Institut für Intermodale
Transport- und
Logistiksysteme

Technische
Universität
Braunschweig

Control Engineering 3 (Regelungstechnik 3)

Lecture Notes

Jürgen Pannek

January 27, 2026

Jürgen Pannek
Institute for Intermodal Transport and Logistic Systems
Hermann-Blenck-Str. 42
38519 Braunschweig

FOREWORD

During winter term 2025/26 I give the lecture to the module *Control Engineering 3 (Regelungstechnik 3)* at the Technical University of Braunschweig. These lecture notes are the 3rd edition, which I prepared to structure the lecture and support my students in their learning process. The notes will be updated to integrate remarks and corrections in due course of the lecture itself.

The aim of the module is to provide participating students with knowledge of advanced control methods, which extend the range of control engineering. After having successfully completed the lecture Modern Control Systems, students are able to define control methods for embedded and networked systems, transfer them to models and applications and apply them. The students can specify and explain the aspects of consistency, stability and robustness as well as areas of application of methods. In addition, they are able to implement the integration of methods in toolchains and apply them to real systems such as vehicles. Students can also reproduce processes of parameter application and automated testing and transfer them to case studies.
To this end, the module will tackle the subject areas

- optimal and robust control as well as
- predictive control

for linear as well as nonlinear systems. In particular, we discuss the methods

- LQR – linear quadratic control,
- H_2 regulator – output feedback control,
- H_∞ regulator – robust control,
- MPC – model predictive control, and
- DCS – distributed control systems.

within the lecture and support understanding and application within the tutorial classes. The module itself is accredited with 5 credits.
An electronic version of this script can be found at

https://www.tu-braunschweig.de/itl/lehre/skripte

Literature for further reading

- Stability and observability
  - SONTAG, E.D.: *Mathematical Control Theory: Deterministic Finite Dimensional Systems*. Springer, 1998. – 531 S
  - HINRICHSEN, D. ; PRITCHARD, A.J.: *Mathematical system theory I: Modeling, state space analysis and robustness*. Springer, 2010
  - ISIDORI, A.: *Nonlinear Control Systems*. 3rd edition. Springer, 1995
- LQR, H_2 and H_∞ control
  - ANDERSON, B.D.O. ; MOORE, J.B.: *Optimal control: linear quadratic methods*. Courier Corporation, 2007
  - SKOGESTAD, S. ; POSTLETHWAITE, I.: *Multivariable feedback control: analysis and design*. John Wiley & Sons, 2005
  - KALMAN, R.E. u. a.: Contributions to the theory of optimal control. In: *Bol. soc. mat. mexicana* 5 (1960), No. 2, pp. 102–119
- Predictive and distributed control
  - GRÜNE, L. ; PANNEK, J.: *Nonlinear Model Predictive Control: Theory and Algorithms*. 2. Springer, 2017
  - RAWLINGS, J.B. ; MAYNE, D.Q. ; DIEHL, M.: *Model predictive control: theory, computation, and design*. Second. Nob Hill Publishing Madison, WI, 2017
  - RICHARDS, A. ; HOW, J.: A Decentralized Algorithm for Robust Constrained Model Predictive Control. In: *Proceedings of the American Control Conference*, 2004, pp. 4241–4266

Contents

List of Figures

List of Definitions and Theorems

CHAPTER 1

STABILITY AND OBSERVABILITY

In control engineering, *stability* and *observability* are fundamental properties of systems. A system is considered stable if its state and output remain bounded for any bounded input, and the effects of the input diminish over time. Observability implies that the system's state can be uniquely determined based on the known history of inputs and outputs.
This chapter discusses methods for enforcing and evaluating these properties. We will distinguish between linear and nonlinear systems. Linear systems can be evaluated analytically, allowing the use of formulas to prove properties. In contrast, nonlinear systems require complex simulations to evaluate.
We begin by introducing the necessary terms from *system theory* and *control theory*, followed by definitions of stability and observability.

1.1. System

The term *system* as such is typically not defined clearly. In certain areas, a system stands for a connected graph, a dynamically evolving entity or even a simulation or an optimization. While the intention of the latter are quite distinct, they all can be boiled down to the following:

A system is the connection of different interacting components to realize given tasks.

The interdependence of systems with their environment is given by so called *inputs and outputs*. More formally, we define the following:

Definition 1.1 (System).
Consider two sets $\mathcal{U}$ and $\mathcal{Y}$. Then a map $\Sigma : \mathcal{U} \to \mathcal{Y}$ is called a system.

The set $\mathcal{U}$ and $\mathcal{Y}$ are called input and output sets. An element from the input set $\mathbf{u} \in \mathcal{U}$ is called an input, which act from the environment to the system and are not dependent on the system itself or its properties. We distinguish between inputs, which are used to specifically manipulate (or control) the system, and inputs, which are not manipulated on purpose. We call the first ones *control or manipulation inputs*, and we refer to the second ones as *disturbance inputs*. An element from the output set $\mathbf{y} \in \mathcal{Y}$ is called an output. In contrast to an input, the output is generated by the system and influences the environment. Here, we distinguish output variables depending on whether we measure them or not. We call the measured ones *measurement outputs*.

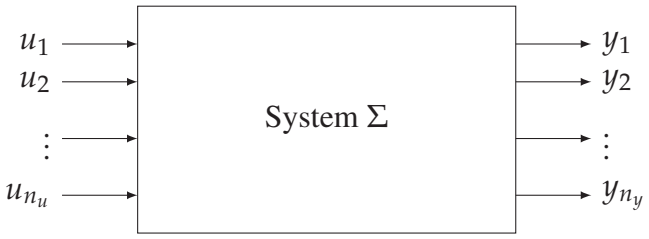


Figure 1.1.: Term of a system

In the literature, certain classes of systems are considered:

- A system is called *linear* if it is linear in inputs and outputs, and *nonlinear* if it is not linear in either the inputs or outputs.
- A system is *time invariant* if all parameters are constants, and *time varying* if at least one parameter is time-dependent.
- Systems can be classified as *static* or *dynamic* depending on whether their outputs depend solely on the input at the same time instant or also on its history.
- *Causal* systems depend only on the history of the inputs, while *acausal* systems include future values.
- If inputs are mapped directly to outputs, then the map is called *input output system*. If the input triggers changes of an internal variable and the output depends on the latter, then the map is called *state space system*.
- If time is measured continuously, the system is said to be in *continuous time*. If time is sampled, it is referred to as *discrete time system*.

To assess systems, we require a formal notation of time:

Definition 1.2 (Time).
A *time set* $\mathcal{T}$ is a subgroup of $(\mathbb{R}, +)$.

Within the lecture, we focus on state space systems, which are time invariant, dynamic and causal. To introduce such systems, we first need to define what we referred to as internal variable:

Definition 1.3 (State).
Consider a system $\Sigma : \mathcal{U} \to \mathcal{Y}$. If the output $\mathbf{y}(t)$ uniquely depends on the history of inputs $\mathbf{u}(\tau)$ for $t_0 \leq \tau \leq t$ and some $\mathbf{x}(t_0)$, then the variable $\mathbf{x}(t)$ is called state of the system and the corresponding set $\mathcal{X}$ is called state set.

Within Definition 1.3, input, output and state refer to tuples

$$\mathbf{u} = \begin{bmatrix} u_1 & u_2 & \dots & u_{n_u} \end{bmatrix}^\top \tag{1.1a}$$

$$\mathbf{y} = \begin{bmatrix} y_1 & y_2 & \dots & y_{n_y} \end{bmatrix}^\top \tag{1.1b}$$

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \dots & x_{n_x} \end{bmatrix}^\top. \tag{1.1c}$$

where u_j is an element within the subset j of the input set $\mathcal{U}$, y_j is an element within the subset j of the output set $\mathcal{Y}$ and x_j is an element within the subset j of the state set $\mathcal{X}$.

Remark 1.4
Here, we use this notation to allow for real valued and other entries such as gears, method characteristics or switches. In the real valued setting, we have $\mathcal{U} \subset \mathbb{R}^{n_u}$, $\mathcal{Y} \subset \mathbb{R}^{n_y}$ *and* $\mathcal{X} \subset \mathbb{R}^{n_x}$.

