HOCHSCHULE
RAVENSBURG-WEINGARTEN
UNIVERSITY
OF APPLIED SCIENCES

In-Domain Control of a 1-Dimensional Heat Equation

Embedded Control Seminar

Wintersemester 2023/2024

Master Electrical Engineering and Embedded Systems
Master Mechatronics

Ravensburg-Weingarten University of Applied Sciences

Abdul Ghani Shahid Qureshi	Matriculation Number: 19841378 (Erasmus Student)
Jan Lauble	Matriculation Number: 36130
Nico Scheiter	Matriculation Number: 36290

19.01.2024

Abstract

Accurate control of heating processes remains an ongoing topic of research due to the wide variety of applications in the process, semiconductor and manufacturing industries. This report investigates the in-domain control of the average temperature in a one dimensional, insulated rod using multiple heat sources.

To this end, the heat diffusion in the rod is modelled by the heat equation which is a partial differential equation. The report introduces a method to spatially discretize the system to obtain a finite dimensional state-space representation in the form of a linear system of ODEs. This model is used to design an LQR feedback controller and a feedforward controller.

The control algorithm is implemented in C-Code and incorporated into a simulation environment by compiling it to a MATLAB executable file in order to increase the computational efficiency and to perform Software-in-the-Loop tests.

Simulations are performed to investigate the open- and closed-loop behavior of the system. The controller is able to track a variety of different reference inputs as long as the bandwidth of the reference is limited and a tracking error is acceptable. Future research directions include the use of output feedback to improve tracking and disturbance rejection as well as methods from optimal control like Model Predictive Control to take constraints of the heat sources into account.

Contents

List of Figures

List of Tables

List of Program Code

Abbreviations

Abbreviation	Description
API	Application Programming Interface
CFD	Computational Fluid Dynamics
FDM	Finite Difference Method
HVAC	Heating, Ventilation and Air Conditioning
LQR	Linear Quadratic Regulator
MEX	MATLAB Executable
MIMO	Multiple-Input-Multiple-Output
MPC	Model Predictive Control
ODE	Ordinary Differential Equation
PDE	Partial Differential Equation
RK4	Runge-Kutta Integration Method
SIL	Software-in-the-Loop

Equations

Symbol	Unit	Description
b	1	Input Coefficient for Heat Source
e	K	Tracking Error
k_i	1	Runge-Kutta Intermediate Variable
m	1	Number of Inputs
n	1	State Dimension
r	K	Reference Input
t	s	Time
u	$\frac{\mathrm{K}}{\mathrm{s}} = \mathrm{W}$	Heat Source Input
x	m	Position along the rod
y	K	System Output
Δt	s	Time Step
Δx	m	Spatial Discretization Stepsize
J	1	Cost Function
L	m	Length of the rod
N	1	Number of Spatial Discretization Steps
T	K or °C	Temperature
$\mathbf{A}$	$\frac{1}{\mathrm{s}}$	System Dynamics Matrix
$\mathbf{A_{cl}}$	$\frac{1}{\mathrm{s}}$	Closed-Loop Dynamics Matrix
$\mathbf{B}$	1	System Input Matrix
$\mathbf{C}$	1	System Output Matrix
$\mathbf{I}$	1	Identity Matrix
$\mathbf{K}$	$\frac{1}{\mathrm{s}}$	Feedback Matrix
$\mathbf{W}$	$\frac{1}{\mathrm{s}}$	Feedforward Matrix
$\mathbf{Q}$	1	LQR State Cost Matrix
$\mathbf{P}$	1	Solution to the Riccati Equation
$\mathbf{R}$	1	LQR Input Cost Matrix
$\mathcal{C}$	1	Controllability Matrix
α	$\frac{\mathrm{cm}^2}{\mathrm{s}}$	Thermal Diffusivity
λ	$\frac{1}{\mathrm{s}}$	Eigenvalue

Symbol	Unit	Description
σ	m	Length-Scale of a Gaussian

1 Introduction

In the realm of control theory and thermodynamics, the regulation of temperature within one-dimensional systems is a fundamental challenge with applications spanning from materials science to industrial processes. The one-dimensional heat equation provides a mathematical foundation for understanding heat transfer in such systems, describing how temperature evolves over time and along a single spatial dimension. Employing control strategies to influence and maintain desired temperature profiles in one-dimensional systems is critical for achieving desired outcomes in various engineering and scientific endeavors.

[1] shows how heat plays a crucial role in the ongoing decarbonization of industry. Novel control methods for heat management play an important role in improving the efficiency of these processes and thereby reducing emissions. Applications of heat control can be found for instance in the production of steel [2] or in heat treatment to improve certain material properties [3][4]. Precise heating is also required for applications in the semiconductor industry [5] and electronics manufacturing [6].

While the concrete mechanism of heating differs depending on the applications, the same physical laws can be used to describe a wide range of systems.

In this report, we will dive into the fundamental concepts and techniques behind state-space control, matrix discretization, and their application to the one-dimensional heat equation. Understanding these principles is crucial first step in solving real-world problems related to temperature regulation and heat management. This report is intended to be a logical starting point before considering more advanced methods for simulation and control.

We consider the problem of controlling the average temperature within an idealized, one-dimensional, insulated rod. While this scenario might seem overly simple, it illustrates many of the same considerations required for more realistic applications.

To start, chapter 2 discusses state-of-the-art simulation and control methods for heating systems. Next, in chapter 3.1 the system under consideration in this report is introduced and a model is developed. This model is then used in chapter 4 to develop an LQR feedback controller and a feedforward controller to regulate the average temperature in the rod. Chapter 5 provides details regarding the implementation of the simulation and control algorithm. Finally, chapter 6 presents simulation results during open- and closed-loop operation and discusses the strengths and weaknesses of the developed control method.

2 State of the Art

Heat simulation and control is a topic of interest in almost every engineering domain spanning from automotive to process to energy applications. This explains the vast number of techniques that evolved to enable modelling and simulation of heat processes.

Lately, modeling and simulation techniques and the use of high-speed computers have greatly improved the accuracy of thermal and heat transfer-related analysis. An increasing amount of models is being developed, tested, and applied. This allows engineers to gather reliable results and to predict system behavior. As a short overview, an excerpt of methods used to simulate heat transfer is listed below [7]

- Molecular Dynamics Simulations
- Monte Carlo Simulation
- Normal Mode (Harmonic) Analysis
- Computational Fluid Dynamics (CFD)
- Finite Difference Method (FDM)

Molecular dynamics is a technique in which physical movements of atoms and molecules are simulated using computers. The method allows atoms and molecules to interact for some time in a simulation. This generates a view of the motion of individual atoms. The results of molecular dynamics simulation can be used in various fields such as thermodynamics, biology, chemistry, materials science and engineering, statistical mechanics and nanotechnology. [8]

Monte Carlo methods/simulations are a set of simulation techniques that rely on repeated random sampling to compute their results. They are often used in computer simulations of physical and mathematical systems. Monte Carlo methods are especially useful for simulating systems with many coupled degrees of freedom such as fluids, disordered materials, strongly coupled solids, and cellular structures. [9]

Normal mode (harmonic) analysis is a method of simulation in which the characteristic vibrations of an energy-minimized system and the corresponding frequencies are determined assuming its energy function is harmonic in all degrees of freedom. Normal mode analysis is less expensive than molecular dynamics simulation but requires much more memory. [10]

Computer Fluid Dynamics is often used to simulate the flow of fluids and heat transfer in systems with fluid involvement, such as air or liquid cooling in electronic devices, HVAC systems, and industrial processes [11]. Finite Difference Method supports numerical analysis of heat transfer problems through discretization and approximation of the governing equations. When analyzing a problem in a CFD application, the first step is to discretize the partial differential equation, then approximate and calculate heat transfer to derive a numerical solution. In different heat flow systems, systems designers can use CFD solvers to apply the finite difference method to heat transfer equations. [12]

The finite difference method is the closest to the approach used in this report. This reduces the continuous simulation domain to a finite dimensional state which enables the application of state-space methods. A short introduction to state-space representation can be found in [13].

Linear state-space representations of Multiple-Input-Multiple-Output (MIMO) systems lend themselves to LQR control. LQR control has successfully been applied to high-dimensional problems [14]. For instance in [15], LQR control is applied to selectively cool steel profiles. In [16], the same authors present an efficient method to numerically solve the LQR problem in the high dimensional case without explicitly solving the Algebraic Ricatti Equation. In [17], ten different methods to control the heat in a rod are compared. The LQR controller ranks among the best performing methods in this comparison.

This motivates the use of an LQR controller for the task of in-domain heat control in a rod that is considered in this report.

3 Development of a Heat Model

In this chapter, the physical system under consideration is introduced. A mathematical model to describe this system is first formulated in continuous time as a Partial Differential Equation (PDE). First, this model is discretized in space to obtain a state-space model. Then, the model is also discretized in time to obtain a discrete-time system of Ordinary Differential Equations (ODEs) which enables numerical simulation. The model developed in the following chapters is subsequently used in Chapter 4 to develop a control system.

3.1 Heat Equation

Figure 3.1 shows a graphical depiction of the physical setup investigated in this report. The rod of length L is assumed to be infinitely thin, i.e. the temperature distribution along its width is constant. Therefore, this reduces to a one-dimensional problem in x. The core objective of this study is to develop an algorithm to control the localized heat sources u in a way that achieves the control objective (e.g. to keep the average temperature along the rod constant). To this end, this chapter introduces the theoretical foundations necessary to develop a simulation which is in turn used to develop and test the control algorithm.

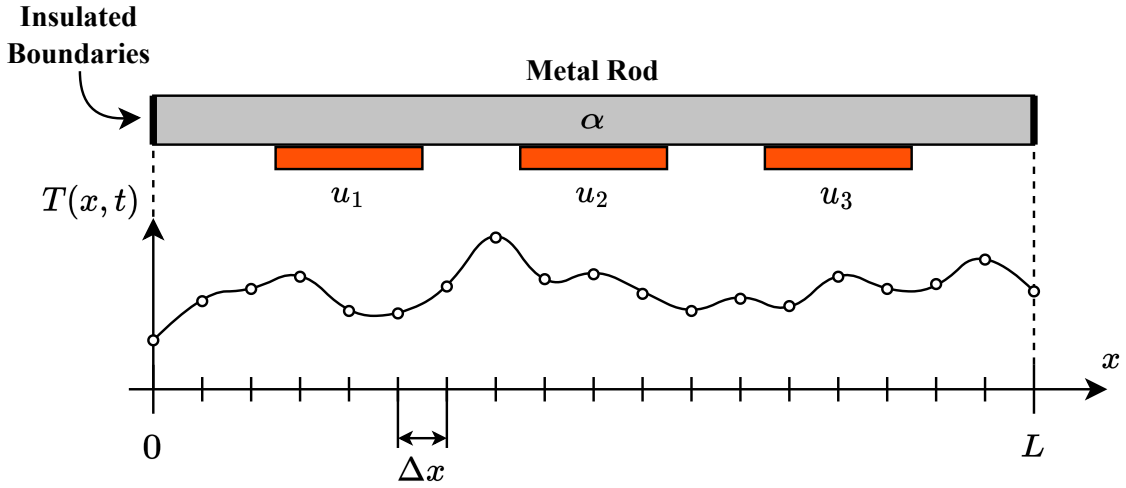


Figure 3.1 – Metallic rod of length L with thermal diffusivity α, three localized heat sources u and insulated boundaries.

This involves solving the heat equation, a PDE that describes the variation of temperature in the rod, using numerical methods [18, p. 3]. Equation 3.1 introduces the heat equation where $T(x,t)$ is the temperature as a function of the position in the rod x and time t.

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2} \tag{3.1}$$

The heat equation relates the time derivative of the temperature with the second spatial derivative. α is the thermal diffusivity [19]. It is a positive constant and describes how quickly heat diffuses. Intuitively, it can be understood as a measure of how fast, large differences in temperature along the object of interest smooth out. A large value of α corresponds to a fast diffusion.

We assume that the rod is completely insulated and that there is no heat radiation. This assumption leads to the boundary conditions for the ends of the rod shown in equation 3.2.

$$\frac{\partial T}{\partial x}(0, t) = 0 \tag{3.2a}$$

$$\frac{\partial T}{\partial x}(L, t) = 0 \tag{3.2b}$$

These boundary conditions are also known as *Neumann Conditions* and they ensure that there is no heat flow across the ends of the rod [20].

