Model Predictive Control

— Applications in Logistics —

Jürgen Pannek
Dynamics in Logistics
Fachbereich 04: Produktionstechnik
Universität Bremen

*EXZELLENT.*
Foreword

This script originates from a correspondent lecture held during the winter term 2014 at the University of Bremen. The central aims of the lecture are the introduction of the so called model predictive control technique (MPC) for dynamic systems. MPC is used in a variety of field including chemical, mechatronic and automotive engineering, financial markets and logistics. In particular, we discuss

- Basic models for control,
- Fundamental definitions from system theory,
- Simulation and discretization,
- Feedforward and feedback controls,
- Optimization methods,
- Model predictive control algorithm, and
- Feasibility, stability and suboptimality.

The lecture is complemented by a series of example classes. These classes focus on

- Programming skills in Matlab, and
- Implementation of a model predictive controller.

At the end of the lecture, students should understand the concept of MPC, know basic algorithms for the required stages, understand the interplay of methods and be able to implement and adapt such a controller. The script is build on my previous lectures regarding MPC at the University of Bayreuth and the book [7].

Additional useful information on MPC may be found in [17]. The books of Sontag [18] and Khalil [11] may be used as a reference regarding the system theoretic background. Regarding simulation and discretization, the book of Hairer and Wanner [8, 9] provides a good overview. Further details on optimization methods can be found in the book of Nocedal and Wright [14].

Jürgen Pannek
Bremen, January 28, 2015
# Contents

<table>
<thead>
<tr>
<th>1 Introduction</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 What is “Model”?</td>
<td>1</td>
</tr>
<tr>
<td>1.2 What is “Predictive”?</td>
<td>3</td>
</tr>
<tr>
<td>1.3 What is “Control”?</td>
<td>4</td>
</tr>
<tr>
<td>1.4 Organization of the Building Block Structure</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2 System Theory</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Dynamical System</td>
<td>9</td>
</tr>
<tr>
<td>2.2 Control Systems</td>
<td>11</td>
</tr>
<tr>
<td>2.3 Control Types and Properties</td>
<td>14</td>
</tr>
<tr>
<td>2.3.1 Feasibility</td>
<td>15</td>
</tr>
<tr>
<td>2.3.2 Stability</td>
<td>16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3 Models in Logistics</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Car Model</td>
<td>21</td>
</tr>
<tr>
<td>3.2 Transport System</td>
<td>23</td>
</tr>
<tr>
<td>3.3 Supply Chain</td>
<td>24</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4 Simulation</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Discrete time Systems</td>
<td>25</td>
</tr>
<tr>
<td>4.2 Continuous time Systems</td>
<td>26</td>
</tr>
<tr>
<td>4.2.1 One Step Methods</td>
<td>26</td>
</tr>
<tr>
<td>4.2.2 Convergence</td>
<td>28</td>
</tr>
<tr>
<td>4.2.3 Adaptive Step Size Methods</td>
<td>29</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5 Optimization and Optimal Control</th>
<th>33</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Optimal Control Problem</td>
<td>33</td>
</tr>
<tr>
<td>5.2 Discretization Methods</td>
<td>34</td>
</tr>
<tr>
<td>5.2.1 Full Discretization</td>
<td>35</td>
</tr>
<tr>
<td>5.2.2 Recursive Discretization</td>
<td>37</td>
</tr>
<tr>
<td>5.3 Optimization Methods</td>
<td>37</td>
</tr>
<tr>
<td>5.3.1 Unconstrained Optimization</td>
<td>37</td>
</tr>
<tr>
<td>5.3.2 Constrained Optimization</td>
<td>41</td>
</tr>
<tr>
<td>5.3.3 Sequential Quadratic Programming</td>
<td>45</td>
</tr>
<tr>
<td>5.3.4 Interior Point Method</td>
<td>46</td>
</tr>
</tbody>
</table>
6  MPC Algorithm and Results  
   6.1  Basic MPC Algorithm ................................................. 49
   6.2  Feasibility .......................................................... 51
   6.3  Stability ............................................................. 53

Bibliography  

Glossary  

Index
Chapter 1

Introduction

Model Predictive Control (MPC) represents a very simple idea for control design, which is intuitively understandable and can be implemented using standard tools. These tools originate from different fields of research such as system theory, modeling, differential and difference equations, simulation, optimization and optimal control. While each of the tools can be applied easily, their connection is slightly more involved, and the analysis of the combination is rather complex. Within this lecture, we will study the basic tools and see how they can be connected to form a MPC. The analysis of MPC is outside the scope of this lecture, and interested readers may refer to the books [7, 17].

The intention of the MPC method is to generate a control law, which represents a so called static state feedback. This means that the control law is evaluated at certain points in time given the current condition/state of the system. The aim of the method is

• to compute an optimal control input for a process Chapter 5
• to stabilize a given reference Chapter 6

based on

• the current state of the process, Chapter 3
• a model of the process, and Chapter 3
• a prediction of possible future states of the model. Chapter 4

The foundations of any dynamical system with and without control are formed by the field of system theory. In the following Chapter 2 we will cover basic notation, concepts and definitions, which will be used throughout the rest of the lecture.

But first, we discuss the terms model, predictive and control separately in a more informal way to get a better understanding of the MPC problem.

1.1 What is “Model”?

Intuitively, we all know what a model is. We have identified it by learning to control our actions using predictions of the effect of these actions. These predictions are based on a model, and form a model of reality in our mind, which we fit to reality using our past experiences. There are simple connection, e.g. “I push a ball, then it rolls”. Yet we may also accumulate very complicated systems such as cars, supply chains or weather forecast. Hence the model is something deterministic, without uncertainty and predictable for all times.
Unfortunately, as we all experienced, models do not represent reality one–to–one. So if we use a model, there may be deviations between model prediction and reality, especially if long time horizons are considered. For a model, we always focus on those aspects we are interested in and do not try to describe all of reality. Hence, the problem is split into two parts,

- the model, which describes what we are interested in, and
- the environment, which contains everything else.

Since we cannot tell anything about the environment (as it is not modeled), interactions between model and environment can only be interpreted as disturbances.

If you look up the term model, you will end up with different explanations. Although these explanations are from different fields, they are remarkably similar:

“A model is a composition of concepts, which are used to help people know, understand, or simulate a subject the model represents.”

\[\text{(Wikipedia: Model – Computer science)}\]

“A model is a theoretical construct representing processes by a set of variables and a set of logical and/or quantitative relationships between them. The model is a simplified framework designed to illustrate complex processes, often but not always using mathematical techniques.”

\[\text{(Wikipedia: Model – Economics)}\]

“A model poses for any modeler as part of the (creative) process.”

\[\text{(Wikipedia: Model – Art)}\]

“A model consists of a set along with a collection of finitary operations, and relations that are defined on it.”

\[\text{(Wikipedia: Model – Mathematics)}\]

Within this lecture, we focus on the quantitative description of a model, i.e. qualitative results such as “a ball will roll downhill” are not the kind of model properties we are looking for. Instead, we utilize laws, which may, e.g., be given by physics or econometrics, and describe at least some part of our impression of reality. Fitting these laws to observations of reality creates new problems:

- For one, we consider noisy measurements. In this context, noisy means that if we take a measurement, e.g. length, weight, time etc., then errors occur since the instruments we use are not perfect.

- And secondly, our laws are imperfect as reality is far more complex than the rules we apply. They also show a stochastic behavior, which makes it impossible to predict exactly their output.
1.2 What is “Predictive”? 

The methodology to obtain a good model despite these shortcomings is not in the scope of this lecture but is treated in “Modeling and System Identification”. Within this lecture, we neglect disturbances.

Within this lecture we consider models satisfying either the so called nonlinear discrete time control systems form

\[ x(n + 1) = f(x(n), u(n)), \tag{1.1} \]

or the so called nonlinear continuous time control systems form

\[ \dot{x}(t) = f(x(t), u(t)), \tag{1.2} \]

where \( x \) represents the internal state of the system, \( u \) the external force on the system, \( f \) the law or dynamics of the system, and \( n, t \) the discrete and continuous time respectively. Note that a model is a specific system. In the following Chapter 2, we will give a formal definition of a system. Examples of models will be stated in Chapter 3.

1.2 What is “Predictive”? 

The model we discussed before can be used to make a projection of future events, a so called prediction. We also know that if the model and the time analysis is correct, so will be the prediction itself. If the model or the projection is not correct, then the prediction may be flawed.

Regarding the wording, one has to be very careful. If we look up prediction, we see in the following:

A prediction or forecast is a statement about the way things will happen in the future, often but not always based on experience or knowledge. While there is much overlap between prediction and forecast, a prediction may be a statement that some outcome is expected, while a forecast is more specific, and may cover a range of possible outcomes. A prediction may be contrasted with a projection, which is explicitly dependent on stated assumptions.

(Wikipedia: Prediction)

Since we already know what a model is, we only need to understand a projection. In principle, the latter is an imitation of the impact of time on a model of a process. This means that the time component of the model is artificially used to anticipate the future or future outcomes. Today, a projection is typically done by computer software. Except for algebraic software, however, all of these methods are incorrect. The errors of these methods are systemic and arise from the way numbers are stored within a computer, that is as a floating point number. Floats cannot describe all numbers, not even all natural numbers. Yet, these projection methods are fast and for short time periods the error bound is small, which renders them applicable for MPC. We will discuss prediction methods in details in Chapter 4.

How does a prediction look like? Basically, it is an evaluation of the dynamics. In the discrete time case of system (1.1) that is

\[ x(1) = f(x(0), u(0)), \]
\[ x(2) = f(x(1), u(1)), \]
\[ x(3) = f(x(2), u(2)), \]
\[ \vdots \]
In the continuous time case of system (1.2), we require a method to solve the differential equation. Such a method defines a solution operator $\Phi$ such that

$$x(t_1) = \Phi(t_1, t_0, x(t_0), u(t_0)),$$

$$x(t_2) = \Phi(t_2, t_1, x(t_1), u(t_1)),$$

$$x(t_3) = \Phi(t_3, t_2, x(t_2), u(t_2)),$$

$$\vdots$$

Again, the solution operator may introduce errors into the prediction, a matter which we discuss in Section 4.2.

1.3 What is “Control”?

The last term within MPC is control. A control is a possibility to influence the behavior of a system and its future development by external means. Hence, if the model is supposed to be influencable, a respective component must exist within the system — the so called input to a system. In that case, the system is called a control system. The state of a system which can be sensed in any way from outside is called the output of a system. Accordingly, control is not something originating within a system, but it is put on a system. Here, we use the term control system in a very general way:

“A system is a functional unity which processes and assigns signals. It describes the temporal cause–and–effect chain of the input parameter and the output parameter.”

Coming back to the input, the information obtained from the output can be used to check whether the objective is accomplished. Commonly such a system is visualized using a block diagram. Within such a diagram, processing units are represented by boxes while assignments are shown as arrows indicating the direction of the signal. There exist two classes of controls, the so called open loop and closed loop one, cf. Figure 1.1 and 1.2.

![Figure 1.1: General structure of a open loop control system](image)

Figure 1.1: General structure of a open loop control system

![Figure 1.2: General structure of a closed loop control system](image)

Figure 1.2: General structure of a closed loop control system

In either case, the output of the control system is based on its input. Yet in the open loop case, the input (the control) is chosen at random. The closed loop version closes the loop by
utilizing the output of the control system to generate its input by some kind of control law. This control law is also called a feedback.

We like to note that the feedback from is typically preferable. This is due to its ability to reject or at least compensate for disturbances happening at runtime of the process. The logic behind the feedback, however, may be very involved and requires not only skills to design it, but also to maintain it.

By design, the open loop control cannot deal with disturbances. To recall this from the model Section 1.1, a disturbances is anything that is not modeled, which is (almost) always the case. Still, development of an open loop control is much simpler and allows us to introduce overall measures like performance, green goals, money, customer satisfaction etc. For this reason, it is often used to get a first impression of how a “good” behavior of the system should look like. In a second step, this behavior is then mimicked by a feedback loop.

The idea of MPC is similar, but abstains from the mimicking part. Instead, the open loop control is used directly. The idea of MPC was first announced by Propoi in [16], who already used the first–discretize–then–optimize approach most commonly used today. The principle of MPC is at best quoted as follows:

“One technique for obtaining a feedback controller synthesis from knowledge of open loop controllers is to measure the current control process state and then compute very rapidly for the open-loop control function. The first portion of this function is then used during a short time interval, after which a new measurement of the process state is made and a new open-loop control function is computed for this new measurement. The procedure is then repeated.”

(Lee and Markus [12])

This simplicity is also the charm and success background of MPC. Each model predictive controller follows three distinct steps:

1. Based on the initial value and a given finite time horizon, an optimal open loop control is computed for the model at hand.

![Figure 1.3: Open loop optimization step of MPC](image-url)
2. Given the open loop solution, the first element of the control is implemented at the process.

3. And last, the new state of the system is estimated and the horizon is shifted forward in time. This render the method to be iteratively applicable.

Note that – while the MPC method itself is simple – the tasks to be completed in the method are typically not simple. To really understand the properties of MPC, we need to take a closer look at the intermediate task, and at respective methods.

1.4 Organization of the Building Block Structure

The lecture follows the requirements of the MPC approach as indicated in the previous illustration in Figures 1.3, 1.4 and 1.5. To this end, we gradually introduce the required components,
which form the building blocks of the method. Here, we will not treat the problem of how to derive the current state of the process. Instead, we assume measurement to be perfect.
Chapter 2

System Theory

Using E.D. Sontags words [18]:

While on the one hand we want to understand the fundamental limitations that mathematics imposes on what is achievable, irrespective of the precise technology being used, it is also true that technology may well influence the type of question to be asked and the choice of mathematical model.

Now, what is the connection of the statement above to our MPC setting? For one, if we derive a model of a process, this is typically done using physical or economical laws. This results in a set of differential equations, ordinary, partial or stochastic alike. Upon implementation of a control action, however, we face the problem that actuators are driven by digital computers. Hence, the control input typically follows a sample–and–hold fashion. This means that the control input stays fixed for a certain period, the so called sampling period, until a new control is set. So from an implementation point of view, it would be better to consider difference equations instead of differential equations, which exhibit the sampling behavior by design.

Here, we consider so called sampled–data systems — a mixture of both concepts. In particular, our examples are given as differential equations, that is in continuous time, whereas the numerical implementation as well as the analysis of our controller design relies on discrete time systems, i.e. using difference equations.

The aim of this chapter is to define these two kinds of systems and introduce fundamental concepts and terminology which we require throughout the lecture. In particular we give a general definition of a control system, its inputs and its outputs in Section 2.1. Next, we discuss the mentioned discrete time, continuous time and sampled data systems in Section 2.2. Last, we introduce the concept of constraints of a system and the respective feasibility and stability properties in Section 2.3.

2.1 Dynamical System

As indicated by their name

\[
\text{discrete time system} \quad \longleftrightarrow \quad \text{continuous time system}
\]

the notion of time is the difference. To handle both within one concept we first introduce the notion of a time set.
Chapter 2: System Theory

Definition 2.1 (Time set)
A time set $\mathcal{T}$ is a subgroup of $(\mathbb{R}, +)$.

By setting $\mathcal{T} = \mathbb{Z}$ or $\mathcal{T} = \mathbb{R}$, we can formally switch between discrete and continuous time.

Having defined time, we now introduce the states and controls of a system:

Definition 2.2 (State and Control)
We call the set $\mathcal{U}$ the control set and the set $\mathcal{X}$ the state set. Moreover, the set of all maps from a set $\mathcal{I} \subset \mathcal{T}$ to a set $\mathcal{U}$ is denoted by $U^\mathcal{I} = \{ u \mid u : \mathcal{I} \rightarrow \mathcal{U} \}$ and called the set of control functions. The elements $x \in \mathcal{X}$ and $u \in \mathcal{U}$ are called state and control of a system.

The sets $\mathcal{X}$ and $\mathcal{U}$ are chosen very general here, which allows us to apply the entire theory to all cases fitting to this concept. From a practical point of view, one can think of $\mathcal{X}$ as a subset of $\mathbb{R}^n$ and $\mathcal{U}$ as a subset of $\mathbb{R}^m$. Note that the concept also allows $\mathcal{U} = \{ 1, 2, 3, 4, 5, 6, \mathbb{R} \}$ representing the gears of a car.

The next idea we require for a system is a notion of possible connection of points. That is, if we are in point $x \in \mathcal{X}$, and we apply control $u \in \mathcal{U}$, at which point do we end up? More generally speaking, this gives us the concept of a transition map for all possible connections:

Definition 2.3 (Transition map)
A transition map is a map $\varphi : D_\varphi \rightarrow \mathcal{X}$ where

$$D_\varphi \subset \{(\tau, \sigma, x, u) \mid \sigma, \tau \in \mathcal{T}, \sigma \leq \tau, x \in \mathcal{X}, u \in \mathcal{U}^{(\sigma, \tau)}\}$$

satisfying $\varphi(\sigma, \sigma, x, \bullet) = x$. Here, $\bullet \in \mathcal{U}^{(\sigma, \sigma)}$ denotes the empty sequence.

Looking at it differently, the transition map is nothing else than an method where we plug in the start and end point, the current state and the control we want to apply, and it gives as the state at then end point. At a later point we see that this is a simulation tool. The set $D_\varphi$ is then the set of all possibilities. Later, these are called simulation runs.