In the continuous time setting $\mathcal{T} = \mathbb{R}$, we can utilize the short form $\dot{\mathbf{x}}$ for $\frac{d}{dt}\mathbf{x}$ and obtain the following compact notation:

Definition 1.5 (State space – continuous time system).
Consider a system $\Sigma : \mathcal{U} \to \mathcal{Y}$ in continuous time $\mathcal{T} = \mathbb{R}$ satisfying the property from Definition 1.3. If $\mathcal{X}$ is a vector space, then we call it *state space* and refer to

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t), t), \quad \mathbf{x}(t_0) = \mathbf{x}_0 \tag{1.2a}$$

$$\mathbf{y}(t) = h(\mathbf{x}(t), \mathbf{u}(t), t). \tag{1.2b}$$

as *continuous time system*. Moreover, $\mathbf{u}$, $\mathbf{y}$ and $\mathbf{x}$ are called *input*, *output* and *state* of the system.

The state of a system at time instant t can then be depicted as a point in the n_x–dimensional state space. The curve of points for variable time t in the state space is called *trajectory* and is denoted by $\mathbf{x}(\cdot)$.

Remark 1.6
Systems with infinite dimensional states are called distributed parametric systems and are described, e.g., via partial differential equations. Examples of such systems are beams, boards, membranes, electromagnetic fields, heat etc..

Similarly, in discrete time $\mathcal{T} = \mathbb{Z}$ we define the following:

Definition 1.7 (State space – discrete time system).
Consider a system $\Sigma : \mathcal{U} \to \mathcal{Y}$ in discrete time $\mathcal{T} = \mathbb{Z}$ satisfying the property from Definition 1.3. If $\mathcal{X}$ is a vector space, then we refer to

$$\mathbf{x}(k+1) = f(\mathbf{x}(k), \mathbf{u}(k), k), \quad \mathbf{x}(0) = \mathbf{x}_0 \tag{1.3a}$$

$$\mathbf{y}(k) = h(\mathbf{x}(k), \mathbf{u}(k), k). \tag{1.3b}$$

as *discrete time system*. Again, $\mathbf{u}$, $\mathbf{y}$ and $\mathbf{x}$ are called *input*, *output* and *state* of the system.

While we have $t \in \mathbb{R}$ in continuous time, for discrete time systems the matter of time refers to an index $k \in \mathbb{Z}$. Therefore, trajectories are no longer represented as curves but as sequences of points within their respective set. Digitalization typically results in discrete time systems, which are obtained by sampling continuous time systems using an A/D and D/A converter. The outcome of this process is a time grid. The simplest case is equidistant sampling with a fixed sampling time T, which produces

$$\mathcal{T} := \{t_k \mid t_k := t_0 + k \cdot T\} \subset \mathbb{R}. \tag{1.4}$$

where t_0 is some fixed initial time stamp. Apart from equidistant sampling, other types such as event based or sequence based are possible.

Remark 1.8
Note that the class of discrete time systems is larger and contains the class of continuous time systems, i.e. for each continuous time system there exists a discrete time equivalent, but for some discrete time systems no continuous time equivalent exists.

Note that in both discrete and continuous time, the map shows a *flow* within the state space. A trajectory is obtained by specifying an initial value and an input sequence. Figure 1.2 illustrates the concept of flow and trajectory. In this case, the flow is colored to indicate its intensity whereas the arrows indicate its direction. The trajectory is evaluated for a specific initial value and „follows“ the flow accordingly.

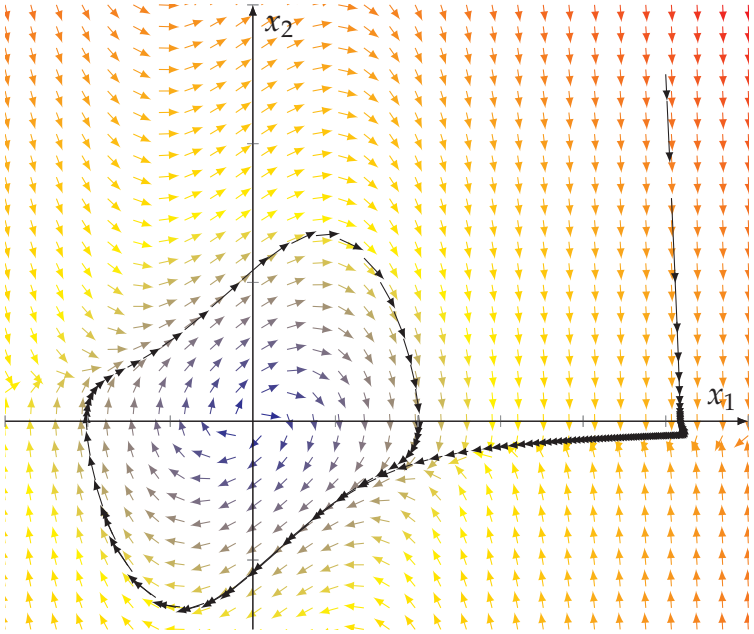


Figure 1.2.: Sketch of a dynamic flow and a trajectory

As stated in the introduction, stability refers to the ability to control a system to achieve a specific goal, such as boundedness or convergence. In order to achieve this, the input must have an impact on the states, either directly or indirectly. Observability, on the other hand, refers to the ability to identify the status of a system, that is, to be able to measure states directly or indirectly. This context is illustrated in Figure 1.3. The figure demonstrates that not all states can be manipulated, even indirectly, and not all states can be observed. However, we will see that even in this case methods can be applied to ensure stability and observability.
In order to discuss the terms stability and observability in details, we focus on the special class of linear control systems:

Definition 1.9 (Linear control system).
For matrices $A \in \mathbb{R}^{n_x \times n_x}$, $B \in \mathbb{R}^{n_x \times n_u}$, $C \in \mathbb{R}^{n_y \times n_x}$, $D \in \mathbb{R}^{n_y \times n_u}$, we call the system

$$\dot{\mathbf{x}}(t) = A \cdot \mathbf{x}(t) + B \cdot \mathbf{u}(t), \qquad \mathbf{x}(0) = \mathbf{x}_0 \tag{1.5a}$$

$$\mathbf{y}(t) = C \cdot \mathbf{x}(t) + D \cdot \mathbf{u}(t) \tag{1.5b}$$

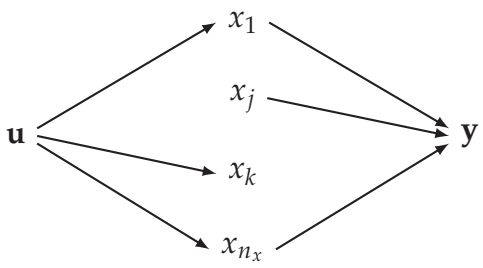


Figure 1.3.: Flow of information for controllability and observability

linear time invariant control system in continuous time with initial value $\mathbf{x}_0 \in \mathbb{R}^{n_x}$. The time discrete equivalent reads

$$\mathbf{x}(k+1) = A \cdot \mathbf{x}(k) + B \cdot \mathbf{u}(k), \qquad \mathbf{x}(0) = \mathbf{x}_0 \tag{1.6a}$$

$$\mathbf{y}(k+1) = C \cdot \mathbf{x}(k) + D \cdot \mathbf{u}(k). \tag{1.6b}$$

This class is of particular interest as we can directly give its solution

Theorem 1.10 (Solution of linear control system).
Consider a linear control system (1.5)*. Then for any initial condition* $\mathbf{x}(t_0) = \mathbf{x}_0$ *and any piecewise continuous control function* $\mathbf{u} \in \mathcal{U}$ *there exists a unique solution*

$$\mathbf{x}(t; t_0, \mathbf{x}_0, \mathbf{u}) = \exp^{A\cdot(t-t_0)} \mathbf{x}_0 + \int_{t_0}^{t} \exp^{A\cdot(t-s)} \cdot B \cdot \mathbf{u}(s) ds. \tag{1.7}$$

In the discrete time case (1.6) *the solution reads*

$$\mathbf{x}(k) = A^k \cdot \mathbf{x}_0 + \sum_{j=0}^{k-1} A^{k-1-j} \cdot B \cdot \mathbf{u}(j). \tag{1.8}$$

From the solution, we directly obtain the so called superposition property and the time shifting property:

Corollary 1.11 (Superposition and time shift).
Consider a linear control system from Definition 1.9. Then the superposition principle

$$\mathbf{x}(t; t_0, \mathbf{x}_0, \mathbf{u}) = \mathbf{x}(t; t_0, \mathbf{x}_0, 0) + \mathbf{x}(t; t_0, 0, \mathbf{u}) \tag{1.9}$$

and the time shift property

$$\mathbf{x}(t;t_0,\mathbf{x}_0,\mathbf{u}) = \mathbf{x}(t;s,\mathbf{x}(s;t_0,\mathbf{x}_0,\mathbf{u}),\mathbf{u}) = \mathbf{x}(t-s;t_0-s,\mathbf{x}_0,\mathbf{u}(s+\cdot)) \tag{1.10}$$

hold.

