R u_t + (Ax_t + Bu_t)^T P (Ax_t + Bu_t) - x_t^T P x_t, \tag{44}$$

where $H(x_t, u_t)$ is a Hamiltonian function, which is an LQR equivalent to the temporal difference error e_t, used in (18). [2]

The similarity between (17), (18) and (43), (44) is caused by the fact that (43) is a Bellman equation (17) for the LQR version of this problem. [2]

To find the gain K, the necessary optimality condition $\partial H(x_t, u_t)/\partial u_t = 0$ has to be fulfilled. The procedure is shown below, starting by modifying (44).

$$H(x_t, u_t) = x_t^T Q x_t + u_t^T R u_t + \left(x_t^T A^T + u_t^T B^T\right) P (Ax_t + Bu_t) - x_t^T P x_t, \tag{45}$$

$$H(x_t, u_t) = x_t^T Q x_t + u_t^T R u_t + x_t^T A^T P A x_t + x_t^T A^T P B u_t + u_t^T B^T P A x_t + u_t^T B^T P B u_t - x_t^T P x_t, \tag{46}$$

$$\frac{\partial H(x_t, u_t)}{\partial u_t} = 2Ru_t + 2B^T P A x_t + 2B^T P B u_t. \tag{47}$$

Then by setting $\partial H(x_t, u_t)/\partial u_t \overset{!}{=} 0$, we get

$$0 = 2Ru_t + 2B^T P A x_t + 2B^T P B u_t, \tag{48}$$

$$0 = \left(R + B^T P B\right) u_t + B^T P A x_t, \tag{49}$$

$$u_t = -\left(R + B^T P B\right)^{-1} B^T P A x_t = -K x_t. \tag{50}$$

Therefore, the optimal strategy π with gain

$$K = \left(R + B^T P B\right)^{-1} B^T P A \tag{51}$$

was found.

By evaluating K using the Lyapunov equation (42), we are performing policy evaluation. If P obtained by such evaluation is used to generate new K (using (51)), policy improvement is being executed. Therefore, by alternating between (42) and (51), we are performing the policy iteration algorithm (2.2.2), which is in control theory known as Kleinman algorithm [17].

By modifying the Lyapunov equation (42) and substituting the gain (51) into it, we get

$$0 = A^T P A - A^T P B K - K^T B^T P A + K^T B^T P B K - P + Q + K^T R K, \tag{52}$$

$$0 = A^T PA - P + Q - A^T PBK + K^T \left(B^T PBK + RK - B^T PA\right), \tag{53}$$

$$0 = A^T PA - P + Q - A^T PBK + K^T \left(\left(B^T PB + R\right) K - B^T PA\right), \tag{54}$$

$$\begin{aligned} 0 = A^T PA - P + Q - A^T PB \left(R + B^T PB\right)^{-1} B^T PA \\ + K^T \left(\left(B^T PB + R\right) \left(R + B^T PB\right)^{-1} B^T PA - B^T PA\right), \end{aligned} \tag{55}$$

$$0 = A^T PA - P + Q - A^T PB \left(R + B^T PB\right)^{-1} B^T PA, \tag{56}$$

where (56) is an Algebraic Riccati Equation, which is an LQR equivalent of the Bellman optimality equation (7).

This result, combined with (34) shows that the value of a policy given by $u_t = \pi(x_t) = -Kx_t$ can be obtained by solving the Algebraic Riccati Equation, given the system dynamics are known. In other words, if the system dynamics are known, there are two ways to obtain the value of a policy. One is to use the Algebraic Riccati or Lyapunov equation and the second one was described in section 2.2.1. The fact that there is more than one way to obtain the value of a policy can be used to verify that the machine learning approach gives the same results as the already known and rigorous approach in the form of the Algebraic Riccati or Lyapunov equation. Such comparison will be given in section 5.3.2.

2.2.6 Q-Learning for Stochastic Systems

For simplicity, stochastic systems were not considered until now. Although Q-learning or reinforcement learning methods in general are perfectly usable for deterministic systems, they are intended to be used on stochastic systems. There are several reasons supporting it, one of them being that there already are simpler methods for deterministic systems, which achieve the same goal as reinforcement learning. Simply put, using reinforcement learning on deterministic systems is a bit of an overkill.

The main characteristic of stochastic systems is that, even though the current state of the system or Markov decision process and the applied action are known, the next state cannot be predicted with 100 % certainty.

In Markov decision processes this means that there are some transition probabilities of transitioning from the current state x_t to all states $x_{t+1} \in X$, given the applied input at time t, as described in section 2.1.1. If we were to use the discrete time-invariant model (27), the stochastic part e_t would have to be added (57).

$$x_{t+1} = Ax_t + Bu_t + e_t, \tag{57}$$

The Bellman equation (6) and Bellman optimality equation (7) change their forms to (58) and (59) respectively. Notice the newly added expected value operator,

$$V^\pi(x_t) = r(x_t, u_t^\pi) + \gamma \mathcal{E} \left\{V^\pi(x_{t+1}) \mid x_t, u_t\right\}, \tag{58}$$

$$V^*(x_t) = \min_{u_t} \left[r(x_t, u_t) + \gamma \mathcal{E} \left\{ V^*(x_{t+1}) \mid x_t, u_t \right\} \right]. \tag{59}$$

Using the Q-function, their respective forms are

$$Q^\pi(x_t, u_t) = r(x_t, u_t) + \gamma \mathcal{E} \left\{ Q^\pi(x_{t+1}, u^\pi_{t+1}) \mid x_t, u_t \right\}, \tag{60}$$

$$Q^*(x_t, u_t) = r(x_t, u_t) + \gamma \mathcal{E} \left\{ \min_{u_{t+1}} Q^*(x_{t+1}, u_{t+1}) \mid x_t, u_t \right\}. \tag{61}$$

The transition from deterministic to stochastic systems has a significant influence on the Q-function identification (2.2.1). If the system is deterministic, only a finite number of equations (18) (time steps) has to be obtained to find the Q-function, which minimizes e_t in each equation. On the other hand, if the system is stochastic, one would need an infinite number of time steps to truly identify the Q-function.

2.3 Model Predictive Control (MPC)

The main reason for the use of Model Predictive Control in this thesis is its ability to deal with hard constraints, which are generally caused by physical limitations of actuators. In our case, the need for constraint support rises mostly from the fact that while controlling the temperature in a room, a negative action input (cooling) is often not possible and certainly not wanted, as such cooling would be seen as a waste of energy. This chapter includes information acquired from [5].

The MPC works on a finite horizon of length N from time 0 to time N. Its goal is to minimize given criterion $J(x, u)$ over the set of inputs and states

$$J^*(x_0) = \min_{\substack{u_0, u_1, \dots, u_{N-1} \\ x_0, x_1, \dots, x_N}} J(x, u) = \min_{\substack{u_0, u_1, \dots, u_{N-1} \\ x_0, x_1, \dots, x_N}} \phi(x_N) + \sum_{k=0}^{N-1} L_k(x_k, u_k). \tag{62}$$

The minimization can be performed with respect to constraints, such as

$$x_{k+1} = f(x_k, u_k), \tag{63}$$

$$x_0 = \text{given initial state}, \tag{64}$$

$$u_{\min} \leq u_k \leq u_{\max}, \tag{65}$$

$$x_{\min} \leq x_k \leq x_{\max}, \tag{66}$$

where (63) describes the system dynamics, (64) is the initial state, and (65) and (66) are constraints on input and state.

2.3.1 MPC Without Constraints

If the controlled system is linear and there are no constraints on state and input, equations (63–66) take the form

$$x_{k+1} = Ax_k + Bu_k, \tag{67}$$

$$x_0 = \text{constant}. \tag{68}$$

Then we consider a quadratic criterion, such as

$$J(x,u) = \frac{1}{2}x_N^T Q_N x_N + \frac{1}{2}\sum_{k=0}^{N-1}\left(x_k^T Q x_k + u_k^T R u_k\right) = \frac{1}{2}\bar{X}^T\bar{Q}\bar{X} + \frac{1}{2}\bar{U}^T\bar{R}\bar{U} + \frac{1}{2}x_0^T Q x_0. \tag{69}$$

The minimization problem then changes to

$$\min_{\bar{X},\bar{U}} J(\bar{X},\bar{U}) = \min_{\bar{X},\bar{U}} \frac{1}{2}\bar{X}^T\bar{Q}\bar{X} + \frac{1}{2}\bar{U}^T\bar{R}\bar{U} + \frac{1}{2}x_0^T Q x_0, \tag{70}$$

where

$$\begin{gathered}\bar{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix},\ \bar{U} = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix}, \\ \bar{Q} = \operatorname{diag}\left(\begin{bmatrix} Q & Q & \dots & Q & Q_N \end{bmatrix}\right),\ \bar{R} = \operatorname{diag}\left(\begin{bmatrix} R & R & \dots & R \end{bmatrix}\right),\end{gathered} \tag{71}$$

and $\frac{1}{2}x_0^T Q x_0$ is a constant. Since the constant does not change the position of the minimal value, this term can be omitted. Constraints, as described in (67), can be written as

$$\bar{X} = \bar{A}\bar{X} + \bar{B}\bar{U} + A_0 x_0, \tag{72}$$

where

$$\bar{A} = \begin{bmatrix} 0 & & & & \\ A & 0 & & & \\ & A & \ddots & & \\ & & \ddots & \ddots & \\ & & & A & 0 \end{bmatrix},\ \bar{B} = \operatorname{diag}\left(\begin{bmatrix} B & B & \dots & B \end{bmatrix}\right),\ A_0 = \begin{bmatrix} A \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{73}$$

Equation (72) can be further rewritten as

$$0 = \left(\bar{A} - I\right)\bar{X} + \bar{B}\bar{U} + A_0 x_0, \tag{74}$$

$$0 = \begin{bmatrix} \left(\bar{A} - I\right) & \bar{B} \end{bmatrix} \begin{bmatrix} \bar{X} \\ \bar{U} \end{bmatrix} + A_0 x_0, \tag{75}$$

$$0 = \widetilde{A} w + \tilde{b}. \tag{76}$$

If we rewrite the minimization criterion (69), the minimization problem without constraints on the input or state can be written as

$$\min_{w} w^T \begin{bmatrix} \bar{Q} & 0 \\ 0 & \bar{R} \end{bmatrix} w. \tag{77}$$

$$\text{Subject to } \; 0 = \widetilde{A} w + \tilde{b}, \tag{78}$$

which is a problem which can be solved by quadratic programming (Section 2.3.3).

2.3.2 Constraints on the Input Vector

For us to be able to add constraints on the input, the set of state vectors $\bar{X}$ has to be eliminated from the criterion. Fortunately, $\bar{X}$ can be eliminated in a simple step-by-step way, as shown below.

$$x_1 = Ax_0 + Bu_0, \tag{79}$$

$$x_2 = Ax_1 + Bu_1 = A^2 x_0 + ABu_0 + Bu_1, \tag{80}$$

$$x_k = A^k x_0 + \begin{bmatrix} A^{k-1}B & A^{k-1}B & \dots & AB & B \end{bmatrix} \bar{U}. \tag{81}$$

Therefore, $\bar{X}$ can be written as

$$\bar{X} = \hat{B}\bar{U} + \hat{A}x_0 = \begin{bmatrix} B & & & & \\ AB & B & & & \\ A^2B & AB & B & & \\ \vdots & \vdots & \ddots & \ddots & \\ A^{N-1}B & A^{N-2}B & \dots & AB & B \end{bmatrix} \bar{U} + \begin{bmatrix} A \\ A^2 \\ \vdots \\ A^N \end{bmatrix} x_0. \tag{82}$$

By substituting (82) into (69) we get

$$J(\bar{U}, x_0) = \frac{1}{2} \left(\hat{B}\bar{U} + \hat{A}x_0\right)^T \bar{Q} \left(\hat{B}\bar{U} + \hat{A}x_0\right) + \frac{1}{2}\bar{U}^T \bar{R}\bar{U}, \tag{83}$$

$$J(\bar{U}, x_0) = \frac{1}{2}\bar{U}^T \hat{B}^T \bar{Q}\hat{B}\bar{U} + x_0^T \hat{A}^T \bar{Q}\hat{B}\bar{U} + x_0^T \hat{A}^T \bar{Q}\hat{A}x_0 + \frac{1}{2}\bar{U}^T \bar{R}\bar{U}, \tag{84}$$

$$J(\bar{U}, x_0) = \frac{1}{2}\bar{U}^T \left(\hat{B}^T \bar{Q}\hat{B} + \bar{R}\right) \bar{U} + x_0^T \hat{A}^T \bar{Q}\hat{B}\bar{U} + x_0^T \hat{A}^T \bar{Q}\hat{A}x_0, \tag{85}$$

$$J(\bar{U}, x_0) = \frac{1}{2}\bar{U}^T\left(\hat{B}^T\bar{Q}\hat{B} + \bar{R}\right)\bar{U} + x_0^T\hat{A}^T\bar{Q}\hat{B}\bar{U}. \tag{86}$$

$$J(\bar{U}, x_0) = \frac{1}{2}\bar{U}^T H\bar{U} + x_0^T F^T\bar{U}. \tag{87}$$

Because $x_0^T\hat{A}^T\bar{Q}\hat{A}x_0$ is a constant and does not change the position of the minimal value, it could be omitted. The minimization problem then changes to

$$\min_{\bar{U}} \frac{1}{2}\bar{U}^T H\bar{U} + x_0^T F^T\bar{U}. \tag{88}$$

With constraints

$$u_k \leq u_{\max} \text{ and } u_k \geq u_{\min}, \tag{89}$$

which is also a problem which can be solved by quadratic programming.

2.3.3 Quadratic Programming

Quadratic programming is a general term denoting a set of algorithms solving the quadratic optimization (minimization for $M > 0$ or maximization for $M < 0$) problem with linear constraints. Since our goal is to minimize the reward, then without loss of generality, only the minimization problem will be considered henceforth. The optimization problem can be written as

$$\min_{x} \frac{1}{2}x^T Mx + n^T x, \tag{90}$$

subject to the constraints

$$Ax \leq b, \tag{91}$$

$$A_{\text{eq}}x = b_{\text{eq}}, \tag{92}$$

$$b_l \leq x \leq b_u, \tag{93}$$

where (91) represents inequality constraints, (92) are equality constraints and b_l and b_u in (93) represent the lower and upper bounds respectively.