Asking the other way around, if we are in point $A$, can we apply control $u \in \mathcal{U}$? Note that, e.g., if the state of the gearing is top gear, then we cannot increase it anymore.

Definition 2.4 (Admissibility)
Given time instances $\tau, \sigma \in \mathcal{T}$, $\sigma < \tau$, a control $u \in \mathcal{U}^{(\sigma, \tau)}$ is called admissible for a state $x \in \mathcal{X}$ if $(\tau, \sigma, x, u) \in D_\varphi$.

Using these definitions, we can introduce the notion of a system. The definition basically says that the system must not be void, i.e. it exists, and that we can split and join it in time:

Definition 2.5 (System)
A tupel $\Sigma = (\mathcal{T}, \mathcal{X}, \mathcal{U}, \varphi)$ is called system if the following conditions hold:

- For each state $x \in \mathcal{X}$ there exists at least two elements $\sigma, \tau \in \mathcal{T}$, $\sigma < \tau$, and some $u \in \mathcal{U}^{(\sigma, \tau)}$ such that $u$ is admissible for $x$. (Nontriviality)
• If \( u \in U^{(\sigma, \mu)} \) is admissible for \( x \) then for each \( \tau \in [\sigma, \mu) \) the restriction \( u_1 := u|_{[\sigma, \tau)} \) of \( u \) to the subinterval \( [\sigma, \tau) \) is also admissible for \( x \) and the restriction \( u_2 := u|_{(\tau, \mu]} \) is admissible for \( \varphi(\tau, \sigma, x, u_1). \) (Restriction)

• Consider \( \sigma, \tau, \mu \in T, \sigma < \tau < \mu. \) If \( u_1 \in U^{(\sigma, \tau)} \) and \( u_2 \in U^{(\tau, \mu)} \) are admissible and \( x \) is a state such that \( \varphi(\tau, \sigma, x, u_1) = x_1, \varphi(\mu, \tau, x_1, u_2) = x_2, \) then the concatenation

\[
u = \begin{cases} u_1, & t \in [\sigma, \tau) \\ u_2, & t \in [\tau, \mu) \end{cases}
\]

is also admissible for \( x \) and we have \( \varphi(\mu, \sigma, x, u) = x_2. \) (Semigroup)

Comparing this definition of a system to the reality of a process, we identify \( x \in X \) as the state of the plant. Note that the state is only a snapshot. The control \( u \in U \) is an exogenous variable and can be used to manipulate the future development of the system.

Before we can introduce constraints and the properties of feasibility and stability, we require one more notion. Implicitly, we already used it within Definition 2.5, where we split one control into two revealing a gluing point. Indeed, the transition map allows us to predict future states \( x(t) \in X \) for all \( t \in T \) telling us how the system evolves. Hence, given an initial state \( x(\sigma) = x_0 \) and a control function \( u(\cdot) \in U^{(\sigma, \tau)} \), we can fully describe the development of the state in time. This description object is called a trajectory.

**Definition 2.6 (Trajectory)**

Given a system \( \Sigma \), an interval \( I \subseteq T \) and a control \( u(\cdot) \in U^I \) we call \( x \in X^I \) a trajectory on the interval \( I \) if it satisfies

\[
x(\tau) = \varphi(\tau, \sigma, x(\sigma), u|_{[\sigma, \tau)}) \quad \forall \sigma, \tau \in I, \sigma < \tau.
\]

Based on these general definitions we now specify the systems we are going to deal with.

## 2.2 Control Systems

Since there exist fundamental differences between the continuous time and discrete time settings, we introduce respective systems separately. Moreover, we define the notion of a sampled–data system. The latter concept allows us to treat continuous time examples in a discrete time setting.

**Definition 2.7 (Discrete time Control System)**

Consider a function \( f : X \times U \to X \). A system of difference equations

\[
x_u(k + 1, x_0) := f(x_u(k, x_0), u(k)), \quad k \in \mathbb{N}_0 \quad (2.1)
\]

is called a discrete time control system. Moreover \( x_u(k, x_0) \in X \) is called state vector and \( u(k) \in u \) control vector.
Chapter 2: System Theory

Existence and uniqueness of a solution of (2.1) is clear by induction. In particular, we obtain a unique solution in positive time direction for a certain maximal existence interval $I$ if $(x_0, u)$ with $x_0 \in X$ and $u \in U^I$ is an admissible pair, cf. Definition 2.4.

**Remark 2.8**
If the system (2.1) is independent of $u$, then it is called dynamical system. In control theory, one aims at designing a control based on the state of the system, which renders the control system to be independent of the control. The controller is designed to induce certain properties.

In the continuous time setting, a control system is given as follows:

**Definition 2.9** (Continuous time Control System)
Consider a function $f : X \times U \to X$. A system of first order ordinary differential equations

$$\dot{x}_u(t) := \frac{d}{dt} x_u(t, x_0) = f(x_u(t, x_0), u(t)), \quad t \in \mathbb{R}$$

(2.2)

is called a continuous time control system.

The control system itself only gives us the state change over time. To compute a possible future — a trajectory in the sense of Definition 2.6 — we require additional information on the starting point.

**Definition 2.10** (Initial Value Condition)
Consider a point $x_0 \in X$. Then the equation

$$x(0) = x_0 \in X$$

(2.3)

is called the initial value condition.

Note that existence and uniqueness of a trajectory is guaranteed if the system is Lipschitz or if the requirements of Caratheodory’s Theorem are met, cf. [11] and [18] respectively.

**Remark 2.11**
Even for simple problems, the optimal control function $u(\cdot)$ is discontinuous [18]. Hence, considering only the set of continuous control functions appears to be too strict. Moreover, the semigroup property stated in Definition 2.5 is violated by a concatenation of two continuous functions if $U$ is restricted to the class of continuous functions. For this reason, one typically relies on the class of measurable functions.

Utilizing existence and uniqueness, we can introduce the notion of a trajectory or solution:

**Definition 2.12** (Solution)
We call the unique function $x_u(t, x_0)$ a solution for $t \in T$ if it satisfies the initial value condition

$$x(0) = x_0 \in X$$

(2.3)
and the control system equation
\[ x_u(k + 1, x_0) := f(x_u(k, x_0), u(k)), \quad k \in \mathbb{N}_0 \] (2.1)
or
\[ \dot{x}_u(t) := \frac{d}{dt}x_u(t, x_0) = f(x_u(t, x_0), u(t)), \quad t \in \mathbb{R}. \] (2.2)

In real life systems, we often face the problem that the control system is continuous time, yet the control is applied in a discrete time nature. This is typically the case for controls being implemented by digital computers. The interconnection between these two settings is called sampling.

**Definition 2.13** (Sampling Solution) Consider a fixed time grid \( T = \{ t_0, t_1, \ldots \} \) with \( t_0 = 0 \) and \( t_k < t_{k+1} \forall k \in \mathbb{N}_0 \). The we call \( x_T(t, x_0, u) \) a sampling solution if the initial value condition
\[ x(0) = x_0 \in X \] (2.3)
and the control system equation
\[ \dot{x}_u(t) := \frac{d}{dt}x_u(t, x_0) = f(x_u(t, x_0), u(t)), \quad t \in \mathbb{R}. \] (2.2)

Together with Caratheodory’s Theorem hold and \( u \) is a concatenation of control functions \( u_k \in \mathcal{U} \) with \( u(t) = u_k(t) \forall t \in [t_k, t_{k+1}) \). The sampling solution is inductively defined via
\[ x_T(t, x_0, u) := x_{u_k}(t - t_k, x_T(t_k, x_0, u)) \quad \forall t \in [t_k, t_{k+1}). \] (2.4)

The time distance \( \Delta_k := t_{k+1} - t_k \) is called sampling period and its reciprocal \( \Delta_k^{-1} \) is called sampling rate.

Upon implementation of controllers, digital computers use piecewise constant control functions. This gives us the following:

**Definition 2.14** (Sampling with zero–order hold) Consider the situation of Definition 2.13 with constant control functions \( u_k \equiv c_k \forall t \in [t_k, t_{k+1}) \). Then \( x_T(\cdot, x_0, u) \) is called sampling solution with zero–order hold.

Now we can derive a discrete time system (2.1) from a continuous time system (2.2).

**Definition 2.15** (Sampled–data System) Consider the sampling solution with zero–order hold \( x_T \) as given by Definition 2.14. Then we call the discrete time system
\[ x_u(k, x_0) := x_T(t_k, x_0, u) \] (2.5)
a sampled–data system and \( x_u(k, x_0) \) is called sampled–data solution for all \( k \in \mathbb{N}_0 \).
So far, we have been looking at control systems themselves. In the following, we concentrate on types of control laws, and properties of the control system, which we want to induce by designing the control law.

### 2.3 Control Types and Properties

Given a continuous time control system (2.2), we can derive a respective sampled–data system (2.5), which corresponds to a discrete time control system (2.1). Hence, if the latter discrete time system shows certain properties, so will the sampled–data and the continuous time control system.

Here, we are looking for the converse, i.e. we seek control laws inducing certain properties. In control theory, so called open–loop and closed–loop control laws are considered.

**Definition 2.16** (Open–loop or Feedforward Control Law)

Consider the setting of Definition 2.7, 2.9 or 2.15. A function \( u : T \rightarrow U \) based on some initial condition \( x_0 \) is called an open–loop or feedforward control law.

**Definition 2.17** (Closed–loop or Feedback Control Law)

Consider the setting of Definition 2.7, 2.9 or 2.15. A function \( \mu : \mathcal{X} \rightarrow U \) is called a closed–loop or feedback control law and is applied by setting \( u(\cdot) := \mu(x(\cdot)) \).

Definitions 2.16 and 2.17 also show the two main lines of work in control theory. Open–loop control is based on the assumption that a good model is available and we wish to modify/optimize its behavior. The corresponding techniques have emerged from the classical calculus of variations and from other areas of optimization theory. This approach typically leads to a control law \( u(\cdot) \) which has been computed offline before the start of the system like a preprogrammed flight plan.

\[
u(1), u(2), \ldots, f : \mathcal{X} \times U \rightarrow \mathcal{X},\ x(1), x(2), \ldots\]

Figure 2.1: General structure of a open loop control system

In particular, one computes the function \( u(\cdot) \) based on the initial conditions \( x_0 \) and the control system dynamics \( f(\cdot, \cdot) \) and applies it directly without cross checking available measurements. The result is the so called open loop solution

\[
x(k + 1) = f(x(k), u(k))
\]

emanating from the initial value \( x(0) = x_0 \). Unfortunately, unknown perturbations and uncertainties may occur in reality. In that case, applying an open–loop control law \( u(\cdot) \) over a long time horizon may lead to large deviations of the state trajectory.

The second line of work is the attempt to integrate these aspects about the model or about the operating environment of the system into the control law. The central tool is the use of feedback correcting deviations from the desired behavior, i.e. we implement a control
2.3 Control Types and Properties

\[ u(\cdot) = \mu(x(\cdot)) \] depending on the actual state of the system. This method reveals the closed loop solution

\[ x(n + 1) = f(x(n), \mu(x(n))) \] (2.7)

emanating from the initial value \( x(0) = x_0 \). Note that implementing the feedback controller requires the states of the system to be continuously monitored and the control law to be evaluated online.

\[ f : \mathcal{X} \times \mathcal{U} \to \mathcal{X} \]

\[ u(1), u(2), \ldots \]

\[ \mu : \mathcal{X} \to \mathcal{U} \]

\[ x(1), x(2), \ldots \]

Figure 2.2: General structure of a closed loop control system

Remark 2.18

Today, these two methods are used in an integrated manner. For one, an a priori path is computed using the open loop method. And in a second step, closed loop methods are applied to prevent deviations from such a path. The advantage of such a combined approach is its reduced complexity which allows for much higher sampling rates.

We now turn towards the properties which we like to induce into our control system at hand.

2.3.1 Feasibility

In order to analyze the longterm behavior of a system, we need to consider time tending to infinity. For our general system from Definition 2.5, the property of admissibility (cf. Definition 2.4) needs to be extended to infinity. Note that there exists a difference between limits points and the limit of a sequence. Hence, the definition needs slight adaptations:

Definition 2.19 (Infinite Admissibility)

Given a system \( \Sigma \) and a state \( x \in \mathcal{X} \), an element of \( \mathcal{U}^{(\sigma, \infty)} \) is called admissible for \( x \) if every restriction \( u|_{[\sigma, \tau)} \) is admissible for \( x \) and each \( \tau > \sigma \).

Such a property is of particular interest in the presence of constraints. Constraints are motivated by boundaries of processes, e.g. that there exists only a finite number of gears in a gearbox or that the capacity of a road is bounded. The most general approach to incorporate constraints in the control system setting is via sets:

Definition 2.20 (Constraints)

For given state and control sets \( \mathcal{X} \) and \( \mathcal{U} \), we call the subsets

\[ \mathcal{X} \subset \mathcal{X} \text{ and } \mathcal{U} \subset \mathcal{U} \] (2.8)

the constrained state and control sets.
Based on these constraints, we can now introduce the concept of *feasibility sets*. To this end, we incorporate the constraints into the definition of admissibility. In principle, this reduces to a change of sets in the definition of the transition map in Definition 2.3. Yet, there is an important difference:

Admissibility deals with controls, feasibility is about states. In particular, a control is called admissible for a specific state. And a state is called feasible if there exists a control sequence such that future states satisfy the state constraints.

Since we have to anticipate future events in the state space, feasibility sets require us to change the perspective in time. Hence, a reverse time view is needed. This leads to the following definition:

**Definition 2.21 (Feasible Set and Admissible Set)**

Consider a control system

\[ x_u(k+1, x_0) := f(x_u(k, x_0), u(k)), \quad k \in \mathbb{N}_0 \]  

(2.1)

with constraints

\[ X \subset \mathcal{X} \quad \text{and} \quad U \subset \mathcal{U} \]  

(2.8)

and \( X^0 \subset X \). For any time frame \( I = [0, N] \subset \mathbb{N}_0 \) the *feasible set* is defined via

\[ X^N := \{ x_0 \mid \exists u : x_u(N, x_0) \in X^0, \quad x_u(k, x_0) \in X, \quad u(k) \in U \forall k \in \{0, \ldots, N-1\} \}. \]  

(2.9)

Moreover, the *admissible set* is given by

\[ U^N_{X^N}(x_0) := \{ u \mid x_u(N, x_0) \in X^0, \quad x_u(k, x_0) \in X, \quad u(k) \in U \forall k \in \{0, \ldots, N-1\} \}. \]  

(2.10)

With regards to the MPC procedure outlined in Chapter 1, we see that the control must be chosen such that the feasible set will never be empty. To guarantee this property of the MPC algorithm, we will discuss different methods in Section 6.2.

### 2.3.2 Stability

Apart from guaranteeing that the controller renders the state of the system to be feasible, we also want to steer a system into a certain state and keep it there. This is the so called *stability property* of a system. To this end, one usually considers equilibrium points and characterizes their stability properties using sets, comparison functions, or Lyapunov functions. Here, we discuss all three methods, which are indeed equivalent. So what do we want to show:

Roughly speaking, an equilibrium point is considered to be *stable* if all solutions starting at nearby points stay close to this point, otherwise it is called *unstable*. Moreover, it is called *asymptotically stable* if all solutions starting at nearby points not only stay nearby but also tend to the equilibrium point as time tends to infinity.

Note that in order to show (asymptotic) stability, we require that the control is infinitesimally admissible. The control is not expected to be unique, and hence the solution \( x_u(\cdot, x_0) \) may not be defined uniquely as well and we may have to consider the case of more than one solution emanating from an initial value \( x_0 \). Since the control is independent from the system, we can
set it arbitrarily. Hence, we need to distinguish properties which are independent of the control, and which are induced by the control.

**Definition 2.22 (Stability Concepts for Equilibrium Points)**
Consider a control system
\[
x_u(k+1, x_0) := f(x_u(k, x_0), u(k)), \quad k \in \mathbb{N}_0
\]
with constraints
\[
\mathbb{X} \subset \mathcal{X} \quad \text{and} \quad \mathbb{U} \subset \mathcal{U}. \tag{2.8}
\]

(i) A point \( x^* \in \mathbb{X} \) is called *strong or robust equilibrium* if \( x_u(k, x^*) = x^* \) holds for all \( u \in \mathcal{U} \) and all \( k \in \mathcal{I} = \mathbb{N}_0 \).

(ii) A point \( x^* \in \mathbb{X} \) is called *weak or controlled equilibrium* if there exists a control law \( u \in \mathcal{U} \) such that \( x_u(k, x^*) = x^* \) holds for all \( k \in \mathcal{I} = \mathbb{N}_0 \).

This definition naturally induces two concepts of stability and asymptotic stability, robustness and controllability. These concepts depend on the interpretation of \( u \) as an external control or a disturbance.

**Definition 2.23**
The equilibrium point \( x^* = 0 \) of a control system (2.1) is

- *strongly or robustly stable* if, for each \( \varepsilon > 0 \), there exists a real number \( \delta = \delta(\varepsilon) > 0 \) such that for all \( u \in \mathcal{U} \) we have
  \[
  \|x_0\| \leq \delta \implies \|x_u(k, x_0)\| \leq \varepsilon \quad \forall k \in \mathcal{I} = \mathbb{N}_0
  \]  \hspace{1cm} \tag{2.11}

- *strongly or robustly asymptotically stable* if it is stable and there exists a positive real constant \( r \) such that for all \( u \in \mathcal{U} \)
  \[
  \lim_{k \to \infty} x_u(k, x_0) = 0
  \]  \hspace{1cm} \tag{2.12}

holds for all \( x_0 \) satisfying \( \|x_0\| \leq r \). If additionally \( r \) can be chosen arbitrary large, then \( x^* \) is called *globally strongly or robustly asymptotically stable*.