The superposition principle allows us to separate the uncontrolled solution ($\mathbf{u} = 0$) and the unforced solution ($\mathbf{x}_0 = 0$).

1.2. Stability

Stability is a crucial characteristic of control systems and is linked to specific points in the state space known as the *operating point*. At these points, the system's dynamics should be zero. In other words, the input (as a control) must be selected appropriately to ensure that the system remains stable.

Definition 1.12 (Operating point).
For continuous time systems (1.2) the pairs $(\mathbf{x}^\star, \mathbf{u}^\star)$ satisfying

$$f(\mathbf{x}^\star, \mathbf{u}^\star) = 0 \tag{1.11}$$

are called *operating points* of the system. For discrete time systems (1.3) we call $(\mathbf{x}^\star, \mathbf{u}^\star)$ operating point if

$$f(\mathbf{x}^\star, \mathbf{u}^\star) = \mathbf{x}^\star \tag{1.12}$$

If (1.11) or (1.12) hold true respectively for any $\mathbf{u}^\star$, then the operating point is called *strong* or *robust operating point*.

Note that for autonomous systems, that is (1.2) or (1.3) being independent of time t or k, the control $\mathbf{u} \in \mathbb{R}^{n_u}$ is required to be constant and fixed to $\mathbf{u} = \mathbf{u}^\star$ in order to compute the operating points.
Based on this definition, the property of stability can be characterized by boundedness and convergence of solutions:

Definition 1.13 (Stability and Controllability).
For a system (1.2) we call $\mathbf{x}^\star$

- *strongly* or *robustly stable* operating point if, for each $\varepsilon > 0$, there exists a real number $\delta = \delta(\varepsilon) > 0$ such that for all $\mathbf{u}$ we have

$$\|\mathbf{x}_0 - \mathbf{x}^\star\| \leq \delta \implies \|\mathbf{x}(t) - \mathbf{x}^\star\| \leq \varepsilon \qquad \forall t \geq 0 \tag{1.13}$$

- *strongly* or *robustly asymptotically stable* operating point if it is stable and there exists a positive real constant r such that for all $\mathbf{u}$

$$\lim_{t\to\infty} \|\mathbf{x}(t) - \mathbf{x}^\star\| = 0 \tag{1.14}$$

  holds for all $\mathbf{x}_0$ satisfying $\|\mathbf{x}_0 - \mathbf{x}^\star\| \leq r$. If additionally r can be chosen arbitrary large, then $\mathbf{x}^\star$ is called *globally strongly* or *robustly asymptotically stable*.

- *weakly stable* or *controllable* operating point if, for each $\varepsilon > 0$, there exists a real number $\delta = \delta(\varepsilon) > 0$ such that for each $\mathbf{x}_0$ there exists a control $\mathbf{u}$ guaranteeing

$$\|\mathbf{x}_0 - \mathbf{x}^\star\| \leq \delta \implies \|\mathbf{x}(t) - \mathbf{x}^\star\| \leq \varepsilon \qquad \forall t \geq 0. \tag{1.15}$$

- *weakly asymptotically stable* or *asymptotically controllable* operating point if there exists a control $\mathbf{u}$ depending on $\mathbf{x}_0$ such that (1.15) holds and there exists a positive constant r such that

$$\lim_{t\to\infty} \|\mathbf{x}(t) - \mathbf{x}^\star\| = 0 \qquad \forall \|\mathbf{x}_0 - \mathbf{x}^\star\| \leq r. \tag{1.16}$$

  If additionally r can be chosen arbitrary large, then $\mathbf{x}^\star$ is called *globally asymptotically stable*.

Stability and controllability are important properties of a system. Stability allows the input to be considered as a disturbance while still retaining the mentioned properties. Controllability, on the other hand, refers to inducing these properties to the system by means of the input.
In the linear case, we can derive sufficient properties for the system to be stable using the Eigenvalue criterion:

Theorem 1.14 (Eigenvalue criterion).
Consider a system (1.5) *with* $\mathbf{u} \equiv 0$. *Let* $\lambda_1, \ldots, \lambda_j \in \mathbb{C}$ *be the Eigenvalues of* A.

- *Then the operating point* $\mathbf{x}^\star = 0$ *is stable iff all Eigenvalues have non-positive real part and for all Eigenvalues with real part* 0 *the corresponding Jordan block is one-dimensional.*

- *Then the operating point $\mathbf{x}^\star = 0$ is locally asymptotically stable iff all Eigenvalues have negative real part.*

Remark 1.15
If all Eigenvalues of a matrix A exhibit negative real part, then the matrix is called Hurwitz.

Given the Eigenvalue criterion, it is straightforward to derive an input, which induces the stability property.

Theorem 1.16 (Linear feedback).
Consider a system (1.5) *with $\mathbf{u} = F \cdot \mathbf{x}$. Then the operating point $\mathbf{x}^\star = 0$ is locally asymptotically stable iff all Eigenvalues of $A + B \cdot F$ for a feedback F have negative real part.*

So technically, that would be it. Yet, we don't know

1. whether or not it is actually possible that a feedback F can be constructed such that the conditions of Theorem 1.16 hold, nor
2. how such a feedback can be constructed.

To answer the first question, we take a look at controllability of a system. Here, Kalman formulated that idea to reach points by combinations of dynamics and input, that is A and B. Since the dimension of the set reachable by the dynamics only cannot grow larger after $n_x - 1$ iterations, he introduced the so called Kalman criterion:

Theorem 1.17 (Kalman criterion).
The system (1.5) *is controllable iff for the controllability matrix*

$$rk\left(B \mid A \cdot B \mid \ldots \mid A^{n_x-1} \cdot B\right) = n_x \tag{1.17}$$

holds. Then the pair (A, B) is called controllable.

Remark 1.18
The reachable set is typically defined as the set of points, which can be reached from $\mathbf{x}_0 = 0$ within a certain time $t \geq 0$ via

$$\mathcal{R}(t) := \{\mathbf{x}(t, 0, \mathbf{u}) \mid \mathbf{u} \in \mathcal{U}\}.$$

Similarly, the controllable set refers to those points $\mathbf{x}_0$*, for which a control* $\mathbf{u}$ *can be found to drive the solution to the origin, i.e.*

$$\mathcal{C}(t) := \{\mathbf{x}_0 \mid \exists \mathbf{u} \in \mathcal{U} : \ \mathbf{x}(t, \mathbf{x}_0, \mathbf{u}) = 0\}.$$

Unfortunately, Kalman assumed that the control needs to affect all dimensions of the state space for the system to be controllable. However, if a part of the system is already controllable without the control affecting it, then only the controllability of the remaining part needs to be ensured. Therefore, Hautus introduced separability in the state space:

Theorem 1.19 (Separability).
For any system (1.5)*, which is not controllable, there exists a linear transformation* T *such that*

$$\tilde{A} := T^{-1} \cdot A \cdot T = \begin{pmatrix} A_1 & A_2 \\ 0 & A_3 \end{pmatrix}, \qquad \tilde{B} := T^{-1} \cdot B = \begin{pmatrix} B_1 \\ 0 \end{pmatrix} \tag{1.18}$$

where (A_1, B_1) *is controllable.*

Now, the idea is to simply apply the Kalman criterion to the separated part of the dynamics/state space:

Theorem 1.20 (Hautus criterion).
Consider a system (1.5)*. Then* (A, B) *is controllable iff*

$$rk\,(\lambda Id - A \mid B) = n_x \tag{1.19}$$

holds

1. *for all* $\lambda \in \mathbb{C}$ *or*
2. *for all eigenvalues* $\lambda \in \mathbb{C}$ *of* A*.*

After addressing whether feedback can be constructed, we will now shift our focus to computing such feedback. We will achieve this by applying basic linear algebra, which will provide us with the controllable canonical form. We will begin with the simpler Kalman case.

Theorem 1.21 (Controllable canonical form).
Consider a system (1.5). *Then* (A, B) *is controllable iff there exists a linear transformation* T *with*

$$\tilde{A} = T^{-1} \cdot A \cdot T = \begin{pmatrix} 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_{n_x} \end{pmatrix} \qquad \tilde{B} = T^{-1} \cdot B = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} \tag{1.20}$$

with coefficients α_j *of the assigned polynomial* $\Xi_A = z^{n_x} - \alpha_{n_x} z^{n_x - 1} - \cdots - \alpha_2 z - \alpha_1$.