Unconstrained Optimization

There are many algorithms able to solve such optimization problems. If only the easiest form of this problem is considered, which is just the optimization problem (90) without any constraints, the solution is simple (provided the function $f(x)$ is convex). The first order necessary condition of optimality states that the gradient of the optimized function equals zero. In our case, if we denote

$$f\left(x\right)=\frac{1}{2}x^TMx+n^Tx, \tag{94}$$

then

$$\nabla f\left(x\right)\overset{!}{=}0, \tag{95}$$

$$df=\nabla^T f\left(x\right)dx, \tag{96}$$

$$df=\frac{1}{2}\left(dx^TMx+x^TMdx\right)+n^Tdx, \tag{97}$$

$$df=\frac{1}{2}\left(x^TM^Tdx+x^TMdx\right)+n^Tdx, \tag{98}$$

$$df=\left(x^T\frac{M^T+M}{2}+n^T\right)dx. \tag{99}$$

Therefore, according to (96) and (99),

$$\nabla f\left(x\right)=\frac{M^T+M}{2}x+n. \tag{100}$$

Under the assumption of $M^T=M$, which is satisfied in our case, (100) further simplifies to

$$\nabla f\left(x\right)=Mx+n. \tag{101}$$

Since the initial condition (95) has to be fulfilled, equation (101) can be rewritten as

$$Mx+n=0, \tag{102}$$

which is just a set of linear equations, whose solution is denoted as the critical point, where the minimum of $f\left(x\right)$ may be found. The second order necessary and sufficient conditions are given by the Hessian

$$\nabla^2 f\left(x\right)=M. \tag{103}$$

The so-called "second derivative test" states that if the Hessian is positive definite, the critical point is the minimum of the optimized function $f\left(x\right)$. On the other hand, if the Hessian is negative definite, the critical point forms the maximum of the function $f\left(x\right)$. If there are both positive and negative eigenvalues of the Hessian, then the critical point is a saddle point. In all other cases, the test does not give an answer.

Constrained Optimization

The constrained optimization problem is usually more complex than the unconstrained one, and its solution cannot be obtained just by solving a set of linear equations. Therefore, more sophisticated methods have to be used. Since most, if not all, of the work done on this thesis, was done using the Matlab [6] or Matlab/Simulink environment [7], this section will concentrate on the constrained quadratic optimization solving algorithms used by the Matlab environment.

According to [8], Matlab uses three algorithms to solve quadratic programming problems described by (90-93). Those algorithms are:

- Interior-point-convex: solves convex problems with any combination of constraints
- Trust-region-reflective: solves bound constrained or linear equality constrained problems
- Active-set: solves problems with any combination of constraints

A detailed description of those algorithms is given in the Matlab documentation [9]. Even though there are three distinct algorithms used by Matlab, there is only one function for solving quadratic programming problems. The function is called `quadprog`, and its description can be found in the documentation [10].

2.3.4 Receding Horizon

Notice that the solution to the minimization problem, the control sequence, does not use any feedback from the controlled system besides the initial state x_0. It purely relies on the model. Such control is called a feed-forward control. The most significant disadvantage of this approach is a possible divergence between the state vector predicted by the model and the state of the real controlled system. This divergence is caused by some combination of the inaccuracy of the model and the process noise of the system.

The MPC works on a principle called “receding horizon”. This principle solves this problem in an elegant way. The quadratic programming solves the given optimization problem on the horizon of length N and produces an input sequence u. Then, only the first input from this sequence is actually applied, and the state of the system is measured. This new state is used as an initial state for the optimization problem in the next step.

The receding horizon approach ensures that the aforementioned divergence does not affect control of the system. This approach of using the receding horizon is sometimes called “open-loop feedback”.

3 Problem Description

As stated in the introduction, the goal of this thesis is to develop an automatic self-tuning controller for regulating temperature (thermostat). Such controller should be able to find an optimal control law solely based on the system's input, output and predefined reward, which itself is a function of the input and output. The final control law should be able to deal with constraints on the input introduced by the boiler.

3.1 Controlled System

For simplicity, the performance of our controllers will be tested on one zone (room) only, which is a SISO problem. However, all results developed in this thesis are applicable to the whole building (MIMO problem). The temperature regulation in a building depends on several factors. The first one is the heat capacity ($C\ \left[\frac{\mathrm{J}}{\mathrm{K}} = \frac{\mathrm{kg\,m^2}}{\mathrm{K\,s^2}}\right]$), which says how much heat is required to be added to the system to change the temperature by 1 K. This quantity is mostly affected by proportions of the building, by the material used during the construction and also by the furnishing.

The second important factor is the heat transfer coefficient ($U\ \left[\frac{\mathrm{W}}{\mathrm{m^2\,K}} = \frac{\mathrm{kg}}{\mathrm{K\,s^3}}\right]$), which is in our case used to determine how much heat transfers through 1 m^2 area of some object (e.g. a wall) from one side to the other if the difference in temperature between those sides is 1 K. Since the heat transfer coefficient is just an inverse of the thermal resistance, the value is mostly affected by the amount and type of insulation used on the building and the insulation properties of the construction material itself.

Apart from the control input, the temperature regulation can also be affected by additional inputs, such as solar heat gain (increase in temperature as the solar radiation is absorbed), number of people in the building or even the presence of powerful computers or servers.

3.2 Demands on the Control Quality

The difference between typical control and reinforcement learning problem does not lie only in maximizing or minimizing reward or cost. There are differences in the demanded performance and convergence speed. Most of the algorithms applied to machine learning problems, which had been proved to be optimal need a massive amount of data (which takes time to generate) to reach the convergence. Therefore, these approaches cannot be used in the intelligent thermostat. The thermostat has to be able to control the room temperature since the moment of installation. There definitely cannot be a week, month or even a year-long learning period.

As mentioned above, the learning period has to be few days long at maximum (given those days contain enough useful data – more in section 5.4). However, this is not the only requirement on the learning period. The thermostat has to be able to control the system even during the learning period, though obviously, the control does not have to be optimal. There are only two mild demands on the control during the learning phase. The controller has to be stabilizing, and its actions must not cause the room temperature to diverge from the reference temperature to a degree which could be recognized by the residents.

3.3 Constraints Introduced by the Boiler

In a typical control problem, the applicable control input is usually bounded equally around the origin. While controlling the altitude of an aircraft, the controller can increase or decrease the plane's altitude by applying an elevator defection in a range between e.g. $-30°$ and $30°$. The controller inside a self-driving car can turn the front wheels e.g. from $-45°$ to $45°$. On the other hand, (as mentioned in section 2.3) while controlling a temperature in a room, a negative action input (active or passive cooling) is often not possible and certainly not wanted, as such cooling would be seen as a waste of energy. Therefore, the first constraint introduced by the boiler is the fact that the lower bound of a set of applicable action inputs would be 0 kW (The heat flow generated by the boiler to the system is measured in kilowatts (kW).). The upper bound is given by the maximum power capability of the boiler.

The second type of constraints introduced by the boiler is given by the fact that most of the commonly used boilers can only operate in a range of circa 30 % to 100 % of their maximum power potential. If the demanded power value would be under the 30 %, the boiler shuts down to 0 %. Therefore, the boiler can be regulated by action inputs from a disconnected set $\{0, \langle 30, 100 \rangle\}\%$.

4 Thermal Model Description

Successful testing of the theoretical results requires a realistic model. The model used in this thesis was developed by Honeywell Prague Laboratory. It utilizes a Hydronic Heating Library (personal communication, August, 2017) specially developed for modeling heating systems in the Matlab/Simulink framework.

The model can be divided into two parts. The first part includes the hydronic network and all components placed in it. Simply said, the first part is the one responsible for heat supply via water in the pipes. Therefore, components in this section include a pump, boiler, pipes, radiators, and valves. The pump simulates a hydraulic pressure source. It can work either in an ideal pressure source mode or in a real pump mode, where the water pressure is modeled as a quadratic function of flow. The boiler block simulates a hot water source. Again, two modes are available. Either the boiler behaves as an ideal hot water source, supplying hot water of set temperature to the network or it models a real hot water source, where the returning water with some temperature is heated in a water tank to a set temperature by a burner, which is controlled by a PI controller. Either way, the boiler introduces a pressure drop caused by fluid friction as a quadratic function of water flow. Finally, the radiator block simulates the hydraulic and thermodynamic behavior of a radiator, a control valve, and connecting pipes. The thermodynamic behavior of the radiator depends mostly on the heat exchange area of the radiator and the total heat exchange coefficient from water to air. The valve can work either in a linear mode, where the flow through the valve is a linear function of the opening or as (Semi-)Equal-percentage valve mode. Pipes are modeled as pressure drop, which is a quadratic function of the flow, proportional to the inner diameter and length of the pipe.

The second part models the thermal dynamics of the room or building. In other words, how is the heat gained from the radiators distributed and lost in the building. There are three main important components from the thermodynamic point of view: room, wall, and vent. The inputs and outputs of the room block represent the in and out heat flow. The room is defined by its area, height, part of space occupied by furniture and its initial temperature. There are three types of wall in the Hydronic Heating Library. The simplest one is the wall between two rooms. Aside from its dimensions, the wall has three additional parameters to be set: thermal conductivity of the material the wall is built of, the heat transfer on the boundary between the wall itself and the air in the room and thermal capacity of the material the wall is built of. The "neighbors wall" is same as the wall between two rooms, except the temperature in the other (neighbor) room is not controlled and is modeled as constant. Next is the external wall, which in addition to the parameters of the simple wall has other parameters. Those describe if the wall is exposed to solar radiation, the area of the wall occupied by a window, heat transmission and solar radiation transmittance of the window and several other parameters describing the interior and exterior facing (insulation thickness and conductivity). The vent represents an interface between two thermal zones without any barrier.

4.1 Capabilities and Settings

The Hydronic Heating Library enables the creation of a whole range of possible floor plans with various room heights, areas, radiator types, construction materials, insulation types and boiler settings. The setting used for our purposes (Figure 6) consists of two neighboring rooms (A and B), each having two external walls and one "neighbors wall". The only difference is that room A has only one window and one radiator, whereas room B has two windows and two radiators. Area of each room is $60\,\mathrm{m}^2$, height of the room is $3\,\mathrm{m}$ with initial temperature of

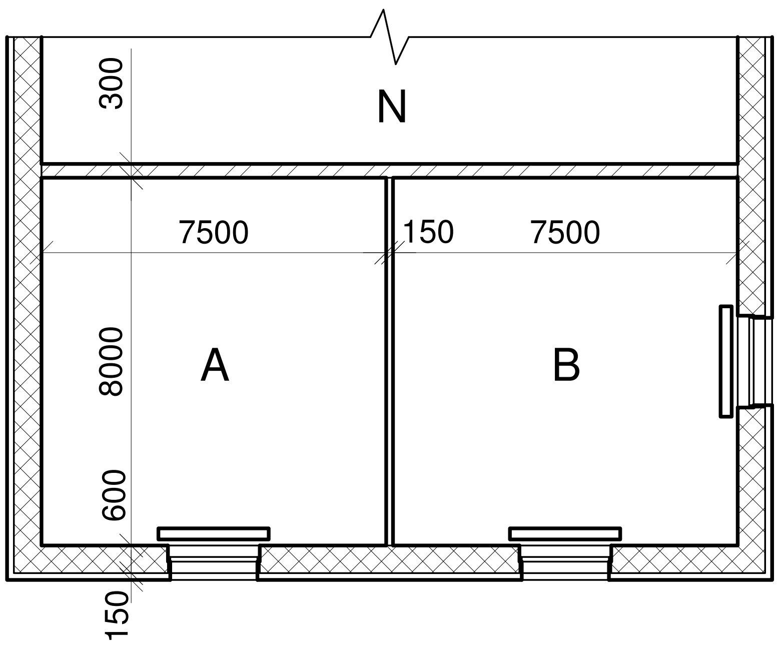


Figure 6: Thermal Model Floor Plan

12 °C. The construction material of the external wall is bricks with 60 cm thickness, and 15 cm thick mineral wool insulation is added on the exterior. The inner wall between the two rooms is 15 cm thick and the "neighbor wall" is 30 cm thick with a constant neighbor (labeled as "N" in Figure 6) temperature of 18 °C. Mean ambient temperature is initially set to 0 °C, although it can be changed later to facilitate specific needs of conducted experiments.

5 Solution

This part shows how were the theories described in chapter 2 modified, combined and applied to the problem. This section includes the description of the major techniques, which were achieved by combining machine learning with classical control approach. Selection of tests needed to produce those techniques is also demonstrated here. This section also includes solutions to a selection of minor problems which had to be solved. The chapter ends with several complications which were found during the testing phase.

5.1 Major Derived Techniques

There are three main solutions produced by this thesis. All of them will be presented in this chapter. Those solutions can be further combined to create the final intelligent thermostat.

5.1.1 Controllers Comparison

The first approach chosen for the intelligent thermostat has the following format. Suppose the thermostat is started with a finite number of seed controllers and its goal is to find the best one. The performance of a given controller is computed using the quadratic reward which was introduced in section 2.2.5 as

$$r(x_t, u_t) = \frac{1}{2}\left(x_t^T Q x_t + u_t^T R u_t\right). \tag{104}$$

First, the Q-function for each controller (policy) has to be evaluated, using the off-policy learning. As shown in 2.2.1, during the off-policy learning, one policy (controller) has to be used to actually control the system, while the Q-function of all controllers can be evaluated. The problem of choosing the right policy to be controlling the system during the off-policy learning is not trivial and will be dealt with later (5.2.1). Let us denote the number of time steps needed to identify Q-functions as N.

Once all Q-functions are evaluated, they (and with them the controllers themselves) have to be compared. For us to be able to compare Q-functions, some measure has to be defined first. Let us define value C^π (cost) of given policy π as

$$C^\pi = \frac{1}{N}\sum_{t=1}^{N} Q^\pi(x_t, u_t^\pi) = \frac{1}{N}\sum_{t=1}^{N}\sum_{k=0}^{\infty}\gamma^k r(x_{t+k}, u_{t+k}^\pi). \tag{105}$$

Assume that some policy π_0 was used to control the system during the N steps of off-policy learning. At each step, there was an opportunity to switch to the policy π and use it (and generate reward – cost) from that point onward. The newly defined cost C^π states the average (over all N steps) cost, which would be paid after switching from π_0 to π.