Since we are concerned with control of some aspect of the heat distribution, the heat sources u need to be introduced in the formulation of the heat equation. This is shown in equation 3.3 where the dependence on the position x along the rod is made explicit.

$$\frac{\partial T}{\partial t}(x) = \alpha \frac{\partial^2 T}{\partial x^2}(x) + \sum_i b_i(x) u_i(t) \tag{3.3}$$

u_i refers to the i-th heat source input and is in general a function of time. Depending on the control law, it might also be formulated as a function of the state i.e. the temperature.

b_i is a function that describes the effect of heat source u_i at position x. Figure 3.2 shows the input functions for idealized, uniform heating. The functional description of $b_i(x)$ can be formulated as follows where $x_{left,i}$ and $x_{right,i}$ are the left and right boundary of the heat source u_i and b_i is a scaling factor

$$b_i(x) = \begin{cases} b_i & \text{if } x \in [x_{left,i}, x_{right,i}] \\ 0 & \text{otherwise.} \end{cases} \tag{3.4}$$

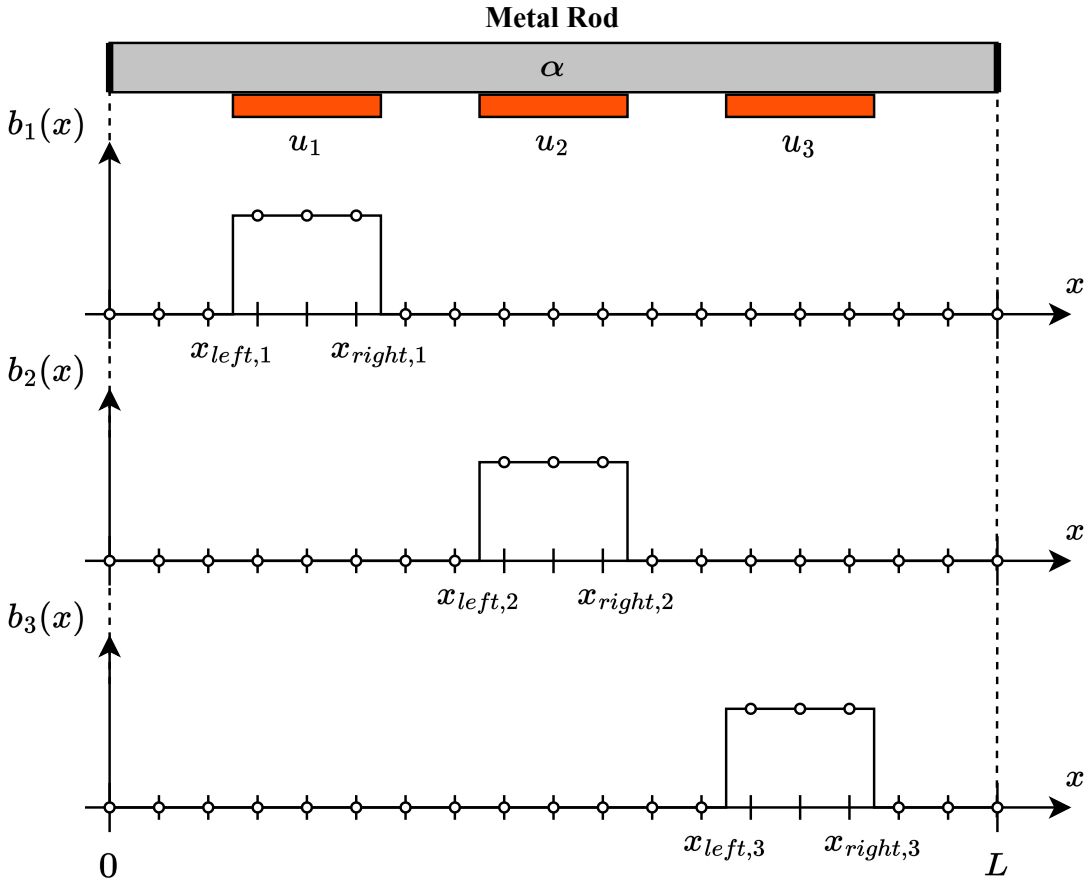


Figure 3.2 – The functions $b_i(x)$ model the effect of the heat source inputs u_i as a function of the position along the rod. Here, an idealized, uniform heating behavior is shown.

This is often a limiting assumption since it is physically unrealistic to have a discontinuous heat input that does not affect neighboring positions. One possible way to make the model more realistic is by using a continuous functional representation that more accurately reflects the physical characteristics of a heat source. Figure 3.3 shows the functions $b_i(x)$ for smooth, gaussian-like heat inputs.

Equation 3.5 is one way to mathematically model this behavior.

$$b_i(x) = b_i \exp\left(-\frac{(x - x_i)^2}{2\sigma_i^2}\right) \tag{3.5}$$

σ_i is a lenght-scale parameter that is used to adjust the steepness of the bell curve. A low value for σ_i corresponds to a steep, impulse like heating and a large value for σ_i describes a smoother, more uniform heat source.

To complete the modelling of the physical system and to obtain a well-posed problem, an

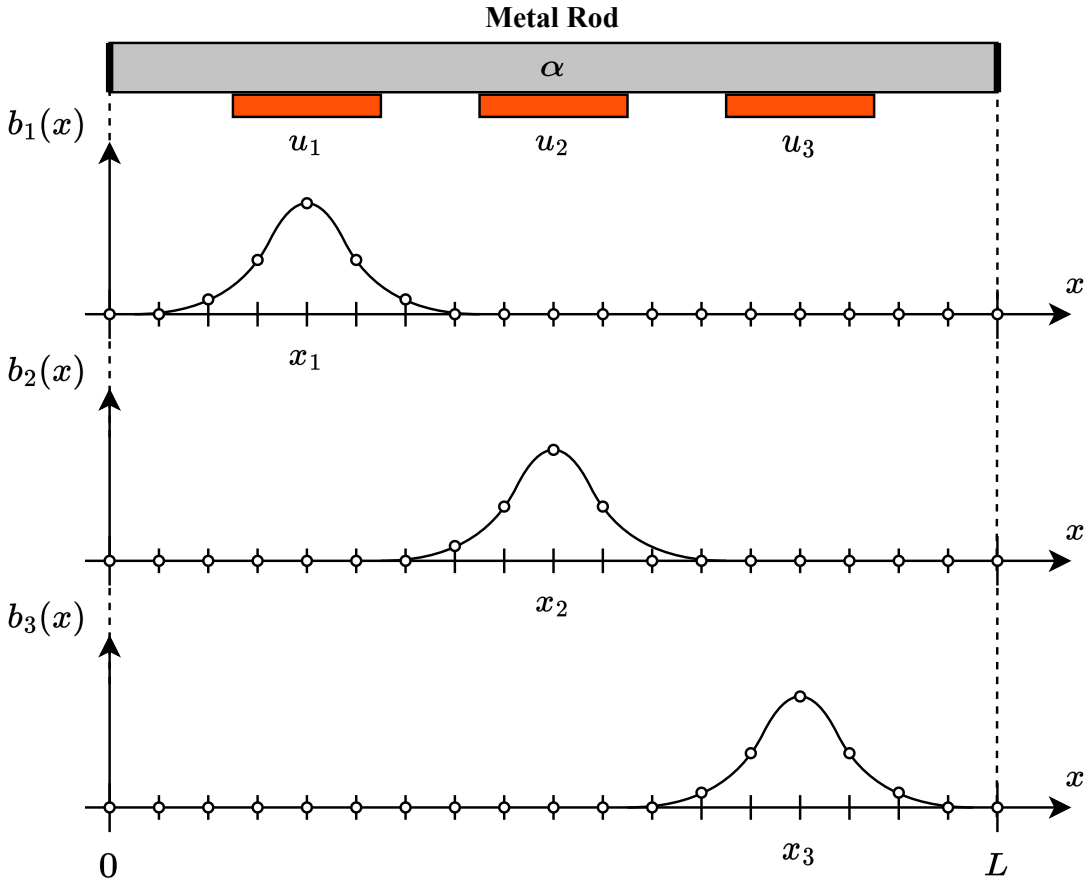


Figure 3.3 – To model a non-uniform heat input, a more complex functional form like a gaussian function can be chosen for b_i.

initial condition has to be provided to specify the heat distribution at time $t = 0$

$$T(x, 0) = T_0(x). \tag{3.6}$$

3.2 Spatial Discretization

The model developed in this chapter aims to provide insights into the thermal dynamics of solid objects, which is crucial in many engineering applications, including material science, thermal management, and the design of cooling systems. To simulate the heat in the rod, a discretized model formulation is necessary. More importantly, a finite-dimensional state is necessary for the development of a suitable control algorithm in Chapter 4.

The methodology adopted in this report revolves around the spatial discretization of the continuous system (the rod). This is achieved by applying the finite difference method for spatial discretization.

More precisely, the second order central difference method is used to discretize the spatial domain as shown in equation 3.7 [21].

$$\frac{\partial^2 T}{\partial x^2}(x) \approx \frac{\frac{T(x+\Delta x)-T(x)}{\Delta x} - \frac{T(x)-T(x-\Delta x)}{\Delta x}}{\Delta x} = \frac{T(x+\Delta x) - 2T(x) + T(x-\Delta x)}{\Delta x^2} \tag{3.7}$$

Δx is the spatial discretization step size as depicted in Figure 3.1. The choice of Δx is a trade-off between simulation fidelity and computational cost. This is crucial for the development of a controller on hardware with limited computational resources. Since simulation and control have different hardware constraints, a finer discretization could be chosen for simulation and a larger discretization step could be used for control. To keep the following sections consistent and easy to follow along, the same discretization step is used for simulation and control.

Spatial discretization only allows the evaluation of the temperature at discrete steps Δx. If required, the values in between can be obtained for example through linear interpolation or by using piece wise cubic splines. This is an alternative way to obtain a smooth result with a comparatively large discretization step.

Applying the spatial discretization to equation 3.1 results in equation 3.8.

$$\frac{\partial T}{\partial t}(x) = \alpha \frac{T(x+\Delta x) - 2T(x) + T(x-\Delta x)}{\Delta x^2} \tag{3.8}$$

Through the spatial discretization, the heat equation is now expressed as a linear system of coupled ODEs in time which can be solved by standard methods for linear state-space models and ODE systems. The resulting linear system is shown in equation 3.9.

$$\begin{bmatrix} \partial T_1/\partial t \\ \partial T_2/\partial t \\ \vdots \\ \partial T_N/\partial t \end{bmatrix} = \begin{bmatrix} -2 & 2 & 0 & \dots & \dots & \dots & 0 \\ 1 & -2 & 1 & 0 & \dots & \dots & 0 \\ 0 & 1 & -2 & 1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & 0 & 1 & -2 & 1 \\ 0 & \dots & \dots & \dots & 0 & 2 & -2 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_N \end{bmatrix} \tag{3.9}$$

The first and last lines are different to account for the boundary conditions introduced in equation 3.2. As a shorthand, standard state-space notation is adopted by defining a

state vector $\vec{T}$

$$\vec{T} = \begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_N \end{bmatrix} \tag{3.10}$$

and by rewriting the system in matrix-vector notation

$$\dot{\vec{T}} = \mathbf{A}\vec{T}. \tag{3.11}$$

To include the heat sources u in the spatially discretized model, an input matrix $\mathbf{B}$ is added to the system for the example with three inputs u_1, u_2, and u_3

$$\dot{\vec{T}} = \mathbf{A}\vec{T} + \mathbf{B}\vec{u} = \mathbf{A}\vec{T} + \begin{bmatrix} | & | & | \\ b_1(x) & b_2(x) & b_3(x) \\ | & | & | \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}. \tag{3.12}$$

with the input vector $\vec{u}$ defined as

$$\vec{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}. \tag{3.13}$$

Matrix $\mathbf{B}$ contains the input functions $b_i(x)$, evaluated at the spatial discretization steps. For the uniform input function shown in Figure 3.2 the input matrix $\mathbf{B}$ is defined in equation 3.14.

$$\mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.14}$$

The size of the input matrix is $\mathbf{B} \in \mathbb{R}^{(N,m)}$ where N is the number of spatial discretization steps and m is the number of inputs u_i. Here, the heat sources act uniformly across a range corresponding to three discretization steps Δx.

Figure 3.4 shows the block diagram of the state space model developed in this section with dynamics matrix $\mathbf{A}$, input matrix $\mathbf{B}$ and output matrix $\mathbf{C}$.