Based on the latter, we directly obtain controllability if we can assign any polynomial.

Theorem 1.22 (Assignable polynomial).
Consider a system (1.5). *Then the pair* (A, B) *is controllable iff every polynomial of degree* n_x *is assignable.*

To enforce the stability property, we require that the roots of an assignable polynomial are in the negative complex half-plain. Hence, if any polynomial is assignable, we choose one for which the root criterion holds.

Theorem 1.23 (Stabilizing polynomial).
Consider a system (1.5). *Then the operating point* $\mathbf{x}^\star = 0$ *is locally asymptotically stable iff there exists an assignable polynomial, for which all roots in* $\mathbb{C}$ *have negative real part.*

Coming back to Hautus's case, we basically require that the uncontrollable part is already stable, that is:

Corollary 1.24 (Polynomial for Hautus criterion).
For any system (1.5)*, the following is equivalent:*

- *There exists an assignable polynomial, for which all roots in* $\mathbb{C}$ *have negative real part.*
- *The pair* (A, B) *is controllable or* (A, B) *is not controllable but* A_3 *has only eigenvalues with negative real part.*

Combining these lines of argumentation, Figure 1.4 provides an overview of the results.

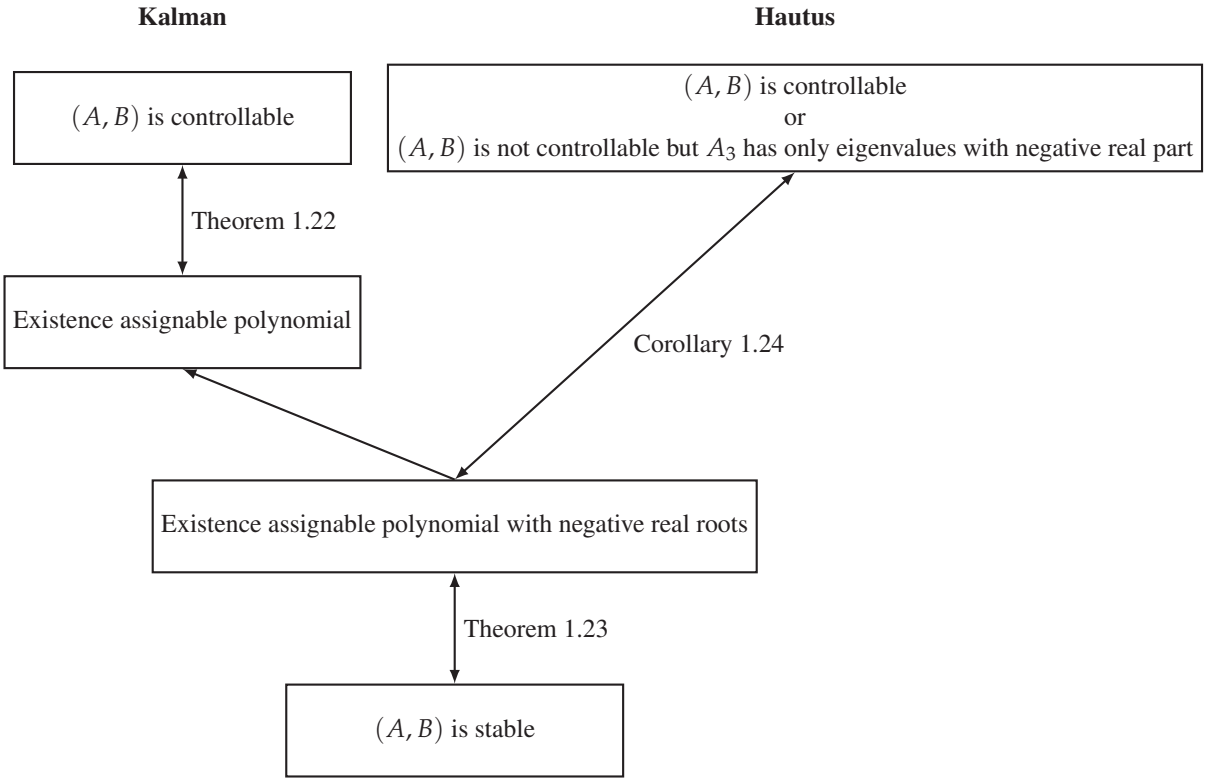


Figure 1.4.: Connection of controllability and stability

Remark 1.25
Theorem 1.23 and Corollary 1.24 are often called pole shifting theorem as the roots of the characteristic polynomial are equivalent to the poles of the transfer matrix of the system.

Regarding stability, we obtain the (dis-)advantages shown in Table 1.1.

Table 1.1.: Advantages and disadvantages of stability results

Advantage	Disadvantage
✓ Allows for canonical form	✗ Is limited to linear systems
✓ Uses simple computation	✗ Unable to treat instabilities
✓ Provides analytic solution	✗ May require separation

1.3. Observability

In many cases, only a reasonable subset of manipulable inputs are controllable, with regards to controllability. Similarly, regarding observability, in most cases only a subset of measurable outputs are actually measured. For our linear time invariant system (1.5) or (1.6) this means that the matrices C, D are not full rank matrices. In practice, states are typically measured while inputs remain unmeasured, i.e. $D = 0$.
The task for observability is to derive information on the system from the outputs $\mathbf{y}(\cdot) \in \mathcal{Y}$ by utilizing the values themselves and the history of values.

Definition 1.26 (Distinguishability).
For a system (1.2) we call

- two states $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$ *distinguishable* if there exists an input $\mathbf{u} \in \mathcal{U}$ such that

$$h(\mathbf{x}_1(t), \mathbf{u}(t)) \neq h(\mathbf{x}_2(t), \mathbf{u}(t)) \tag{1.21}$$

  for some time $t \in \mathcal{T}$.

- the system *observable* if any two states $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$ are distinguishable.

As in the previous Section 1.2, we now focus on the linear time invariant case. For such systems, we have that equation (1.21) reads

$$C \cdot \mathbf{x}(t, \mathbf{x}_1, \mathbf{u}(t)) \neq C \cdot \mathbf{x}(t, \mathbf{x}_2, \mathbf{u}(t)). \tag{1.22}$$

By superposition, we can simplify the latter using linearity:

Lemma 1.27 (Necessary and sufficient condition for distinguishability).
Consider the system (1.5). *Then two states* $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$ *are distinguishable iff condition*

$$C \cdot \mathbf{x}(t, \mathbf{x}_1 - \mathbf{x}_2, 0) \neq 0 \tag{1.23}$$

holds for some $t \geq 0$.

Note that the lemma states that whether states are distinguishable and observable does not depend on the input $\mathbf{u}$ in the linear case.

Remark 1.28
The set of non-observable states is defined as those states $\mathbf{x}_0$ *such that the output for* $\mathbf{u} = 0$ *is always zero, i.e.*

$$\mathcal{N}(t) := \{\mathbf{x}_0 \mid C \cdot \mathbf{x}(t, \mathbf{x}_0, 0) = 0 \; \forall t \geq 0\}.$$

So again as in Section 1.2, we

1. need to identify conditions to ensure that a system is observable, and
2. have to construct an *observer*.

Based on Lemma 1.27, we can apply the Eigenvalue criterion from Theorem 1.14 to the pair (A, C).

Theorem 1.29 (Kalman criterion).
The system (1.5) *is observable and the pair* (A, C) *is called observable iff for the observability matrix*

$$rk\left(C^\top \mid A^\top \cdot C^\top \mid \ldots \mid \left(A^\top\right)^{n_x - 1} \cdot C^\top\right) = n_x \tag{1.24}$$

holds.

Following Hautus' approach, we can use the same separation within the dynamics to expand the applicability of Kalman's criterion.

Theorem 1.30 (Separability).
For any system (1.5)*, which is not observable, there exists a linear transformation* T *such that*

$$\tilde{A} := T^{-1} \cdot A \cdot T = \begin{pmatrix} A_1 & A_2 \\ 0 & A_3 \end{pmatrix}, \; \tilde{B} := T^{-1} \cdot B = \begin{pmatrix} B_1 \\ 0 \end{pmatrix}, \; \tilde{C} := C \cdot T = \begin{pmatrix} 0 & C_2 \end{pmatrix} \tag{1.25}$$

where (A_3, C_2) *is observable.*

Now, we are faced with the challenge of lacking an equivalent for stability in the context of observable systems. However, it has been observed that there are significant similarities between controllability and observability, which also apply on a systemic level:

Definition 1.31 (Dual system).
Consider the system (1.5) defined by (A, B, C). Then we define the *dual system* as given by $(A^\top, C^\top, B^\top)$.