- *weakly stable or controllable* if, for each \( \varepsilon > 0 \), there exists a real number \( \delta = \delta(\varepsilon) > 0 \) such that for each \( x_0 \) there exists a control \( u \in \mathcal{U} \) guaranteeing
  \[
  \|x_0\| \leq \delta \implies \|x_u(k, x_0)\| \leq \varepsilon \quad \forall k \in \mathcal{I} = \mathbb{N}_0.
  \]  \hspace{1cm} \tag{2.13}

- *weakly asymptotically stable or asymptotically controllable* if there exists a control \( u \in \mathcal{U} \) depending on \( x_0 \) such that (2.13) holds and there exists a positive constant \( r \) such that
  \[
  \lim_{k \to \infty} x_u(k, x_0) = 0 \quad \forall \|x_0\| \leq r.
  \]  \hspace{1cm} \tag{2.14}

If additionally \( r \) can be chosen arbitrary large, then \( x^* \) is called *globally asymptotically stable*. 
Chapter 2: System Theory

Note that strongly asymptotically stable control systems are boring since the chosen control does not affect the stability property of the system. Still, it can be used to improve the performance of the system. Moreover, strong asymptotic stability is interesting in the presence of significant measurement or discretization errors.

The concept of weak stability, on the other hand, naturally leads to the question how to compute a control law such that \( x^\star \) is weakly stable, and, in particular, how to characterize the quality of a control law.

Apart from Definition 2.23, alternative definitions can be found in the literature, and these definitions are all equivalent. The first one utilizes so called comparison functions.

### Definition 2.24 (Comparison Functions)

The following classes of functions are called comparison functions:

- A continuous non-decreasing function \( \gamma : \mathbb{R}^+_0 \to \mathbb{R}^+_0 \) satisfying \( \gamma(0) = 0 \) is called class \( \mathcal{G} \) function.
- A function \( \gamma : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0} \) is of class \( \mathcal{K} \) if it is continuous, zero at zero and strictly increasing.
- A function is of class \( \mathcal{K}_\infty \) if it is of class \( \mathcal{K} \) and also unbounded.
- A function is of class \( \mathcal{L} \) if it is strictly positive and it is strictly decreasing to zero as its argument tends to infinity.
- A function \( \beta : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0} \) is of class \( \mathcal{KL} \) if for every fixed \( t \geq 0 \) the function \( \beta(\cdot, t) \) is of class \( \mathcal{K} \) and for each fixed \( s > 0 \) the function \( \beta(s, \cdot) \) is of class \( \mathcal{L} \).

The functions allow us to geometrically include solutions emanating from a given initial value by inducing a bound for the worst case. This directly leads to the following result, cf. [11]:

### Theorem 2.25 (Stability Concepts)

Consider a control system (2.1).

(i) An equilibrium \( x^\star = 0 \) is strongly asymptotically stable or robustly asymptotically stable if there exists an open neighborhood \( \mathcal{N} \) of \( x^\star \) and a function \( \beta \in \mathcal{KL} \) such that

\[
\|x_u(k, x_0)\| \leq \beta(\|x_0\|, k)
\]

holds for all \( x_0 \in \mathcal{N}, \ u \in U^I \) and all \( k \in I = \mathbb{N}_0 \).

(ii) An equilibrium \( x^\star = 0 \) is weakly asymptotically stable or asymptotically controllable if there exists an open neighborhood \( \mathcal{N} \) of \( x^\star \) and a function \( \beta \in \mathcal{KL} \) such that for every \( x_0 \in \mathcal{N} \) there exists a control law \( u \in U^I \) such that

\[
\|x_u(k, x_0)\| \leq \beta(\|x_0\|, k)
\]

holds for all \( k \in I = \mathbb{N}_0 \).

The second concept utilizes the Lyapunov functions, which can be interpreted as energy function of the system state. The main difference lies in considering a minimizing control in
the neighborhood of the considered state.

**Definition 2.26 (Control–Lyapunov–Function)**
Consider an equilibrium $x^* = 0$, a control system (2.1) with $f(0,0) = 0$ and a neighborhood $\mathcal{N}$ of $x^*$. Then a continuous function $V : \mathcal{N} \to \mathbb{R}_0^+$ is called a Control–Lyapunov–Function if there exist functions $\alpha, \alpha_1, \alpha_2 \in \mathcal{K}_\infty$ such that for all $x \in \mathcal{N}$ there exists a control function $u \in \mathcal{U}$ such that the inequalities

$$\alpha_1(\|x\|) \leq V(x) \leq \alpha_2(\|x\|) \quad (2.15)$$

$$\inf_{u \in \mathcal{U}} V(f(x,u)) - V(x) \leq -\alpha(x) \quad (2.16)$$

hold for all $x \in \mathcal{N} \setminus \{x^*\}$.

The idea of the Lyapunov function is comparable to a salad bowl: If we put a ball into the bowl, it will run downhill and remain at the lowest point. The lowest point regarding the Lyapunov function marks the desired equilibrium. Hence, the bowl has to be “put” at the right spot. A second thing is that energy needs to be drawn from the system continuously. Hence, if the height is only measurement, then the ball in the bowl is not allowed to move uphill again. If height and velocity are combined, this is a different and more applicable energy measure.

In our last step, we apply this energy concept to obtain stability by energy draining arguments, cf. [18]:

**Theorem 2.27 (Asymptotic Stability)**
Consider a control system (2.7) where $f(0,0) = 0$, a neighborhood $\mathcal{N}$ of $x^*$ and a continuous function $V : \mathcal{N} \to \mathbb{R}_0^+$.

(i) An equilibrium $x^* = 0$ is strongly asymptotically stable or robustly asymptotically stable if (2.15) and

$$\sup_{u \in \mathcal{U}} V(f(x,u)) - V(x) \leq -\alpha(x)$$

hold for all $x \in \mathcal{N}$.

(ii) An equilibrium $x^* = 0$ is weakly asymptotically stable or asymptotically controllable if for all $x \in \mathcal{N}$ there exists a control $u \in \mathcal{U}$ such that (2.15), (2.16) hold.

Again, the strong or robust concept means that no matter which control we consider, energy is drawn from the system. The weak concept requires additional work to design a control such that the stability property is induced.
Chapter 3

Models in Logistics

Within this chapter, we take a look at very different examples from the field of logistics. The first of these is a car model given by only one differential equation. The initial problem is to control the car such that it arrives at its target position. The second example deals with a transport system. Such a problem stems from an aggregated point of view. We do not look at distinct goods flowing through the network or cars driving through a street system, but at aggregated numbers. In contrast to this more centralized example, the last one deals with supply chains. Each stage within such a system is controlled locally, which gives us a distributed setting.

3.1 Car Model

Modeling of traffic scenarios can be done in various ways. The typical approach is to model each car with sufficient details using one-track or multi-body dynamics. Additionally, each car is constrained by the road boundaries and of course other cars. Utilizing such a model offers the highest degree of freedom in the control. Unfortunately, its complexity is high as well which renders the computation of an optimal solution for such a model to be inapplicable in real time.

Alternatively, a more abstract approach can be used. We define the route of each vehicle via routing points, which we interpolate using splines. The car is then controlled along the arc of the spline, cf. Figure 3.1 for a schematic sketch. Here, we utilize the velocity along the arc length as a control which gives a single ordinary differential equation per car.

Note that deriving the routing points is a decoupled problem, which may be solved by a traffic guidance system. For simplicity, the center of the traffic lane can be chosen. Regarding a production process or a single machine, these routing points can be regarded as a feedforward control.

To formalize this approach, we call $M \in \mathbb{N}$ the number of routine points. The routing points themselves are denoted by

$$\begin{pmatrix} x_m \\ y_m \end{pmatrix}, \quad m = 0, \ldots, M,$$

where $(x_0, y_0)^T$ is the initial position of the car on the traffic lane and $(x_M, y_M)^T$ is its target position.

Denoting the entire arc length by $L$, the routing points are interpolated via the cubic spline

$$S(\ell) = \begin{pmatrix} S_x(\ell) \\ S_y(\ell) \end{pmatrix}, \quad 0 \leq \ell \leq L,$$

where
Chapter 3: Models in Logistics

Figure 3.1: Definition of the driving path via splines for given routing points

which is parametrized by $\ell$ representing the position on the arc. The arc length is approximated by

$$
\ell_0 := 0, \quad \ell_{m+1} := \ell_m + \sqrt{(x_{m+1} - x_m)^2 + (y_{m+1} - y_m)^2}, \quad L := \ell_M.
$$

Last, we re-obtain the parametrized driving route via

$$
\begin{pmatrix}
  x(\ell) \\
  y(\ell)
\end{pmatrix}
= \begin{pmatrix}
  S_x(\ell) \\
  S_y(\ell)
\end{pmatrix}
\quad \text{for } 0 \leq \ell \leq L.
$$

The spline gives us the route of each car, and its velocity is the time derivative of the current position on the arc. Hence, driving along the route is equivalent to solving the initial value problem

$$
\dot{\ell}(t) = u(t), \quad \ell(0) = 0
$$

where $t$ denotes time and $u(t)$ represents the velocity of the car at time instant $t$. By choosing the velocity $u \in \mathbb{U}$ we can control the car along the route. The corresponding position at time instant $t$ is given by

$$
\begin{pmatrix}
  x(\ell(t)) \\
  y(\ell(t))
\end{pmatrix}
= \begin{pmatrix}
  S_x(\ell(t)) \\
  S_y(\ell(t))
\end{pmatrix}
$$

Remark 3.1

(i) Instead of the velocity along the route, we could also use the acceleration or jerk. These choices result in a differential equation of higher order. Additionally, the bounds on the velocity are then state constraints, which drastically increase the complexity of the problem.

(ii) As mentioned before, we could also impose more complex models for each car and the respective dynamics. However, these model would lead to an increase in the computational cost. Since the modeled arcs are locally controlled by sublayer controllers of the car, these arcs represent reality close enough. Hence, such an approach is more efficient.
3.2 Transport System

From a higher level point of view, cars are pieces flowing around in a network. Such a network can be given by an undirected graph $G = (V,E)$, where $V = \{x_1, x_2, \ldots, x_K\} \subset \mathbb{R}^2$ denotes the set of $K$ nodes and $E \subseteq V \times V$ the set of edges connecting these nodes. Each node $x_i$ represents a cardinal point, in the traffic context this corresponds to a crossing. At these cardinal points, several edges meet. Similar to Kirchhoff conditions for electrical networks – if there exists a flow on these edges, they must satisfy boundary conditions of the edges. Hence, the nodes are candidates for controllers. Each edge $e_{i,j}$ serves as a connection between two cardinal points $x_i$ and $x_j$. The flow along such an edge is modeled by the transport equation

$$\frac{\partial y}{\partial t}(x,t) + c \frac{\partial y}{\partial x}(x,t) = 0$$

for $(x,t) \in (x_i, x_j) \times [0,\infty)$, 

$$\frac{\partial y(x_k,t)}{\partial v} = u_k(t)$$

for $k \in \{i,j\}$,

$$y(x,0) = y_0(x)$$

for $x \in (x_i, x_j)$ (3.3)

Here, the state $y(x,t)$ represents the flow at time $t$ and position $x$. The control $u_k(\cdot)$ is a Neumann boundary condition representing the inward and outward flow at a cardinal point $x_i$, which can be set to $\{0,1\}$, i.e. stop or go. In a more aggregated setting, it may also be modeled continuously by representing the average throughput at a node. Note that this is not a contradiction but a generalization of Kirchhoff’s law. If only one edge is considered, the inward and outward flow must only satisfy the boundary conditions. If more than one edge is considered, $u_k(\cdot)$ influences the adjacent edges as well and neighboring edges must be considered. The boundary conditions therefore connect different edges. The full network is the combined model defined on the set $\Omega := V \cup E \subset \mathbb{R}^2$. Here, edge $e_{i,j}$ is considered to be the interval $(x_i, x_j)$ for $(i,j) \in E$.

**Remark 3.2**

In contrast to the car model from Section 3.1, we now have several controls that can be set. The setting is called centralized since all these controls can be set by a central entity. Note that we do not consider individual cars moving across the graph, instead we consider the flow of cars and control it via the nodes.
### 3.3 Supply Chain

A supply chain is a multi-stage network, which is driven by the dynamics

\[
\begin{align*}
\dot{s}_p(t) &= f_s(a^p(t), \ell^p(t)) \quad \text{(Stock)} \\
\dot{o}_u^p(t) &= f_o(o^p(t), a^p(t)) \quad \text{(Unfulfilled order to stock)} \tag{3.4} \\
\dot{b}^p(t) &= f_b(d^p(t), \ell^p(t)) \quad \text{(Backlog from stock)}
\end{align*}
\]

where \( p \in S = \{S, M, R\} \) denotes the stages, cf. [4, 15, 19]. Typically, the stage set contains supplier (S), manufacturer (M) and retailer (R). Moreover, \( a^p, \ell^p, o^p \) and \( d^p \) denote the arriving and leaving as well as the order and demand rates. For all times \( t \geq 0 \) and stages \( p \in S \), the system is subject to the constraints

\[
\begin{align*}
0 &\leq o^p(t) \leq o^p_{\max} \\
0 &\leq o_u^p(t) \leq o_u^p_{\max} \\
0 &\leq s^p(t) \leq s^p_{\max} \\
0 &\leq b^p(t) \leq b^p_{\max} \tag{3.5}
\end{align*}
\]

as well as unknown customer orders \( o^C \) and fixed delivery delays \( \tau_{ij} \), where \( i, j \in S \) represent consecutive stages, see also Figure 3.3 for a schematic sketch of a three stage supply chain.

![Supply Chain Diagram](Supplier-Manufacturer-Retailer-Customer.png)

Figure 3.3: Sketch of a three stage supply network consisting of supplier, manufacturer and retailer including the customer.

We define the state vector as the abbreviation \( x^p = (s^p, o_u^p, b^p)\). The stages need to be linked since arrival/leaving as well as demand/order information is required to evaluate the dynamics. Here, we use \( a^j(t + \tau_{ij}) = \ell^i(t) \) and \( d^j(t) = o^i(t) \) for consecutive nodes \( i, j \in S \) and \( a^i(\tau_{ii}) = o^i(t) \) for the supplier to define these connections.

**Remark 3.3**

This last example is different from both the car model from Section 3.1 and the transport system from Section 3.2 as the control is computed locally. For both the car and the transport system model, these control values are computed by one entity, hence the actions are coordinated. In the so called distributed setting, the communication plays an important role in controlling such a system.
Chapter 4

Simulation

In this chapter, we focus on numerically evaluating the solution of a control system. This will be required in the model predictive control method in order to derive an optimal trajectory for a given cost functional over a finite time horizon. Furthermore, generating such solutions is also important in terms of simulation, which refers to the following:

Simulation is the use of a mathematical model to recreate a situation, often repeatedly, so that the likelihood of various outcomes can be more accurately estimated.

Here, we will not focus on the stochastic issues of how to generate and interpret probability measures of the outcome. Instead, we focus on the part to generate solutions. To this end, we present suitable methods, discuss the convergence theory for one step methods and give an introduction into step size control algorithms.

4.1 Discrete time Systems

Using discrete time systems of the form

\[ x_u(k+1, x_0) := f(x_u(k, x_0), u(k)), \quad k \in \mathbb{N}_0, \]

all we need is a fixed control sequence \( u(0), u(1), \ldots \) to generate a trajectory via the iteration

\[
\begin{align*}
    x_u(1, x_0) &= f(x_u(0, x_0), u(0)), \\
    x_u(2, x_0) &= f(x_u(1, x_0), u(1)), \\
    x_u(3, x_0) &= f(x_u(2, x_0), u(2)), \\
    &\vdots
\end{align*}
\]

From this iteration, it is simple to see that the impact of a chosen control sequence can be analyzed by evaluating the trajectory. Hence, an idea to study stochastic properties regarding probability measures can be done, e.g., via a Monte–Carlo–Simulation, cf. [1]. Note that in simulation, one typically does not analyze the impact of controls but of disturbances or parameter uncertainties.

Hence, generating a open loop trajectory is easy once the open loop control is given. How to actually compute this control will be discussed in the following Chapter 5.
4.2 Continuous time Systems

After dealing with discrete time systems, we now focus on the more complex continuous time case. Recall that our control system is of the form

\[ \dot{x}_u(t) = f(x_u(t, x_0), u(t)), \quad t \in \mathbb{R}. \]  

(2.2)

For this system, we like to introduce an operator \( \varphi(T; 0, x, u) \) such that we re-obtain a control system in discrete time form

\[ x_u(k+1, x_0) := f(x_u(k, x_0), u(k)) = \varphi(T; 0, x_u(k, x_0), u(k)), \quad k \in \mathbb{N}_0. \]  

(2.1)

Proceeding this way allows us to apply the solution operator as a black box to evaluate a continuous time system and base the MPC method on top of a discrete time system.

Note that in the notation above, the solution operator is nothing else than a sampling system, cf. Definition 2.13. Yet the sampling system assumes that we can evaluate the continuous system precisely. Here, we apply numerical methods to evaluate (2.2). Unfortunately, such a method typically provides an approximation \( \tilde{\varphi}(T; 0, x, u) \) of the solution only.