The quadratic form of value $V^\pi(x_t)$ was proved in 2.2.5. The quadratic form of $Q^\pi(x_t, u_t^\pi)$ can be proved analogically and results in

$$Q^\pi(x_t, u_t^\pi) = \frac{1}{2}\begin{bmatrix} 1 \\ x_t \\ u_t^\pi \end{bmatrix}^T P \begin{bmatrix} 1 \\ x_t \\ u_t^\pi \end{bmatrix} = \frac{1}{2} z_t^T P z_t, \tag{106}$$

with some quadratic kernel matrix P. If we substitute (106) into (105), we get

$$C^{\pi} = \frac{1}{N}\sum_{t=1}^{N}\frac{1}{2}z_t^T P z_t = \frac{1}{2N}\sum_{t=1}^{N} z_t^T P z_t, \tag{107}$$

which can be further rewritten as

$$C^{\pi} = \frac{1}{2N}\sum_{t,i,j} z_{t,i}^T P_{i,j} z_{t,j}, \tag{108}$$

where $P_{i,j}$ denotes value on i-th row and j-th column of matrix P. An analogical notation is used for vectors z and z^T. Apparently, the kernel matrix P does not directly depend on time. On the contrary, the data vector z does. The outer product of the data vector and its transposition forms a new matrix, which will be denoted as data kernel D.

$$D_{i,j} = \frac{1}{N}\sum_{t=1}^{N} z_{t,i} z_{t,j}^T \tag{109}$$

The advantage of this approach is that the data kernel D can be updated online as

$$D_t = \frac{t-1}{t}D_{t-1} + \frac{1}{t}z_t z_t^T. \tag{110}$$

If we then substitute (109) into (108) and rearrange the equation, we get

$$C^{\pi} = \frac{1}{2}\sum_{i,j} P_{i,j} D_{i,j} = \frac{1}{2}\mathrm{tr}\left(PD\right). \tag{111}$$

Since the goal is to minimize the cost, the policy (or controller) with the lowest C^{π} is considered to be the best one. As mentioned before, this method brings a problem. How to choose the policy, which will control the process during the off-policy learning. The most straightforward solution is random switching between all compared controllers. This problem will be further addressed in section 5.2.1.

However, there is another problem associated with this approach, which was not solved yet. The cost of some policy π is computed as an average reward which would be gained after switching from the controlling policy to the policy π at some state x. The problem is that if the policy π would be used since the beginning, it may not drive the system to that particular state x. Therefore, calculating how good is the policy π at continuing from the state x onward may be biased towards those policies which would go through state x if used from the beginning.

From a different point of view, the bias could be caused by the fact that some policies may be better at starting from various initial conditions, than others. Therefore, a situation may occur when a policy π_1 is optimal, but has problems with starting from miscellaneous initial states and would not naturally drive the system through state x. Suppose further that a policy π_2 is not optimal, but better at dealing with various initial conditions and would not go through state x if used from the beginning either. If the measure mentioned above were used to compare those two policies, π_2 would be deemed as better, even though it is not.

Unfortunately, the effect of this bias does not decrease with increasing amount of learning data. However, it does not increase either.

5.1.2 Multi-Component Reward

The reward, introduced at the beginning of this thesis can be defined as a weighted sum of elementary rewards

$$r(x_t, u_t) = \omega_1 \rho_1(x_t, u_t) + \omega_2 \rho_2(x_t, u_t) + \ldots \omega_L \rho_L(x_t, u_t) = \sum_{l=1}^{L} \omega_l \rho_l(x_t, u_t), \tag{112}$$

where ω_l represents the weight of the elementary reward $\rho_l(x_t, u_t)$. By substituting (112) into (2), we get a definition of the value function, which supports the idea of multi-component reward.

$$V^\pi(x_t) = \sum_{k=0}^{\infty} \gamma^k \sum_{l=1}^{L} \omega_l \rho_l(x_{t+k}, u_{t+k}^\pi), \tag{113}$$

$$V^\pi(x_t) = \sum_{l=1}^{L} \omega_l \sum_{k=0}^{\infty} \gamma^k \rho_l(x_{t+k}, u_{t+k}^\pi), \tag{114}$$

$$V^\pi(x_t) = \sum_{l=1}^{L} \omega_l v_l^\pi(x_t), \tag{115}$$

where $v_l^\pi(x_t)$ represents elementary value function of the policy π. Analogically, the Q-function can be defined as a weighted sum of partial Q-functions (q_l^π)

$$Q^\pi(x_t, u_t) = \sum_{l=1}^{L} \omega_l q_l^\pi(x_t, u_t^\pi). \tag{116}$$

Therefore, partial Q-functions can be evaluated each using only its respective partial reward. The final Q-function can then be obtained by summing the partial Q-functions with the same weights as those, used to sum the partial rewards.

This approach can be further applied to the result of the previous section. The constant C^π used to compare individual controllers can be divided into partial costs, where each c_l^π is computed in a manner described in 5.1.1 as a cost of its respective partial Q-function q_l^π. Then, the total cost can be again computed as the weighted sum of partial costs.

$$C^\pi = \sum_{l=1}^{L} \omega_l c_l^\pi \tag{117}$$

If not stated otherwise, the elementary rewards $\rho_l(x_t, u_t)$ used in this thesis further on will be in a form:

$$\rho_1(x_t, u_t) = u_t^2 \text{ and } \rho_2(x_t, u_t) = (s_t - y_t)^2, \tag{118}$$

where s_t is the set-point or reference temperature and y_t is the output or in this case actual temperature in the room. Both values are present in the x_t state vector.

Performance/Energy Trade-Off

The ability to compute the cost for each reward component is especially important for the control engineering aspect of this thesis. The so-called "Performance/Energy Trade-off" describes a typical control engineering problem. Simply put, the higher the control performance (a low difference between the controlled state and reference – error), the higher is the energy needed to obtain such results. Therefore, by lowering the control error, the used energy increases, and vice versa.

Therefore, if one reward component is defined to show the controller's ability to keep low error and second reward component represents the energy used by the controller's actions, then using their respective partial costs, controllers can be compared based on those two measures or any of their linear combinations.

5.1.3 Optimal Controller

The second approach to the intelligent thermostat problem utilizes the policy iteration method described in section 2.2.2. This method had to be modified to satisfy the demands introduced in 3.2. The only constraint on the initial policy given by the theory is that it is stabilizing. However, a different "learning" policy has to be used during the learning period. This policy (although most likely being sub-optimal) has to be not only stabilizing, it has to be reasonable as well, as it will control the system during the learning period. It is assumed that this "learning" policy will be designed by a human. After the policy iteration method converges to the optimal policy (controller), it replaces the "learning" controller and is used from that point onward.

This method can be combined with the first approach (5.1.1) to create an algorithm which takes the initial set of controllers, compares them, and uses them to generate new, assumably optimal controller, which then replaces the worst controller form the initial set. Since the method used to generate the controller is based only on data (no model is used), the controller may not actually be optimal even if the algorithm considers it optimal as it will be optimal only with respect to the obtained data. Therefore, this optimization and replacement of the worst controller should be performed with adequate frequency indefinitely.

A second motivation for frequently generating new "optimal" controllers comes from the fact that the system can be evolving in time. The evolving system causes the "optimal" controller to be sub-optimal because it was generated based on a Q-function, which does not reflect the new (evolved) system dynamics.

The controller produced by this method has a significant disadvantage. It is not able to follow the reference signal with zero error. This is caused by the fact that the system state does not include the estimate of the disturbance. The absence of the perturbation estimate is usually solved by using the Kalman filter. Unfortunately, Kalman filter cannot be used in this context since it uses a model of the system, which is in our case unknown.

5.1.4 Merging Q-Learning and MPC

The last challenge tackled by this thesis was to somehow incorporate constraints into the process of generating the optimal controller. Only a few commonly used controllers are able to satisfy constraints. One of them being the Model Predictive Control (MPC) (2.3). Unfortunately, as the name suggests, MPC uses the model in the form of the differential equation set. Therefore, a way how to substitute the model by a Q-function had to be found.

Take first the Bellman equation (15) utilizing the Q-function framework

$$Q^{\pi}(x_t, u_t) = r(x_t, u_t) + \gamma Q^{\pi}(x_{t+1}, u^{\pi}_{t+1}). \tag{119}$$

There are several ways how to view this equation. Some of them had already been used in this thesis (algebraic equation, update rule, temporal difference, etc.). Still, the Bellman equation can also be seen as a one-step prediction. Input u_t is applied at time t and state x_{t+1} is observed. However, there is no need to restrict the prediction to just one step. Let us introduce the generalized Bellman equation with the h-step prediction

$$\begin{aligned} &Q^{\pi}_h(x_t, u_t, u_{t+1}, u_{t+2}, \ldots, u_{t+h}) = \\ &\qquad = r(x_t, u_t) + \gamma r(x_{t+1}, u_{t+1}) + \gamma^2 r(x_{t+2}, u_{t+2}) + \ldots + \gamma^h r(x_{t+h}, u_{t+h}) \\ &\qquad\qquad + \gamma^{h+1} Q^{\pi}(x_{t+h+1}, u^{\pi}_{t+h+1}), \end{aligned} \tag{120}$$

$$Q^{\pi}_h(x_t, u_{t:t+h}) = \sum_{k=0}^{h} \gamma^k r(x_{t+k}, u_{t+k}) + \gamma^{h+1} Q^{\pi}(x_{t+h+1}, u^{\pi}_{t+h+1}). \tag{121}$$

Notice, the simplified notation $u_{t:t+h}$ for the $\{u_t, u_{t+1}, u_{t+2}, \ldots, u_{t+h}\}$ sequence was used in the second version of the equation.

To find the constraints-satisfying controller, first, an optimal Q-function has to be found. For instance, the policy iteration method 2.2.2 can be used, just as it was used in the previous section. Once the optimal Q^* is found, it can be extended to obtain the optimal Q-function Q^*_h as described above. The problem then changes to minimization of $Q^*_h(x_t, u_{t:t+h})$ (122).

$$u^*_{t:t+h} = \arg\min_{u_{t:t+h}} Q^*_h(x_t, u_{t:t+h}) \tag{122}$$

Such minimization can be subjected to equality and inequality constraints. However, only inequality constraints will suffice for our purposes. The input itself can be constrained by lower and upper bound (b_l and b_u)

$$b_l \leq u_{t+k} \leq b_u, \tag{123}$$

or the difference between two consecutive inputs $\Delta u_{t+k} = u_{t+k+1} - u_{t+k}$ can be subjected to constraints

$$b_{\Delta l} \leq \Delta u_{t+k} \leq b_{\Delta u} \quad (\text{or} \quad b_{\Delta l} \leq |\Delta u_{t+k}| \leq b_{\Delta u}). \tag{124}$$

The equation containing the quadratic kernel P of the optimal extended Q-function $Q^*_h(x_t, u_{t:t+h})$ can be written as follows

$$Q^*_h(x_t, u_{t:t+h}) = \frac{1}{2} \begin{bmatrix} 1 \\ x_t \\ u_{t:t+h} \end{bmatrix}^T P \begin{bmatrix} 1 \\ x_t \\ u_{t:t+h} \end{bmatrix}. \tag{125}$$

Although this problem could be solved using the explicit MPC [12], another approach involving quadratic programming is used here. The symmetric kernel matrix P can be divided into 9 sections based on the matrix multiplication rules

$$Q_h^*(x_t, u_{t:t+h}) = \frac{1}{2}\begin{bmatrix} 1 \\ x_t \\ u_{t:t+h} \end{bmatrix}^T \begin{bmatrix} P_{11} & P_{1x} & P_{1u} \\ P_{x1} & P_{xx} & P_{xu} \\ P_{u1} & P_{ux} & P_{uu} \end{bmatrix} \begin{bmatrix} 1 \\ x_t \\ u_{t:t+h} \end{bmatrix}. \tag{126}$$

Using symmetry of the P matrix ($P = P^T \Rightarrow$ e.g. $P_{x1} = P_{1x}^T$), result of the multiplication takes the form

$$Q_h^*(x_t, u_{t:t+h}) = \frac{1}{2}u_{t:t+h}^T P_{uu} u_{t:t+h} + x_t^T P_{xu} u_{t:t+h} + P_{1u} u_{t:t+h} + \frac{1}{2}x_t^T P_{xx} x_t + P_{1x} x_t + \frac{1}{2}P_{11}. \tag{127}$$

The minimization problem (122) then changes to

$$u_{t:t+h}^* = \arg\min_{u_{t:t+h}} \left[\frac{1}{2}u_{t:t+h}^T P_{uu} u_{t:t+h} + x_t^T P_{xu} u_{t:t+h} + P_{1u} u_{t:t+h} + \frac{1}{2}x_t^T P_{xx} x_t + P_{1x} x_t + \frac{1}{2}P_{11}\right]. \tag{128}$$

Because x_t is known at this point, $\frac{1}{2}x_t^T P_{xx} x_t + P_{1x} x_t + \frac{1}{2}P_{11}$ is a constant and since constant does not change the position of the minimum, it can be omitted. Therefore, the minimization further simplifies to

$$u_{t:t+h}^* = \arg\min_{u_{t:t+h}} \left[\frac{1}{2}u_{t:t+h}^T P_{uu} u_{t:t+h} + \left(P_{ux} x_t + P_{u1}\right)^T u_{t:t+h}\right], \tag{129}$$

which can be solved by the quadratic programming (2.3.3). Here, the receding horizon principle (2.3.4) used in MPC can be applied. Therefore, only the first input from the $u_{t:t+h}^*$ sequence is used, then the whole computation is repeated at time $t + 1$ and so forth. No learning of the extended Q-function takes place here. To obtain new, updated extended Q-function, the optimal normal one-step Q-function has to be found first.

As mentioned above, the Q-function used to generate the constraints satisfying input sequence has to be optimal. Therefore, this method is suited to be used as an extension of the previous method.