Using this definition, we obtain

Theorem 1.32 (Duality).
Consider a system (A, B, C) *and its dual* $(A^\top, C^\top, B^\top)$. *Then we have*

$$(A, B, C) \text{ controllable} \iff (A^\top, C^\top, B^\top) \text{ observable} \tag{1.26}$$
$$(A, B, C) \text{ observable} \iff (A^\top, C^\top, B^\top) \text{ controllable} \tag{1.27}$$

Remark 1.33
In particular, we have that the reachable set of the dual system is identical to the observable set

$$\left(\bigcup_{t\geq 0} \mathcal{R}(t)\right)\top =: \mathcal{R}^\top = \mathcal{N}^\perp := \left(\bigcap_{t\geq 0} \mathcal{N}(t)\right)^\perp.$$

and vice versa.

Using duality, we define the property detectability, which resembles stability of the dual system.

Definition 1.34 (Detectability).
A system (1.5) is called *detectable* if

$$\lim_{t\to\infty} \mathbf{x}(t, \mathbf{x}_0, 0) = 0 \tag{1.28}$$

holds for all $\mathbf{x}_0 \in \mathcal{X}$.

Detectability therefore means that information on the non-observable part (cf. Theorem 1.30) is not required as respective solutions are asymptotically stable.
Hence, we now have the means to transfer the Hautus criterion to observability.

Theorem 1.35 (Hautus criterion).
Consider a system (1.5). *Then* (A, C) *is observable iff*

$$rk\left(\lambda Id - A^\top \mid C^\top\right) = n_x \tag{1.29}$$

holds

1. *for all* $\lambda \in \mathbb{C}$ *or*
2. *for all eigenvalues* $\lambda \in \mathbb{C}$ *of* A.

Similar to the canonical form for controllability, for observable systems a respective transformation can be found.

Theorem 1.36 (Observable canonical form).
Consider a system (1.5). *Then* (A, C) *is observable iff there exists a linear transformation* T *with*

$$\tilde{A} = T^{-1} \cdot A \cdot T = \begin{pmatrix} 0 & \cdots & \cdots & 0 & \alpha_1 \\ 1 & \vdots & \ddots & \vdots & \alpha_2 \\ \vdots & 1 & \ddots & 0 & \vdots \\ 0 & 0 & \vdots & 1 & \alpha_{n_x} \end{pmatrix} \qquad \tilde{C} = C \cdot T = \begin{pmatrix} 0 & \cdots & 0 & 1 \end{pmatrix} \tag{1.30}$$

with coefficients α_j *of the assigned polynomial* $\Xi_A = z^{n_x} - \alpha_{n_x} z^{n_x - 1} - \cdots - \alpha_2 z - \alpha_1$.

Using duality, we particularly have

Theorem 1.37 (Duality of detectability and controllability).
A system (A, C) *is detectable iff the system* $(A^\top, C^\top)$ *is controllable.*

Combining these lines of argumentation together with the core of stability, Figure 1.5 provides an overview of the results.
We like to point out that the properties controllability and observability are independent from one another and only connected for the respective dual system. Consequently, there exist four classes of systems

1. controllable and observable,
2. controllable and not observable,
3. not controllable and observable, and
4. not controllable and not observable.

These classes can also be seen in Figure 1.3, which served as starting point for these terms.
Regarding observability, we obtain the (dis-)advantages shown in Table 1.2.

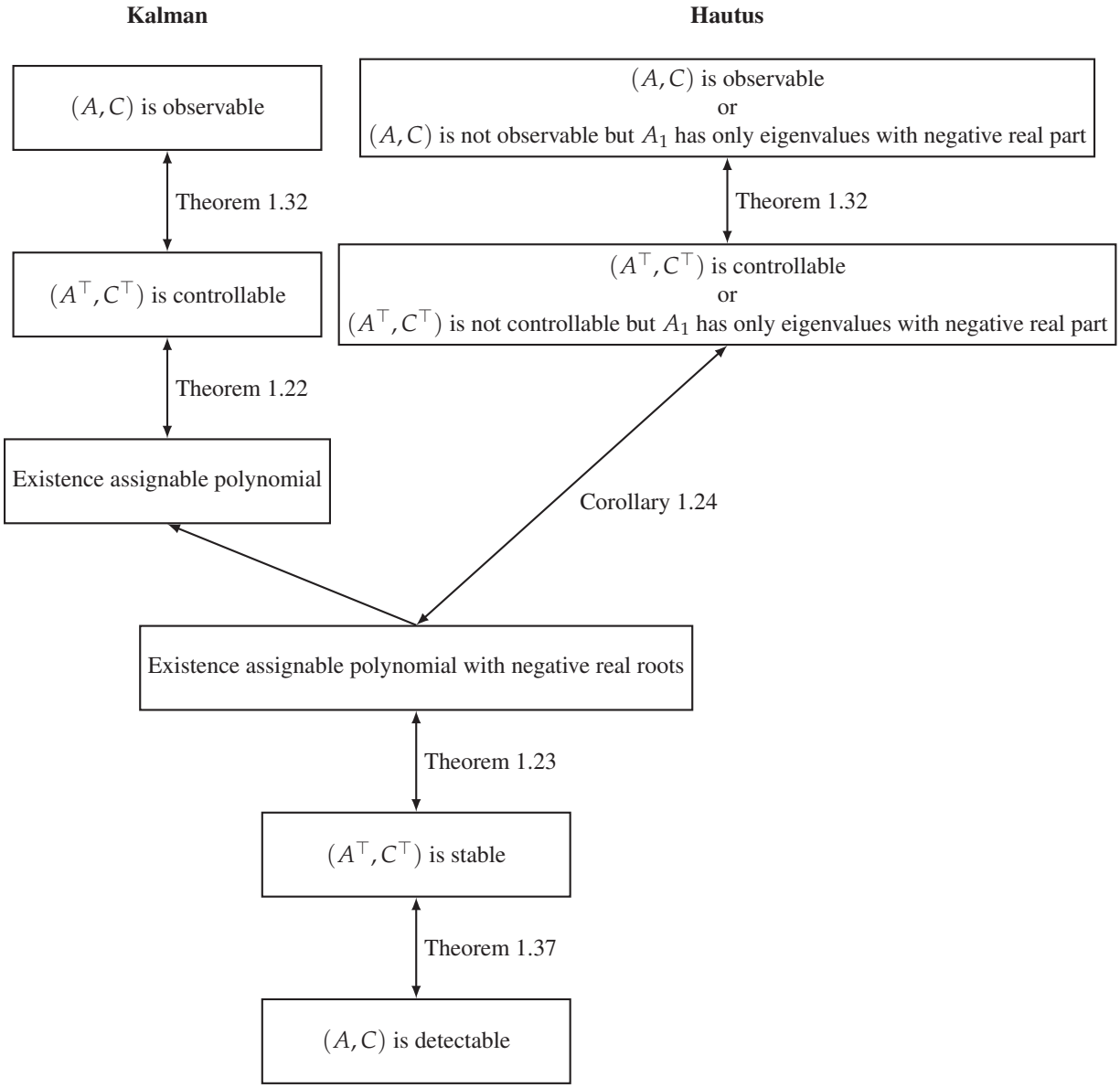


Figure 1.5.: Connection of observability and detectability

Table 1.2.: Advantages and disadvantages of observability results

Advantage		**Disadvantage**	
✓	Allows for canonical form	✗	Is limited to linear systems
✓	Analog to stability	✗	Requires detectability

Part I.

Linear systems

CHAPTER 2

OPTIMAL STABILIZATION

Regarding stabilization, we found the Eigenvalue criterion for computing stabilizing feedbacks in Chapter 1. While this is sufficient to guarantee stability, it only addresses a qualitative property, while quantitative aspects such as performance or the dynamics itself are not considered. Specific examples of quantitative aspects that should be avoided are large trajectory overshoots and large control values.

To deal with such quantitative issues, we discuss methods that incorporate the latter directly into their construction. To do this, we first clarify what is good and what needs to be avoided, and then quantify these aspects. This is achieved by using so called *key performance indicators* within a *cost function* that is optimized according to the state and the dynamics of the problem.

Throughout this chapter, we consider the case of stabilizing an operating point. More general settings will be considered for more advanced methods. In addition, we limit ourselves to linear time invariant systems of the form (1.5) and assume that the full state is measureable.