To introduce these methods, we first define a grid function and its properties:

\textbf{Definition 4.1} (Grid function)
Consider time instants \( 0 = t_0 < t_1 < \ldots < t_M = T \).

(i) A set \( G = \{t_0, t_1, \ldots, t_M\} \) of time instants is called a time grid on the interval \([0, T]\). The values \( h_i := t_i+1 - t_i \) and \( h := \max_{i=0,\ldots,M-1} h_i \) are called step sizes and maximal step size, respectively.

(ii) A function \( \tilde{\varphi} : G \times G \times \mathcal{X} \times \mathcal{U} \to \mathcal{X} \) is called grid function.

(iii) Assume that the solution \( \varphi(t; t_0, x_0, u) \) of (2.2) exists for \( t \in [0, T] \). Then a family of grid functions \( \tilde{\varphi}_j, j \in \mathbb{N} \), on time grids \( G_j \) on the interval \([0, T]\) with maximal step sizes \( h_j \) is called (discrete) approximation of \( \varphi(t; t_0, x_0, u) \), if it is convergent, i.e.,

\[ \max_{t_i \in G_j} \| \tilde{\varphi}_j(t_i; t_0, x_0, u) - \varphi(t_i; t_0, x_0, u) \| \to 0 \quad \text{as } h_j \to 0. \]  

(4.1)

The convergence of the approximation is said to be of order \( p > 0 \) if for all compact sets \( K \subset \mathbb{R}^d, Q \subset \mathcal{U} \) there exists a constant \( M > 0 \) such that

\[ \max_{t_i \in G_j} \| \tilde{\varphi}_j(t_i; t_0, x_0, u) - \varphi(t_i; t_0, x_0, u) \| \leq M h_j^p \]  

(4.2)

holds for all \( x_0 \in K \), all \( u \in Q \) and all sufficiently fine grids \( G_j \) on \([0, T]\).

In other words, an approximation \( \tilde{\varphi}(t_i; t_0, x_0, u) \) is a grid function defined on \( G \). It approximates the values of the true solution \( \varphi(t_i; t_0, x_0, u) \) at the grid points \( t_i \) and becomes the more accurate the finer the grid becomes, i.e., for \( h_j \to 0 \). Moreover, we see that if the order of convergence \( p \) is larger, the approximation will converge faster.

4.2.1 One Step Methods

One class of such approximations is called the class of one step methods. One step methods compute the grid function \( \tilde{\varphi} \) iteratively via

\[ \tilde{\varphi}(t_0; t_0, x_0, u) := x_0, \quad \tilde{\varphi}(t_{i+1}; t_0, x_0, u) := \Phi(\tilde{\varphi}(t_i; t_0, x_0, u); u, h_i) \]  

(4.3)
for \( i = 0, \ldots, M - 1 \) starting from the given initial value \( x_0 \). The mapping \( \Phi : \mathcal{X} \times \mathcal{U} \times \mathbb{R} \to \mathcal{X} \) is called the one step function. The mapping itself should be easy to implement, cheap to evaluate on a computer, and provide a convergent approximation in the sense of Definition 4.1(iii).

To design such a map \( \Phi \), we use the integral equation

\[
\varphi(t_{i+1}; t_0, x_0, u) = \varphi(t_i; t_0, x_0, u) + \int_{t_i}^{t_{i+1}} f(\varphi(t; t_0, x_0, u), u) \, dt
\]

is satisfied by the solution of the differential equation (2.2) at consecutive grid points \( t_i \) and \( t_{i+1} \). Then, we approximate the integral via the rectangle rule

\[
\int_{t_i}^{t_{i+1}} f(\varphi(t; t_0, x_0, u), u) \, dt \approx (t_{i+1} - t_i) f(\varphi(t_i; t_0, x_0, u), u)
\]

which give us

\[
\varphi(t_{i+1}; t_0, x_0, u) \approx \varphi(t_i; t_0, x_0, u) + h_i f(\varphi(t_i; t_0, x_0, u), u)
\]

Hence, we can approximate the solution via

\[
\tilde{\varphi}(t_{i+1}; t_0, x_0, u) = \varphi(t_i; t_0, x_0, u) + h_i f(\varphi(t_i; t_0, x_0, u), u)
\]

(4.4)

revealing the so called Euler iteration via (4.3) with

\[
\Phi(x; u, h) := x + hf(x, u).
\]

The Euler scheme is the most simple one step method. In general, these methods can be written as follows:

**Definition 4.2 (Explicit Runge–Kutta Scheme)**

An \( s \)-stage (explicit) Runge–Kutta method is given by

\[
k_i := f \left( t + c_i h, x + h \sum_{j=1}^{i-1} a_{ij} k_j, u \right) \quad \text{for } i = 1, \ldots, s
\]

\[
\Phi(x; u, h) := x + h \sum_{i=1}^{s} b_i k_i.
\]

The value \( k_i = k_i(x, u, h) \) is called the \( i \)th stage of the method.

The methods are sometimes also given via so called Butcher tableaus. Depending on the number of stages, these tableaus are quite extensive and require some thought of how to actually implement them as the parameters may be very small or are given as quotients. The most prominent methods are the Euler, Heun and the classical Runge–Kutta one, which are given in Table 4.2. Please note that these methods are designed for ordinary differential equations. Partial differential equation additionally require a discretization of the state space, cf. finite difference or finite element methods [2,13].
Table 4.1: General form of a Butcher tableau

<table>
<thead>
<tr>
<th>c_1</th>
<th>a_{21}</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>c_2</td>
<td>a_{31}</td>
<td>a_{32}</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>c_s</td>
<td>a_{s1}</td>
<td>a_{s2}</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>b_1</td>
<td>b_2</td>
<td>...</td>
</tr>
</tbody>
</table>

Table 4.2: Butcher tableaus for the Euler, Heun and classical Runge-Kutta method

4.2.2 Convergence

We still need to know whether the class of one step methods is appropriate, i.e. if respective results converge towards the true solution. To this end, we define the error at time \( t_i \) as

\[
e(t_i) := \| \tilde{\varphi}(t_i; t_0, x_0, u) - \varphi(t_i; t_0, x_0, u) \|.
\]

The main idea to show convergence is to use the triangle inequality in order to separate the error sources in the iteration (4.3) into

(a) the error caused by the previously accumulated error

(b) and the local error.

Abbreviating \( \varphi(t_i) = \varphi(t_i; t_0, x_0, u) \) and \( \tilde{\varphi}(t_i) = \tilde{\varphi}(t_i; t_0, x_0, u) \), this leads to the estimate

\[
e(t_{i+1}) = \| \tilde{\varphi}(t_{i+1}) - \varphi(t_{i+1}) \| = \| \Phi(\tilde{\varphi}(t_i); u, h_i) - \varphi(t_{i+1}) \|
\leq \| \Phi(\tilde{\varphi}(t_i); u, h_i) - \Phi(\varphi(t_i); u, h_i) \| + \| \Phi(\varphi(t_i); u, h_i) - \varphi(t_{i+1}) \|. \tag{4.5}
\]

To show convergence, we need to cope with these error sources. Since the errors are separated, we can deal with them individually using so called Lipschitz and consistency conditions:

**Definition 4.3** (Lipschitz and Consistency Condition) (i) A one step method satisfies the *Lipschitz condition* if for all compact subsets \( K \subset X \) and \( Q \subset U \) there exists a constant \( \Lambda > 0 \) such that for all sufficiently small \( h > 0 \) the inequality

\[
\| \Phi(x_1; u, h) - \Phi(x_2; u, h) \| \leq (1 + \Lambda h) \| x_1 - x_2 \| \tag{4.6}
\]

holds for all \( x_1, x_2 \in K \) and all \( u \in Q \).
(ii) A one step method is called consistent with order of consistency $p > 0$ if for all compact subsets $K \subset X$ and $Q \subset U$ there exists a constant $C > 0$ such that for all sufficiently small $h > 0$ the inequality

$$
\| \Phi(x; u, h) - \varphi(h; 0, x, u) \| \leq Ch^{p+1}
$$

(4.7)

holds for all $x \in K$ and all $u \in Q$.

Inequality (4.6) guarantees that the propagation of previous errors (a) within a one step method stays bounded. This property can be checked very easily by computing the Lipschitz constant of the method. The consistency condition (4.7), on the other hand, ensures that the local error (b) remains small. Its validation is more complex and requires the Taylor approximation and a comparison of summands.

**Remark 4.4**

To guarantee the order of consistency $p$ the dynamics $f$ must be $p$ times continuously differentiable with respect to $x$. If we consider a sampled data system (2.4) on an interval $[t_i, t_{i+1}]$ with sampling instant $t_k \in (t_i, t_{i+1})$ for some, this becomes a major issue since the control and thus the map is in general discontinuous and nonsmooth at the sampling instants. Here, we need to make sure that all sampling instants are elements of the time grid $G$.

Using the given assumptions, we can now state the main result of this section:

**Theorem 4.5** (Convergence of One Step Methods)

If a one step method $\Phi$ satisfies the Lipschitz condition (4.6) and the consistency condition (4.7) with order $p$, then the approximation $\tilde{\varphi}$ from (4.3) is convergent in the sense of Definition 4.1(iii) with order of convergence $p$.

### 4.2.3 Adaptive Step Size Methods

So far, we considered a priori fixed step sizes. Note that while these step sizes are not necessarily equally long, they are not subject to changes at runtime. Typically, there is a link between the sampling time $T$ and the step size $h$, which in most cases is given by $T = ih$ with $i \in \mathbb{N}$. Since for practical applications, $T$ might be rather large, setting $i = 1$ reveals $h = T$ and results in a large error. Setting $i \gg 1$ requires a lot of computations and may render the simulation process to be intractable.

Using adaptive step size methods, the step length is computed online, and not linked to the sampling time $T$. Adjusting $h$ appropriately to the solution, we can make the error term in (4.2) arbitrarily small without changing $T$. To this end, we allow for large $h_i$ if the error is small and use small $h_i$ when large errors are observed.

In order to obtain an efficient way to construct an adaptive grid $G$, we use two different one step methods $\Phi_1, \Phi_2$ with different orders of consistency $p_1 < p_2$ to compute a step length $h_i = t_{i+1} - t_i$ which guarantees a predefined local error bound $\text{tol}_{\text{ODE}}$. Here, by $p_1 < p_2$ we mean that for $\Phi = \Phi_1$ inequality (4.7) cannot hold for $p = p_2$, i.e., no matter how $C$ is chosen we have that (4.7) is violated for sufficiently small $h$. Since the term $\varphi(t_i; t_0, x_0, u)$ in (4.5) is not available at runtime, and we cannot use our previous idea to guarantee the local error (a)
to satisfy
\[ \| \Phi(\varphi(t; t_0, x_0, u); u, h_i) - \varphi(t_{i+1}; t_0, x_0, u) \| \leq \text{tol}_{\text{ODE}}. \]

Instead, we insert the term \( \varphi(t_{i+1}; t_i, \tilde{\varphi}(t_i; t_0, x_0, u), u) \) which gives us
\[
\| \tilde{\varphi}(t_{i+1}; t_i, x_0, u) - \varphi(t_{i+1}; t_i, x_0, u) \| \leq \| \Phi(\tilde{\varphi}(t_i; t_i, x_0, u); u, h_i) - \varphi(t_{i+1}; t_i, \tilde{\varphi}(t_i; t_0, x_0, u), u) \|
+ \| \varphi(t_{i+1}; t_i, \tilde{\varphi}(t_i; t_0, x_0, u), u) - \varphi(t_{i+1}; t_i, \varphi(t_i; t_0, x_0, u), u) \|
\]

The second term is independent of the choice of \( h_i = t_{i+1} - t_i \), which leave us with considering the first term only. Hence, we want to choose \( h_i \) such that
\[
\| \Phi(\tilde{\varphi}(t_i; t_i, x_0, u); u, h_i) - \varphi(t_{i+1}; t_i, \tilde{\varphi}(t_i; t_0, x_0, u), u) \| \leq \text{tol}_{\text{ODE}}
\]
holds for a predefined tolerable error bound \( \text{tol}_{\text{ODE}} \). Since the term \( \varphi(t_{i+1}; t_i, \tilde{\varphi}(t_i; t_0, x_0, u), u) \) is unknown, we need to circumvent this problem and impose a second one step method, which possesses a higher order of consistency \( p_2 > p_1 \). This allows for the following theorem:

**Theorem 4.6 (Adaptive Step Sizing Method)**
Consider two one step methods \( \Phi_1, \Phi_2 \) with orders of consistency \( p_1, p_2 \) satisfying \( p_2 \geq p_1 + 1 \). Then there exist constants \( k_1, k_2 > 0 \) such that for all sufficiently small \( h_i > 0 \) the computable error
\[
\varepsilon := \| \Phi_1(\tilde{\varphi}(t_i; 0, x_0, u), u, h_i) - \Phi_2(\tilde{\varphi}(t_i; 0, x_0, u), u, h_i) \| \quad (4.8)
\]
and the local error of the one step method \( \Phi_1 \)
\[
\varepsilon := \| \Phi_1(\tilde{\varphi}(t_i; 0, x_0, u), u, h_i) - \varphi(t_{i+1}; t_i, \tilde{\varphi}(t_i; 0, x_0, u), u) \| \quad (4.9)
\]
satisfy the inequality
\[
k_1 \varepsilon \leq \varepsilon \leq k_2 \varepsilon. \quad (4.10)
\]

For small step sizes it follows that \( k_1 \approx k_2 \approx 1 \), i.e. \( \varepsilon \approx \varepsilon_{i,1} \approx c_i h_i^{p_1+1} \) which gives us the estimate \( \varepsilon_{i} \approx \frac{\varepsilon}{h_i^{p_1+1}} \) for the coefficient \( c_i \). Hence, the error tolerance \( \text{tol}_{\text{ODE}} \) is satisfied (approximately) for the step size
\[
\text{tol}_{\text{ODE}} = c_i h_i^{p_1} = \frac{\varepsilon}{h_i^{p_1+1}} \implies h_{i,\text{new}} = \frac{1}{\sqrt{\text{fac} \frac{\text{tol}_{\text{ODE}}}{\varepsilon}}} h_i \quad (4.11)
\]
Due to these approximations, a security factor \( \text{fac} \in (0,1) \) is introduced to compensate for these errors. \( \text{fac} = 0.9 \) is a typical choice for this factor in many algorithms.

A schematic implementation of a one step scheme with adaptive step size is given in the following algorithm:

**Algorithm 4.7**
Suppose an initial value \( x_0 \), a control value \( u \), a tolerance \( \text{tol}_{\text{ODE}} \) and sampling period \( T \) are given.
4.2 Continuous time Systems

(1) Set \( \tilde{\phi}(0; 0, x_0, u) = x_0, i = 0, t_0 = 0, h_0 = T. \)

(2) If \( t_i = T \) stop; If \( t_i + h_i > T \) set \( h_i = T - t_i. \)

(3) Set \( t_{i+1} = t_i + h_i \) and compute \( \Phi_1(\tilde{\phi}(t_i; 0, x_0, u), u, h_i) \) and \( \Phi_2(\tilde{\phi}(t_i; 0, x_0, u), u, h_i). \)

(4) Compute \( \varepsilon \) and \( h_{i,new} \) according to (4.8) and (4.11).

(5) If \( \varepsilon > \text{tolODE} \) set \( h_i = h_{i,new} \) and goto (3).

(6) If \( \varepsilon \leq \text{tolODE} \) set \( \tilde{\phi}(t_{i+1}; t_0, x, u) = \Phi_2(\tilde{\phi}(t_i; 0, x_0, u), u, h_i), h_{i+1} = h_{i,new}, i = i + 1 \) and goto (2).

For two reasons, step size control algorithms are usually much more efficient than the use of equidistant time grids: Firstly, there typically exists regions, which allow for larger time steps and thus allow for a faster progress of the adaptive iteration procedure. And secondly, the additional effort of simultaneously evaluating two methods can be reduced significantly by embedding. This means that both methods use the same stages \( k_i \), thus the stages \( k_i \) only need to be evaluated once, cf. Table 4.3 for the Butcher tableau of the DoPri5 method.

| 0  | 1/5 | 1/5 |
| 3/10 | 3/10 | 9/10 |
| 4/5 | 14/15 | -16/15 | 32/9 |
| 8/9 | 19372 | -25360 | 64448 | 212 |
| 9/5 | 6561 | -2187 | 6561 | 729 |
| 1 | 9017 | -355 | 46732 | 49 | -5103 |
| 3168/8 | 33 | 5247 | 176 | 18566 |
| 1 | 35 | 0 | 560 | 125 | -2187 | 11 |
| 384 | 1113 | 192 | 6784 | 85 |
| 35 | 0 | 500 | 125 | -2187 | 11 | 84 |
| 384 | 1113 | 192 | 6784 | 85 |
| 5179 | 0 | 7571 | 393 | -21297 | 187 | 1 |
| 57600 | 16695 | 640 | -339900 | 2100 | 40 |

Table 4.3: Butcher tableau of the DoPri(4)5 method

**Remark 4.8**

Since Theorem 4.6 and the derivation of (4.11) require \( h_i \) to be sufficiently small, adaptive step size selection schemes usually do not rigorously maintain the specified error tolerance. While step sizing methods are reliable, in general only equidistant grids with sufficiently small maximal step size provide rigorous error bounds.
Chapter 5

Optimization and Optimal Control

So far, we have seen how we can generate a trajectory from a control system given a predefined control sequence. This chapter focuses on the question how an optimal control can be derived. To this end, we first require a notion of costs, which is used to assess each control sequence. This will give us a so called optimal control problem. We then transform the latter via discretization methods into an optimization problem. Here, we discuss the full and recursive discretization method. The resulting optimization problems are either sparse structured (full discretization) or dense (recursive discretization), and require respective algorithms from nonlinear optimization to solve them. Within the lecture, we shortly discuss the most prominent methods, the so called Interior Point (IP) and the Sequential Quadratic Programming (SQP) method.