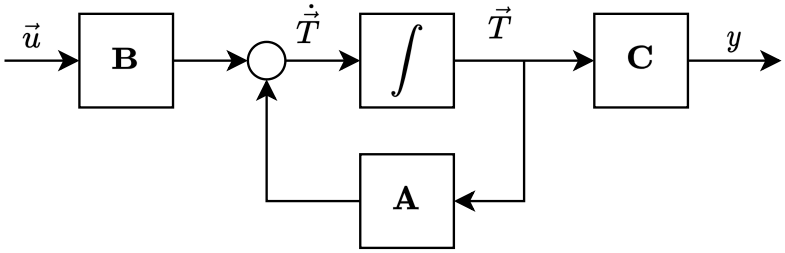


Figure 3.4 – Block diagram of the linear state-space model obtained by spatially discretizing the heat equation with input $\vec{u}$.

The output matrix/vector $\mathbf{C}$ is used to calculate the output y as a linear combination of states. $\mathbf{C}$ is defined according to the control objective. In this report we consider the regulation of the average temperature in the rod. The average temperature is calculated

according to equation 3.15 and can directly be expressed in matrix-vector notation. In this case $\mathbf{C}$ is a row vector.

$$y = \frac{1}{N}(T_1 + T_2 + \cdots + T_N) = \begin{bmatrix} \frac{1}{N} & \frac{1}{N} & \dots & \frac{1}{N} \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_N \end{bmatrix} = \mathbf{C}\vec{T} \tag{3.15}$$

3.3 Discretization in Time

To simulate the behavior of the system over time, standard methods from numerical simulation of ODE systems are considered. While we are considering a linear system of ODEs which can be solved by more efficient methods using matrix exponentials [22, pp. 35–41], we choose to adopt a framework that enables the solution of arbitrary nonlinear systems of ODEs to allow future extensions of the model that incorporate nonlinear effects.

The simplest possible method is the discretization of the continuous time dynamics of equation 3.12 through *Euler's Method* [23, Ch.2]. *Euler's method* for integration of a system of ODEs $\dot{\vec{T}} = f(\vec{T}, \vec{u})$ with state-vector $\vec{T}$ is shown in equation 3.16. In our case, the function f is the right-hand side of equation 3.12.

$$\vec{T}_{k+1} = \vec{T}_k + \Delta t \cdot f(\vec{T}_k, \vec{u}_k) \tag{3.16}$$

Forward Euler is a first order method i.e. the error scales linearly with the discretization step size Δt. To improve the accuracy of the discretization, higher order methods where the error scales more favorably can be used. This requires a higher computational effort but the resulting increase in accuracy of the numerical solution can be significant.

For the simulations in this report, we choose the standard fourth-order *Runge-Kutta (RK4)* method to solve the linear state-space system [23, p. 74]. As shown in equation 3.17, this method calculates intermediate steps within each time interval to estimate the value of the state at the next timestep, significantly improving the accuracy of the solution compared to first order methods like *Forward Euler*.

$$
\begin{aligned}
k_1 &= f(\vec{T}_k, \vec{u}_k) \\
k_2 &= f(\vec{T}_k + \frac{\Delta t}{2} \cdot k_1, \vec{u}_k) \\
k_3 &= f(\vec{T}_k + \frac{\Delta t}{2} \cdot k_2, \vec{u}_k) \\
k_4 &= f(\vec{T}_k + \Delta t \cdot k_3, \vec{u}_k) \\
\vec{T}_{k+1} &= \vec{T}_k + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4)
\end{aligned} \tag{3.17}
$$

- k_1 is the slope of the system of equations at the beginning of the interval, using $\vec{T}_k$ (current temperature distribution at timestep k)
- k_2 uses the midpoint of the interval between the current and next timestep, to make a slope estimate by evaluating the ODE system at $\vec{T}_k + \frac{\Delta t}{2} \cdot k_1$.
- k_3 is similar but uses the slope k_2 for the midpoint estimate
- k_4 calculates the slope at the end of the interval, by evaluating the ODE system at $\vec{T}_k + \Delta t \cdot k_3$

The final value for the temperature distribution $\vec{T}_{k+1}$ at the next timestep $k+1$ is obtained by combining these four estimates to give a weighted average for the next step.

Alternative approaches to solve the system of ODEs include implicit methods like the *Backward Euler* method [24] and adaptive step-size solvers. These methods will not be considered further in this report since the RK4 method provides a sufficient accuracy and computational efficiency.

To conclude, we summarize the physical parameters used to describe the system and the hyperparameters required for discretization, simulation and control below:

- α: Thermal diffusivity of the rod material
- L: Total length of the rod
- $T_0(x)$: Initial temperature distribution of the rod
- N: Number of spatial discretization steps along the length of the rod
- Δx: Spatial step size, determined based on L and N

- Δt: Temporal step size for simulation and control

The implementation of the simulation in MATLAB using the model developed in this chapter is discussed in Chapter 5.

4 Control Algorithm Development

This chapter focuses on designing and implementing an LQR (Linear Quadratic Regulator) controller to optimize the rod heating system. LQR controllers are known for their efficacy in stabilizing linear systems by minimizing a carefully crafted quadratic cost function through a linear full-state feedback control law. The cost function quantifies system performance, considering both, the state variables and control inputs. LQR control strikes an optimal balance between stability and control effort, ensuring efficient system operation. This trade-off can be adjusted by modifying the tuning parameters of the cost function.

4.1 Controllability Analysis

The controllability of a system indicates the extent to which a control algorithm can influence the system. It is therefore an important prerequisite for designing a controller. The controllability depends on the way the inputs influence the states which is described by $\mathbf{B}$. To check whether a system is controllable, the controllability matrix is calculated as shown in equation 4.1.

$$\mathcal{C} = \begin{bmatrix} \mathbf{B} & \mathbf{AB} & \mathbf{A}^2\mathbf{B} & \dots & \mathbf{A}^{N-1}\mathbf{B} \end{bmatrix}. \tag{4.1}$$

According to [25], a system is fully controllable if the controllability matrix has full rank

$$\operatorname{rank}(\mathcal{C}) \stackrel{!}{=} N. \tag{4.2}$$

As a result, the state can be driven to any desired value by providing suitable inputs. In MATLAB, the controllability matrix can be checked with the command `rank(ctrb(A,B))`. For the input model $\mathbf{B}$ and parameters that are chosen in this report, the controllability condition is fulfilled. The system is therefore fully controllable.

4.2 Optimization Problem

Without loss of generality, the goal of the controller is to quickly drive the state to zero with minimal control effort. The cost function of an LQR controller evaluates the performance of the control loop under these considerations. It is based on the states

and control variables of the system. Specifically, the quadratic cost function for a linear time-invariant system in state space that defines an LQR controller is [22, p. 113]

$$J = \int_0^\infty \left(\vec{T}^\top(t)\mathbf{Q}\vec{T}(t) + \vec{u}^\top(t)\mathbf{R}\vec{u}(t) \right) dt. \tag{4.3}$$

This cost function must now be minimized subject to the constraints imposing the dynamics and initial condition of the system

$$\dot{\vec{T}}(t) = \mathbf{A}\vec{T}(t) + \mathbf{B}\vec{u}(t) \tag{4.4a}$$

$$\vec{T}(0) = \vec{T}_0. \tag{4.4b}$$

The matrices $\mathbf{Q}$ and $\mathbf{R}$ determine how the LQR controller balances the individual costs on the state and control input variables. Appropriate selection of these factors is crucial to achieve the desired performance of the control loop. It often requires experimental adjustments and testing to find optimal values that meet the specific requirements of the system.

The $\mathbf{Q}$ matrix must be positive semidefinite ($\vec{T}^\top\mathbf{Q}\vec{T} \geq 0 \ \forall\vec{T}$) and influences the performance by penalizing deviations of the state variables from the goal state. By choosing high weights in $\mathbf{Q}$, the state will rapidly converge but this happens at the expense of a large control effort. An additional requirement is that the pair $(\mathbf{A}, \mathbf{Q})$ is observable. This can be achieved by making $\mathbf{Q}$ a positive definite matrix.

The matrix $\mathbf{R}$ has to be positive definite ($\vec{u}^\top\mathbf{R}\vec{u} > 0 \ \forall\vec{u} \neq 0$) such that the integrant is always positive. $\mathbf{R}$ is the weighting matrix for the control effort. If large values for the elements in $\mathbf{R}$ are chosen, the state only decays slowly to its final value but the control effort is minimized.

The choice of weighting matrices depends on the specific requirements of the system. An attempt is often made to find a compromise between tracking the reference signals and minimizing the control effort.

The weight matrices $\mathbf{Q}$ and $\mathbf{R}$ are chosen as diagonal matrices in the following form

$$\mathbf{Q} = q \cdot \mathbf{I}_{NxN} \tag{4.5a}$$

$$\mathbf{R} = \mathbf{I}_{mxm}. \tag{4.5b}$$

Since all state variables are equally important, the state weights are all set to q. Similarly, the control effort is penalized equally for all inputs. The weights of $\mathbf{R}$ are set to 1 since only the relative weights between $\mathbf{Q}$ and $\mathbf{R}$ are relevant. This reduces the manual tuning effort to a single parameter.

With the state and input vectors as defined previously in equations 3.10 and 3.13, the cost function can be rewritten explicitly as

$$J = \int_0^\infty \begin{bmatrix} T_1(t) & T_2(t) & \dots & T_N(t) \end{bmatrix} \begin{bmatrix} q & 0 & \dots & 0 \\ 0 & q & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & q \end{bmatrix} \begin{bmatrix} T_1(t) \\ T_2(t) \\ \vdots \\ T_N(t) \end{bmatrix} + \begin{bmatrix} u_1(t) & u_2(t) & u_3(t) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1(t) \\ u_2(t) \\ u_3(t) \end{bmatrix} dt. \tag{4.6}$$

4.3 Solution to the Optimization Problem

The optimal feedback control law of the LQR controller has the following form

$$\vec{u}(t) = -\mathbf{K}\vec{T}(t). \tag{4.7}$$

where the feedback gain matrix $\mathbf{K}$ is given by

$$\mathbf{K} = \mathbf{R}^{-1}\mathbf{B}^\top\mathbf{P}. \tag{4.8}$$

As described in [22, p. 113], $\mathbf{P}$ is the solution to the symmetric, positive definite algebraic Matrix Riccati equation given by

$$\mathbf{A}^\top\mathbf{P} + \mathbf{P}\mathbf{A} - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^\top\mathbf{P} + \mathbf{Q} = 0. \tag{4.9}$$

Solving the Riccati equation for the matrix $\mathbf{P}$ allows to derive the gain matrix $\mathbf{K}$ that minimizes the associated cost function. Due to its complexity, the algebraic Riccati equation can generally no longer be solved analytically. The calculation is done by a specialized solver that is called by the `lqr` function in MATLAB.

Once the gain matrix $\mathbf{K}$ is determined, the closed-loop equation can be defined as

$$\dot{\vec{T}}(t) = \mathbf{A}\vec{T} + \mathbf{B}\vec{u} = \mathbf{A}\vec{T} + \mathbf{B}\left(-\mathbf{K}\vec{T}\right) = (\mathbf{A} - \mathbf{BK})\,\vec{T}. \tag{4.10}$$

This equation describes the closed-loop dynamics of the control loop in which the feedback control is applied to the state vector $\vec{T}$ using the gain matrix $\mathbf{K}$. The closed-loop dynamics matrix $\mathbf{A}_{cl} = \mathbf{A} - \mathbf{BK}$ fully describes the behavior and stability of the system.

4.4 Feedforward Control

To enable tracking of reference signals, a feedforward matrix $\mathbf{W}$ is added to the input of the system as shown in Figure 4.1. The feedforward filter matrix scales the reference input to achieve the desired input-output behavior. It is determined empirically by applying a constant step input to the system and observing the steady-state output value. The ratio between these two values is used to calculate the elements of the (one-dimensional/vector) feedforward matrix $\mathbf{W}$ such that the steady-state gain is 1. For the system parameters assumed in this report, the following feedforward matrix values have been determined to enable accurate tracking of the average temperature

$$\mathbf{W} = \begin{bmatrix} 0.258 & 0.258 & 0.258 \end{bmatrix}. \tag{4.11}$$

As can be seen in Figure 4.1, the final control law therefore consists of the feedback and feedforward terms as follows

$$\vec{u}(t) = -\mathbf{K}\vec{T}(t) + \mathbf{W}\vec{r}(t). \tag{4.12}$$

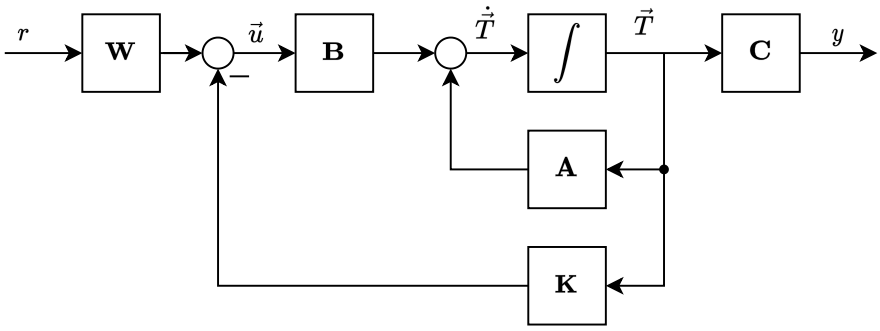


Figure 4.1 – Block Diagram of the LQR control method with linear feedback gain matrix $\mathbf{K}$ and feedforward matrix $\mathbf{W}$.