2.1. Linear quadratic regulator — LQR

The starting point for optimal feedback design is the quantification of good performance. For this purpose, inputs, outputs and functional dependencies of the system can be used to derive a quantification. For LQR we consider the state space representation, but for H_2 and H_∞ controllers the frequency representation is used. To handle both concepts, we use so-called key performance criteria.

Definition 2.1 (Key performance criterion).
A key performance criterion is a function, which measures defined information retrieved from the system against a standard.

Focusing on the state space, we typically speak of cost functions. These combined information on state and input of the system to quantify performance of the control.

Definition 2.2 (Cost function).
We call a key performance criterion given by a function $\ell : \mathcal{X} \times \mathcal{U} \to \mathbb{R}_0^+$ a *cost function*.

The value of a key performance indicator only shows a snapshot, i.e. the evaluation at a single point in time $t \in \mathcal{T}$. To get the performance, we need to evaluate it over the lifetime of the system. Since we are defining a function of a function, this is called a functional.

Definition 2.3 (Cost functional).
Consider a key performance criterion $\ell : \mathcal{X} \times \mathcal{U} \to \mathbb{R}_0^+$. Then we call

$$J(\mathbf{x}_0, \mathbf{u}) := \int_0^\infty \ell(\mathbf{x}(t, \mathbf{x}_0, \mathbf{u}), \mathbf{u}(t))dt \tag{2.1}$$

cost functional.

Now we can combine the criteria to evaluate and optimize the dynamics over an operating period. This allows us to quantify not only operating points, but also the transients from the current state of the system to such an operating point.

Definition 2.4 (Optimal control problem).
Consider a system (1.2) and a cost functional (2.1). Then we call

$$\begin{aligned} &\min\ J(\mathbf{x}_0, \mathbf{u}) = \int_0^\infty \ell(\mathbf{x}(t, \mathbf{x}_0, \mathbf{u}), \mathbf{u}(t))dt \quad \text{over all } \mathbf{u} \in \mathcal{U} \\ &\text{subject to } \dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t), t), \quad \mathbf{x}(t_0) = \mathbf{x}_0 \end{aligned} \tag{2.2}$$

an *optimal control problem*. The function

$$V(\mathbf{x}_0) := \inf_{\mathbf{u} \in \mathcal{U}} J(\mathbf{x}_0, \mathbf{u}) \tag{2.3}$$

is called *optimal value function*.

The idea of the optimal control problem is to enforce the stability property of a system and to compute a feedback that is optimal in terms of the key performance indicator. A simple way to check whether a feedback stabilizes a system is to use the following condition.

Definition 2.5 (Null controlling).
Consider a system (1.2) and a cost function $J : \mathcal{X} \times \mathcal{U} \to \mathbb{R}_0^+$. If the condition

$$J(\mathbf{x}_0, \mathbf{u}) < \infty \qquad \Longrightarrow \qquad \mathbf{x}(t, \mathbf{x}_0, \mathbf{u}) \to 0 \quad \text{for } t \to \infty \tag{2.4}$$

holds, then we call the optimal control problem *null controlling*.

The relationship between condition (2.4) and stability is quite simple: If we design the key performance criterion to be zero at the desired operating point, then once the operating point is reached, no additional costs will be incurred over the operating period. Therefore, the state of the system will remain at the operating point. Note that by Definition 1.12 for each operating point there exists an input such that the state remains unchanged.

Corollary 2.6 (Null controlling stability).
If a optimal control problem is null controlling, then the system is stabilizable.

Now, we focus on the LTI case (1.5). For this particular case, it is sufficient to consider a norm like criterion, that is a way to measure the distance from current state to operating point. The first distance which we consider is the Euclidean distance.

Definition 2.7 (Quadratic cost function).
We call a key performance criterion $\ell : \mathcal{X} \times \mathcal{U} \to \mathbb{R}_0^+$ a quadratic cost function if it is given by

$$\ell(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} \mathbf{x}^\top & \mathbf{u}^\top \end{bmatrix} \cdot \begin{pmatrix} Q & N \\ N^\top & R \end{pmatrix} \cdot \begin{bmatrix} \mathbf{x} \\ \mathbf{u} \end{bmatrix} \tag{2.5}$$

where $Q \in \mathbb{R}^{n_x \times n_x}$, $N \in \mathbb{R}^{n_x \times n_u}$ and $R \in \mathbb{R}^{n_u \times n_u}$ form a symmetric and positive definite matrix in (2.5).

Combining linear dynamics with quadratic costs gives us the so called LQ problem.

Definition 2.8 (LQ problem).
Consider the optimal control problem given by the LTI system (1.5) and the quadratic cost function (2.5). Then we refer to this setting as *linear quadratic problem* or *LQ* problem.

The nice property of the LQ problem is that its solution is null controlling and therefore the solution also stabilizes the system.

Theorem 2.9 (Null controlling).
The LQ problem is null controlling.

The central question now is to compute the solution of the LQ problem. In particular, we are not simply interested in a solution but in a solution which can be evaluated based on the state of the system, i.e. a feedback. To this end, we utilize the idea of the value function and suppose it can be chosen in the ansatz

$$V(\mathbf{x}) = \mathbf{x}^\top \cdot P \cdot \mathbf{x} \tag{2.6}$$

for $P \in \mathbb{R}^{n_x \times n_x}$. If this ansatz is right, we obtain the following:

Theorem 2.10 (LQR feedback).
If the LQ problem exhibits a value function of the form (2.6)*, then the solution to the LQ problem*

$$\mathbf{u}^\star(t) = F \cdot \mathbf{x}(t, \mathbf{x}^\star, F) \tag{2.7}$$

is asymptotically stable with feedback matrix $F \in \mathbb{R}^{n_u \times n_x}$ given by

$$F = -R^{-1} \cdot \left(B^\top \cdot P + N\right) \tag{2.8}$$

and $\mathbf{x}(t, \mathbf{x}^\star, F)$ represents the solution of the closed loop

$$\dot{\mathbf{x}}(t) = (A + B \cdot F) \cdot \mathbf{x}(t), \qquad \mathbf{x}(0, \mathbf{x}^\star, F) = \mathbf{x}^\star.$$

To evaluate the feedback, we require the matrix P of the value function ansatz. This matrix can be computed using the so called algebraic Riccati equation. The idea of this equation is that the solution reaches the operating point and calculate the minimum of the ansatz (2.6), i.e. take the derivative and set it to zero. Since the ansatz is quadratic, the necessary condition is also sufficient for optimality.

Theorem 2.11 (Algebraic Riccati equation).
The optimal value function of the LQ problem is given by (2.6) *iff the matrix $P \in \mathbb{R}^{n_x \times n_x}$ is semi positive definite and solves the algebraic Riccati equation*

$$P \cdot A + A^\top \cdot P + Q - (P \cdot B + N) \cdot R^{-1} \left(B^\top \cdot P + N^\top\right) = 0. \tag{2.9}$$

When computing of a solution P of (2.9), we have to be careful about the requirements of the solution for the following reason: While the algebraic Riccati equation can have more than one solution, there exists at most one semi positive definite P. Combining the latter results, we obtain the following procedure to compute the linear quadratic regulator (LQR):

Algorithm 2.12 (Computation of LQR)
Consider an LQ problem

$$\min\ J(\mathbf{x}_0, \mathbf{u}) = \int_0^\infty \begin{bmatrix}\mathbf{x}(t)^\top & \mathbf{u}(t)^\top\end{bmatrix} \cdot \begin{pmatrix} Q & N \\ N^\top & R \end{pmatrix} \cdot \begin{bmatrix}\mathbf{x}(t) \\ \mathbf{u}(t)\end{bmatrix} dt \quad \text{over all } \mathbf{u} \in \mathcal{U} \tag{2.10}$$
$$\text{subject to } \dot{\mathbf{x}}(t) = A \cdot \mathbf{x}(t) + B \cdot \mathbf{u}(t), \quad \mathbf{x}(t_0) = \mathbf{x}_0$$

to be given. Then we obtain the LQR feedback F via

1. Compute a semipositive definite solution P of the algebraic Riccati equation (2.9)

$$P \cdot A + A^\top \cdot P + Q - (P \cdot B + N) \cdot R^{-1} \left(B^\top \cdot P + N^\top\right) = 0.$$

2. Compute the optimal linear feedback F via (2.8)

$$F = -R^{-1} \cdot \left(B^\top \cdot P + N\right).$$

The connections between the latter results are visualized in Figure 2.1.