5.1 Optimal Control Problem

The notion of a so called optimal control problem extends our model notion by assigning costs to each trajectory. These costs are given via the functional

\[ J_N(x_0, u) = \sum_{k=0}^{N-1} \ell \left( x_u(k, x_0), u(k) \right) + L(x_u(N, x_0)) \]  

where \( \ell : \mathcal{X} \times \mathcal{U} \to \mathbb{R} \) and \( L : \mathcal{X} \to \mathbb{R} \) are the so called stage and terminal costs. A typical choice of these functions is the quadratic version

\[ \ell(x, u) = \|x\|^2 + \lambda \|u\|^2 \]
\[ L(x) = \|x\|^2. \]

We like to note that computing a control

\[ u^* = \arg\min_{u \in U^N} J_N(x_0, u) \]  

may not be tractable if \( N \) is very large or even \( N = \infty \). While the infinite horizon solution, i.e. \( u^* \) for \( N = \infty \) would be our most preferred one, we know from Chapter 1 that \( N \) will be a positive finite and preferably small integer number.

Combining the cost functional (5.1) with the control system dynamics (2.1), the initial value condition (2.3) and the feasibility condition (2.9), we obtain the following:
Definition 5.1 (Optimal Control Problem (OCP))

We call the problem

\[
\text{Minimize } J_N(x_0, u) := \sum_{k=0}^{N-1} \ell(x_u(k), u(k)) + L(x_u(N, x_0)),
\]

with respect to \( u(\cdot) \in U^N_{\mathcal{X}^N}(x_0) \), subject to

\[
x_u(0, x_0) = x_0 \in \mathcal{X}^0, \quad x_u(k+1, x_0) = f(x_u(k, x_0), u(k)).
\]

an optimal control problem.

Here, we particularly emphasize the case of a sampled data continuous time system (2.4) inducing a discrete time system. Still, all results also apply for discrete time systems not linked to a continuous time one via sampling.

Remark 5.2

The equivalent continuous time problem can be obtained by replacing (2.1) by (2.2) and the cost function (5.1) by

\[
J_N(x_0, u) = \int_{t=0}^{NT} \ell(x_u(t, x_0), u(t)) \, dt + L(x_u(NT, x_0)).
\]

Within the last Chapter 4 we discussed how a solution for a control system (2.1) can be computed for a given initial value \( x_0 \) and control \( u \) using differential equation solvers. The aim of this chapter is to numerically solve the optimal control problem (OCP) as to compute a minimizing control \( u^* \in U^N_{\mathcal{X}^N}(x_0) \). To this end, we follow the so called “first discretize then optimize” approach. In a transformation step, we discretize the optimal control problem (OCP) into a nonlinear optimization problem in standard form, and in a solution step, we generate a control sequence as a solution of the discretized problem.

Remark 5.3

Apart from the “first discretize then optimize” approach there also exists a so called “first optimize then discretized” method. Applying the latter requires in deep knowledge of Pontryagin’s minimum principle, which is beyond the scope of this lecture. The basic idea is to introduce the adjoint differential equation as part of an integrated solution. This extended dynamics is then solved for the given boundary conditions, which consist of the initial value conditions of the system dynamics, and the endpoint conditions arising from the minimum principle for the adjoint equations. Such a solution method typically relies on numerical methods including Newton’s method and the differential equation solvers introduced in the previous Chapter 4.

5.2 Discretization Methods

The aim of discretization is to transform the optimal control problem (OCP) into a nonlinear optimization problem in standard form:
5.2 Discretization Methods

**Definition 5.4** (Nonlinear Optimization Problem)

We call the problem

\[
\text{Minimize } F(z) \\
\text{with respect to } z \in \mathbb{R}^n \\
\text{subject to } G(z) = 0 \text{ and } H(z) \geq 0.
\]

(NLP)

with maps \( F : \mathbb{R}^n \rightarrow \mathbb{R} \), \( G : \mathbb{R}^n \rightarrow \mathbb{R}^m \) and \( H : \mathbb{R}^n \rightarrow \mathbb{R}^r \) a nonlinear optimization problem in standard form.

Even though (OCP) is already a discrete time problem, the process of converting (OCP) into (NLP) is called discretization. Here, we will stick with this commonly used term even though in a strict sense we only convert one discrete problem into another.

As we will see, the (NLP) problem related to (OCP) can be formulated in different ways. The first variant, called full discretization, incorporates the dynamics (2.1) as additional constraints into (NLP). This approach is very straightforward but causes large computing times for solving the problem (NLP) due to its dimensionality, unless special techniques such as condensing for handling these constraints can be used on the optimization algorithm level.

The second approach is designed to deal with this dimensionality problem. It recursively computes \( x_u(k, x_0) \) from the dynamics (2.1) outside of the optimization problem (NLP) and is hence called recursive discretization. Proceeding this way, the dimension of the optimization variable \( z \) and the number of constraints is reduced significantly. However, the method has some drawbacks regarding parallelization, warm start and sensitivity.

### 5.2.1 Full Discretization

The full discretization technique to obtain an optimization problem in standard form (NLP) is the most simplest and most common one. The discrete time trajectory \( x_u(k, x_0) \) in (OCP) is given by the dynamics (2.1) via

\[
x_u(k + 1, x_0) := f(x_u(k, x_0), u(k)), \quad k \in \mathbb{N}_0,
\]

(2.1)

and in the continuous time control system case the map a numerical approximation

\[
\hat{\varphi}(t; t_0, x_0, u) := x_0, \quad \hat{\varphi}(t_{i+1}; t_0, x_0, u) := \Phi(\hat{\varphi}(t_i; t_0, x_0, u); u, h_i)
\]

(4.3)

is used.

In Definition 4.1 we introduced a time grid \( \mathcal{G} \), which allowed us to define a discrete time approximation of the continuous time solution of a given differential equation. Here, we use the same idea, but not only for the state but also for the control trajectories. Since the control trajectories in continuous time are considered to be constant between two sampling points, we define the discretization time grid \( \mathcal{G} \) as the set of all sampling points within one finite horizon

\[
\mathcal{G} := \{ t_k \in \mathbb{R} \mid t_k \text{ is sampling point} \}.
\]

(5.3)

Note that in the discrete time case we have \( \mathcal{G} := \{0, \ldots, N\} \). This definition is similar to Definition (4.1)(i) except that in the previous chapter we did not assume a finite horizon.

Now, each control value \( u(k), k \in \{0, \ldots, N - 1\} \) is an optimization variable in (OCP) and also an optimization variable in (NLP). The idea of the full discretization is now to treat each
point on the trajectory $x_u(k, x_0)$ as an additional independent $n_u$-dimensional optimization variable and define the total optimization variable via
\[
z := (x_u(0, x_0)^\top, \ldots, x_u(N, x_0)^\top, u(0)^\top, \ldots, u(N - 1)^\top)^\top. \tag{5.4}
\]
To guarantee that the solution of (NLP) also corresponds to a trajectory of (2.1), we add respective equality constraints to (NLP), which read
\[
x_u(k + 1, x_0) - f(x_u(k, x_0), u(k)) = 0 \quad \text{for } k \in \{0, \ldots, N - 1\} \tag{5.5}
\]
\[
x_u(0, x_0) - x_0 = 0 \tag{5.6}
\]
Additionally, we have to reformulate the constraints $u \in \mathbb{U}^N_{X_n}(x_0)$, which can be written as
\[
x_u(k, x_0) \in \mathbb{X} \quad k \in \{0, \ldots, N\}
\]
\[
u(k) \in \mathbb{U} \quad k \in \{0, \ldots, N - 1\} \tag{5.7}
\]
Note that the setting is easily extended to the case of time varying constraints.
In the following, we assume $\mathbb{X}$ and $\mathbb{U}$ to be given by a set of functions
\[
G^S_i : \mathbb{R}_x^n \times \mathbb{R}_u^n \to \mathbb{R}, \quad i \in \mathcal{E}^S = \{1, \ldots, n_G\}
\]
\[
H^S_i : \mathbb{R}_x^n \times \mathbb{R}_u^n \to \mathbb{R}, \quad i \in \mathcal{I}^S = \{1, \ldots, n_H\}
\]
via equality and inequality constraints of the form
\[
G^S_i(x_u(k, x_0), u(k)) = 0, \quad i \in \mathcal{E}^S, k \in K_i \subseteq \{0, \ldots, N\} \tag{5.8}
\]
\[
H^S_i(x_u(k, x_0), u(k)) \geq 0, \quad i \in \mathcal{I}^S, k \in K_i \subseteq \{0, \ldots, N\}. \tag{5.9}
\]
where the index sets $K_i, i \in \mathcal{E}^S \cup \mathcal{I}^S$ formalize the possibility that some of these constraints are not required at time instant $k \in \{0, \ldots, N\}$.

Summarizing, we obtain the following definition:

**Definition 5.5 (Full Discretization)**
The nonlinear programming problem in standard form (NLP)

\[
\text{Minimize} \quad F(z) := \sum_{k=0}^{N-1} \ell(x_u(k, x_0), u(k)) + L(x_u(N, x_0))
\]

with respect to
\[
z := (x_u(0, x_0)^\top, \ldots, x_u(N, x_0)^\top, u(0)^\top, \ldots, u(N - 1)^\top)^\top \in \mathbb{R}^{n_z}
\]

subject to $G(z) = \begin{bmatrix} G^S_i(x_u(k, x_0), u(k)) \end{bmatrix}_{i \in \mathcal{E}^S, k \in K_i} = 0$

and $H(z) = \begin{bmatrix} H^S_i(x_u(k, x_0), u(k)) \end{bmatrix}_{i \in \mathcal{I}^S, k \in K_i} \geq 0$

is called the full discretization of Problem (OCP).

The advantage of the full discretization is its simplicity. On the backside, the method results in a high dimensional optimization variable $z \in \mathbb{R}^{(N+1)n_z + Nn_u}$ and a large number of both equality and inequality constraints. Since computing times of solvers for (NLP) depend massively on the size of the problem, this is unwanted.
5.2.2 Recursive Discretization

In contrast to the full discretization, the recursive discretization avoids the problem of high dimensional optimization problems (NLP) regarding both the optimization variable itself and the number of equality constraints. The methodology of the recursive discretization is inspired by the (hierarchical) divide and conquer principle. Basically, the control system dynamics is decoupled and treated as a subproblem of the optimization problem. These two layers exchange information regarding the control sequence $u$ and the initial value $x_0$ from the (NLP) to the simulation, and the state sequences $x_u(\cdot, x_0)$ in the opposite direction.

The optimization variable $z$ reduces to

$$z := (u(0)^\top, \ldots, u(N-1)^\top)^\top$$

and the constraint functions $G^S_i : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}$, $i \in E^S$ are given by (5.8). The inequality constraints $H^S_i : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}$, $i \in I^S$ and the cost function $F$ remain unchanged. Hence, the recursively discretized problem takes the following form:

**Definition 5.6 (Recursive Discretization)**

The nonlinear programming problem in standard form (NLP)

$$\begin{align*}
\text{minimize} & \quad F(z) := \sum_{k=0}^{N-1} \ell(x_u(k, x_0), u(k)) + L(x_u(N, x_0)) \\
\text{subject to} & \quad G(z) = \left[ G^S_i(x_u(k, x_0), u(k)) \right]_{i \in E^S, k \in K_i} = 0 \\
& \quad H(z) = \left[ H^S_i(x_u(k, x_0), u(k)) \right]_{i \in I^S, k \in K_i} \geq 0
\end{align*}$$

is called the recursive discretization of Problem (OCP).

Analyzing the dimension of the optimization variable and the number of equality constraints, we see that using the recursive discretization the optimization variable consists of $N \cdot n_u$ scalar components and the number of equality constraints is reduced to the number of conditions in (5.8). We can conclude that this discretization is minimal in these regards.

Unfortunately, the method has some drawbacks regarding parallelization, warm start and sensitivity. These shortcomings can to some extend be circumvented by incorporating multiple shooting techniques, which are beyond the scope of this lecture. The basic idea is to find a suitable compromise between the full and the recursive discretization by introducing few breaking points into the recursive discretization.

5.3 Optimization Methods

Now that we derived an (NLP) from an (OCP), we discuss the two main methods for solving the optimization problem. Both methods rely on an approximation of the (NLP) by quadratic functions, yet they are quite different in their approach and their applicability regarding the size of the (NLP). For details we refer to [7,14].

5.3.1 Unconstrained Optimization

Now that we have discretized the optimal control problem (OCP) and transformed it into a nonlinear optimization problem (NLP) in standard form, our aim is to compute a minimizer $z$, ...
which then gives us an optimal control $u$ for our original problem. In this section, we discuss the foundations of all optimization techniques; details for methods like the popular Sequential Quadratic Programming (SQP) or Interior-Point Methods (IP) are then given in the subsequent section. On the one hand, this allows us to characterize the main principles of such algorithms. On the other hand, it reveals an abstract method, exposes the computationally expensive parts of such algorithms and allows for rearranging the ordering of these steps to reduce the computational cost.

Although the problem (NLP) is actually a constrained optimization problem, in this section we present solution methods for dealing with unconstrained optimization problems since the basic principles are similar to those in constrained optimization. Since there do not exist any restrictions on the optimization variable $z$, the unconstrained optimization problem is a special case of the standard (NLP) problem and is given by

$$\begin{align*}
\text{minimize} & \quad F(z) \\
\text{with respect to} & \quad z \in \mathbb{R}^n
\end{align*}$$

Due to the size of such a problem, in practice we need to solve it on a computer. The computer, however, cannot deal with this problem in an abstract way, it can only evaluate the given functions like the cost function $F$ and possibly its derivative $\nabla_z F$ for finitely many $z$. Hence, the goal in constructing a solution method for any (NLP) problem is to find a strategy for choosing these evaluation points in order to reliably identify a solution of (NLP). While the identification of and the search strategy for a solution are primary goals for a solution method, one still has to keep in mind some secondary goals like computing time, required memory storage and also the number of required function evaluations whose detailed analysis is beyond the scope of this chapter but can be found in the standard literature for nonlinear optimization.

Focussing on the primary goals, we first need to characterize a solution of a problem (NLP) and how it can be checked whether a given point $z \in \mathbb{R}^n$ is a solution. In principle, we are interested in so called global minimizers.

**Definition 5.7** (Global Minimizer)
A point $z^* \in \mathbb{R}^n$ is a global minimizer of the function $F : \mathbb{R}^n \rightarrow \mathbb{R}$ if $F(z^*) \leq F(z)$ holds for all $z \in \mathbb{R}^n$.

Unfortunately, for general nonlinear — and in particular nonconvex — problems such global minimizers are hard to find in practice since we only have local knowledge of the function $F$ and its derivative $\nabla_z F$. Due to this local knowledge and our intention to evaluate only a small number of vectors $z$, we cannot cover the entire definition space of $F$. As a consequence, if we construct an algorithm under these restrictions we can never be sure if we reached a global minimizer. Nevertheless we are often able to identify a so called local minimizer.

**Definition 5.8** (Local Minimizer)
A point $z^* \in \mathbb{R}^n$ is a local minimizer of the function $F : \mathbb{R}^n \rightarrow \mathbb{R}$ if there exists a neighborhood $\mathcal{N}(z^*)$ such that $F(z^*) \leq F(z)$ holds for all $z \in \mathcal{N}$.

In some cases not all hopes for identifying a global minimizer are lost since we may have additional information on the function $F$. For example, if $F$ is convex, then every local minimizer is also a global minimizer. But even if we know that $F$ is convex, we still need to find such a minimizer. In order not to have to check all values $z$ in a certain area, we will assume
the function $F$ to be at least twice continuously differentiable which allows us to use a more practical way of locating a minimizer using Taylor’s theorem. Here and in the following we denote derivatives using the following notation which is common in nonlinear optimization. For a continuously differentiable function $g = (g_1, \ldots, g_p) : \mathbb{R}^n \to \mathbb{R}^p$ we use the gradient notation for the Jacobian matrix

$$
\nabla_z g(z) = 
\begin{pmatrix}
\frac{\partial g_1}{\partial z_1} & \cdots & \frac{\partial g_1}{\partial z_n} \\
\vdots & \ddots & \vdots \\
\frac{\partial g_p}{\partial z_1} & \cdots & \frac{\partial g_p}{\partial z_n}
\end{pmatrix}
$$

which we abbreviate to $\nabla g$ if there is no ambiguity. For a twice continuously differentiable function $g : \mathbb{R}^n \to \mathbb{R}$ we write the so-called Hessian as

$$
\nabla^2 z g(z) = 
\begin{pmatrix}
\frac{\partial^2 g}{\partial z_1^2} & \cdots & \frac{\partial^2 g}{\partial z_1 z_n} \\
\vdots & \ddots & \vdots \\
\frac{\partial^2 g}{\partial z_n z_1} & \cdots & \frac{\partial^2 g}{\partial z_n^2}
\end{pmatrix}
$$

which we abbreviate to $\nabla^2 g$ if there is no danger of confusion.