5.2 Partial Problems Solutions

While deriving the major solutions introduced in 5.1, several sub-problems had to be dealt with, namely, which policy is to be followed during the off-policy learning and which technique is best for Q-function evaluation. Both of these partial problems are solved in this section.

5.2.1 Policy Choosing During Off-Policy Learning

During the off-policy learning, some policy π_0 has to be chosen to control the system. Regarding this matter, [1] states the following:

"In this case (Off-Policy Learning)*, the learned action-value function* (Q-function according to our notation) *Q, directly approximates Q^*, the optimal action-value function, independent of the policy being followed. (...) The policy still has an effect in that it determines which state-action pairs are visited and updated. However, all that is required for correct convergence is that all pairs continue to be updated."*

This statement is undoubtedly true. However, it can only be applied to the MDP framework with finite discrete state and input spaces, where the Q-function is represented as a table, where each cell corresponds to a state-action pair. Since in our case, the Q-function is represented by a quadratic kernel P, the demand on the policy π_0 is milder. If the off-policy learning is used to compare n controllers, it is enough for the policy π_0 to be randomly switching between those n controllers. However, it is not enough to use each controller just once. All controllers have to be evaluated in multiple states for us to be able to decide which controller is the best one.

Although the random switching approach works, it is certainly a primitive one. There are many algorithms which would solve this problem better in any sense possible. One of them is the Thompson sampling algorithm [15], which optimally solves the exploration/exploitation dilemma and was originally used for the Multi-armed bandit problem [16].

5.2.2 Q-Function Evaluation

As stated before, considering the linear system and quadratic criterion, the Q-function has a quadratic form

$$Q^\pi(x_t, u_t^\pi) = \frac{1}{2}\begin{bmatrix} 1 \\ x_t \\ u_t^\pi \end{bmatrix}^T P \begin{bmatrix} 1 \\ x_t \\ u_t^\pi \end{bmatrix} = \frac{1}{2} z_t^T P z_t. \tag{130}$$

Therefore, it can be represented as a weighted sum of features ϕ

$$Q^\pi(x_t, u_t^\pi) = \sum_i w_i \phi_i(x_t, u_t^\pi), \tag{131}$$

where w_i are the weights, associated with the features. In our case, the set of all features Φ consists of all monomials of the quadratic or lower order, which can be created by combining $1 + n_x + n_u$ elements from $\{1, x_t, u_t^\pi\}$. The challenge is now to find the weight vector $W = \begin{bmatrix} w_1 & \dots & w_M \end{bmatrix}^T$, where M is the number of monomials, which can be computed as binomial coefficient

$$M = \begin{pmatrix} 1 + n_x + n_u + 1 \\ 2 \end{pmatrix}, \tag{132}$$

where n_x is the length of the state vector and n_u is the length of the input vector.

Recursive Least Squares Method

As stated in section 2.2.1, one way to obtain the vector is to use the recursive least squares method. The goal of the classical least squares method is to find such θ in

$$y_k = z_k^T \theta + e_k, \tag{133}$$

where y_k is the vector of observed values at step k, z_k is a regressor, which is the independent variable, and e_k is the error at step k, so that

$$\sum_{k=1}^{K} e_k^T e_k \tag{134}$$

is minimized. Solution to this problem is expressed by equation

$$\hat{\theta}_K = \left(Z_K^T Z_K\right)^{-1} Z_K^T Y_K, \tag{135}$$

where $\hat{\theta}_K$ is the estimate of vector θ, $Z_K = \begin{bmatrix} z_1 & \dots & z_K \end{bmatrix}^T$ and $Y_K = \begin{bmatrix} y_1 & \dots & y_K \end{bmatrix}^T$.

Suppose now that a new pair $\{z_{K+1}, y_{K+1}\}$ is available. There is no need to compute the new estimate $\hat{\theta}_{K+1}$ by using all $K+1$ data. The $\hat{\theta}_{K+1}$ can be computed as an update of $\hat{\theta}_K$ using the new pair $\{z_{K+1}, y_{K+1}\}$.

Let us denote the $Z_K^T Z_K$ as S_K. Then S_{K+1} takes form

$$S_{K+1} = Z_{K+1}^T Z_{K+1} = \begin{bmatrix} Z_K^T & z_{K+1} \end{bmatrix} \begin{bmatrix} Z_K \\ z_{K+1}^T \end{bmatrix} = Z_K^T Z_K + z_{K+1} z_{K+1}^T = S_K + z_{K+1} z_{K+1}^T. \tag{136}$$

By multiplying (135) by S_K, it can be derived that

$$S_K \hat{\theta}_K = S_K \left(Z_K^T Z_K\right)^{-1} Z_K^T Y_K = Z_K^T Y_K. \tag{137}$$

If we compute the value at $K+1$, we get

$$S_{K+1} \hat{\theta}_{K+1} = Z_{K+1}^T Y_{K+1} = \begin{bmatrix} Z_K^T & z_{K+1} \end{bmatrix} \begin{bmatrix} Y_K \\ y_{K+1}^T \end{bmatrix} = Z_K^T Y_K + z_{K+1} y_{K+1}^T. \tag{138}$$

By combining (136),(137) and (138), we get

$$S_{K+1} \hat{\theta}_{K+1} = S_K \hat{\theta}_K + z_{K+1} y_{K+1}^T \tag{139}$$

Therefore, $\hat{\theta}_{K+1}$ can be evaluated as

$$\hat{\theta}_{K+1} = S_{K+1}^{-1} \left(S_K \hat{\theta}_K + z_{K+1} y_{K+1}^T\right) = \left(S_K + z_{K+1} z_{K+1}^T\right)^{-1} \left(S_K \hat{\theta}_K + z_{K+1} y_{K+1}^T\right). \tag{140}$$

Recursive Least Squares for Q-Function Evaluation

The temporal difference equation derived in 2.2.1 states

$$e_t = r(x_t, u_t) - Q^\pi(x_t, u_t) + \gamma Q^\pi(x_{t+1}, u_{t+1}^\pi) \tag{141}$$

and using the equation (131), it can be rewritten as

$$r(x_t, u_t) = Q^\pi(x_t, u_t) - \gamma Q^\pi(x_{t+1}, u_{t+1}^\pi) + e_t, \tag{142}$$

$$r(x_t, u_t) = \sum_i w_i \phi_i(x_t, u_t^\pi) - \gamma \sum_i w_i \phi_i(x_{t+1}, u_{t+1}^\pi) + e_t, \tag{143}$$

$$r(x_t, u_t) = \sum_i \left[w_i \left(\phi_i(x_t, u_t^\pi) - \gamma \phi_i(x_{t+1}, u_{t+1}^\pi) \right) \right] + e_t, \tag{144}$$

$$r(x_t, u_t) = \left(\Phi_t - \gamma \Phi_{t+1} \right)^T W + e_t, \tag{145}$$

where $\Phi_t = \begin{bmatrix} \phi_1(x_t, u_t^\pi) & \phi_2(x_t, u_t^\pi) & \dots & \phi_M(x_t, u_t^\pi) \end{bmatrix}^T$ and analogically $\Phi_{t+1} = \begin{bmatrix} \phi_1(x_{t+1}, u_{t+1}^\pi) & \phi_2(x_{t+1}, u_{t+1}^\pi) & \dots & \phi_M(x_{t+1}, u_{t+1}^\pi) \end{bmatrix}^T$. As mentioned before, recursive least squares method minimizes $\sum_{k=1}^{K} e_k^T e_k$ by finding the best θ in equation

$$y_k = z_k^T \theta + e_k. \tag{146}$$

Apparently, recursive least squares method can be used for Q-Function evaluation, given the connection between (145) and (146). The observed value y_k becomes the reward $r(x_t, u_t)$. The regressor z_k consists of the difference between sets of monomials at time t and $t+1$. The estimated vector θ is the weight vector W.

Unbiased Estimator

According to [14], the recursive least squares method described above is biased and should not be used in this context. The reason behind the bias is that the method assumes that the stochastic part of the problem is contained within the observed data y_k, or reward $r(x_t, u_t)$ in our case. However, it is actually included in the regressor (z_k), concretely in the set of monomials at time $t+1$ ($\Phi(x_{t+1}, u_{t+1}^\pi)$).

The paper [14] further suggests using

$$\hat{W} = \hat{\theta} = \left(\Phi^T \Phi - \gamma \Phi^T \Psi \right)^{-1} \Phi^T \mathit{\Gamma}, \tag{147}$$

where $\Phi = \begin{bmatrix} \Phi_1 & \Phi_2 & \dots & \Phi_K \end{bmatrix}^T$, $\Psi = \begin{bmatrix} \Phi_2 & \Phi_3 & \dots & \Phi_{K+1} \end{bmatrix}^T$ and $\mathit{\Gamma} = \begin{bmatrix} r(x_1, u_1) & r(x_2, u_2) & \dots & r(x_K, u_K) \end{bmatrix}^T$ instead of (135) to avoid the bias. Equation (147) can be modified to support recursive estimation.

5.3 Tests

This section demonstrates on practical examples why is Q-learning useful for comparing controllers. It is also presented here that Q-learning gives the same results as the Lyapunov equation. Finally, this section shows why using constraints is superior to simply saturating the control input.

5.3.1 Q-Learning – Motivation

The first test which is worth mentioning is the "Motivation test". This test shows the motivation for using the Q-learning to evaluate and compare controllers.

Suppose there are two PID controllers. Those controllers obey the performance/energy trade-off described in 5.1.2. Controller K_1 keeps the control error minimal at the cost of high energy usage. The other controller (K_2) has lower energy consumption but produces a larger error.

The naive way to compare these two controllers would be to let them one by one regulate the system for some limited time (e.g. 48 hours) and then compute their performance as pair

$$\left\{ \tfrac{1}{T} \sum_t u_t, \;\; \tfrac{1}{T} \sum_t |s_t - y_t| \right\}, \tag{148}$$

where T is the total time duration in seconds, u_t is the controller's output (input to the system), s_t is the set-point or reference temperature and y_t is the output or in this case actual temperature in the room. Therefore, the performance is computed as a sum of energy used to control the system and accumulated error.

The other approach would be to use the Q-learning and its cost (C^π) measure introduced in 5.1.1 to compare K_1 and K_2. Particularly, the multi-component cost described in 5.1.2 with partial rewards defined as (118) would be used here so that a pair similar to (148) is obtained.

First, a control (or reference) test is performed. Both controllers are successively let to control the system under identical conditions, i.e. the set-point and outside temperature sequences are identical. Then, under the same conditions, the Q-learning comparing approach is applied. Both controllers are evaluated during only one 48-hour period using the off-policy learning. The policy used to control the system during the evaluation is a random mixture of K_1 and K_2 (see 5.2.1).

	Naive Approach		Q-Learning	
	$\frac{1}{T}\sum_t u_t$ [kW]	$\frac{1}{T}\sum_t \lvert s_t - y_t \rvert$ [°C]	Energy Cost Component [-]	Error Cost Component [-]
K_1	4.49	1.14	252.8	232.6
K_2	4.42	1.42	192.5	235.7

Table 1: Q-Learning Motivation (Control) Test – Results

Since both methods give results in different units (mostly as a result of discounting in (105)), they are not directly comparable to each other. However, because the goal is to minimize both input energy consumption and control error, lower values (green in the table) always mean better controller. As expected, both techniques give the same final results (Table 1). Controller K_1 wins at keeping the error minimal but loses at energy usage and vice versa.

Now, a similar test is performed. This time, however, the difference between the mean outside temperature during the first (K_1) and second (K_2) run will be 2 °C, which certainly could occur during a real test. The Q-learning run will experience this two-degree shift in the middle of its 48-hour period. Outside temperatures used during all experiments are shown in Figure 8.

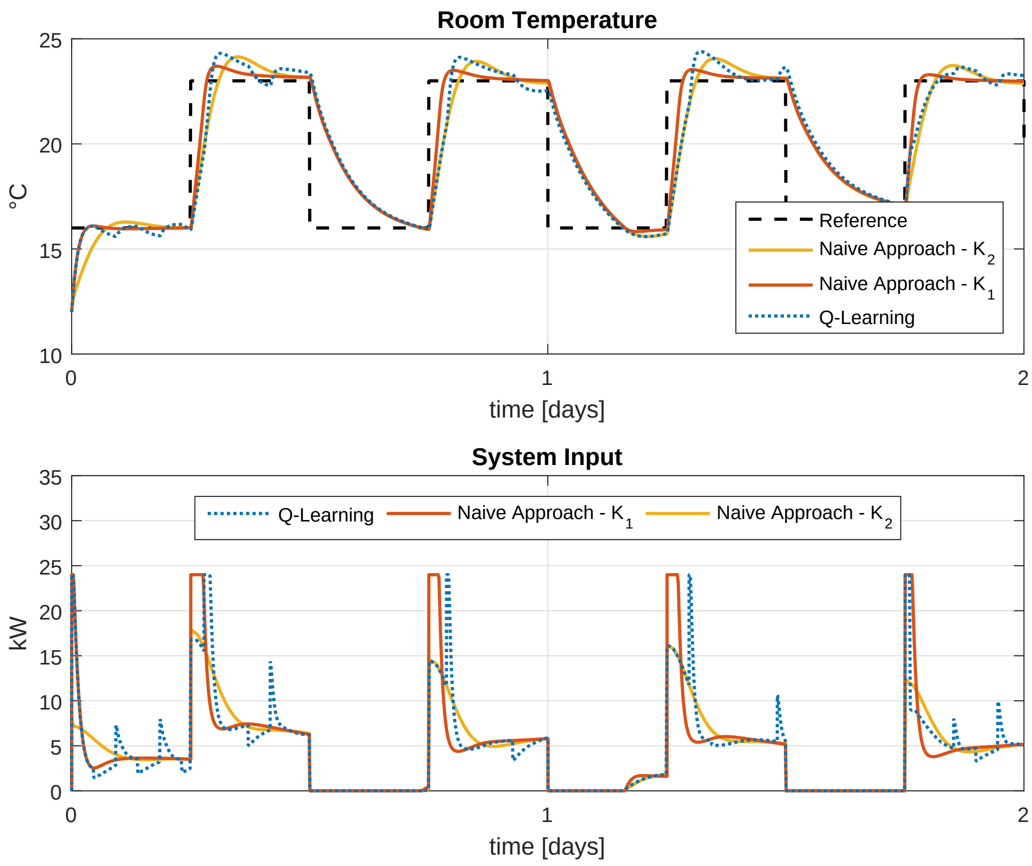


Figure 7: Room Temperature and System Input During Control by K_1, K_2 and Q-Learning (Control Test)

	Naive Approach		Q-Learning	
	$\frac{1}{T}\sum_t u_t$ [kW]	$\frac{1}{T}\sum_t \lvert s_t - y_t \rvert$ [°C]	Energy Cost Component [-]	Error Cost Component [-]
K_1	4.37	1.16	246.6	220.9
K_2	4.54	1.4	171.9	223

Table 2: Q-Learning Motivation Test – Results

Apparently, there is no way to decide which controller is better, just by looking at the room temperature development. In both cases (naive approach and Q-learning), the test results graph (Figure 9) looks very similar to the control test results graph (Figure 7). However, if the actual results are compared (Table 2), it is apparent that according to the naive approach, K_1 is better at both energy and error costs, which is definitely not true, if compared with the control test. On the other hand, Q-learning approach correctly concluded that controller K_1 is better at keeping the error minimal but worse at energy usage and vice versa, as in the control test.