Simulations and results obtained from this controller architecture with various tuning parameters for q are presented in Chapter 6.

4.5 Stability Analysis

To show that the system is stable, we need to calculate the eigenvalues of the closed-loop matrix $\mathbf{A}_{cl}$ which is defined as

$$\mathbf{A}_{cl} = \mathbf{A} - \mathbf{BK}. \tag{4.13}$$

This matrix describes the dynamics of the controlled system. For higher order systems, the calculation of the eigenvalues becomes complex, so the calculation is done numerically in MATLAB. The diagrams show that all eigenvalues are on the left-hand side of the imaginary axis in the complex plane, which means that all eigenvalues have a negative real part. This indicates that the system is stable [26]. The position of the eigenvalues provides information about the reaction speed of the system. Eigenvalues that are further away from the origin make the system react faster. Figure 4.2 shows that the pole furthest from the origin is at $\lambda = -0.16$, which indicates that it reacts the fastest.

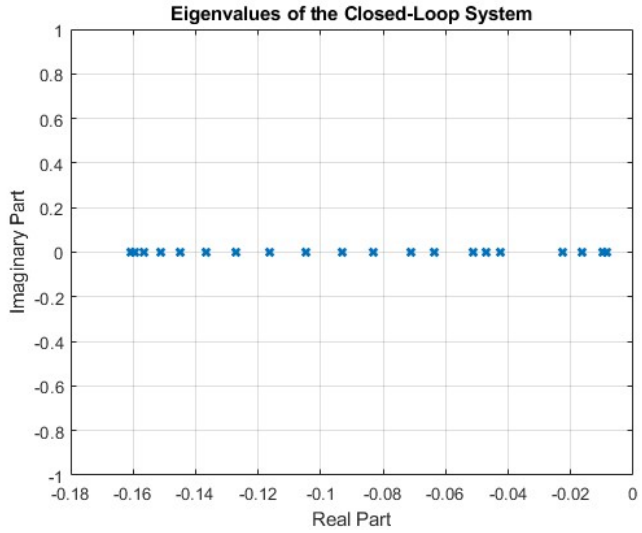


Figure 4.2 – Eigenvalues of the closed-loop matrix $\mathbf{A}_{cl}$

The slowest pole is at $\lambda = -0.01$. Eigenvalues with an imaginary part indicate oscillatory behavior. In this specific case, no oscillation can be observed as all eigenvalues have a purely real part. Therefore, the closed-loop system behavior is well-damped.

5 Implementation

5.1 Simulation of the Control Algorithm

The simulation of heat evolution in the rod is implemented in MATLAB, allowing for intricate data handling, complex calculations, and detailed visualization.

To summarize the theoretical considerations from Chapter 3, the different parts of the simulation are outlined in the following sections.

State-Space Representation

The spatially discretized dynamics of heat transfer are described using the state-space representation first introduced in equation 3.12

$$\dot{\vec{T}} = \mathbf{A}\vec{T} + \mathbf{B}\vec{u}. \tag{5.1}$$

The state equations relate the rate of change of temperatures (state variables) to the current temperatures and external heat source inputs.

Here, $\vec{T}$ is the temperature vector, $\mathbf{A}$ is a matrix capturing the thermal interactions between different segments, $\mathbf{B}$ is a matrix mapping the heat sources' effects into temperature changes, and $\vec{u}$ is the control input vector representing the heat sources.

Program code 5.1 shows the calculation of the the right-hand side of the differential equation in MATLAB. The function takes in the current state, control inputs, and the system matrices $\mathbf{A}$ and $\mathbf{B}$ to return the rate of change of temperature. This function is called multiple times per timestep by the RK4 algorithm to calculate the various slopes as previously explained in section 3.3.

```
1 %% State Space Equation for the Dynamics
2 function dT = heat_equation(T, u, A, B)
3     dT = A * T + B * u;
4 end
```

Program Code 5.1 – Implementation of the **B** Matrix

Matrix A - System Matrix

The Matrix **A** is central to the state-space representation, encapsulating the spatial aspect of the heat equation. It is formed considering the second spatial derivative requirement of the heat equation, accounting for the conduction between neighboring segments, and influenced by the thermal diffusivity α. Program Code 5.2 shows the implementation of the matrix **A** in MATLAB. The function requires the user to specify the number of states and the size of the spatial discretization step Δx.

```
function A = compute_A_matrix(N, alpha, dx)
    A = full(gallery('tridiag',N,1,-2,1));

    A(1,2) = 2;
    A(N,N-1) = 2;

    A = A .* alpha/dx^2;
end
```

Program Code 5.2 – Implementation of the **A** Matrix

Matrix B - Input Matrix

The Matrix **B** specifies how the control inputs (heat sources) affect the temperature in the system. It defines how the individual spatially discretized segments of the rod are influenced by each heat source. As an example, program code 5.3 shows the calculation of the input matrix for the case with uniform heat inputs of Figure 3.2.

Simulation Loop

The simulation loop is outlined in Algorithm 1. The simulation is initialized by specifying the initial temperature distribution and the reference trajectory that is used as the input to the system. In each loop iteration, the next controller value is calculated based on the current state and reference input. With this information, the temperature distribution at the next timestep is calculated using the RK4 algorithm. With the new state, the output y is calculated which is then used to evaluate the tracking error e. The tracking error e can then be used to evaluate the performance of the controller.

```
function [B] = compute_B_matrix(N)
    N5 = N/5;
    N10 = N/10;

    n_raw = zeros(6,1);
    n_raw(1) = N10+1;
    n_raw(2) = n_raw(1)+N5-1;
    n_raw(3) = n_raw(2)+N10+1;
    n_raw(4) = n_raw(3)+N5-1;
    n_raw(5) = n_raw(4)+N10+1;
    n_raw(6) = n_raw(5)+N5-1;

    n = zeros(6,1);
    n(1:2:5) = ceil(n_raw(1:2:5));
    n(2:2:6) = floor(n_raw(2:2:6));

    B = zeros(N,3);
    B(n(1):n(2),1) = 1;
    B(n(3):n(4),2) = 1;
    B(n(5):n(6),3) = 1;
end
```

Program Code 5.3 – Implementation of the **B** Matrix

Results and Visualizations

The study employs several visual aids to elucidate the temperature dynamics within the rod. Examples of these are shown when discussing the results of the simulation in Chapter 6.

5.2 Low-Level Implementation of the Control Algorithm

One of the main considerations in developing a control algorihm for embedded systems is the computational cost of the algorithm. High-level programming languages like MATLAB and Python are often not suitable in these systems. This is explained in part by the dynamically typed nature of these languages which leads to a lower computational efficiency with regard to the speed and memory usage.

In real-time systems, runtime guarantees and a safe memory usage are among the main priorities. Reliability is also a key factor in these systems because typically there is no way to update the software once the system is deployed. For these reasons, low-level programming languages like C or C++ are used for the implementation of control

Algorithm 1 Simulation Loop

1: Initialize buffers for the state vector $\vec{T}$, control inputs $\vec{u}$, output y and tracking error e
2: Initialize the state vector $\vec{T}$ with the initial conditions
3: Calculate the initial output y
4: **for** each timestep k **do**
5: Update the reference input r
6: Calculate the control input $\vec{u}$
7: Calculate the state vector $\vec{T}$ at time $k+1$ using RK4 integration
8: Calculate the new output y
9: Evaluate the tracking error e
10: **end for**

algorithms running in embedded systems. These languages are statically typed. This allows for stricter control of the memory usage and a higher computational efficiency.

Nevertheless, low-level programming languages are often not suitable for prototyping of control algorithms because of a longer development time. Development of low-level code comes with a higher complexity caused by the need to consider implementation details that are of secondary importance in the functional design of the algorithm itself.

One typical approach used in the development of control algorithms is to write high-level code (e.g. MATLAB/Simulink) for prototyping and once the algorithm is validated, to rewrite the algorithm in a low-level language.

An alternative approach that is gaining increased importance is the use of automatic code generation that automatically compiles the high-level code to a low-level language. Due to the simple computational nature of the control algorithm developed in the previous section, automatic code generation will not be further considered here.

Instead, the control algorithm is manually rewritten as C-Code, using the MATLAB C-Code API [27]. The C-Code is then compiled as a *MATLAB Executable (MEX)* [28]. This allows the C-Code to be called from within the MATLAB environment. Therefore, the previous simulation code can be reused with the same C-Code that could then theoretically be deployed to actual hardware.

This constitutes a simplified form of a *Software-in-the-Loop (SIL)* simulation. SIL refers to methods that test software in a simulated environment (strictly speaking, the software-under-test should be run in a separate process) [29]. In this report, the actual physical environment i.e. the rod is simulated to test the performance of the control software in the simulation. This approach of combining low-level MEX files with high-level code is also

useful to accelerate simulations by replacing parts of the code that are computationally expensive with a MEX file.

While the design of the linear feedback controller described in Chapter 4 requires careful consideration of the control task at hand, the actual computation of the control law is straightforward. The control law is restated in equation 5.2

$$\mathbf{u} = -\mathbf{K} \cdot \mathbf{x} + \mathbf{W} \cdot r. \tag{5.2}$$

In the next part, the dimensions are defined as follows: $\mathbf{u} \in \mathbb{R}^m$ is the control input, $\mathbf{x} \in \mathbb{R}^n$ is the state vector, $\mathbf{K} \in \mathbb{R}^{m \times n}$ is the feedback gain matrix, $\mathbf{W} \in \mathbb{R}^{m \times 1}$ is the feedforward gain or static filter matrix and $r \in \mathbb{R}$ is the reference input. Therefore, the task of the control algorithm is to compute two matrix multiplications and one vector addition. The code that implements this control algorithm is shown in program code 5.4.

```
1  //Functionality - Linear Regulator to compute u = W * r - K * x
2  void linear_regulator_filter(double *x, double *K, double *r, double *
       W, double *u, mwSize N, mwSize P, mwSize M) {
3      //Initialize u to zero
4      for(mwSize i = 0; i < M; i++) {
5          u[i] = 0;
6      }
7
8      //Matrix-Vector multiplication u = -K*x
9      //K is stored sequentially with the columns concatenated
10     //Row index i, Column index j
11     for(mwSize i = 0; i < M; i++) {
12         for(mwSize j = 0; j < N; j++) {
13             u[i] -= K[i+j*M] * x[j];
14         }
15     }
16
17     //Matrix-Vector multiplication u += W*r
18     //Column index i, row index j
19     for(mwSize i = 0; i < M; i++) {
20         for(mwSize j = 0; j < P; j++) {
21             u[i] += W[i+j*M] * r[j];
22         }
23     }
24 }
```

Program Code 5.4 – C-Code implementation of the control algorithm for the linear regulator.

First, the control input vector $\mathbf{u}$ is initialized to zero. Then, the two matrix multiplications

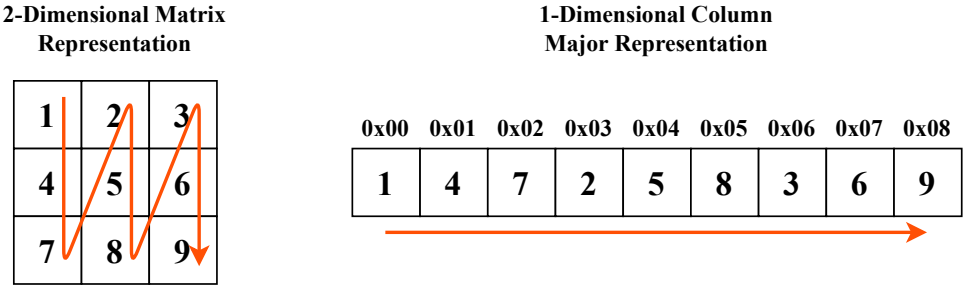


Figure 5.1 – Illustration of the column-major array layout.

are computed. The matrix multiplications are implemented as nested *for-loops*. The two dimensional matrices **K** and **W** are stored as one dimensional arrays in *Column-Major Array Layout* [30]. This means that all elements of the matrix are stored sequentially in memory/in an array with the columns concatenated, i.e. starting with the elements of the first column, then the second column and so on. Using i as the row and j as the column index for the matrix elements, the index of the element (i, j) in a matrix is computed as $i + j \cdot M$ with M as the number of rows in the matrix. This is illustrated in Figure 5.1. On the left, the elements of a matrix are shown in their two-dimensional layout. On the right, the corresponding representation in memory with the elements stored sequentially is shown.