**Theorem 5.9 (Taylor’s Theorem)**

Consider a function $F : \mathbb{R}^n \to \mathbb{R}$ which is continuously differentiable and a direction vector $d \in \mathbb{R}^n$. Then we have

$$
F(z + d) = F(z) + \nabla F(z + td)^\top d
$$

for some $t \in (0, 1)$. If $F$ is twice continuously differentiable, then we also have

$$
F(z + d) = F(z) + \nabla F(z)^\top d + \frac{1}{2} d^\top \nabla^2 F(z + td)d
$$

for some $t \in (0, 1)$.

The advantage of Taylor’s theorem is that it allows us to introduce knowledge on the gradient $\nabla F(z^*)$ and the Hessian $\nabla^2 F(z^*)$ into the search for a local minimizer $z^*$. In particular, first order necessary conditions are derived very easily.

**Theorem 5.10 (First Order Necessary Conditions)**

Consider a vector $z^* \in \mathbb{R}^n$ and a function $F : \mathbb{R}^n \to \mathbb{R}$ where $F$ is continuously differentiable in an open neighborhood of $z^*$ and $z^* \in \mathbb{R}^n$ is a local minimizer of $F$. Then we have $\nabla F(z^*) = 0$.

In a similar manner, information on the Hessian can be used to derive second order necessary conditions from equation (5.12).

**Theorem 5.11 (Second Order Necessary Conditions)**

Consider a vector $z^* \in \mathbb{R}^n$ and a function $F : \mathbb{R}^n \to \mathbb{R}$ where $F$ is twice continuously differentiable in an open neighborhood of $z^*$ and $z^* \in \mathbb{R}^n$ is a local minimizer of $F$. Then we have $\nabla F(z^*) = 0$ and the Hessian $\nabla^2 F(z^*)$ is positive semidefinite.
The results from Theorems 5.10 and 5.11 reveal guidelines to what we are looking for, i.e., which properties a local minimizer must fulfill. However, these results cannot be used to identify a local minimizer once we have found a candidate satisfying the previous conditions. In order to perform such a check, the following theorem can be used.

**Theorem 5.12 (Second Order Sufficient Conditions)**

Consider a vector \( z^* \in \mathbb{R}^n \) and a function \( F : \mathbb{R}^n \rightarrow \mathbb{R} \) where \( F \) is twice continuously differentiable in an open neighborhood of \( z^* \). If \( \nabla F(z^*) = 0 \) and \( \nabla^2 F(z^*) \) is positive definite, then \( z^* \) is a local minimizer of \( F \).

Before we consider constraints we give a short description of the standard strategies for nonlinear optimization, the line-search and the trust-region strategy. Both methods have in common that they approximately compute local minimizers by iteratively computing values \( z_k \) converging to \( z^* \). Hence, an initial guess \( z_0 \) needs to be supplied by the user for starting the iteration. A good initial guess, i.e., a vector close to a minimizer, can usually only be obtained by utilizing knowledge on the process. If such knowledge is not at hand, the starting point can be chosen arbitrarily, however, the convergence speed of the sequence \( z_k \) towards a minimizer is drastically reduced in general.

Within the line-search strategy, the approximation method computes a direction \( d_k \) in its \( k \)-th step and searches along the vector \( d_k \) starting from the current iterate \( z_k \) for a new iterate \( z_{k+1} = z_k + \alpha_k d_k \) with lower cost function value \( F(z_{k+1}) \). Here the direction \( d_k \) is typically obtained from minimizing a model function \( m_k \) which catches the local behavior of the cost function \( F \) at the current iterate \( z_k \) and is easy to minimize numerically. Often, quadratic functions of the form

\[
m_k(z_k + d_k) = F(z_k) + d_k^T \nabla F(z_k) + \frac{1}{2} d_k^T B_k d_k
\]

are used for this purpose, where \( B_k \) is either the Hessian \( \nabla^2 F(z_k) \) or an approximation of it. The corresponding step length \( \alpha_k > 0 \) can then, e.g., be determined by solving the one-dimensional minimization problem

\[
\min_{\alpha_k > 0} F(z_k + \alpha_k d_k). \tag{5.13}
\]

Once an (approximated) solution to the step length problem has been found, a new search direction and a new step length are computed and the scheme is applied iteratively.

In contrast to the line-search approach, the trust-region method takes the only local approximation properties of the model function \( m_k \) into account when minimizing this function in order to determine \( d_k \). Since \( m_k \) can only be guaranteed to be a good approximation close to \( z_k \) — i.e., it can only be “trusted” in a neighborhood of \( z_k \) — the search region for a minimizer of \( m_k \) is restricted to a so called trust-region which is usually given by a ball \( B_\Delta(z_k) \). Hence, the problem consists in computing a suitable next iteration candidate by solving

\[
\min_{d_k} m_k(z_k + d_k) \text{ where } z_k + d_k \in B_\Delta(z_k). \tag{5.14}
\]

If the candidate \( z_k + d_k \) does not show a sufficient decrease in the cost function \( F \), then the trust-region is considered to be too large. Hence, the radius \( \Delta \) is reduced and the new minimization problem (5.14) is solved again. Like in the line search approach, the model function \( m_k \) in (5.14) is often generated by quadratic functions.
5.3 Optimization Methods

Remark 5.13
Taking an abstract look on both line-search and trust-region method, the difference between those two approaches lies in the ordering of the basic steps, i.e. finding a search direction and a suitable step length. While the line-search method fixes the search direction first and then computes a step length $\alpha_k$, the trust-region method first defines the maximal step length $\Delta$ and then searches for a minimizer using the model $m_k$.

5.3.2 Constrained Optimization

So far we have dealt with unconstrained optimization problems and shown the fundamental results and basic algorithmic ideas which can be used to solve such problems. In the NMPC algorithm, however, we face the constrained nonlinear optimization problem

$$
\begin{align*}
\text{minimize} & \quad F(z) \\
\text{with respect to} & \quad z \in \mathbb{R}^{n_z} \\
\text{subject to} & \quad G_i(z) = 0 \text{ for all } i \in G \text{ and } H_i(z) \geq 0 \text{ for all } i \in I
\end{align*}
$$

(NLP)

in every step of the NMPC iteration where the functions $F$, $G$ and $H$ are defined by one of the discretizations of the problem (OCP). Note that all these discretizations lead to a problem of type (NLP), hence all subsequent results hold for either of these discretizations.

The index sets in (NLP) are given by $E = \{1, \ldots, n_G\}$ and $I = \{n_G + 1, \ldots, n_G + n_H\}$, respectively, and the functions $G_i$ and $H_i$ are called equality and inequality constraints, respectively. These constraints induce the following feasible set which will be important for our upcoming analysis.

Definition 5.14 (Feasible Set)
For a problem (NLP) the set

$$
\Omega = \{z \mid G_i(z) = 0, \ i \in E; \ H_i(z) \geq 0, \ i \in I\}
$$

(5.15)

is called the feasible set and the elements $z \in \Omega$ are called feasible points.

Since a minimizer for the problem (NLP) has to be an element of $\Omega$ by definition, we have to modify the definition of a local minimizer in the context of constrained optimization problems which we want to approximate later:

Definition 5.15 (Local Minimizer)
A point $z^* \in \mathbb{R}^{n_z}$ is a local minimizer of the problem (NLP) if there exists a neighborhood $\mathcal{N}$ of $z^*$ such that $F(z^*) \leq F(z)$ holds for all $z \in \mathcal{N} \cap \Omega$.

In a similar way as for unconstrained optimization problems, we now want to derive necessary and sufficient conditions which will allow us to construct numerical methods to compute a local minimizer $z^*$ of a problem (NLP). The mathematical background of the necessary and sufficient conditions given in Theorems 5.10, 5.11 and 5.12 is Taylor’s Theorem 5.9 stating results for a linear or quadratic approximation. In constrained optimization, the functions $G$ and $H$ will now also be replaced by suitable approximations. Here we use linear approximations

$$
G(z + d) \approx G(z) + \nabla G(z)^T d \quad \text{and} \quad H(z + d) \approx H(z) + \nabla H(z)^T d
$$

for this purpose. This, however, only makes sense if the geometry of the feasible set $\Omega$ is — at least locally — reflected properly when $G$ and $H$ are replaced by approximations. To this end so called constraint qualifications are imposed which we are going to deal with next.

Before we state the popular linear independent constraint qualification (LICQ), we need some definitions. We first introduce the tangent cone $T_\Omega(z)$ to the feasible set $\Omega$.

**Definition 5.16 (Tangent Cone)**
A vector $v \in \mathbb{R}^n$ is called *tangent vector* to $\Omega$ at a point $z \in \Omega$ if there exists a sequence of feasible points $(z_k)_{k \in \mathbb{N}}$ with $z_k \to z$, $z_k \in \Omega$ and a sequence of positive scalars $(t_k)_{k \in \mathbb{N}}$ with $t_k \to 0$ such that

$$\lim_{k \to \infty} \frac{z_k - z}{t_k} = v$$

(5.16)

holds. The set of all tangent vectors to $\Omega$ at $z$ is called the *tangent cone* and is denoted by $T_\Omega(z)$.

Note that $T_\Omega$ only depends on the geometry of $\Omega$. The set $T_\Omega(z)$ can be seen as a local approximation of all feasible directions at a given feasible point $z \in \Omega$. The feasible directions are all vectors $d \in \mathbb{R}^n$ for which $z + \alpha d \in \Omega$ holds for all sufficiently small $\alpha > 0$ and the definition of $T_\Omega$ implies that each feasible direction is contained in $T_\Omega(z)$. Conversely, for each element $v \in T_\Omega(z)$ and each $\epsilon > 0$ there exists a feasible direction $d$ with $\|d - v\| < \epsilon$.

Obviously, all equality constraints $G_i$ restrict these feasible directions but not necessarily all inequality constraints: if $H_i(z) > 0$ holds, then since $H_i$ is continuous we get $H_i(z + \alpha d) > 0$ for all $d \in \mathbb{R}^n$ provided $\alpha > 0$ is sufficiently small. If, however, $H_i(z) = 0$ holds, then an arbitrarily small change of $z$ in the “wrong” direction may lead to $H_i(z + \alpha d) < 0$. Hence, all inequality constraints $H_i$ with $H_i(z) = 0$ and all equality constraints $G_i$ may restrict feasible moving directions. These constraints are called *active* and their indices are characterized by the following definition.

**Definition 5.17 (Active Set)**
The active set $A(z)$ at any feasible point $z \in \Omega$ consists of the equality constraint indices from $E$ together with the indices of the inequality constraints $i \in I$ where $H_i(z) = 0$ holds, that is $A(z) := E \cup \{i \in I \mid H_i(z) = 0\}$.

**Definition 5.18 (Active Constraints)**
Consider the active set $A(z)$ of a feasible point $z \in \Omega$. Then we call

$$A(z) := \left( (G_i)_{i \in E} \quad (H_i)_{i \in A(z) \cap I} \right)$$

(5.17)

the set or vector of active constraints and $n_A = \sharp A(z)$ the number or dimension of active constraints at $z$. Moreover, we denote the corresponding Lagrange multiplier vector by $\lambda^A$.

Using the active set we can now define a set of “linearized” feasible directions obtained from the linearizations of $H$. 

5.3 Optimization Methods

Definition 5.19 (Linearized Feasible Directions)
For a feasible point \( z \in \Omega \) and the active set \( \mathcal{A}(z) \) we call the set
\[
\mathcal{F}(z) = \left\{ v \in \mathbb{R}^n \left| \begin{array}{l}
v^\top \nabla G_i(z) = 0 \text{ for all } i \in \mathcal{E} \\
v^\top \nabla H_i(z) \geq 0 \text{ for all } i \in \mathcal{A}(z) \cap \mathcal{I}
\end{array} \right. \right\}
\] (5.18)
the set (or cone) of linearized feasible directions.

Since \( T_\Omega(z) \subseteq \mathcal{F}(z) \), it is important to have that these sets coincide for the proof of necessary optimality conditions based on the linearizations of the \( G_i \) and \( H_i \) as well as for using the linearized \( G_i \) and \( H_i \) in our algorithms. The idea of constraint qualifications is to guarantee that these sets indeed coincide, i.e., that the geometry of \( T_\Omega \) is captured by the linearizations of \( G_i \) and \( H_i \). The linear independence constraint qualification is probably the most popular one.

Definition 5.20 (LICQ)
Consider a feasible point \( z \) and the active set \( \mathcal{A}(z) \). Suppose that \( F, G \) and \( H \) are continuously differentiable. If the elements of the gradient set \( \{ \nabla G_i(z) \mid i \in \mathcal{E} \} \cup \{ \nabla H_i(z) \mid i \in \mathcal{A}(z) \cap \mathcal{I} \} \) are linearly independent then we say that the linear independence constraint qualification (LICQ) holds.

Under this condition we obtain \( T_\Omega(z) = \mathcal{F}(z) \), see [5, Lemma 9.2.1].

Now we want to proceed as in the unconstrained case, i.e., give characterizations of minimizers of the cost function \( F \) amongst the feasible points \( z \in \Omega \). Note that in the constrained case we cannot simply use Taylor’s Theorem 5.9 to conclude that if \( z^* \in \Omega \) is a local minimizer for the problem (NLP) then we have \( \nabla F(z^*) = 0 \).

Example 5.21
In order to see that Taylor’s Theorem 5.9 cannot be used in the constrained case, consider the example of minimizing \( F(z) = z \) over \( \Omega = [-1, 1] \). Obviously \( z = -1 \) is a local minimizer, yet we have \( \nabla F(-1) = 1 \).

The problem with applying Theorem 5.10 in the constrained case is that no boundaries of the constraints sets are included in the analysis in this theorem. In constrained optimization, however, we often face the situation of a minimizer lying on the boundary of the feasible set \( \Omega \). To deal with this matter, the following auxiliary function \( L : \mathbb{R}^n \times \mathbb{R}^{n_G+n_H} \rightarrow \mathbb{R} \), the so called Lagrangian is introduced. For its definition, we combine the constraints \( G_i \) and \( H_i \) into one function \( C : \mathbb{R}^n \rightarrow \mathbb{R}^{n_G+n_H} \) given by
\[
C : z \mapsto \left[ \begin{array}{c}
(G_i(z))_{i \in \mathcal{E}} \\
(H_i(z))_{i \in \mathcal{I}}
\end{array} \right]
\]
and define as a modification of the cost function \( F \) by
\[
L(z, \lambda) := F(z) - \lambda^\top C(z).
\] (5.19)
The idea behind this definition is that the additional term \( -\lambda^\top C(z) \) penalizes violations of the state constraints. The vector \( \lambda \in \mathbb{R}^{n_G+n_H} \) is called Lagrange multiplier.
Similar to Theorem 5.10, we can now state a first order necessary optimality condition — usually called KKT (Karush–Kuhn—Tucker) condition — in the constrained case using the Lagrangian (5.19) which will serve as a guideline to find local minimizers, see [5, Theorem 9.1.1].

**Theorem 5.22** (KKT Conditions)
Consider the problem (NLP) with local minimizer $z^* \in \Omega$. Moreover suppose the functions $F$, $C$ and $H$ to be continuously differentiable and the (LICQ) to hold at $z^*$. Then there exists a Lagrange multiplier $\lambda^* \in \mathbb{R}^{n_G+n_H}$ such that the following conditions hold.

$$\nabla_z L(z^*, \lambda^*) = 0 \quad (5.20)$$
$$G_i(z^*) = 0 \quad \forall i \in \mathcal{E} \quad (5.21)$$
$$H_i(z^*) \geq 0 \quad \forall i \in \mathcal{I} \quad (5.22)$$
$$\lambda_i^* \geq 0 \quad \forall i \in \mathcal{I} \quad (5.23)$$
$$\lambda_i^* G_i(z^*) = 0 \quad \forall i \in \mathcal{E} \quad (5.24)$$
$$\lambda_i^* H_i(z^*) = 0 \quad \forall i \in \mathcal{I}. \quad (5.25)$$

The identity (5.25) is a so called strict complementarity condition which says that either $\lambda_i^* = 0$ or $H_i(z^*) = 0$ must hold. A special case which is important for nonlinear optimization algorithms is the following.

**Definition 5.23**
Consider the problem (NLP) with local minimizer $z^* \in \Omega$ and Lagrange multiplier $\lambda^* \in \mathbb{R}^{n_G+n_H}$ satisfying (5.20) - (5.25). Then we say that the strict complementarity condition holds if $\lambda_i^* > 0$ for all $i \in \mathcal{I} \cap \mathcal{A}(z^*)$.

Interpreting the KKT conditions we see that they connect the gradient of the cost function to active constraints. In particular, Theorem 5.22 states that for a given minimizer $z^*$ moving along an arbitrary vector $v \in \mathcal{F}(z^*)$ either increases the value of the first order approximation of the cost function, i.e. $v^\top \nabla F(z^*) > 0$, or keeps its value at the same level in the case $v^\top \nabla F(z^*) = 0$.

In the second case — the so called “critical” case $v^\top \nabla F(z^*) = 0$ — it is unknown if the cost function value is increasing or decreasing along $v$. Here second order conditions come into play and the curvature information can be used to obtain more information about change of $F$ along these directions, see [5, Theorem 9.3.1] for a corresponding proof.