In conclusion, there are several factors which can affect the comparison of n controllers if performed in the naive way. Those factors include the outside temperature and many others, such as solar heat gain, wind or even presence/absence of people in the room. One can never assure identical conditions of those factors during the run of all n controllers. Therefore, the comparison will never be fair to all controllers. The Q-learning solves this problem by comparing

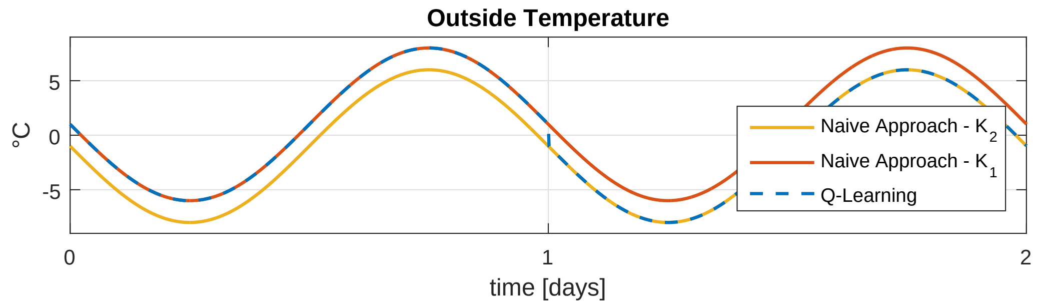


Figure 8: Outside Temperature During Controllers Comparison

the controllers on the same set of data, which assures the fairness. A pleasant side-effect of the Q-learning approach is its time-saving aspect. Since all controllers are evaluated at once, the total time needed to perform the comparison using the Q-learning is only $1/n$ of the time needed to compare the controllers using the naive approach.

5.3.2 Q-Learning Versus Lyapunov Equation

As indicated in section 2.2.5, the purpose of this test is to verify that Q-learning gives the same result as the already known and rigorous approach in the form of the Lyapunov equation. Since knowledge of the system dynamics is essential for this test, a random discrete stable second-order system will be generated for this test instead of using our realistic thermal model.

The test will be performed in the following way. A set of 100 controllers (K_i with $i \in \begin{bmatrix} 1, & 100 \end{bmatrix}$) with quadratic criterion (149) will be tested, and therefore 100 tests will be performed in total. Weight matrices of said controllers are given by the rules specified in (150).

$$J_i = \frac{1}{2}\left(x^T Q_i x + u^T R_i u + 2x^T N_i u\right) \tag{149}$$

$$Q_i \in \begin{bmatrix} 0.01, & 0.99 \end{bmatrix},\ R_i = 1 - Q_i,\ N_i = 0 \tag{150}$$

Two evaluation methods will be applied. First, the C_i value derived in 5.1.1, which uses the quadratic kernel P_i obtained using Q-learning will be used to evaluate the performance of each controller over a 500-second period. Next, a solution of a Lyapunov equation (42) $\bar{P}_i$ for given system dynamics and all controllers K_i will be computed. Then, the P_i kernel will be substituted by $\bar{P}_i$, and $\bar{C}_i$ will be computed for each controller.

This way, a set of 100 C_i and $\bar{C}_i$ values was obtained. Ideally, $|\Delta C_i| = \left|C_i - \bar{C}_i\right|$ should equal zero for each i, or at least be as small as possible. Table 3 displays several important characteristics of $|\Delta C| = \begin{bmatrix} |\Delta C_1|, & \dots, & |\Delta C_{100}| \end{bmatrix}$. Although $|\Delta C|$ does not equal zero, this test clearly confirms that Q-learning gives the same result as the Lyapunov equation when the system dynamics is known and therefore can be used even for problems, where the system dynamics are unknown.

Mean value of $\lvert\Delta C\rvert$	Median value of $\lvert\Delta C\rvert$	Min. value of $\lvert\Delta C\rvert$	Max. value of $\lvert\Delta C\rvert$
8.68×10^{-15}	7.13×10^{-15}	1.73×10^{-18}	3.69×10^{-14}

Table 3: Q-Learning Versus Lyapunov Equation – Results

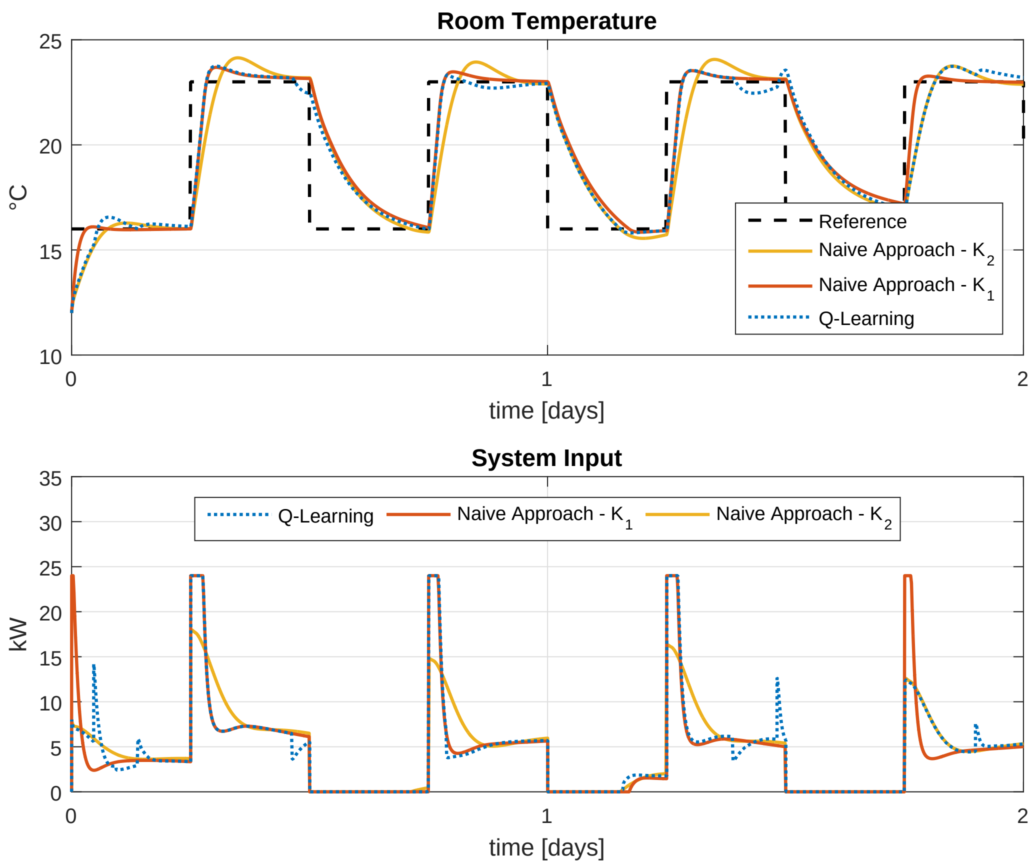


Figure 9: Room Temperature and System Input During Control by K_1, K_2 and Q-Learning

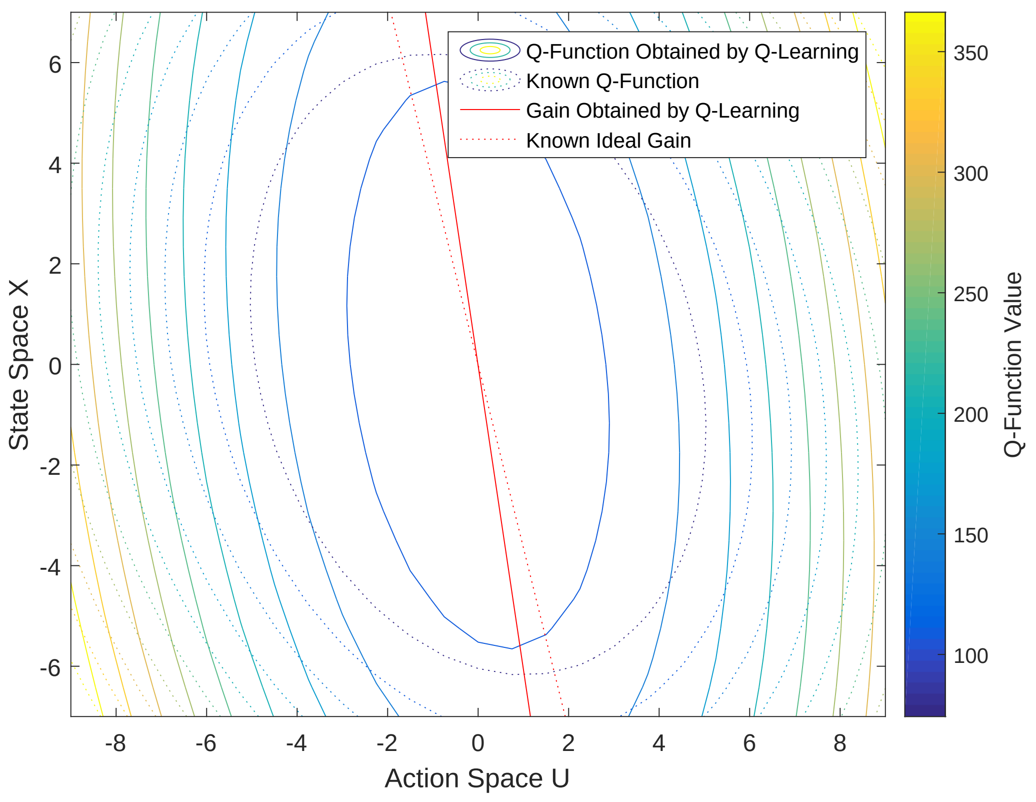
5.3.3 Biased Versus Unbiased Estimator

As the title suggests, this section compares Q-function estimations using the biased (Least Squares Method) and unbiased estimators.

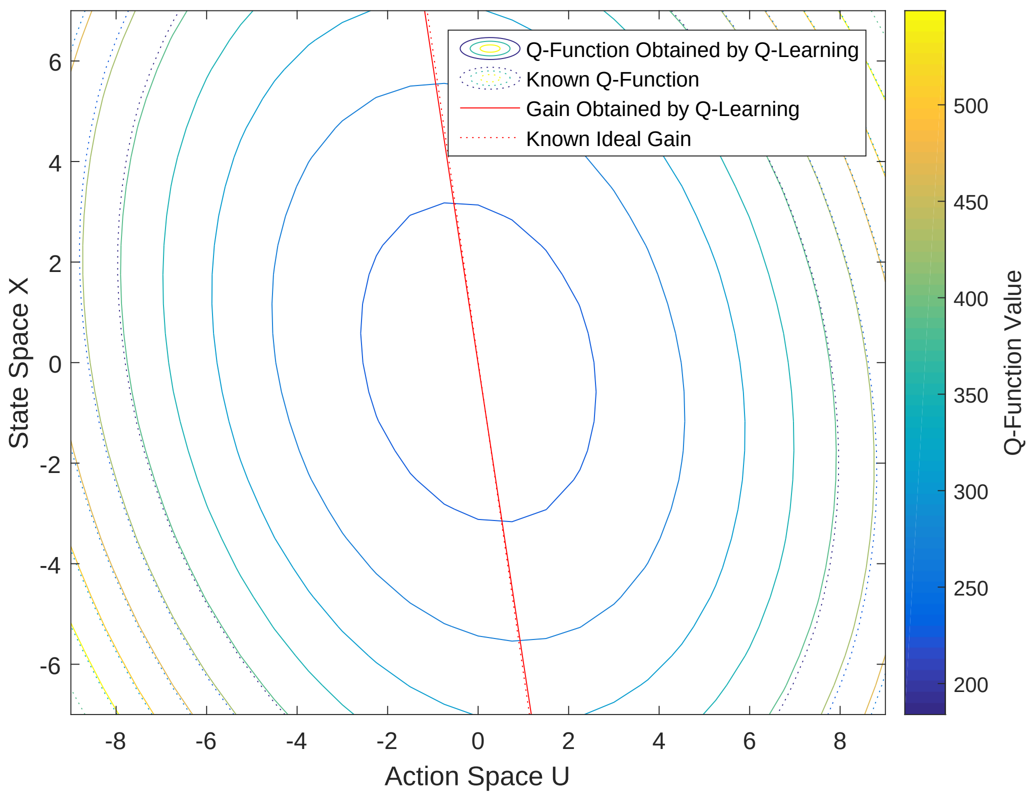
For this test a white noise with standard deviation $\sigma = 1$ is considered, modeling the stochastic part of the regressor as mentioned in 5.2.2. For us to be able to plot the results of the experiment, a first order system with one input and known dynamics was used to generate the data. Identified Q-functions and corresponding derived control gains obtained by both methods were compared to the true Q-function and control gain.

The result is shown in Figure 10. It can be seen that using the biased estimator, the gain obtained by using the Q-learning is not the same as the already known ideal gain. However, using the unbiased estimator results in a gain that is an accurate representation of the known ideal gain.