In addition to the function of code 5.4, an interface function to map between the MATLAB and C-Code is required. A snippet of this function is presented in code 5.5.

The function input and output arguments are passed to the C-Code via pointers to arrays. Error checks are performed to ensure the correct usage of the function. New pointers are declared and bound to the input and output arguments before being passed to the control algorithm in program code 5.4. Considering that the direct implementation of the linear regulator code in MATLAB is a single line of code, the overhead is clearly evident. Hence, a careful consideration between the need for computational efficiency and the effort required to implement functionality in low-level code is required. Once the C-Code is fully written, it is compiled to a MATLAB executable file which allows the function to be called as if it was a native MATLAB function. Unit tests are written in MATLAB to ensure the correct functionality of the C-Code before integrating it with the simulation code.

To confirm the runtime improvement of the C-Code, profiling of the code is conducted for a typical simulation with a timestep of 50 ms for a total time of 300 s. With these parameters, there are a total of 6000 calls to the control algorithm.

```c
//Gateway to MATLAB
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *
    prhs[]) {
    //nlhs: Number of output arguments --> size of plhs array
    //nrhs: Number of input arguments --> size of prhs array
    //plhs, prhs: Arrays containing output and input arguments

    // Checking of the input arguments omitted for brevity
    ...

    //Declaration of variables for computational routine with
    functionality
    mwSize N;           //size of state vector x
    mwSize P;           //size of reference vector P
    mwSize M;           //size of control vector u
    double *x;          //Nx1 state vector x
    double *K;          //MxN gain matrix K
    double *r;          //Px1 state vector x
    double *W;          //MxP filter matrix W
    double *u;          //Mx1 control vector u

    u = -K*x + W*r;     //Compute control input u

    //Read input data
    x = mxGetPr(prhs[0]);   //Get state vector x input argument
    K = mxGetPr(prhs[1]);   //Get gain matrix K input argument
    r = mxGetPr(prhs[2]);   //Get reference vecotr r input argument
    W = mxGetPr(prhs[3]);   //Get filter matrix W input argument

    N = mxGetM(prhs[0]);   //Get number of rows of state vector
    P = mxGetM(prhs[2]);   //Get number of rows of reference vector
    M = mxGetM(prhs[1]);   //Get number of rows of gain matrix

    //Prepare a real-valued Mx1 matrix as the output
    plhs[0] = mxCreateDoubleMatrix(M,1,mxREAL);
    u = mxGetPr(plhs[0]);  //Get a pointer to the output matrix

    //Call the actual function
    linear_regulator_filter(x, K, r, W, u, N, P, M);
}
```

Program Code 5.5 – Implementation of the interface code between MATLAB and C.

The following time measurements are obtained:

- MATLAB: 0.024 s
- C-Code: 0.017 s

While the absolute time difference is small, the relative difference is clearly evident. The difference in runtime corresponds to an improvement of 29%. For the simple control algorithm considered here, this is insignificant but for more complex methods and simulations with a longer time horizon, the C-Code implementation leads to significant runtime improvements.

Simulation results with the C-Code implementation instead of the MATLAB prototyping code are omitted since both implementations are mathematically identical.

To summarize, the use of low-level C-Code compiled as a MATLAB exectuable enables testing of hardware-ready code in a simulation environment. This allows for early performance analysis and testing of the control algorithm. Additionally, complex simulations can be accelerated by integrating computationally expensive parts of the code as MEX files.

6 Results

6.1 Open-Loop Behavior of the Heat in the Rod

To validate the model, open loop simulations without a control input are performed.

The following simulations consider a rod of length 100 cm discretized into 20 sections of 5 cm each resulting in a 21-dimensional state vector.

Without an external input $\vec{u}$, the initial heat along the rod is expected to diffuse until an equilibrium is reached and the temperature at each position is the same.

To show this, the temperature is initialized such that it linearly increases from 0 at $x = 0$ cm to 100°C at $x = 100$ cm.

The temperature distribution is simulated for 3000 s (50 minutes) with a time step of 50 ms for a rod with thermal diffusivity of $\alpha = 5\frac{\text{cm}^2}{\text{s}}$. Figures 6.1 and 6.2 shows that the temperature profile matches the expected behavior and the diffusion can be observed.

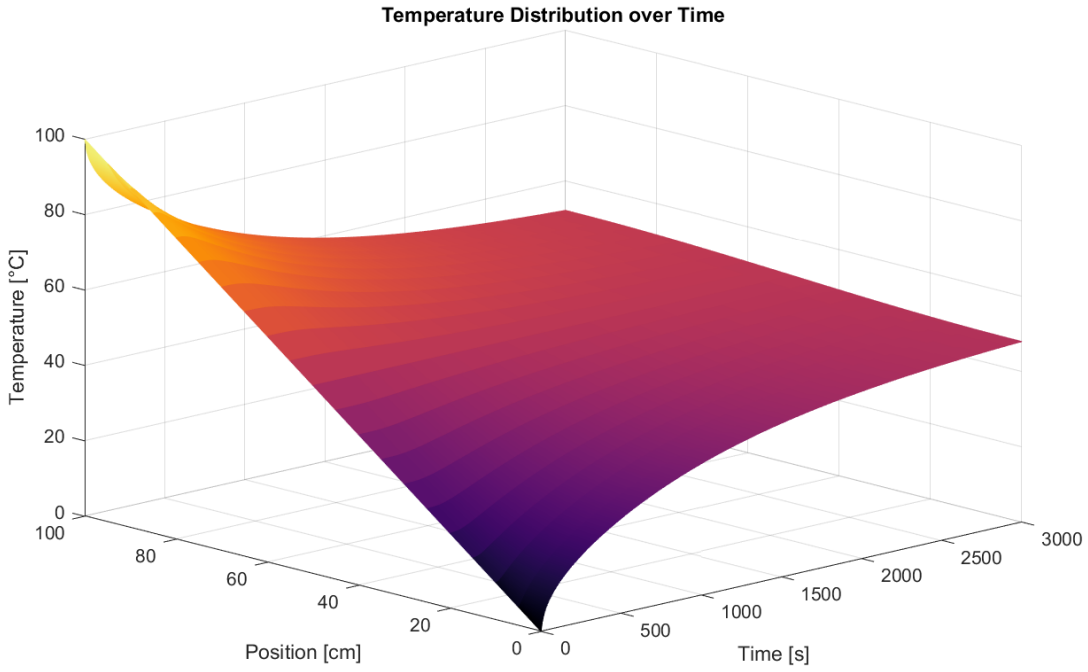


Figure 6.1 – Simulation of the heat diffusion along the rod until an equilibrium is reached.

This shows that the behavior of the rod is inherently stable as proven in Chapter 4.5.

To show the effect of the thermal diffusivity α, the simulation is repeated with $\alpha = 1\frac{\text{cm}^2}{\text{s}}$

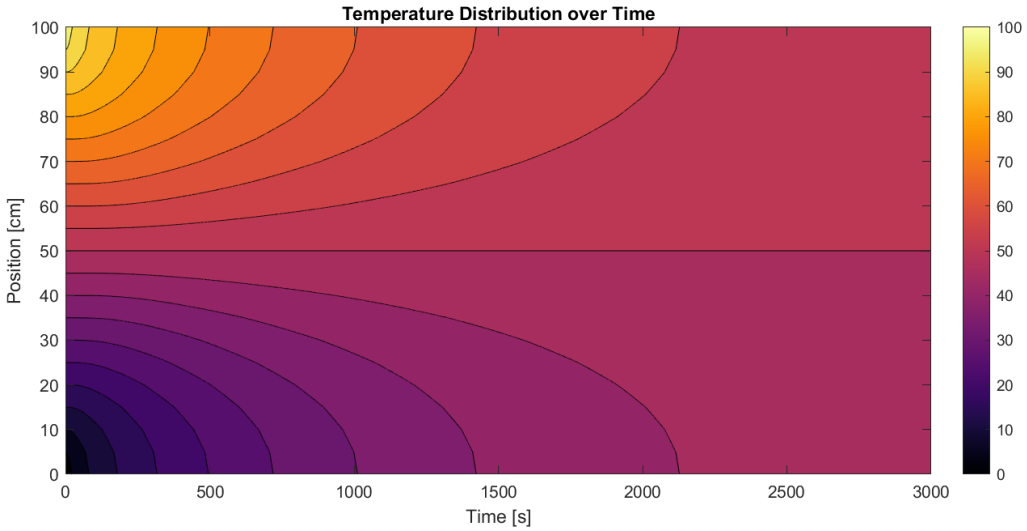


Figure 6.2 – Contour plot of the heat diffusion along the rod.

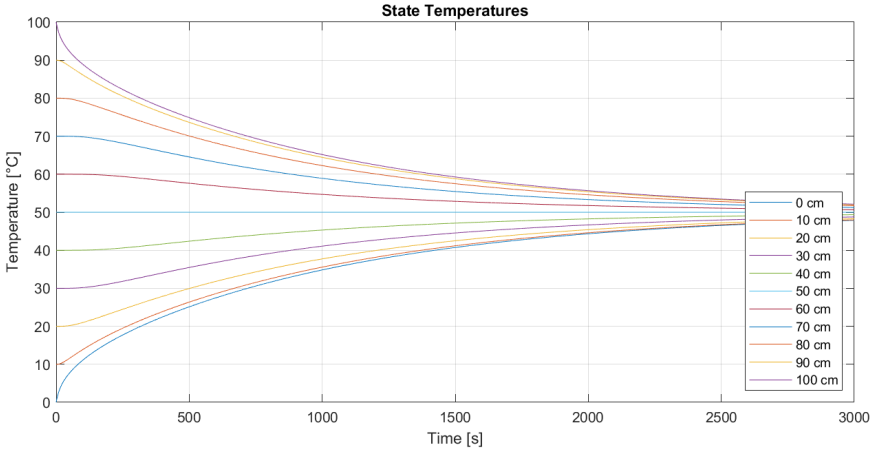
and $\alpha = 10\frac{cm^2}{s}$.

The evolution of the temperatures for the different diffusivities is shown in Figure 6.3. As expected, the diffusion is faster when α is higher. This is also evident in the eigenvalues of the spatially discretized system matrix $\mathbf{A}$. Table shows the maximum eigenvalue for the three values of α used in Figure 6.3. With increasing α, the maximum eigenvalue becomes more negative and the system dynamics become faster.

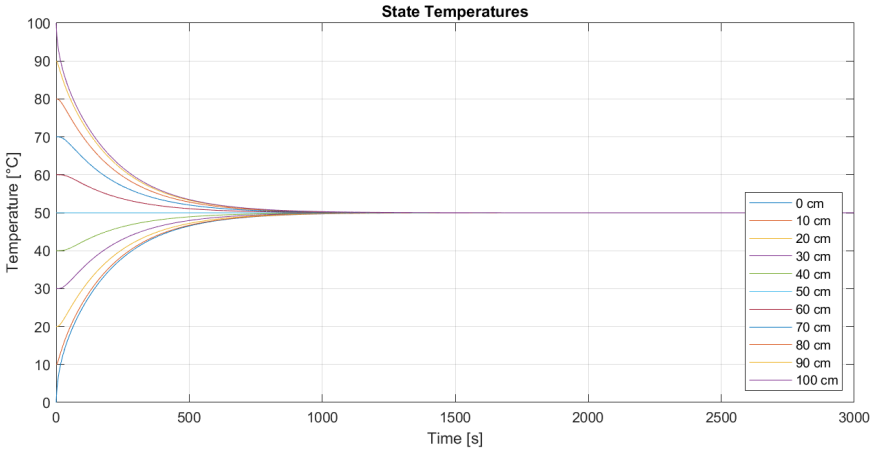
α	$\max(\lambda(\mathbf{A}))$
$1\frac{cm^2}{s}$	-0.136
$5\frac{cm^2}{s}$	-0.683
$10\frac{cm^2}{s}$	-1.366

Table 6.1 – Maximum eigenvalue of the spatially discretized system matrix $\mathbf{A}$ for different thermal diffusivities α.