**Theorem 5.24** (Second Order Necessary Conditions)
Consider the problem (NLP) with local minimizer $z^* \in \Omega$. Suppose the functions $F$, $G$ and $H$ to be continuously differentiable and the (LICQ) to hold at $z^*$. Let $\lambda^* \in \mathbb{R}^{n_G+n_H}$ be a Lagrange multiplier satisfying the KKT conditions (5.20)–(5.25). Then the inequality

$$v^\top \nabla^2_{zz} L(z^*, \lambda^*) v \geq 0 \quad (5.26)$$

holds for all

$$v \in \mathcal{C}(z^*, \lambda^*) := \left\{ v \in \mathcal{F}(z^*) \mid v^\top \nabla H_i(z^*) = 0 \text{ for all } i \in \mathcal{A}(z^*) \cap \mathcal{I} \text{ with } \lambda_i^* > 0 \right\}. \quad (5.27)$$
The set $C$ is also called the critical cone. It contains all directions which leave the active inequality constraints with $\lambda_i > 0$ as well as all equality constraints active if one moves a sufficiently small step along these directions. This, however, does not need to hold for those active inequality constraints with $\lambda_i = 0$. In particular, we have the equivalence
\[
v \in C(z^*, \lambda^*) \iff \begin{cases} 
\nabla G_i(z^*)^\top v = 0, & \text{for all } i \in E, \\
\nabla H_i(z^*)^\top v = 0, & \text{for all } i \in A(z^*) \cap I \text{ with } \lambda_i^* > 0, \\
\nabla H_i(z^*)^\top v \geq 0, & \text{for all } i \in A(z^*) \cap I \text{ with } \lambda_i^* = 0.
\end{cases}
\]

Now, we want to get a converse result, i.e. we want to check whether a given feasible point is actually a local minimizer. As it turns out, the only differences between the previous necessary conditions and the sufficient conditions presented next is that the constraint qualification is not required whereas inequality (5.26) needs to be strengthened to a strict inequality, cf. [5, Theorem 9.3.2]:

**Theorem 5.25** (Second Order Sufficient Conditions)
Consider a feasible point $z^* \in \Omega$ and suppose a Lagrange multiplier $\lambda^* \in \mathbb{R}^{n_G + n_H}$ to exist satisfying (5.20) – (5.25). If we have
\[
v^\top \nabla^2_{zz} L(z^*, \lambda^*) v > 0 \quad (5.28)
\]
for all $v \in C(z^*, \lambda^*)$ with $v \neq 0$, then $z^*$ is a strict local minimizer of problem (NLP).

We will now focus on the currently common approaches to solve nonlinear constrained optimization problems (NLP), these are the so called Sequential Quadratic Programming (SQP) based on the active set $A$ and the Interior-Point Method (IP). In the remainder of this section we describe the main ideas of these strategies, cf. [5,14] for details.

### 5.3.3 Sequential Quadratic Programming

The sequential quadratic programming approach (SQP) introduces a so called working set $W_k$ of the current operating point $z_k$. This working set contains all indexes of constraints which are currently active, that is all equality constraints $i \in E^S$ and all inequality constraints $i \in I^S$ satisfying equality. For the working set $W_k$, the constraints are linearized and the cost functional is approximated using a second order Taylor approximation of the Lagrangian, which reveals

\[\begin{align*}
\text{Minimize} & \quad F(z_k) + \nabla F(z_k)^\top d_k + \frac{1}{2} d_k^\top \nabla^2_{zz} L(z_k, \lambda_k^W) d_k \\
\text{with respect to} & \quad d_k \in \mathbb{R}^{n_z} \\
\text{subject to} & \quad C_i(z_k) + \nabla C_i(z_k)^\top d_k = 0 \quad \text{for all } i \in W_k
\end{align*}\]

where $L$ denotes the Lagrangian of the problem (NLP) and $d_k$ the search direction step. We can solve this problem by computing the solution of the linear equation
\[
\begin{bmatrix}
\nabla^2_{zz} L(z_k, \lambda_k^W) & -\nabla C^W(z_k)^\top \\
\nabla C^W(z_k) & 0
\end{bmatrix}
\begin{bmatrix}
d_k \\
\lambda_k^W
\end{bmatrix} + 
\begin{bmatrix}
\nabla F(z_k) \\
C^W(z_k)
\end{bmatrix} = 0.
\]

The next iterate is then given by
\[z_{k+1} = z_k + d_k.\]

At each iterate, the working set is updated and a new search direction step is computed until the first order optimality conditions are satisfied sufficiently well.
Algorithm 5.26 (Basic Active Set (SQP) Algorithm)
Suppose a pair of initial values \((z_0, \lambda_0)\) and an initial working set \(W_0 \subseteq A(z_0)\) to be given and set \(k := 0\).
While convergence test is not satisfied do
1. Compute \(F(z_k), \nabla F(z_k), \nabla^2_{zz} L(z_k, \lambda_k), C(z_k)\) and \(\nabla C(z_k)\)
2. Solve equality constrained problem with \(W_k\) and obtain \(d_k, \lambda_k\) and \(W_{k+1}\)
3. Set \(z_{k+1} := z_k + d_k, \lambda_{k+1} := \lambda_k\)
4. Set \(k := k + 1\)

Remark 5.27
Due to its compactness, the (SQP) approach fits to the recursive discretization.

5.3.4 Interior Point Method
In contrast to the (SQP) approach, the interior point method (IP) introduces so called slack variables into the (NLP). The idea of these slacks is to convert inequality constraints – which are difficult to handle numerically – into equality constraints by adding new optimization variables:

Minimize \(F(z)\)
with respect to \(z \in \mathbb{R}^n, s \in \mathbb{R}^n\)
subject to \(G_i(z) = 0\) for all \(i \in \mathcal{E}\),
\(H_i(z) - s_i = 0\) for all \(i \in \mathcal{I}\) and \(s_i \geq 0\)
where \(s_i\) denotes the slack of the inequality constraint \(i \in \mathcal{I}\). Now, the first order necessary conditions for optimality reveal the equation system

\[
\begin{bmatrix}
\nabla^2_{zz} L(z, s, v, w) & 0 & -\nabla G(z)^\top & -\nabla H(z)^\top \\
0 & W & 0 & S \\
\n\nabla G(z) & 0 & 0 & 0 \\
\n\nabla H(z) & -Id & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
d_z \\
d_s \\
d_v \\
d_w \\
\end{bmatrix}
= -
\begin{bmatrix}
\nabla F(z) - \nabla G(z)^\top v - \nabla H(z)^\top w \\
Sw - \mu e \\
G(z) \\
H(z) - s \\
\end{bmatrix}
\]

with \(s \geq 0, w \geq 0\) and \(\mu = 0\), using the vector \(e := (1, 1, \ldots, 1)^\top\) and the matrices \(S := \text{diag}(s)\) and \(W := \text{diag}(w)\) to simplify the notation and \(\mu > 0\) is introduced to enforce the solution stay away from the boundary. Now, we can apply Newton’s method to compute the root via

and obtained the next iterate by

\[
\begin{align*}
z_{k+1} &= z_k + \alpha_{s}^{\max} d_s \\
v_{k+1} &= v_k + \alpha_{w}^{\max} d_v \\
s_{k+1} &= s_k + \alpha_{s}^{\max} d_s \\
w_{k+1} &= w_k + \alpha_{w}^{\max} d_w
\end{align*}
\]

where we set with \(\tau \in (0, 1)\) (typically 0.995)

\[
\begin{align*}
\alpha_{s}^{\max} &= \max \{ \alpha \in (0, 1] \mid s + \alpha d_s \geq (1 - \tau)s \} \\
\alpha_{w}^{\max} &= \max \{ \alpha \in (0, 1] \mid w + \alpha d_w \geq (1 - \tau)w \}
\end{align*}
\]
Imposing the error function

\[ E(z_k, s_k, v_k, w_k; \mu_k) = \max \{ \| \nabla F(z) - \nabla G(z)^\top v - \nabla H(z)^\top w \|, \| Sw - \mu e \|, \| G(z) \|, \| H(z) - s \| \} \]

we obtain the following basic algorithm

**Algorithm 5.28** (Basic Interior-Point Algorithm)
Suppose a pair of initial values \((z_0, s_0)\) to be given and set \(k := 0\).
Compute multipliers \(v_0\) and \(w_0\), define parameters \(\mu_0 > 0, \sigma, \tau \in (0, 1)\)
While convergence test not satisfied

1. While \(E(z_k, s_k, v_k, w_k; \mu_k) \geq \mu_k\)
   
   (a) Compute search direction \(d = (d_z, d_s, d_v, d_w)\)
   
   (b) Determine \(\alpha_{\max}^s, \alpha_{\max}^w\)
   
   (c) Obtain new iterate \((z_{k+1}, s_{k+1}, v_{k+1}, w_{k+1})\)
   
   (d) Set \(\mu_{k+1} := \mu_k\) and \(k := k + 1\)

2. Choose \(\mu_k \in (0, \sigma \mu_k)\)

**Remark 5.29**

*Due to its structure, the (IP) approach is applied to a full discretization.*
Chapter 6

MPC Algorithm and Results

In the previous chapters we discussed all the building blocks required for the MPC algorithm. As already outlined in the introductory Chapter 1, the idea of MPC is a three step schemes that is to be performed at each sampling instant $n$:

1. Obtain the current state of the system $x_0$

2. Compute an optimal control $u^*$ by predicting the future behavior of the system over a finite time horizon $k = 0, \ldots, N - 1$ of length $N \geq 2$, and

3. Use the first element of the resulting optimal control sequence as a feedback control value $\mu_N(x_0) := u^*(0)$ for the next sampling interval.

The purpose of the present chapter is twofold: For one, we discuss the construction of a basic MPC algorithm and the interplay of the building blocks. And secondly, we take a closer look at properties, which we like to induce in the resulting closed loop solution

$$x(n + 1) = f(x(n), \mu_N(x(n))).$$

(6.1)

resulting from the scheme above. Hence, the first “construction part” in Section 6.1 relies on the simulation, discretization and optimization methods discussed in Chapters 4 and 5. Sections 6.2 and 6.3 focus on properties of solutions such as feasibility and stability which were introduced in Chapter 2.

6.1 Basic MPC Algorithm

The aim of this section is to introduce a basic MPC algorithm using the building blocks from the previous chapters. To this end, we restrict ourselves to the case of stabilizing a constant reference $x^{ref} \equiv x^* \in \mathbb{X}$. More complex schemes with time varying references are possible and can be integrated easily into the basic MPC algorithm. For simplicity of exposition, especially regarding the forthcoming property analysis, we refer to [7,17] for details on these cases.

Note that it is a requirement that $x_*$ is an equilibrium of the closed loop (6.1), that is

$$x_* = f(x_*, \mu(x_*)),$$

in order to find a feedback law which stabilizes the system at $x_*$. In turn, a necessary condition for the latter is the existence of a control $u_* \in \mathbb{U}$ with

$$x_* = f(x_*, u_*).$$
which we assume to hold.

Regarding the cost function, we require that if we are in the equilibrium \( x_\ast \) and use the control value \( u_\ast \) in order to stay in the equilibrium, then the cost should be zero. Outside the equilibrium, the cost should be positive, i.e,

\[
\ell(x, u) = 0 \quad \text{if and only if} \quad x = x_\ast \text{ and } u = u_\ast \\
\ell(x, u) > 0 \quad \text{if } x \neq x_\ast \text{ or } u \neq u_\ast.
\]  

(6.2)

The background for this requirement is connected to the energy concept of the system, similar to the Control–Lyapunov functions from Definition 2.26. If the cost (or energy) of any point in the state space was less than zero, then an optimal solution would be to steer the system into that point (given that the system can be steered there). Hence, there exists a connection between

<table>
<thead>
<tr>
<th>Goal:</th>
<th>Design:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stabilize the system</td>
<td>Cost functional</td>
</tr>
</tbody>
</table>

If our system is defined on vector spaces, that is, e.g., \( \mathcal{X} = \mathbb{R}^n_x \) and \( \mathcal{U} = \mathbb{R}^n_u \), then we may always assume \( x_\ast = 0 \) and \( u_\ast = 0 \) without loss of generality: if this is not the case we can replace \( f(x, u) \) by \( f(x + x_\ast, u + u_\ast) - x_\ast \), which corresponds to a simple linear coordinate transformation on \( \mathcal{X} \) and \( \mathcal{U} \). In this case, a popular choice for \( \ell \) meeting Condition (6.2) is the quadratic function

\[
\ell(x, u) = \|x\|^2 + \lambda\|u\|^2
\]

with the usual Euclidean norms and a weighting parameter \( \lambda \geq 0 \). In the metric space setting with metrics \( d_{\mathcal{X}}, d_{\mathcal{U}} \), we can set

\[
\ell(x, u) = d_{\mathcal{X}}(x) + \lambda d_{\mathcal{U}}(u)
\]

Other choices of \( \ell \) are possible and often reasonable.

Given such a cost function \( \ell \) and a prediction horizon length \( N \geq 2 \), we can now formulate the basic MPC algorithm:

**Algorithm 6.1** (Basic MPC Algorithm)

At each sampling time \( t_n, n = 0, 1, 2 \ldots \):

1. Measure the state \( x(n) \in \mathcal{X} \) of the system
2. Set \( x_0 := x(n) \), solve the optimal control problem

\[
\begin{align*}
\text{Minimize} & \quad J_N(x_0, u) := \sum_{k=0}^{N-1} \ell(x_u(k), u(k)) + L(x_u(N, x_0)) \\
\text{with respect to} & \quad u(\cdot) \in U^N_{\mathcal{X} \mathcal{U}}(x_0), \quad \text{subject to} \\
x_u(0, x_0) = x_0 \in \mathcal{X}^0, & \quad x_u(k+1, x_0) = f(x_u(k, x_0), u(k))
\end{align*}
\]  

(OCP)

and denote the obtained optimal control sequence by \( u^*(\cdot) \in U^N_{\mathcal{X} \mathcal{U}}(x_0) \).
3. Define the MPC feedback value \( \mu_N(x(n)) := u^*(0) \in \mathcal{U} \) and use this control value in the next sampling period.
Note that in Algorithm 6.1 we assumed that an optimal control sequence \( u^*(\cdot) \) to exists, cf. [10] for sufficient conditions to guarantee existence.

During the theoretical analysis in the next sections, we neglect the fact that solving (OCP) in Step (2) of the algorithm usually requires some computation time. If the computing time is large compared to the sampling time \( T \), then this fact is not negligible in a real life application. To circumvent this problem, we refer to the advanced step design given in [7].

**Remark 6.2**

In Algorithm 6.1 we apply only the first element of the minimizing control sequence and do not use the remaining values. In an implementation, this data can be utilized for a warm start of the numerical solution schemes from the previous Chapter 5.

From Algorithm 6.1, we see that the problem (OCP) is at the core of the method. From the previous Chapter 5 we know that such a problem can be solved by discretization and nonlinear programming, which gave us the problem (NLP). Within the discretization part, we required the solution of the control system dynamics, which we discussed in Chapter 4. Depending on the chosen discretization, the simulation part is either fully integrated or performed outside the problem (NLP). This gives us the following tree:

![Figure 6.1: Building blocks within the MPC Algorithm 6.1](image)

### 6.2 Feasibility

One of the main reasons for the success of MPC is its ability to explicitly account for constraints. We introduced these constraints in Definition 2.20 via sets \( X \subset \mathcal{X} \) and \( U \subset \mathcal{U} \). The property that the trajectory lies within \( X \) for all \( n \in \mathbb{N}_0 \) is called feasibility, and the property that the control corresponding to a state lies within \( U \) is called admissibility, cf. Section 2.3.1.

Within optimization, the constraint sets are typically compact and convex. For the analysis, the so called viability (or controlled forward invariance) property is sufficient:
Assumption 6.3 (Viability)
For each $x \in X$ there exists $u \in U$ such that $f(x, u) \in X$ holds.

The property excludes the situation that there are states $x \in X$ from which the trajectory leaves the set $X$ for all admissible control values. Therefore, $U_{X_N}^N(x) \neq \emptyset$ is guaranteed for all $x_0 \in X$ and all $N \in \mathbb{N}_\infty$. We will use this property to ensure the feasibility of (OCP):

Definition 6.4 (Feasibility of (OCP))
An optimal control problem (OCP) is called feasible for an initial values $x_0$ if the set $U_{X_N}^N(x_0)$ is non empty.

Note that in the previous Chapter 5, we allowed the case that there exists no solution for (OCP). Utilizing Assumption 6.3, we can conclude the following:

Corollary 6.5 (Feasibility of (OCP))
Suppose Assumption 6.3 holds for a given (OCP). Then

- (OCP) is feasible for all $x_0 \in X$,
- the feedback $\mu_N(x_0)$ is well defined for each $x_0 \in X$, and
- any finite admissible control sequence $u \in U_{X_N}^N(x_0)$ can be extended to an infinite one.

Remark 6.6
In the presence of stabilizing terminal constraints, Assumption 6.3 can be dropped. This is due to the backwards construction of the feasible set, which in the presence of terminal constraints will not allow for infeasible initial states.

The following theorem shows that the viability assumption ensures that the MPC closed loop (6.1) satisfies the constraints:

Theorem 6.7 (Admissibility)
Consider Algorithm 6.1 using $U_{X_N}^N(x_0)$ from Definition 2.21 in problem (OCP). Moreover, suppose Assumption 6.3 to hold and $x_0 \in X$. Then we have

$$x(n) \in X \quad \text{and} \quad \mu_N(x(n)) \in U$$

for all $n \in \mathbb{N}$, i.e. the constraints are satisfied along the solution of (6.1). Thus, the MPC feedback $\mu_N$ is admissible.