A second test shows how the bias progresses with increasing standard deviation of the white noise ($\sigma \in \begin{bmatrix} 0, & 0.25 \end{bmatrix}$). Figure 11 shows individual elements of the Q matrix after being estimated using both biased and unbiased estimator.



(a) Known and Obtained Q-function and Control Gain Using Biased Estimator



(b) Known and Obtained Q-function and Control Gain Using Unbiased Estimator

Figure 10: Biased versus Unbiased Estimator

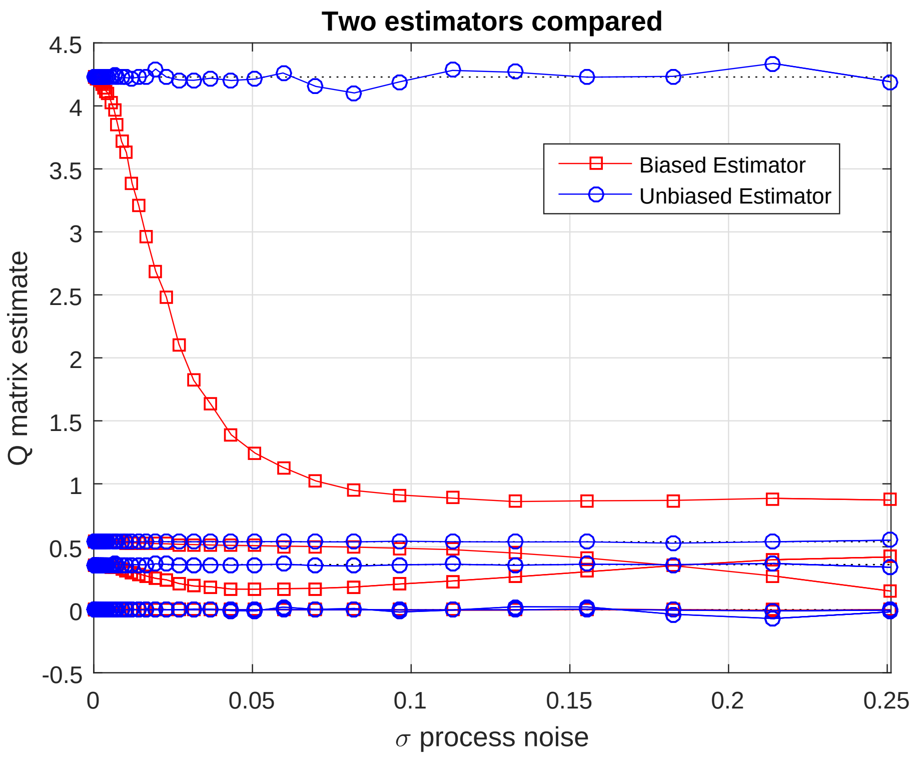


Figure 11: Bias Progression with Increasing Standard Deviation σ

5.3.4 Constrained Control – Motivation

The last test shows motivation for merging the Q-learning with MPC to include constraints and reference preview in the controller design. Once again, the MPC approach will be compared to a naive one. In this case, the naive approach consists of saturating simple LQR-designed state feedback controller.

The test will be performed on a simple discrete second-order system. In both cases, the controller will be designed with weight matrices $Q = 10$ and $R = 0.1$. Allowed input values represented as constraints for the MPC and saturation for the LQR will lie in the interval $\begin{bmatrix} 0, & 35 \end{bmatrix}$.

The performance of both controllers will be compared based on a simple quadratic metrics

$$\text{Cost} = \frac{1}{N} \sum_{i=1}^{N} (s_i - y_i)^T Q (s_i - y_i) + u_i^T R u_i, \tag{151}$$

where N is number of samples, s_i is set-point (reference) at sample i, y_i is output value and u_i is input.

Results of the experiment are summarized in Table 4 and example of the control can be found in Figure 12. The figure shows the performance of MPC and both saturated and unsaturated LQR control. As expected, the saturated LQR, although being designed with the same weight matrices as MPC, results in higher cost. It is also clearly visible, how the MPC “prepares” for the reference shift in advance, instead of acting when it actually occurs. This behavior results in reducing both error and control input energy.

Controller	Saturated LQR	MPC
Cost	136.2	109.7

Table 4: Constrained Control Motivation Test – Results

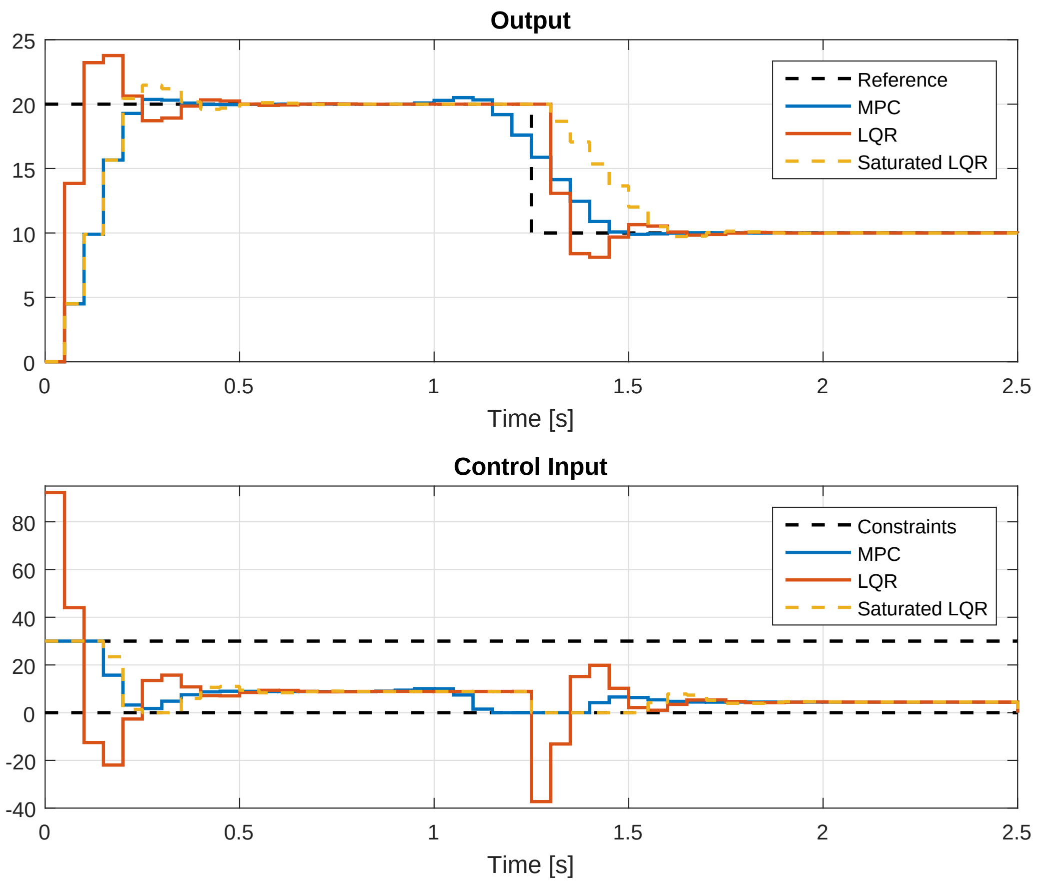


Figure 12: Constrained Control Motivation Test

5.4 Additional Requirements on the Learning Policy

The policy used during the learning period has to have certain properties. As explained in 5.2.1, while comparing n controllers, a random mixture of those controllers is enough to be used during the off-policy learning. However, if our goal is to find the optimal controller, the requirements on the policy are more strict.

Starting from the obvious ones, the boiler must be active during the learning period. Therefore, for example, data obtained during summer are not useful.

The second requirement is that the policy cannot be a linear combination of state values. Therefore, a state feedback controller without any modifications is not useful here. Figure 13 shows the result of using such controller on a simple first-order system. Data obtained during the learning period all lie on the line representing the state feedback controller used during that period. As a result of this, the Q-learning is able to identify the Q-function only on that particular line and the rest of the Q-function is chosen randomly, because there is an infinite number of quadratic Q-functions, which fit the data.

A seemingly simple solution to this problem is to add an exploration noise to the input generated by the state feedback controller (or any other controller used during the learning period). This solution does in fact work, but only under the condition that the exploration noise is larger than the process noise of the system. Therefore, another problem arises: How large should the exploration noise be to exceed the unknown process noise of the system? This question does not have a definitive answer yet. A series of tests on the realistic thermal model suggests that even a slight exploration noise, which does not affect the room temperature by more than 1 °C as opposed to control with zero exploration noise is enough for the Q-learning to identify the Q-function well enough to produce a satisfying optimal controller.

Figure 14 demonstrates how the use or absence of exploration noise affects the optimized state feedback controller. The effect increases with growing temperature difference between learning data and actual later use of optimized controller. This is shown in Figure 14, where the set-point stays around 16.5 °C during the learning period, but rises to circa 24.5 °C during the comparison of optimized controllers. It is apparent that even if the exploration noise does not affect the temperature during the learning period by more than 1 °C, the optimized controller has no problem following set-point that is 8 °C higher. The cost of the optimized controller obtained by using the exploration noise reaches only 19 % of the cost of the controller obtained without the exploration noise. However, it is not a rigorous conclusion and this question remains open.

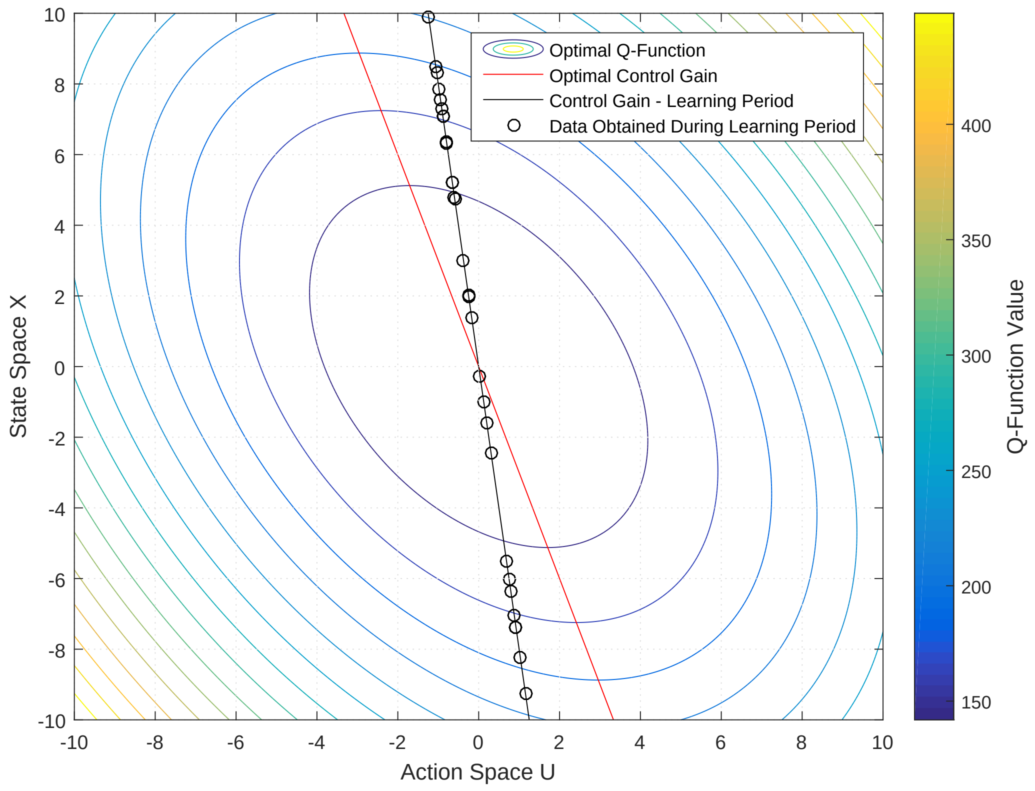


Figure 13: Example of Using State Feedback Controller During the Learning Period

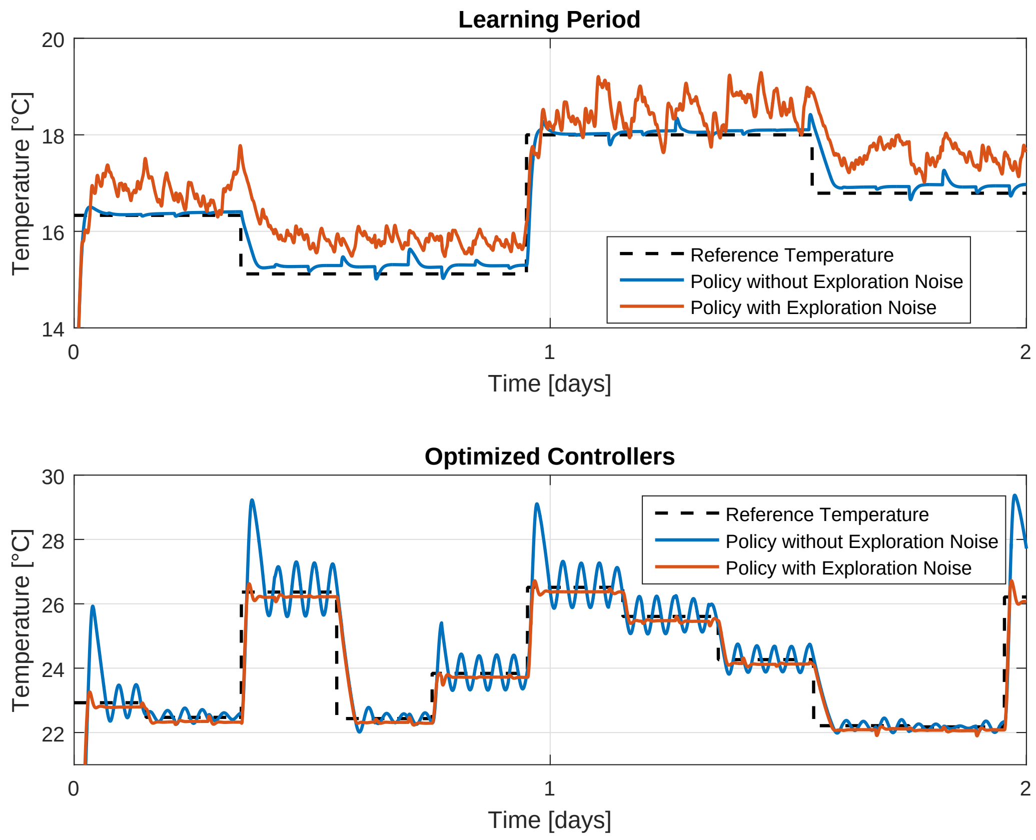


Figure 14: Result of Using and Not Using Exploration Noise During the Learning Period

6 Results

This chapter contains examples of the three main results of this thesis: controllers comparison, generating an optimal controller and merging Q-learning and MPC. All examples in this chapter will be performed using the realistic thermal model introduced in chapter 4.