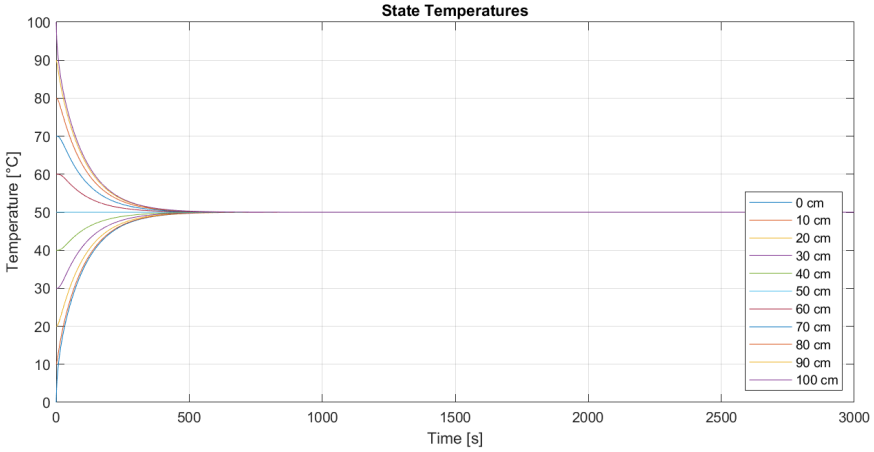
Depending on the material properties, the heat diffusion might be too fast or too slow for a given application. Since changing the material properties or the material itself is often not an option, active control methods are required to achieve the desired dynamic behavior.



(a) Heat along the rod with a thermal diffusivity of $\alpha = 1 \frac{cm^2}{s}$.



(b) Heat along the rod with a thermal diffusivity of $\alpha = 5 \frac{cm^2}{s}$.



(c) Heat along the rod with a thermal diffusivity of $\alpha = 10 \frac{cm^2}{s}$.

Figure 6.3 – Heat diffusion along the rod for different thermal diffusivities.

6.2 Closed-Loop Behavior of the Heat in the Rod

LQR control utilizes the control inputs to achieve the dynamic behavior that optimizes the quadratic cost introduced in Chapter 4.2. Under the (unrealistic) assumption that arbitrarily strong control inputs are permissible (no actuator constraints in the optimization problem), there are no constraints and the system dynamics can be sped up to any desired specification.

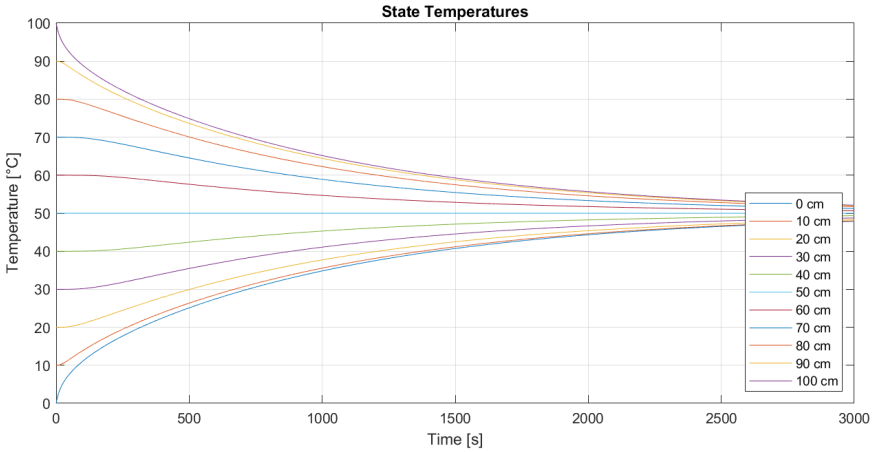
When applying the linear feedback gain matrix obtained from the LQR controller by applying full-state feedback, the eigenvalues of the system are shifted to new locations. For the following investigation, a thermal diffusivity of $\alpha = 1\frac{cm^2}{s}$ is used.

The cost matrices $\mathbf{Q}$ and $\mathbf{R}$ are chosen to be a diagonal matrices. More precisely, all states are weighted the same and all inputs are weighted the same to simplify the analysis.

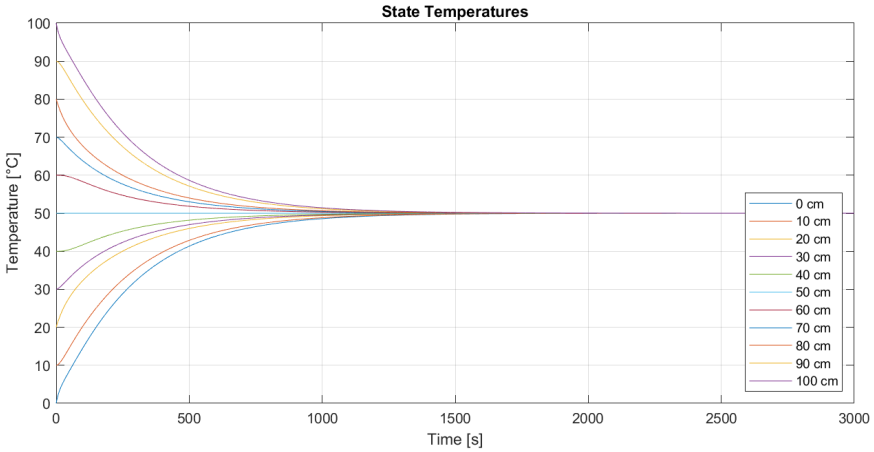
Figure 6.4 shows the effect of the linear regulator in driving the temperature across the rod to $50°C$ which is equivalent to the natural equilibrium. The matrix $\mathbf{R}$ is an identity matrix in all cases. The first subplot shows the open-loop behavior as before. $\mathbf{Q}$ is chosen to be $10^{-5}\mathbf{I}$ in Figure 6.4b and $10^{-3}\mathbf{I}$ in Figure 6.4c.

The LQR controller speeds up the dynamics according to the relative cost on the states specified by $\mathbf{Q}$ and on the inputs specified by $\mathbf{R}$. If the values in $\mathbf{Q}$ are higher, the cost on the state error is higher and therefore larger control inputs are used to drive the state faster to the goal (in this case $50°C$). This can be seen in the faster convergence of Figure 6.4c compared to Figure 6.4b where the cost on the state error is lower by a factor of 100. The resulting trajectories with an LQR controller are similar to the trajectories on a rod with a higher diffusivity. Additionally, the LQR controller introduces feedback and is able to correct for disturbances. Figure 6.5 illustrates the downside of increasing the cost on the state error. The faster dynamics are achieved by larger input signals and therefore require stronger heat sources. In this report we assume that the heat sources are also able to cool the rod which explains the negative values of one of heat input u_3.

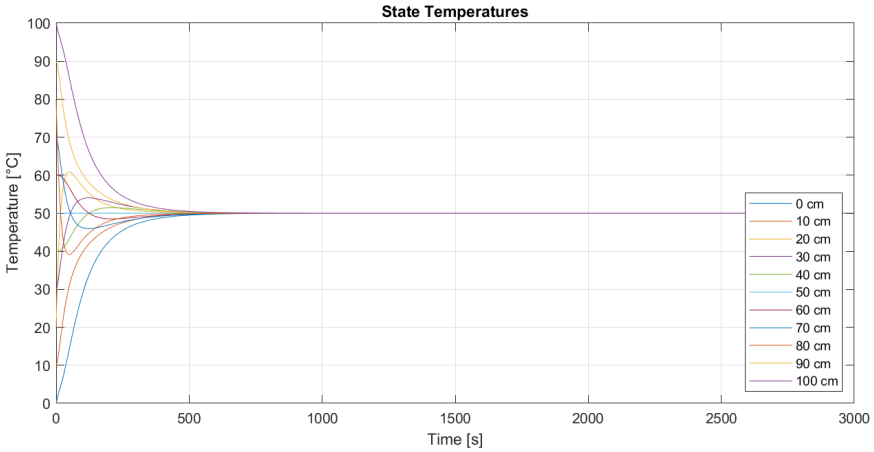
If actuator constraints have to be considered in the LQR optimization problems, different techniques that can directly reason about complex constraints like *Model Predictive Control (MPC)* should be considered.



(a) Open-loop behavior without linear feedback.

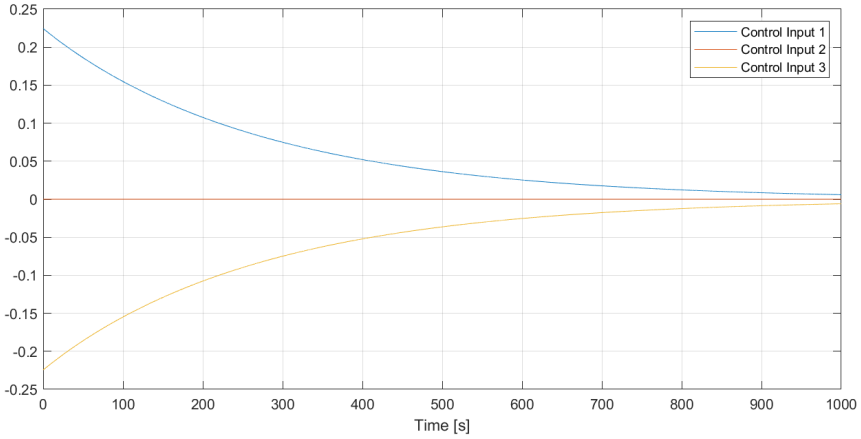


(b) Behavior with linear LQR feedback with $\mathbf{Q} = 10^{-5}\mathbf{I}$

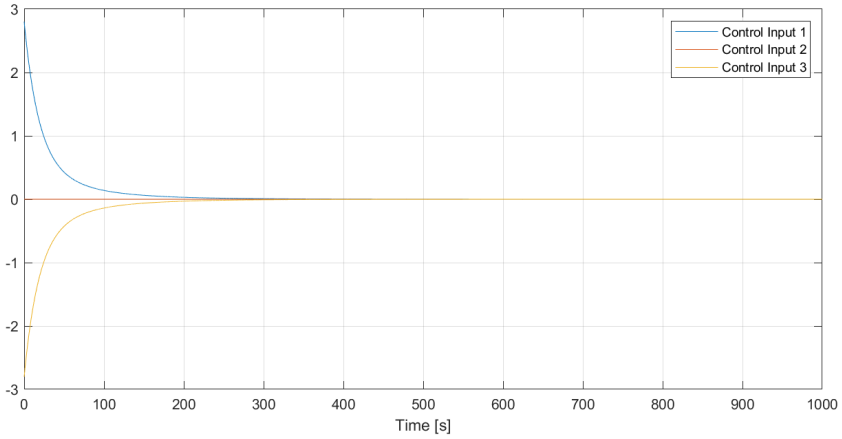


(c) Behavior with linear LQR feedback with $\mathbf{Q} = 10^{-3}\mathbf{I}$.

Figure 6.4 – Dynamics of the heat distribution with linear feedback and different LQR cost function parametrizations.



(a) Heat inputs for linear LQR feedback with $\mathbf{Q} = 10^{-5}\mathbf{I}$.



(b) Heat inputs for linear LQR feedback with $\mathbf{Q} = 10^{-3}\mathbf{I}$

Figure 6.5 – Heat source inputs with linear feedback and different LQR cost function parametrizations.

6.3 Feedforward Control of the Average Temperature

The stabilization, regulation and adjustment of the process dynamics is achieved through LQR control as shown in the last section. This allows direct output control through a simple feedforward structure as was illustrated in Figure 4.1.

Figure 6.6a shows the step response of the average temperature with the initial temperature at a constant $0°C$ along the rod. The step response has aperiodic behavior, it is strongly damped without any overshoot.

Figures 6.6b shows the temperature trajectories at different positions during the step. This illustrates an overshoot in temperature at the positions where the heat sources act. Once the average temperature converges, the individual temperatures also converge to constant values. This becomes easy to see when looking at the heat inputs in Figure 6.6c. During the initial rise, the control inputs are active and provide heat energy to the system. Once the correct amount of heat energy corresponding to the average temperature of $50°C$ is in the system, the heat sources converge to constant values, the average temperature is correct and the system reaches an equilibrium.