Note that Theorem 6.7 implies the following:

Corollary 6.8 (Feasibility of MPC)
If a state $x(n)$ is feasible for (OCP), then the closed loop successor state $x(n + 1) = f(x(n), \mu_N(x(n)))$ is again feasible.
Remark 6.9
In the case of sampled data systems, the constraints are only defined for the sampling times $t_n$, but not for the inter-sampling times $t \neq t_n$. To incorporate this shortcoming, one can use the continuity of the trajectory to tighten the constraints.

6.3 Stability

Next to feasibility, the property of (asymptotic) stability as given in Definition 2.23 is desirable for the closed loop system. The case of robust (asymptotic) stability can be handled pretty easily:

**Theorem 6.10** (Stability of Robust Asymptotic Stable Equilibria)
Consider a robust asymptotically stable equilibrium $x_* \in \mathbb{X}$ of a control system (2.1) and the viability Assumption 6.3 to hold. Then the (MPC) control law $\mu_N : \mathbb{X} \to \mathbb{U}$ is asymptotically stabilizing $x_*$.  

**Proof.** The proof of this theorem is straightforward: Due to viability, each (OCP) is feasible according to Corollary 6.5. By Corollary 6.8, also the (MPC) problem is feasible which gives us an infinite feasible solution trajectory. Using the robust asymptotic stability property from Definition 2.23 and $x_* \in \mathbb{X}$, this solution converges towards the equilibrium independently of the choice of the actual control strategy.

In contrast to that, the case of weakly asymptotic stable equilibria is much more involved. The reason for this lies in the fact that now we require a specific solution inducing the stability property, and cannot rely on that particular property to hold for the system anyway.

In principle, there are three different ideas (and combinations of those) known in the literature. The first of these ideas utilizes on the one hand the finite horizon geometry property of the computation of the (MPC) law in each step, and on the other hand the principle of optimality together with the an imprinted feasibility property. The basic idea of Keerthi and Gilbert [10] is to assume a terminal point condition, that is the solution of each (MPC) step must terminate in the equilibrium, i.e. $x_u(N,x(n)) = x_*$.  

**Theorem 6.11** Stability with Terminal Point Condition Consider a control system (2.1) with equilibrium $x_* \in \mathbb{X}$. Suppose the condition $x_u(N,x(n)) = x_*$ (i.e. $\mathbb{X}_0 = \{x_*\}$) to be included in the (OCP) problem to be solved in each step $n \in \mathbb{N}_0$ of the (MPC) Algorithm 6.1 and the initial value to be an element of the feasible set $x_0 \in \mathbb{X}_N$. Then the (MPC) control law $\mu_N : \mathbb{X} \to \mathbb{U}$ is feasible and asymptotically stabilizing $x_*$ for each $x_0 \in \mathbb{X}_N$.

**Proof.** The idea of the proof is illustrated in Figure 6.2 and can best be explained geometrically. Utilizing the assumption $x_0 \in \mathbb{X}_N$ and condition $x_u(N,x(n)) = x_* \in \mathbb{X}_0$, the exists a feasible solution and an admissible control $u$ to problem (OCP). Since $x_*$ is an equilibrium, there exists a control $u^* \in \mathbb{U}$ such that $f(x_*, u^*) = x_*$. Consequently, if we apply the (MPC) procedure from Algorithm 6.1 and shift the horizon, the control sequence $\hat{u} = (u(1), \ldots, u(N-1), u^*)$ is admissible for the new initial point $x(n+1) = x_u(1, x(n))$, and hence a feasible solution exists rendering the set of feasible solutions to be non empty. Optimizing over this set, we again...
obtain a feasible solution and an admissible control to problem (OCP) of the next (MPC) iteration. Hence, the (MPC) feedback law $\mu_N$ is feasible.

Last, we utilize the geometry of the cost function $\ell$, which exhibits a minimum at $x_*$, cf. (6.2). Hence, if there exists no control with lower costs than the extended control $\hat{u}$, then the state of the system arrives at the equilibrium after $N$ steps of the (MPC) procedure and will remain there. If there exists a control with lower costs than $\hat{u}$, then by principle of optimality and the construction of the costs $\ell$, the state of the system converges to $x_*$ for $n \to \infty$. Hence, the (MPC) control law asymptotically stabilizing $x_*$ for each $x_0 \in X^N$.

Even nowadays, the idea of Keerthi and Gilbert widely used for mechanical systems. The reasons for that are on the one hand its additional property of very fast computing times for the (OCP) problems, and the availability of inverse kinematics which allow a construction of the feasible set $X^N$. Still, there is a large drawback: Terminal conditions which aim for single points – just like $x_u(N, x(n+1)) = x_*$ – are difficult to handle numerically. Typically, only a certain terminal region can be reached.

Chen and Allgöwer [3] were able to extend the terminal point condition used in Theorem 6.11 to such a terminal region. To this end, they introduced a terminal cost function in form of a local Lyapunov function, and a respective local feedback law rendering the terminal region to be forward invariant.

**Theorem 6.12** (Stability with Terminal Costs and Constraints)
Consider a control system (2.1) with equilibrium $x_0 \in X$. Suppose terminal costs $L$, the terminal set $X^0$ and a local feedback $\kappa$ to be given such that

- $X^0 \subseteq X$, $X^0$ is closed and $x_* \in X^0$,
- $\kappa(x) \in U$ for all $x \in X^0$,
- $f(x, \kappa(x)) \in X^0$ for all $x \in X^0$, and
- $L(x) \geq L(f(x, \kappa(x))) + \ell(x, \kappa(x))$ for all $x \in X^0$.

Moreover, suppose the condition $x_u(N, x(n)) \in X^0$ is included in the (OCP) problem to be solved in each step $n \in \mathbb{N}_0$ of the (MPC) Algorithm 6.1 and the initial value is an element of the feasible set $x_0 \in X^N$. Then the (MPC) control law $\mu_N : X \to U$ is feasible and asymptotically stabilizing $x_*$ for each $x_0 \in X^N$. 

Proof. Similar to the proof of Theorem 6.11, we can utilize assumption \( x_0 \in X^0 \) and condition \( x_u(N, x(n)) \in X^0 \) to conclude existence of a feasible solution and an admissible control \( u \) to problem (OCP), cf. Figure 6.3. Since \( \kappa \) renders the set \( X^0 \) to be forward invariant, that is \( f(x, \kappa(x)) \in X^0 \) for all \( x \in X^0 \), we can generate an extended control sequence \( \hat{u} := (u(1), \ldots, u(N-1), \kappa(x_u(N, x(n)))) \) similar to the proof of Theorem 6.11. Consequently, the control sequence \( \hat{u} \) is admissible for the new initial point \( x(n+1) = x_u(1, x(n)) \) of the (MPC) procedure after shifting the horizon, and hence a feasible solution exists rendering the set of feasible solutions to be non empty. Following identical arguments as in the proof of Theorem 6.11, the (MPC) feedback law \( \mu_N \) is feasible.

In order to show asymptotic stability, we consider the cost development over time. Connecting the steps \( n \) and \( n+1 \) utilizing the extended control \( \hat{u} := (u^*(1), \ldots, u^*(N-1), \kappa(x_u(N, x(n)))) \), we obtain

\[
J_N(x(n+1), \hat{u}) = J_N(x(n), u^*) - \ell(x_u(0, x(n)), u^*(0)) - L(x_u(N, x(n))) + \ell(x_u(0, x(n+1)), \hat{u}(0)) + L(x_u(N, x(n+1))).
\]

Utilizing condition \( L(x) \geq L(f(x, \kappa(x))) + \ell(x, \kappa(x)) \) now reveals

\[
J_N(x(n+1), \hat{u}) \leq J_N(x(n), u^*) - \ell(x_u(0, x(n)), u^*(0)).
\]

Last, the geometry of the costs \( \ell \) given in (6.2) reveals that

\[
J_N(x(n+1), \hat{u}) < J_N(x(n), u^*)
\]

whenever \( x(n) \neq x^* \), which shows the asymptotic stability property of the (MPC) feedback law. \( \square \)

Figure 6.3: Geometry of the stability idea utilizing terminal costs and regions

The idea of the terminal region, a local Lyapunov function and a respective local feedback is directly connected to a linearization approach. Indeed, for a small neighborhood around the equilibrium \( x^* \), the dynamics can be linearized and a quadratic Lyapunov function can be found for this linearized dynamics. Using Sontag’s formula, a local feedback can be computed directly to stabilize the equilibrium. Hence, these components can be derived rather easily. The remaining problem to solve is the size of the neighborhood that is to be considered. Here, one has to make sure that this set is control forward invariant, i.e. as soon as a solution enters the neighborhood and the local feedback is applied, the trajectory does not leave the set again.
There remains one open issue connected to the transient behavior of the solution between the initial point \(x(0)\) and the neighborhood \(X^0\): During the transition, the system is driven by a control minimizing both the stage costs \(\ell\), the terminal costs \(L\) and the terminal region \(X^0\). Hence, the performance with respect to the original costs \(\ell\) might be arbitrarily bad. Grün and Pannek [6, 7] dealt with this issue using a so called relaxed Lyapunov condition that is supposed to hold for the optimal costs of two consecutive (MPC) steps:

**Theorem 6.13** (Stability without Terminal Costs and Constraints)
Consider a control system (2.1) with equilibrium \(x_* \in \mathbb{X}\) and suppose the viability Assumption 6.3 to hold. Moreover, suppose the relaxed Lyapunov inequality

\[
V_N(f(x, \mu_N(x))) \leq V_N(x) - \alpha \ell(x, \mu_N(x))
\]  

(6.4)

to hold for some \(\alpha \in (0, 1)\) and all \(x \in \mathbb{X}\). Then the (MPC) control law \(\mu_N : \mathbb{X} \to \mathbb{U}\) is feasible and the suboptimality estimate

\[
\alpha V^\mu_N(x) := \alpha \sum_{n=0}^{\infty} \ell(x(n), \mu_N(x(n))) \leq V_N(x) \leq V^\infty(x)
\]  

(6.5)

holds. If additionally there exist \(\alpha_1, \alpha_2, \alpha_3 \in K_\infty\) such that

\[
\alpha_1(\|x - x_*\|) \leq V_N(x) \leq \alpha_2(\|x - x_*\|), \quad \ell(x, \mu_N(x)) \geq \alpha^3(\|x - x_*\|),
\]

then the (MPC) control law \(\mu_N : \mathbb{X} \to \mathbb{U}\) asymptotically stabilizes \(x_*\) for each \(x_0 \in \mathbb{X}^N\) and \(V_N\) is a Lyapunov function for system (2.1).

**Proof.** From Assumption 6.3, we directly obtain feasibility of the (MPC) feedback law via Corollary 6.8. Utilizing the relaxed Lyapunov inequality

\[
V_N(f(x, \mu_N(x))) \leq V_N(x) - \alpha \ell(x, \mu_N(x))
\]

we obtain

\[
\alpha \ell(x, \mu_N(x)) \leq V_N(x) - V_N(f(x, \mu_N(x)))).
\]

Applying the dynamics of the (MPC) Algorithm 6.1, we obtain

\[
\alpha \ell(x(n), \mu_N(x(n))) \leq V_N(x(n)) - V_N(x(n+1)).
\]

to hold along the closed loop solution. Combining two consecutive steps \(n\) and \(n+1\) as sketched in Figure 6.4, we see that

\[
\alpha \ell(x(n), \mu_N(x(n))) + \alpha \ell(x(n+1), \mu_N(x(n+1))) \leq V_N(x(n)) - V_N(x(n+1))
\]

\[
+ V_N(x(n+1)) - V_N(x(n+2)) = V_N(x(n)) - V_N(x(n+2)).
\]

Hence, we can conclude for multiple consecutive steps along Figure 6.4 that

\[
\alpha \sum_{n=0}^{M} \ell(x(n), \mu_N(x(n))) \leq V_N(x(n)) - V_N(x(n + M + 1)).
\]
Since $V_N(x) \geq 0$ for all $x \in X$, we have
\[
\alpha \sum_{n=0}^{M} \ell(x(n), \mu_N(x(n))) \leq V_N(x(n))
\]
and can go to the limit
\[
\alpha V_N^\infty(x) = \alpha \lim_{M \to \infty} \sum_{n=0}^{M} \ell(x(n), \mu_N(x(n))) \leq V_N(x(n))
\]
showing the first half of the suboptimality estimate (6.5). By optimality of $V_N(x)$, we have that $V_N(x) \leq V_\infty(x)$ showing the second half of the estimate.

As a consequence of the relaxed Lyapunov inequality and the existence of $\alpha_3 \in \mathcal{K}_\infty$, we obtain that the energy of the system is decreasing over time and that a function $\alpha \in \mathcal{K}_\infty$ satisfying (2.16) exists. Together with the bounds $\alpha_1, \alpha_3 \in \mathcal{K}_\infty$ we can conclude that $V_N(x)$ is a Lyapunov function of system (2.1). Hence, by Theorem 2.27, the (MPC) feedback law asymptotically stabilizes the equilibrium $x_*$. 

As an addon to the previous results, Theorem 6.13 also allows us to calculate a bound for the performance of the (MPC) law with respect to the best possible controller, that is the (MPC) feedback law for an infinite horizon. On the backside, as one can directly see from the conditions of Theorem 6.13, the relaxed Lyapunov condition does not give us feasibility of the closed loop as the other conditions did. Hence, different methods must be applied to guarantee feasibility, i.e. via our viability Assumption 6.3 or alternative methods, see [7] for details and references.
Bibliography


Glossary

The following table provides an overview of the notation used within the script. Note that auxiliary notations, which are used in examples etc. only, are not displayed here.

<table>
<thead>
<tr>
<th>Functions</th>
<th>Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(\cdot, \cdot)$ Dynamic of the system, p. 11</td>
<td>$\mathbb{R}$ Set of real numbers, p. 10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sets</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$U$ Constrained control set, p. 15</td>
<td>$f(\cdot, \cdot)$ Dynamic of the system, p. 11</td>
</tr>
<tr>
<td>$X$ Constrained state set, p. 15</td>
<td>$\mathbb{R}$ Set of real numbers, p. 10</td>
</tr>
<tr>
<td>$Z$ Set of integer numbers, p. 10</td>
<td>$U$ Control set, p. 10</td>
</tr>
<tr>
<td>$\mathbb{D}_\varphi$ description, p. 10</td>
<td>$U^I$ Set of all control functions $u : I \rightarrow U$, p. 10</td>
</tr>
<tr>
<td>$\mathcal{T}$ Time set, p. 10</td>
<td>$\mathcal{X}$ State set, p. 10</td>
</tr>
<tr>
<td>$\mathcal{U}$ Control set, p. 10</td>
<td>$\alpha(\cdot)$ Class $\mathcal{K}_\infty$ comparison function, p. 19</td>
</tr>
<tr>
<td>$\mathcal{G}$ Set of all class $\mathcal{G}$ functions, p. 18</td>
<td>$\mathcal{G}$ Set of all class $\mathcal{G}$ functions, p. 18</td>
</tr>
<tr>
<td>$\mathcal{K}$ Set of all class $\mathcal{K}$ functions, p. 18</td>
<td>$\mathcal{K}<em>\infty$ Set of all class $\mathcal{K}</em>\infty$ functions, p. 18</td>
</tr>
<tr>
<td>$\mathcal{K}<em>\infty$ Set of all class $\mathcal{K}</em>\infty$ functions, p. 18</td>
<td>$\mathcal{KL}$ Set of all class $\mathcal{KL}$ functions, p. 18</td>
</tr>
<tr>
<td>$\mathcal{KL}$ Set of all class $\mathcal{KL}$ functions, p. 18</td>
<td>$\mathcal{L}$ Set of all class $\mathcal{L}$ functions, p. 18</td>
</tr>
<tr>
<td>$\mathcal{L}$ Set of all class $\mathcal{L}$ functions, p. 18</td>
<td></td>
</tr>
</tbody>
</table>
Variables

\( T \)  Sampling time, p. 13
\( u \)  Control of a system, p. 10
\( u(\cdot) \)  Control function of a system, \( u(\cdot) \in U^I \), p. 10
\( x \)  State of a system, p. 10
\( x_0 \)  Initial value \( x_0 \in X \), p. 12
\( x_T(\cdot, x_0, u) \)  Sampling solution for sampling time \( T \), initial value \( x_0 \) and control function \( u \), p. 13
\( x_u(\cdot, x_0) \)  Trajectory for control \( u(\cdot) \) and initial value \( x_0 \in X \), p. 13
Index

A
Approximation
  local .................................... 42
C
Complementarity condition ................. 44
Concatenation .............................. 11
Constraint
  equality .................................. 41
  inequality .............................. 41
  number of ................................ 37
Control ..................................... 10
  admissible ............................... 10
D
Differential equation solver
  step size control
    algorithm ............................ 30
F
Feasible
  point .................................... 45
M
Minimizer ................................. 40
  candidate .............................. 40
N
Nontriviality .............................. 11
R
Restriction ............................... 11
S
Semigroup ................................. 11
State
  set .................................... 10
  vector ................................. 10

System ................................... 11

T
Taylor
  theorem .................................. 43
Time set .................................. 10
Tolerance
  optimization routine .................... 46
Trajectory ............................... 11
Transition map ........................... 10