6.1 Controllers Comparison

The first technique derived in 5.1.1 is the ability to compare controllers based on their performance. One of many advantages of the Q-learning is its capability to compare different types of controllers. For instance PIDs with state feedback controllers. Let us start by comparing three PID controllers.

To compare the performance of said PID controllers, the multi-component cost described in 5.1.2 with partial rewards defined as

$$\rho_1(x_t, u_t) = (u_t - u_{t-1})^2 = \Delta u_t^2 \text{ and } \rho_2(x_t, u_t) = (s_t - y_t)^2 = e_t^2, \tag{152}$$

where s_t and y_t are included in x_t and s_t is the set-point or reference temperature and y_t is the output or in this case actual temperature in the room, with corresponding weights $\omega_1 = \omega_2 = 10$ was used here.

	PID$_1$	PID$_2$	PID$_3$
P	4	100	0.2
I	2×10^{-3}	5×10^{-3}	1×10^{-4}
D	0	1×10^{-3}	1×10^{-4}

Table 5: PID Controllers To Be Compared

As displayed in Table 5, the controllers were purposely set so that one controller is extremely fast (PID$_2$), one is extremely slow (PID$_3$), and one is in the middle (PID$_1$). Therefore, given the weights, PID$_1$ should be chosen as the best one.

Figure 15 demonstrates how was the system regulated by those controllers. It also displays how were the controllers switched. Table 6 reveals results of the test. Columns c_1 and c_2 denote partial costs corresponding to partial rewards ρ_1 and ρ_2. The final column C presents the final cost, computed as

$$C = \omega_1 c_1 + \omega_2 c_2. \tag{153}$$

As expected, PID$_2$ is the worst in the c_1 column, PID$_3$ is the worst in the c_2 column, and the final order from best to worst is PID$_1$, PID$_2$ and PID$_3$.

Controller	c_1 (Δu^2)	c_2 (e^2)	C
PID$_1$ – middle	148.9	24.8	**1736**
PID$_2$ – fast	292.9	16.4	**3094**
PID$_3$ – slow	16.2	297.2	**3134**

Table 6: Comparing PID Controllers – Results

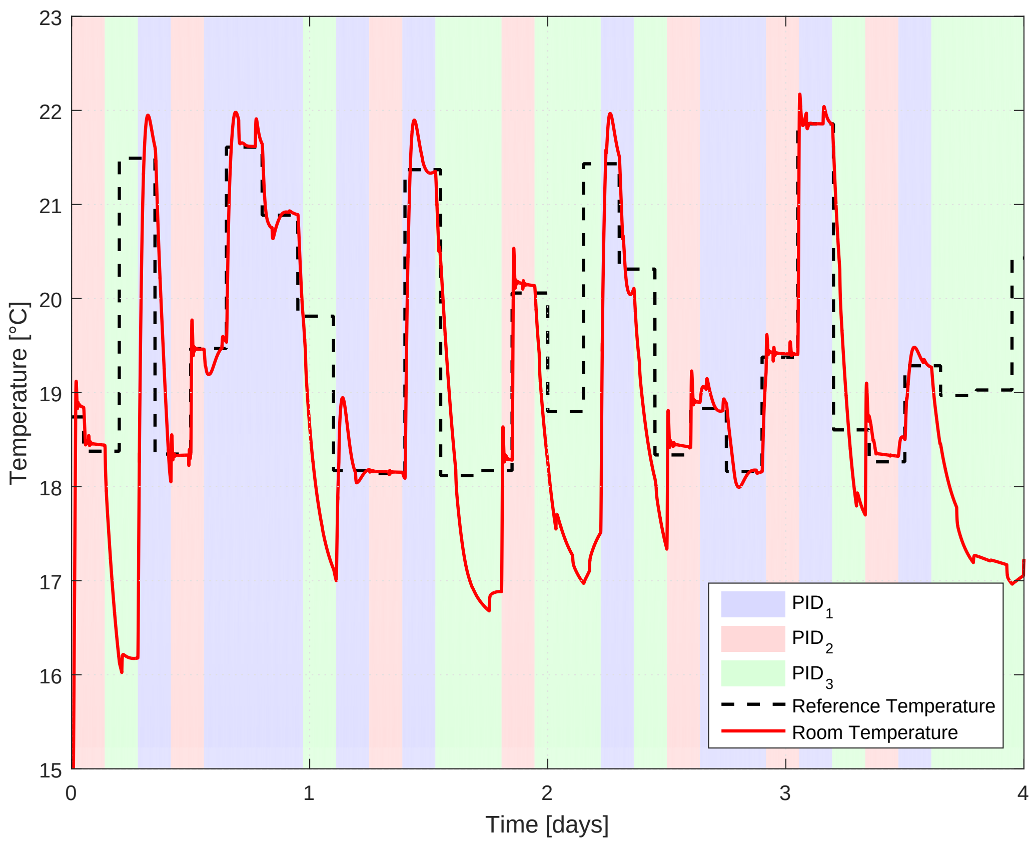


Figure 15: Comparing PIDs

The process of comparing n state feedback controllers among themselves is identical to comparing PIDs. Therefore, it will not be demonstrated here. However, the ability to compare PIDs with state feedback controller will be presented in the next section.

6.2 Optimal Controller

To demonstrate that the Q–learning is able to generate the optimal controller, a similar setup as in the previous section will be used. The definition of partial rewards and their corresponding weights will remain the same. Also, the same three controllers will be used to control the system during the 48-hour learning period. Additionally, an exploration noise will be added to the applied system input to ensure correct Q-function identification (as described in 5.4). The optimal controller will then be compared with the same three PIDs, which were used to its generation.

Results of the comparison are displayed in Table 7. As expected, the total cost C of the optimal state feedback controller is the lowest one among all compared controllers.

Controller	c_1 (Δu^2)	c_2 (e^2)	C
PID$_1$ – middle	36	24.2	**602.4**
PID$_2$ – fast	163.4	19.5	**1829.3**
PID$_3$ – slow	21.3	224.4	**2456.9**
Optimal State Feedback Controller	3.7	22.9	**265.9**

Table 7: Comparing Optimal State Feedback Controller with PIDs – Results

Figure 16 shows how was the room temperature regulated by all four controllers during the comparing phase. However, it does not demonstrate the performance of the optimal state feedback controller well. Therefore, Figure 17 is presented, containing an example of control performed by the optimized state feedback controller over the period of 48 hours. It is clearly visible that the controller easily follows the reference. The only exceptions are the slight perturbations, like the one marked by the red circle, caused by our simple outside temperature model – step changes in outside temperature.

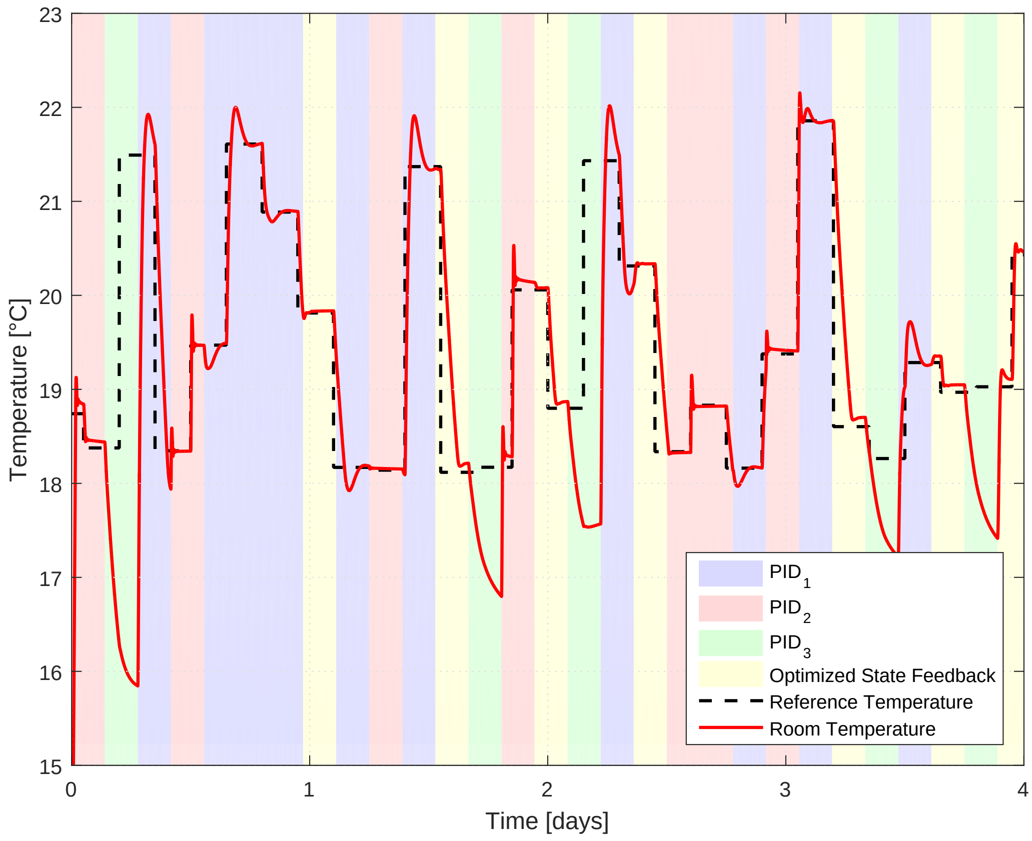


Figure 16: Comparing Optimal State Feedback Controller with PIDs

6.3 Merging Q-Learning and MPC

The final important result of this thesis is the incorporation of constraints and reference preview into the controller design. The theory behind it was explained in 5.1.4. The example shown in 5.3.4 demonstrates how the MPC goes through the reference shift in the middle. For example,

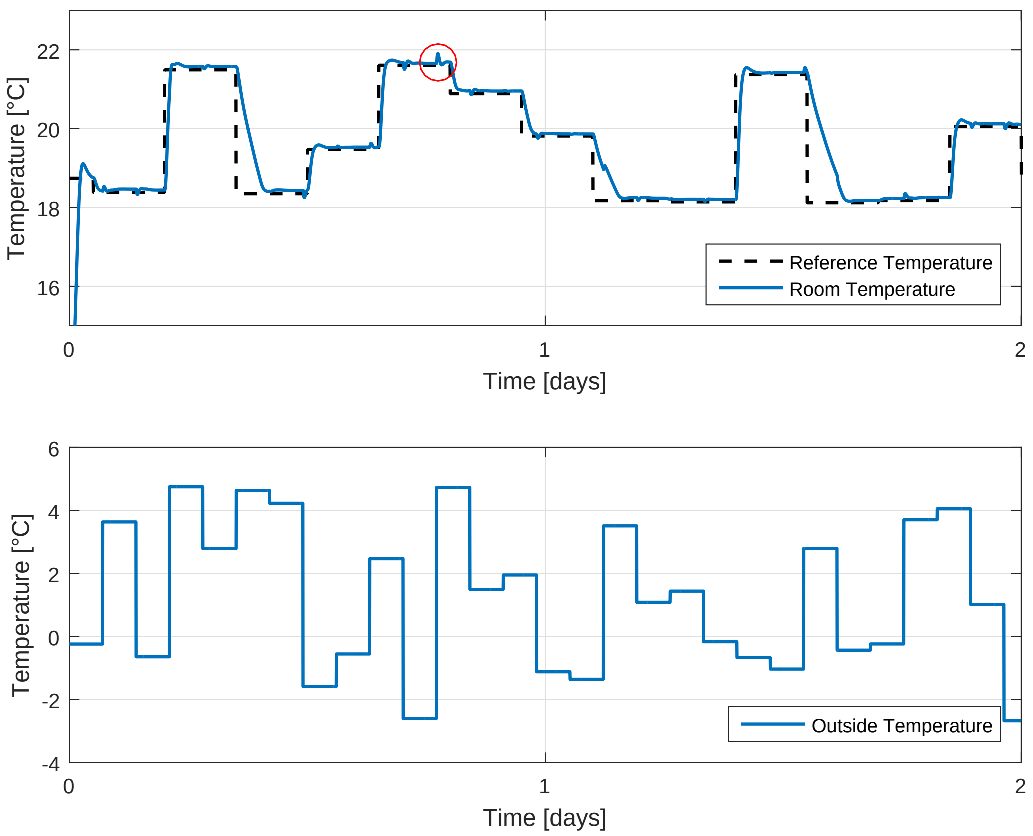


Figure 17: Example of Control Performed by the Optimized State Feedback Controller

if a reference would shift from 18 °C to 22 °C at 8 am, the MPC would control the boiler so that the temperature at 8 am would be 20 °C.

Although this approach indeed minimizes the cost (151), it is not desired when the room temperature is being controlled. The preferred behavior, in this case, is for the room temperature to start increasing soon enough to reach the desired value at the same time the reference does. Similar logic applies for decreasing the reference. The temperature should be the same as the reference until the reference lowers. Therefore, the necessary condition for the desired behavior is for the room temperature to be always higher or equal to the reference temperature.

This behavior can be obtained by lowering the cost of the positive control error (when the room temperature is higher than the reference temperature).

Figure 18 demonstrates the behavior on the realistic thermal model using a boiler with 24 kW maximum power and compares it to an LQ control. The sample time T_s was chosen to be 9 minutes. The whole reference preview set consists of 9 reference previews, where the time difference between each two is T_s. Therefore the reference preview set is $\left\{ \; 9\,\text{min}, \quad 18\,\text{min}, \quad \ldots, \quad 81\,\text{min} \; \right\}$ prior to the reference. Figure 18 (as well as Figure 19) displays only the 81-minute reference preview. It can be seen that the room temperature controlled by the MPC indeed follows the desired behavior. However, there is a slight positive error (circa 0.4 °C) when the reference is at 15 °C. This error is caused by the fact that was already explained in 5.1.3. The system state does not include the estimate of the disturbance, which is usually solved by using the Kalman filter, which cannot be used in this case. The LQR is not able to accomplish the desired behavior, but only if the reference increases. If we were to compare those two controllers only when the reference decreases, the performance would be the same.