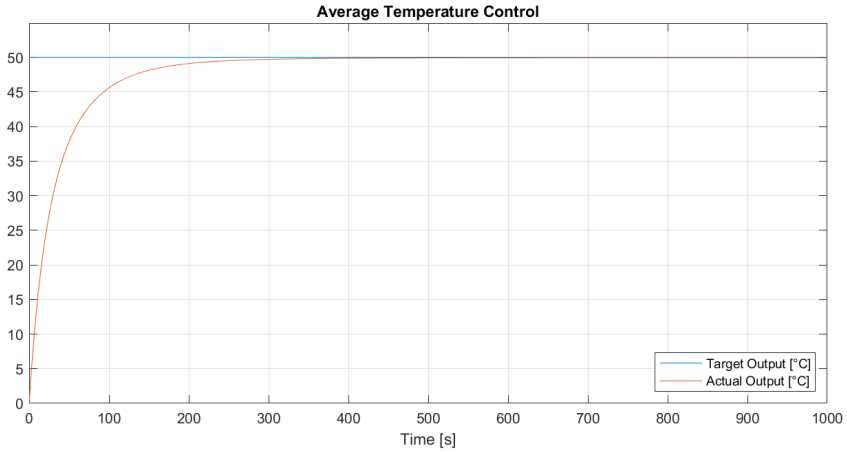
Step responses are important to characterize the system behavior. In practice however, it is clear that step changes in temperature are not achievable. Therefore, the behavior with more practical reference inputs is considered next. To show this behavior, a ramp change in average temperature to the same final value of $50°C$ is applied.

The average temperature tracks the ramp change closely but exhibits a mostly constant tracking error. The state trajectories again converge to constant temperatures but not to the average temperature.

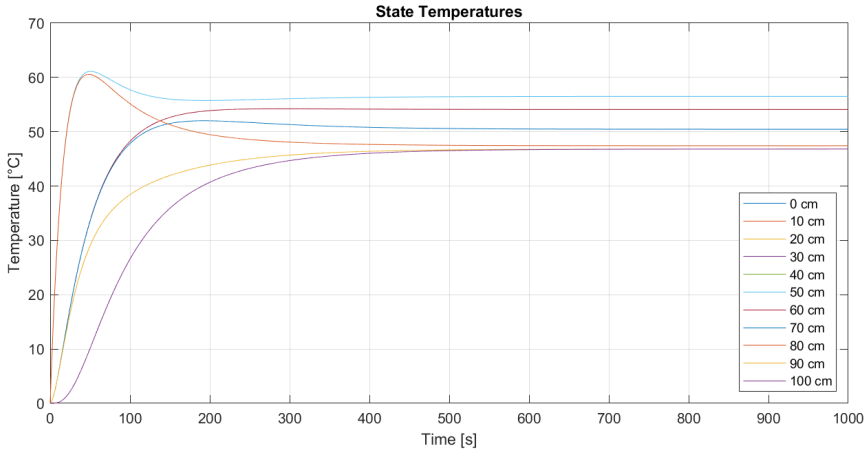
Finally, the frequency domain characteristics are investigated qualitatively by exciting the system with a sinusoidal input as illustrated in Figure 6.8. The temperature in the rod is initialized at a constant $50°C$ and the reference input varies sinusoidally by $20°C$ around this constant bias with a period of $500s$.

The closed-loop behavior exhibits low-pass behavior as is evident in the reduced magnitude in the output and the phase-lag between reference input and the actual average temperature in Figure 6.8a.

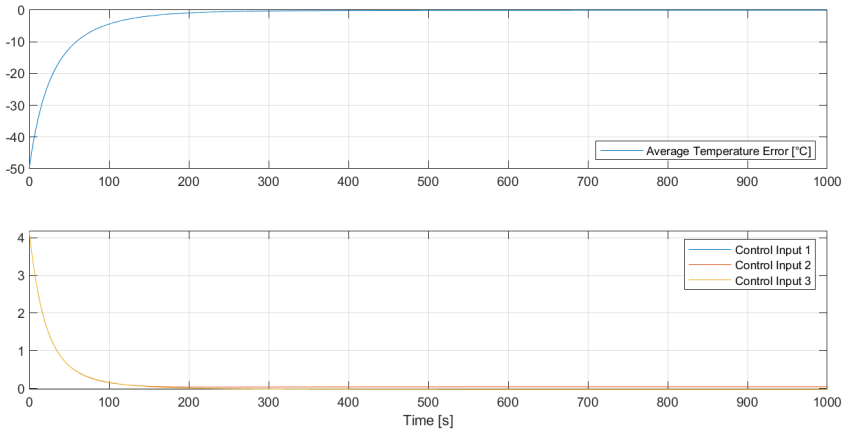
To quantify the bandwidth of the controller, simulations are performed with a sinusoidal input signal with varying frequencies. The magnitude and phase is calculated for each frequency and the result is depicted in the bode plot of Figure 6.9.



(a) Step response of the average temperature.

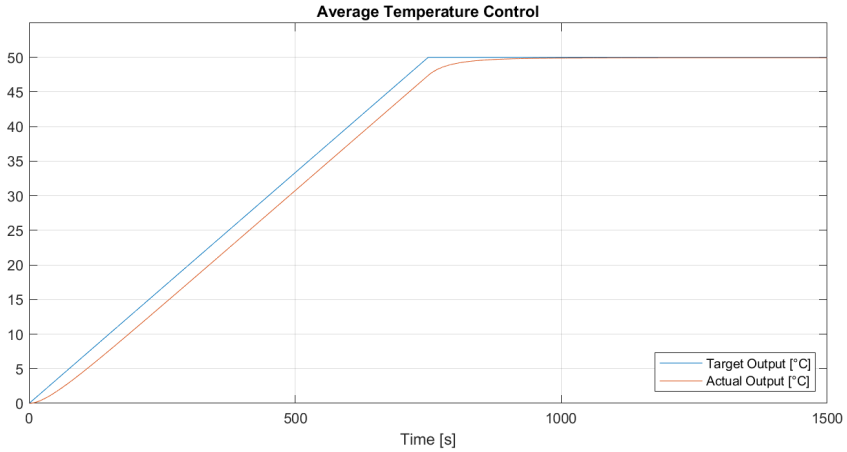


(b) State trajectories during the step change in average temperature.

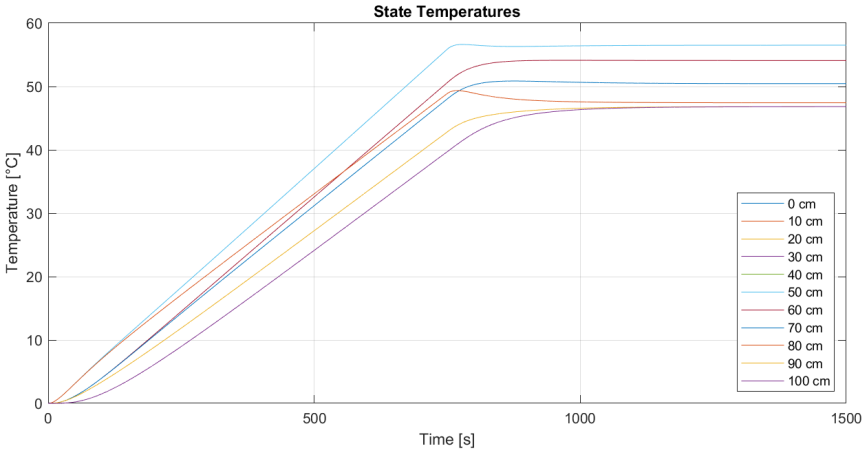


(c) Tracking error and heat inputs during the step change in average temperature.

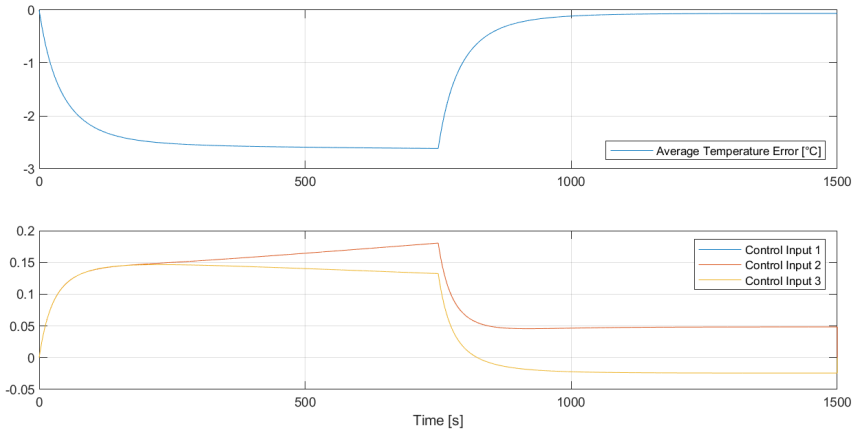
Figure 6.6 – Step response of the average temperature to $50°C$ starting at $0°C$.



(a) Tracking behavior of the average temperature with a ramp input.

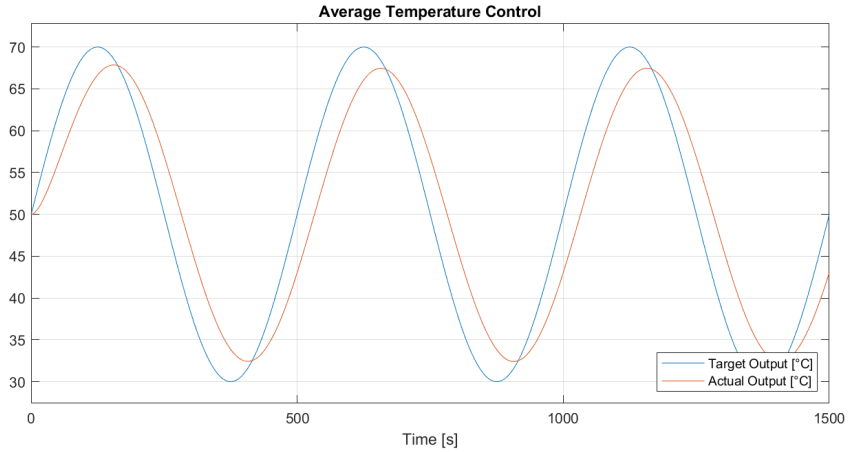


(b) State trajectories during the ramp change in average temperature.

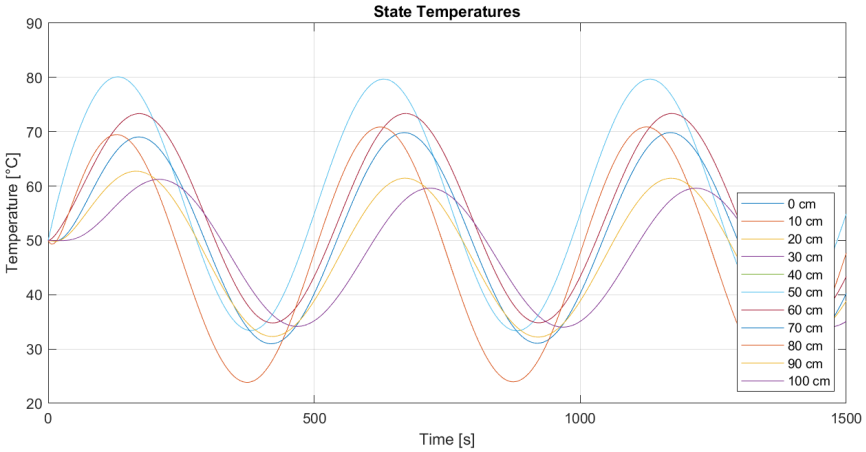


(c) Tracking error and heat inputs during the ramp change in average temperature.

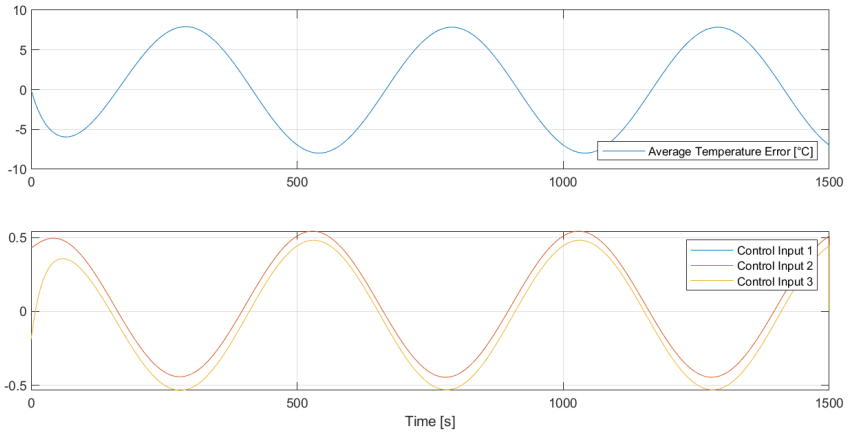
Figure 6.7 – Tracking behavior of the average temperature for a ramp change from $0°C$ to $50°C$.



(a) Tracking behavior of the average temperature with a sinusoidal reference input.



(b) State trajectories during the sinusoidal excitation in average temperature.



(c) Tracking error and heat inputs during the sinusoidal excitation in average temperature.

Figure 6.8 – Tracking behavior of the average temperature for a sinusoidal reference signal with $20°C$ amplitude and a frequency of 500 s.