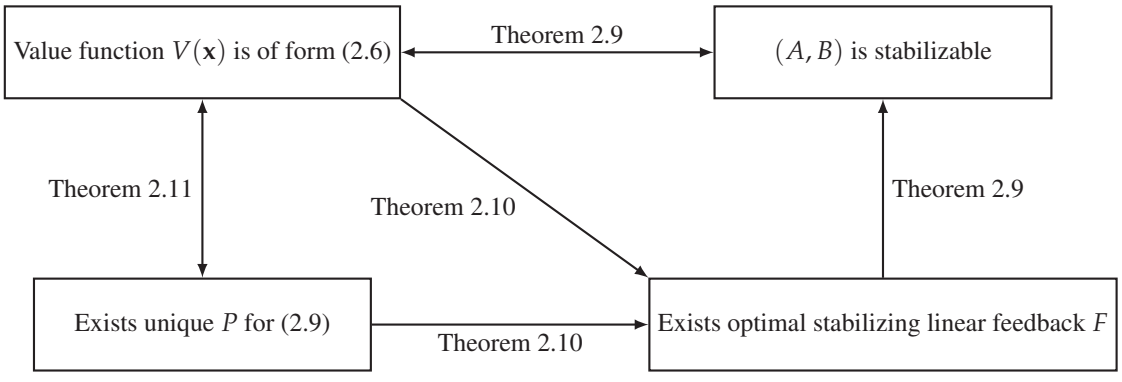


Figure 2.1.: Connection of LQR results

Remark 2.13
The state based setting described within this section can be extended to the output based setting. For this case, we utilize the quadratic cost function

$$\ell(\mathbf{y},\mathbf{u}) = \begin{bmatrix}\mathbf{y}^\top & \mathbf{u}^\top\end{bmatrix} \cdot \begin{pmatrix}\widetilde{Q} & \widetilde{N} \\ \widetilde{N}^\top & R\end{pmatrix} \cdot \begin{bmatrix}\mathbf{y} \\ \mathbf{u}\end{bmatrix} \tag{2.11}$$

with $Q = C^\top \cdot \widetilde{Q} \cdot C$ and $N = C^\top \cdot N$. Given output values $\mathbf{y}$, we obtain that the respective LQ problem is null controlling if the pair (A, C) is observable. In that case, the relations drawn in Figure 2.1 hold.

Recapitulating the present section, we summarize the (dis-)advantages shown in Table 2.1.

Table 2.1.: Advantages and disadvantages of the LQR approach

Advantage		Disadvantage	
✓	Allows explicit computation	✗	Limited to LTI systems
✓	Optimal for quadratic costs	✗	Limited to quadratic costs
✓	Provides continuous feedback	✗	Requires Riccati equation

2.2. H_2 control

In contrast to LQR, which focuses on properties measured within the state space, the H_2 formalism considers a frequency domain idea. To get to this idea, we first introduce the 2-norm for systems.

Definition 2.14 (L_2 norm).
Consider a function $v : \mathbb{R} \to \mathbb{R}^{n_y}$. Then we call

$$\|v\|_2 = \left(\int_0^\infty \sum_{j=1}^{n_y} v_j(t)^2 dt\right)^{\frac{1}{2}} = \left(\int_0^\infty v(t)^\top \cdot v(t) dt\right)^{\frac{1}{2}} \tag{2.12}$$

the L_2 norm of the function. If

$$V(s) := \hat{v}(s) = \mathcal{L}(f(t)) = \int_0^\infty \exp(-st) \cdot f(t)dt, \qquad s = \alpha + i\omega$$

denotes the Laplace transform of v, then we call

$$\|V\|_2 = \left(\frac{1}{2\pi} \int_{-\infty}^{\infty} \sum_{j=1}^{n_y} |V_j(i\omega)|^2 d\omega \right)^{\frac{1}{2}} = \left(\frac{1}{2\pi} \int_{-\infty}^{\infty} V(i\omega)^\top \cdot V(i\omega) d\omega \right)^{\frac{1}{2}} \tag{2.13}$$

the L_2 norm of the transform.

Remark 2.15
In the literature, the term L_2 space is typically found to be the correct one. Yet, talking about function which are bounded and analytic in the right half plane and exhibit finite L_p norms on the imaginary axis – which are fundamental for stable function – are called Hardy spaces, the term H_2 norm has become dominant.

By Parseval's theorem we directly have

Corollary 2.16 (H_2 norm equivalence).
Consider a function $v : \mathbb{R} \to \mathbb{R}^{n_y}$ and its Laplace transform $V := \hat{v}$. Then

$$\|v\|_2 = \|V\|_2 \tag{2.14}$$

holds for the H_2 norms.

In order to apply this result, we reconsider our dynamics. For multivariable systems, we know from control theory that a reformulation via the Laplace transform reveals a transfer matrix connecting inputs to outputs. In particular, for our LTI case (1.5)

$$\begin{aligned} \dot{\mathbf{x}}(t) &= A \cdot \mathbf{x}(t) + B \cdot \mathbf{u}(t) \\ \mathbf{y}(t) &= C \cdot \mathbf{x}(t) + D \cdot \mathbf{u}(t) \end{aligned}$$

the frequency domain equivalent is given by

$$G(s) = C \cdot (s\mathrm{Id} - A)^{-1} \cdot B + D.$$

Computing the solution of the LTI system reveals

$$\mathbf{y}(t) = C \cdot \exp^{A \cdot t} \cdot \mathbf{x}_0 + \int_0^t H(t-\tau) \cdot \mathbf{u}(\tau) d\tau \tag{2.15}$$

where $H(t-\tau)$ is the impulse response

$$H(t) := \begin{cases} C \cdot \exp^{A \cdot t} \cdot B + D, & \text{if } t \geq 0 \\ 0, & \text{if } t < 0. \end{cases}$$

Combined, we obtain the Laplace transform of the impulse response:

Corollary 2.17 (Laplace-transform impulse response).
Consider an LTI system (1.5)*. Then*

$$G(s) = \int_0^\infty H(t) \cdot \exp^{-st} dt \tag{2.16}$$

represents the transfer matrix of the system.

Now we can apply Corollary 2.16 to our dynamics and see the following:

Theorem 2.18 (H_2 norm equivalence for LTI).
Consider an LTI system (1.5) *and let* $G(s)$ *be its Laplace transform. Then we have*

$$\|G\|_2 = \|H\|_2 \tag{2.17}$$

where by (2.12) *we have*

$$\|H\|_2 = \left(\int_{-\infty}^{\infty} \sum_{j=1}^{n_y} \sum_{k=1}^{n_y} |H_{jk}(t)|^2 dt \right)^{\frac{1}{2}} = \left(\int_{-\infty}^{\infty} tr\left(H(t)^\top \cdot H(t) \right) dt \right)^{\frac{1}{2}}. \tag{2.18}$$

Equation (2.18) allows us to evaluate the H_2 norm in frequency domain by means known in the state domain. To this end, we only require the solution and the respective output, which we get

from (2.15). In particular, for the LTI case we have

$$\|G\|_2 = \|H\|_2 = \operatorname{tr}\left(\left(\int_0^\infty C \cdot \exp^{A\cdot t} \cdot B + D\right)^\top \cdot \left(\int_0^\infty C \cdot \exp^{A\cdot t} \cdot B + D\right)\right)^{\frac{1}{2}}. \tag{2.19}$$

Here, we get the first result for a respective controller:

Theorem 2.19 (H_2 stability).
Consider an LTI system (1.5)*. Then the system is stable iff its H_2 norm is finite.*

Having defined the connections between the norms, the aim of the H_2 controller we want to compute now is to minimize the H_2 norm of the closed loop. Note that the term associated to the initial value $\mathbf{x}_0$ in (2.15) is a constant and therefore can be omitted in an optimization.

Definition 2.20 (H_2 problem).
Consider the optimal control problem given by the LTI system (1.5) and the cost functional from (2.18)

$$J(\mathbf{x}_0, \mathbf{u}) := \|H\|_2^2 = \sum_{j=1}^{n_u} \int_0^\infty \mathbf{y}(t)^\top \cdot \mathbf{y}(t) dt \tag{2.20}$$

to be minimized over all $\mathbf{u}(t) = e_j \cdot \delta(t)$ where $\delta(\cdot)$ is the Dirac delta function. Then we refer to this setting as H_2 *problem*.

Within this setting, the input is modeled as noise, which is realized on the j-th input using the Dirac delta and may occur at any time instant t.