Figure 19 shows, how is the room temperature controlled by MPC using several boilers with the maximum power ranging from 8 kW to 24 kW. The preview setting and outside temperature stay the same as in the previous test. As the figure shows, the reference shift from 15 °C to 17.5 °C (labeled by “A” in the figure) does not cause any problems for any boiler. As expected, it can be seen in “B” that the less power the boiler has, the sooner it begins to heat. The problem with low-power boiler settings is shown in regions labeled as “C” and “D”, where both 8 kW and 12 kW boilers are not able to heat the air in the room from 15 °C to 25 °C in time, even though they start at the beginning of the whole 81-minute preview period. It is clearly visible especially in the region “D”, where 16 kW, 20 kW and 24 kW boilers finish their full power heating exactly as the reference shifts, but 8 kW and 12 kW boilers have to use their full potential even beyond this point. This phenomenon, for obvious reasons, grows stronger with increasing shift in reference temperature and decreasing maximum boiler power. Therefore, the length of the preview period should be, to some extent, inversely proportional to the maximum boiler power.

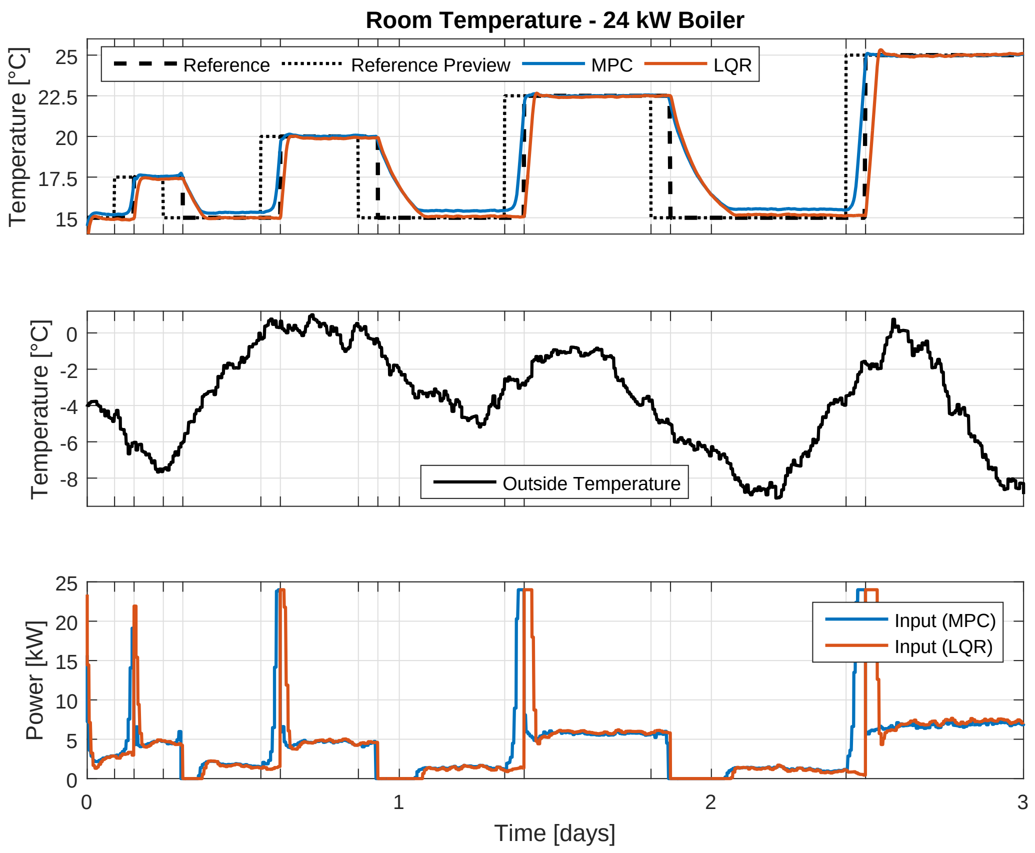


Figure 18: Example of Merging Q-Learning and MPC Compared with LQR – Maximum Boiler Power Constrained at 24 kW

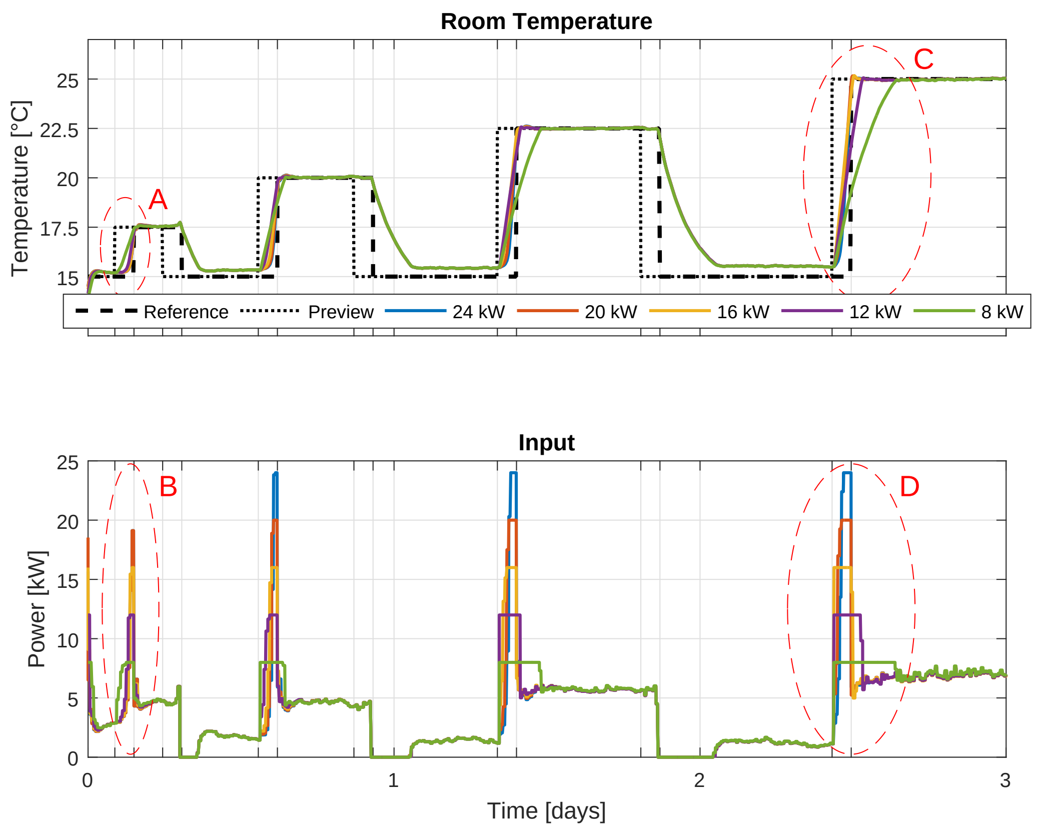


Figure 19: Example of Merging Q-Learning and MPC Using Various Maximum Boiler Power (Constraints)

7 Conclusion

The goal of this thesis was to develop a self-learning algorithm for an intelligent thermostat. The requirement was for the algorithm to be able to find an optimal controller without knowledge of the system dynamics, only from the observed data. From the multitude of artificial intelligence methods, the Q-learning was chosen to accomplish this task. A realistic thermal model was used to test all results and control algorithms mentioned in this thesis.

Before trying to find an optimal controller, the Q-learning was used for an easier task: comparing n controllers (policies). The solution of this problem is presented in section 5.1.1, where a cost C^{π} was defined for us to be able to compare Q-functions corresponding to individual controllers. Examples of using the solution on the realistic thermal model are shown in 6.1.

The Q-learning was then used for the task of finding an optimal controller. The solution of this problem introduced in 5.1.3 involves using the policy iteration method, which was chosen over the value iteration method mostly for its superior convergence speed. The process of determining the optimal controller needs to undergo a certain learning period during which another controller is used to control the system and provide the data needed for the policy iteration method. A connection with the first task is possible here as n controllers may be compared among themselves and a new optimal controller can be designed from the data acquired during the comparing. Such example is presented in 6.2.

The last challenge tackled by this thesis was to somehow incorporate constraints into the process of generating the optimal controller. The need for constraint support rises mostly from the fact that while controlling the temperature in a room, a negative action input (cooling) is often not possible and certainly not wanted, as such cooling would be seen as a waste of energy Therefore, a connection of Q-learning and Model Predictive Control (5.1.4) was needed. This problem was solved by expanding the dimension of Q-function to incorporate the prediction horizon h and using its kernel as an input to the quadratic programming algorithm. An example of control using this modified MPC is displayed in 6.3.

The whole process of developing a smart thermostat from start to finish or in our case from requirements and known theory to fully functional device takes an extensive amount of time. Therefore, the scope of this thesis was limited to the proof of concept. Although this thesis clearly shows that this concept has practical potential, it is not (at least yet) possible to determine its exact energy saving and reference following abilities as it is highly affected by the potential customer settings and configuration of the base-line controller to be compared to.

However, the Q-learning approach definitely brings many benefits. The biggest one being the independence on a model and its ability to find an optimal controller using only real-time data. Compared to a current thermostat, the advantage here is that the initial man-designed controller does not have to be even remotely optimal. Its only role is to stabilize the system during the learning period until the optimal controller is obtained. Although the efficiency of the optimal controller cannot be easily determined, it will surely be at least the same as the one of the initial controller, but most likely higher. This can be seen in 6.2, where the optimal controller reached only 44 % of the best man-designed controller's cost. The Q-learning control concept as described in this thesis can be used in various other fields where the model of the system either cannot or would be hard to obtain.

7.1 Unsolved Problems and Other Future Improvements

The biggest problem yet to be solved is the necessity of adding an exploration noise to the input generated by the controller used during the learning period. And more importantly: How large should the exploration noise be to exceed the unknown process noise of the system? This problem is introduced in section 5.4.

Another unsolved problem concerns the controllers comparison (5.1.1). The policy used during the off-policy learning is created by randomly switching between the compared controllers. Therefore, the comparing technique may be biased towards those controllers that are better at starting from various initial conditions.

Addressing these issues will be the subject of future work. Apart from the unsolved problems, there are other improvements that need to be implemented.

As mentioned multiple times (e.g. 6.3), the state used in the Q-function does not include the estimate of the disturbance. It is usually estimated using the Kalman filter, which cannot be used here because it needs a model of the system. The current solution causes a non-zero steady-state error. Therefore a way to incorporate integral action to the controller has to be found.

Another improvement that should be implemented is the Thompson sampling method mentioned in 5.2.1. This approach will optimally solve the exploration/exploitation trade-off regarding switching between controllers during the off-policy learning.

8 References

[1] R. S. Sutton, A. G. Barto, "*Reinforcement Learning - an Introduction*", Cambridge, MA: MIT Press, 1998.

[2] F. L. Lewis, D. Vrabie, K. G. Vamvoudakis, "*Reinforcement Learning and Feedback Control: Using Natural Decision Methods to Design Optimal Adaptive Controllers*", IEEE Control Systems Magazine, vol. 32, no. 6, pp. 76-105, December 2012.

[3] K. J. Åström, B. Wittenmark, "*Adaptive Control*", Dover, Second edition, 2008, `https://books.google.cz/books?id=L0m_CR-IK24C&printsec=frontcover#v=onepage&q&f=false`, accessed: 2018-05-07.

[4] E. Lavretsky, "*Adaptive Control: Introduction, Overview, and Applications*", `https://www.cds.caltech.edu/archive/help/uploads/wiki/files/140/IEEE_WorkShop_Slides_Lavretsky.pdf`, accessed: 2018-05-07.

[5] Z. Hurák, "*Optimal and Robust Control*", CTU Course, `https://moodle.fel.cvut.cz/mod/folder/view.php?id=63655`, accessed: 2018-05-07.

[6] Matlab environment, `https://www.mathworks.com/products/matlab.html`, accessed: 2018-03-01.

[7] Matlab/Simulink environment, `https://www.mathworks.com/products/simulink.html`, accessed: 2018-03-01.

[8] Quadratic programming by Matlab, `https://www.mathworks.com/discovery/quadratic-programming.html`, accessed: 2018-03-01.

[9] Quadratic programming algorithms by Matlab, `https://www.mathworks.com/help/optim/ug/quadratic-programming-algorithms.html`, accessed: 2018-03-01.

[10] Quadprog function documentation, `https://www.mathworks.com/help/optim/ug/quadprog.html`, accessed: 2018-03-01.

[11] Dynamic programming theorems, `http://lhendricks.org/econ720/ih1/DP_SL.pdf`, accessed: 2018-03-01.

[12] A. Alessio, A. Bemporad, "*A Survey on Explicit Model Predictive Control*", `http://cse.lab.imtlucca.it/~bemporad/publications/papers/nmpc08-survey-explicit.pdf`, accessed: 2018-05-20.

[13] G. Bontempi, S. B. Taieb, "*Statistical Foundations of Machine Learning*", `https://www.otexts.org/1582`, accessed: 2018-03-15.

[14] S. J. Bradtke, A. G. Barto, "*Linear Least-Squares Algorithms for Temporal Difference Learning*", `http://www.incompleteideas.net/bradtke-barto-96.pdf`, accessed: 2018-03-15.

[15] D. J. Russo, B. V. Roy, A. Kazerouni, I. Osband, Z. Wen, "*A Tutorial on Thompson Sampling*", `https://web.stanford.edu/~bvr/pubs/TS_Tutorial.pdf`, accessed: 2018-04-05.

[16] J. Duchi, "*The Multi-Armed Bandit Problem*", `https://web.stanford.edu/class/cs229t/Lectures/bandits.pdf`, accessed: 2018-04-05.

[17] D. L Kleinman, "*On an Iterative Technique for Riccati Equation Computation*", `http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1098829`, accessed: 2018-04-05.