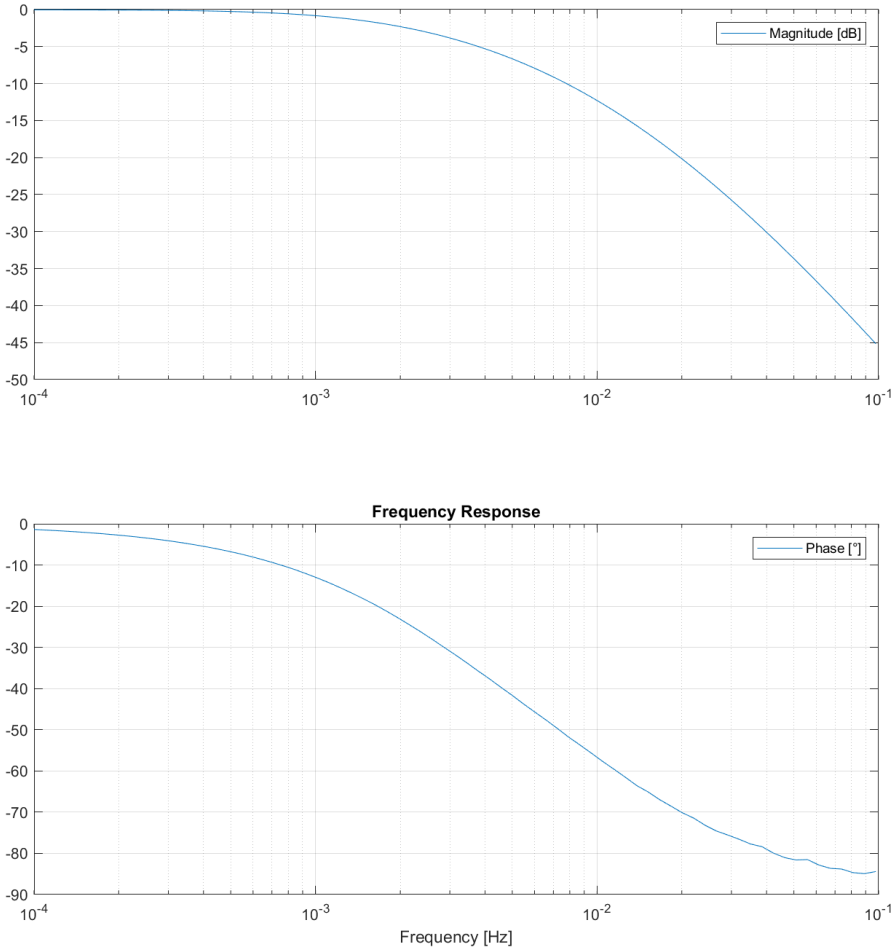


Figure 6.9 – Frequency characteristics of the closed-loop system.

This shows, that the controller bandwidth, defined by the 3 dB cut-off frequency of the closed-loop transfer function, is at 2.5 mHz which is equivalent to 400 s. The system has first order behavior as is evident in the 45° phase lag at the cut-off frequency and approaches 90° for large frequencies.

6.4 Discussion

The previously shown results confirm the fundamental assumptions about the system made in the development of the controller in Chapter 4. Nevertheless, there are multiple aspects of the control architecture that are suboptimal and can be improved in future work.

Tracking might be improved by using more advanced control structures like the state-space integral feedback controller discussed in [22, p. 152]. This is also a promising approach to reject disturbances which have not yet been considered. The main idea is to add output feedback instead of pure feedforward control for reference tracking.

An alternative approach is the use of a model-reference design like the servo-compensator introduced [22, p. 159]. This also works by including output feedback in the control structure. Additionally, a reference model that generates dynamically feasible trajectories for the system is designed.

In this report, we have assumed perfect knowledge of the system dynamics and noiseless measurements of the complete state. This is unrealistic in practice where sensors are expensive and most likely only available at a few positions along the rod. This motivates further investigations into the observability of the system and the possible use of an observer to obtain information about the full state.

We have assumed, that the heat inputs are able to perform heating and cooling. In many applications, this is not the case. The standard LQR control problem does not take these actuator constraints into account. A promising approach to explicitly model the constraints, is the use of MPC. In an MPC framework, a similar control objective to the LQR cost function can used. The formulation of the optimization problem can then be augmented with the constraints of the heat sources. Since the time-dynamics of the system considered here are slow compared to the time scales at which a modern microprocessor operates, complex optimization problems can likely be solved fast enough for real-time control.

7 Conclusion

Based on our study on in-domain control of average temperature in one-dimensional insulated rods, our results demonstrate a promising development in the field of control strategies. Using the heat equation and spatial discretization, a controller that combines feedback and feedforward control has been developed. The LQR feedback controller allows modification of the system dynamics while the feedforward controller leads to accurate tracking control, especially when handling constant inputs. During transient reference signal changes, a substantial tracking error demonstrates a potential area for further improvement.

Identification of first-order low-pass behavior of the closed-loop system provides crucial insight into the system's dynamics for controlling it more effectively. This might also serve as a starting point for the development of a system identification method to determine the parameters of a real-world implementation of the system. In the future, we should capitalize on the insights about the system dynamics obtained in this report while simultaneously addressing tracking errors. With this dual approach, the average temperature of one-dimensional insulated rods will be regulated more accurately and responsively, pushing the limits of accuracy and responsiveness.

By implementing the control algorithm in C-Code, we introduce a Software-in-the-Loop simulation method to accelerate the speed of development and verification. The C-Code implementation leads to an increase in computational efficiency. Compiling the C-Code as a MATLAB executable file allows easy integration of the embedded control software into the simulation environment. This approach will especially benefit the development of more complex models and control strategies for which this report serves as a starting point.

References

[1] G.P. Thiel and A.K. Stark. "To Decarbonize Industry, We Must Decarbonize Heat". In: *Joule* 5.3 (Mar. 2021), pp. 531–550. DOI: 10.1016/j.joule.2020.12.007.

[2] S. Kundu et al. "Modelling of Microstructure and Heat Transfer during Controlled Cooling of Low Carbon Wire Rod". In: *ISIJ International* 44.7 (2004), pp. 1217–1223. DOI: 10.2355/isijinternational.44.1217.

[3] M.S.A. Rahaman, A.F. Ismail, and A. Mustafa. "A Review of Heat Treatment on Polyacrylonitrile Fiber". In: *Polymer Degradation and Stability* 92.8 (Aug. 2007), pp. 1421–1432. DOI: 10.1016/j.polymdegradstab.2007.03.023.

[4] M. Domanski and J. Webb. "A Review of Heat Treatment Research". In: *Lithic Technology* 32.2 (Jan. 2007), pp. 153–194. DOI: 10.1080/01977261.2007.11721052.

[5] D. Miyagi et al. "Improvement of Zone Control Induction Heating Equipment for High-Speed Processing of Semiconductor Devices". In: *IEEE Transactions on Magnetics* 42.2 (Feb. 2006), pp. 292–294. DOI: 10.1109/TMAG.2005.860823.

[6] H. Xu et al. "Local Melting and Shape Controlling of Solder Joint via Induction Heating". In: *Journal of Materials Processing Technology* 209.6 (Mar. 2009), pp. 2781–2787. DOI: 10.1016/j.jmatprotec.2008.06.034.

[7] M.M.A. Rafique. "Modeling and Simulation of Heat Transfer Phenomena". In: *Heat Transfer Studies and Applications*. Ed. by M.S.N. Kazi. InTech, July 2015. DOI: 10.5772/61029. (Visited on 12/12/2023).

[8] Wikipedia. *Molecular Dynamics*. 2023. URL: https://en.wikipedia.org/wiki/Molecular_dynamics (visited on 12/12/2023).

[9] Wikipedia. *Monte Carlo Method*. 2023. URL: https://en.wikipedia.org/wiki/Monte_Carlo_method (visited on 12/12/2023).

[10] P. Steinbach. *Normal Mode (Harmonic) Analysis*. Jan. 2019. URL: https://bioinformatics.niaid.nih.gov/cmm/intro_simulation/node26.html#:~:text=%2C%20the%20potential%20near%20the%20minimum,any%20classical%20system%20behaves%20harmonically. (visited on 12/12/2023).

[11] Wikipedia. *Computational Fluid Dynamics*. 2023. URL: https://en.wikipedia.org/wiki/Computational_fluid_dynamics (visited on 12/12/2023).

[12] Cadence CFD. *Explaining the Finite Difference Method and Heat Transfer*. 2023. URL: https://resources.system-analysis.cadence.com/blog/msa2022-explaining-the-finite-difference-method-and-heat-transfer (visited on 12/12/2023).

[13] D. Rowell. "State-Space Representation of LTI Systems". In: (Oct. 2022). URL: https://web.mit.edu/2.14/www/Handouts/StateSpace.pdf (visited on 12/12/2023).

[14] O. Cano et al. *High-Order Electrical RLC Oscillators - PID and LQR Control*. Jan. 2022. URL: https://forschung.rwu.de/sites/forschung/files/2022-03/EC_Seminar_High-order_electrical_RLC_oscillators_PID_and_LQR_control.pdf (visited on 12/12/2023).

[15] P. Benner and J. Saak. "Linear-Quadratic Regulator Design for Optimal Cooling of Steel Profiles". In: (May 2005). URL: https://monarch.qucosa.de/api/qucosa%3A18598/attachment/ATT-6/ (visited on 12/12/2023).

[16] J. Saak and P. Benner. "Efficient Numerical Solution of the LQR-problem for the Heat Equation". In: *PAMM* 4.1 (Dec. 2004), pp. 648–649. ISSN: 1617-7061, 1617-7061. DOI: 10.1002/pamm.200410305. (Visited on 12/12/2023).

[17] C.J. Rapson. *Spatially Distributed Control - Heat Conduction in a Rod*. 2008. URL: https://www.diva-portal.org/smash/get/diva2:1015367/FULLTEXT01.pdf (visited on 12/12/2023).

[18] D.V. Widder. *The Heat Equation*. Academic Press, Jan. 1976. ISBN: 978-0-08-087383-1.

[19] J.D. Babbitt. "On the Differential Equations of Diffusion". In: *Canadian Journal of Research* 28a.4 (July 1950), pp. 449–474. DOI: 10.1139/cjr50a-037.

[20] N. Subani et al. "Analytical Solution of Homogeneous One-Dimensional Heat Equation with Neumann Boundary Conditions". In: *Journal of Physics: Conference Series* 1551.1 (May 2020), p. 012002. DOI: 10.1088/1742-6596/1551/1/012002.

[21] J.C. Strikwerda. *Finite Difference Schemes and Partial Differential Equations, Second Edition*. Society for Industrial and Applied Mathematics, Jan. 2004. DOI: 10.1137/1.9780898717938.

[22] G. Schulz and C. Graf. *Regelungstechnik 2: Mehrgrößenregelung, Digitale Regelungstechnik, Fuzzy-Regelung*. München: Oldenbourg Wissenschaftsverlag Verlag, 2013. DOI: 10.1524/9783486736151.

[23] K.E. Atkinson, W. Han, and D. Stewart. *Numerical Solution of Ordinary Differential Equations*. 1st ed. Hoboken: Wiley, Jan. 2009. DOI: 10.1002/9781118164495.

[24] M. Zeltkevic. *Forward and Backward Euler Methods*. Apr. 1998. URL: https://web.mit.edu/10.001/Web/Course_Notes/Differential_Equations_Notes/node3.html (visited on 12/02/2023).

[25] Wikibooks. *Control Systems/Controllability and Observability*. 2024. URL: https://en.wikibooks.org/wiki/Control_Systems/Controllability_and_Observability (visited on 01/15/2024).

[26] Wikibooks. *Control Systems/State-Space Stability*. 2024. URL: https://en.wikibooks.org/wiki/Control_Systems/State-Space_Stability (visited on 01/15/2024).

[27] The MathWorks, Inc. *Write C Functions Callable from MATLAB (MEX Files)*. 2023. URL: https://mathworks.com/help/matlab/call-mex-files-1.html (visited on 11/18/2023).

[28] The MathWorks, Inc. *Build MEX Function or Engine Application - MATLAB Mex*. 2023. URL: https://mathworks.com/help/matlab/ref/mex.html (visited on 11/18/2023).

[29] dSPACE GmbH. *SIL Introduction*. 2023. URL: https://www.dspace.com/de/gmb/home/news/engineers-insights/sil-introduction.cfm (visited on 11/18/2023).

[30] The MathWorks, Inc. *Row-Major and Column-Major Array Layouts*. 2023. URL: https://mathworks.com/help/coder/ug/what-are-column-major-and-row-major-representation-1.html (visited on 11/18/2023).