Remark 2.21
If the covariance of the inputs is a unitary matrix, then the input can be interpreted as white noise. Moreover, the result of the H_2 converges in the expected value as all frequencies are accounted for in an equal manner. Therefore, the H_2 control shows a stochastic characterization.

Having defined the H_2 problem, we can solve it using an identical idea as in the LQR case, that is to impose an algebraic Riccati equation. In particular, we obtain the following:

Theorem 2.22 (H_2 feedback).
Consider the H_2 problem and suppose $\|H_2\|$ to be finite. Then the solution to the H_2 problem

$$\mathbf{u}^\star(t) = F \cdot \mathbf{x}(t, \mathbf{x}^\star, F) \tag{2.21}$$

is asymptotically stable with feedback matrix $F \in \mathbb{R}^{n_u \times n_x}$ given by

$$F = -\left(D^\top \cdot D\right)^{-1} \cdot B^\top \cdot P \tag{2.22}$$

where P is the unique symmetric positive semidefinite solution of the algebraic Riccati equation

$$A^\top \cdot P + P \cdot A - P \cdot B \cdot \left(D^\top \cdot D\right)^{-1} \cdot B^\top \cdot P + C^\top \cdot C = 0. \tag{2.23}$$

Similar to the LQR case, we again have to be careful to use the positive definite solution of the algebraic Riccati equation. The approach to evalute the H_2 feedback is almost identical to the LQR case:

Algorithm 2.23 (Computation of H_2 controller)
Consider an H_2 problem

$$\min\ J(\mathbf{x}_0, \mathbf{u}) = \|H\|_2^2 = \sum_{j=1}^{n_u} \int_0^\infty \mathbf{y}(t)^\top \cdot \mathbf{y}(t) dt \quad \text{over all } \mathbf{u}(t) = e_j \cdot \delta(t) \tag{2.24}$$

$$\begin{aligned} \text{subject to } \dot{\mathbf{x}}(t) &= A \cdot \mathbf{x}(t) + B \cdot \mathbf{u}(t), \quad \mathbf{x}(t_0) = \mathbf{x}_0 \\ \mathbf{y}(t) &= C \cdot \mathbf{x}(t) + D \cdot \mathbf{u}(t) \end{aligned} \tag{2.25}$$

to be given. Then we obtain the H_2 feedback F via

1. Compute a semipositive definite solution P of the algebraic Riccati equation (2.23)

$$A^\top \cdot P + P \cdot A - P \cdot B \cdot \left(D^\top \cdot D\right)^{-1} \cdot B^\top \cdot P + C^\top \cdot C = 0.$$

2. Compute the optimal linear feedback F via (2.22)

$$F = -\left(D^\top \cdot D\right)^{-1} \cdot B^\top \cdot P.$$

Remark 2.24
Note that in the LTI case we have that

$$\begin{aligned} J(\mathbf{x}_0, \mathbf{u}) &= \sum_{j=1}^{n_u} \int_0^\infty \mathbf{y}(t)^\top \cdot \mathbf{y}(t) dt \\ &= \sum_{j=1}^{n_u} \int_0^\infty \left(C \cdot \mathbf{x}(t) + D \cdot \mathbf{u}(t)\right)^\top \cdot \left(C \cdot \mathbf{x}(t) + D \cdot \mathbf{u}(t)\right) dt. \end{aligned}$$

If we choose $C = Q^{\frac{1}{2}}$ and $D = R^{\frac{1}{2}}$, then we obtain that H_2 is a special case of LQR.

Similar to LQR, we obtain the (dis-)advantages shown in Table 2.2.

Table 2.2.: Advantages and disadvantages of H_2 control

Advantage	Disadvantage
✓ Optimal in L_2 norm	✗ Requires Riccati equation
✓ Focuses on I/O systems	✗ Disregards dynamics

2.3. H_∞ control

The idea of the H_∞ feedback is similar to the H_2 feedback. Instead of the L_2 norm, where the aim is to minimize the deviation of the output along the trajectory, in the H_∞ case the supremum norm is used to minimize the highest deviation.

Definition 2.25 (L_∞ norm).
Consider a function $v : \mathbb{R} \to \mathbb{R}^{n_y}$. Then we call

$$\|v\|_\infty = \sup_t \|v(t)\| \tag{2.26}$$

the L_∞ norm of the function. If

$$V(s) := \hat{v}(s) = \mathcal{L}(f(t)) = \int_0^\infty \exp(-st) \cdot f(t) dt, \qquad s = \alpha + i\omega$$

denotes the Laplace transform of v, then we call

$$\|\mathbf{y}\|_\infty = \sup_v \left\{ \frac{\|G(i\omega) \cdot v\|}{\|v\|} \mid v \neq 0,\ v \in \mathbb{C}^{n_y} \right\} \tag{2.27}$$

the L_∞ norm of the transform.

Again, the terms L_∞ and H_∞ are used identically in the literature. In the case of H_∞, we will not go into deep but only highlight connections to H_2. The first connection is about conservatism of the controllers. Since we have

$$\begin{aligned}\|G \cdot v\|_2 &= \left(\int_{-\infty}^{\infty} \|G(i\omega) \cdot v(i\omega)\|^2 d\omega \right)^{\frac{1}{2}} = \left(\int_{-\infty}^{\infty} \|G(i\omega)\|^2 \cdot \|v(i\omega)\|^2 d\omega \right)^{\frac{1}{2}} \\ &\leq \sup_\omega \left(\sigma\left(G(i\omega)\right)\right) \cdot \left(\int_{-\infty}^{\infty} \|v(i\omega)\|^2 d\omega \right)^{\frac{1}{2}} = \|G\|_\infty \cdot \|v\|_2\end{aligned}$$

where $\sigma(\cdot)$ denotes the maximal singular value, we obtain

$$\|G\|_\infty \geq \frac{\|G \cdot v\|_2}{\|v\|_2} \qquad \forall v \neq 0.$$

This can be interpreted as the concentrated impact of v close to the frequency range of $\|G\|_\infty$. Hence, the H_∞ norm gives the maximum factor by which the system magnifies the H_2 norm of any input. For this reason, $\|G\|_\infty$ is also referred to as gain of the system.

Remark 2.26
As a consequence, the H_∞ feedback is always more conservative than the H_2 feedback as it aims to hold down the maximal amplification.

Using the H_∞ norm, we define the H_∞ problem similar to the H_2 problem:

Definition 2.27 (H_∞ problem).
Consider the optimal control problem given by the LTI system (1.5) and the cost functional from (2.18)

$$J(\mathbf{x}_0, \mathbf{u}) := \|H\|_\infty^2 = \sup_t \sum_{j=1}^{n_u} \|\mathbf{y}(t)\|^2 \tag{2.28}$$

to be minimized over all $\mathbf{u}(t) = e_j \cdot \delta(t)$ where $\delta(\cdot)$ is the Dirac delta function. Then we refer to this setting as H_∞ *problem*.

Regarding the solutions, again an algebraic Riccati equation is employed and we obtain:

Theorem 2.28 (H_∞ feedback).
Consider the H_∞ problem and suppose $\|H_\infty\| < \gamma$ to be finite. Then the feedbac

$$\mathbf{u}^\star(t) = F \cdot \mathbf{x}(t, \mathbf{x}^\star, F) \tag{2.29}$$

asymptotically stablizes the system with feedback matrix $F \in \mathbb{R}^{n_u \times n_x}$ given by

$$F = -\left(D^\top \cdot D\right)^{-1} \cdot B^\top \cdot P \tag{2.30}$$

where P is the unique symmetric positive semidefinite solution of the algebraic Riccati equation

$$A^\top \cdot P + P \cdot A - P \cdot B \left(D^\top \cdot D\right)^{-1} B^\top P + \gamma^{-2} \cdot P \cdot B \cdot B^\top \cdot P + C^\top \cdot C = 0. \tag{2.31}$$

Different to LQR and H_2, we can summarize (dis-)advantages in the following Table 2.3.

Table 2.3.: Advantages and disadvantages of H_∞ control

Advantage	**Disadvantage**
✓ Optimal in L_∞ norm	✗ May be conservative
✓ Focuses on maximal gain	✗ Requires Riccati equation

CHAPTER 3

OPTIMAL OBSERVATION

In the previous chapters, we discussed stability as a system property and how we can manage to ensure that a system is asymptotically stable by computing a feedback law. The feedback, however, is based on the state of the system $\mathbf{x}$. Since typically not all states are actually measured but instead only a restricted output $\mathbf{y}$ is known, the feedback cannot be evaluated.
To complete this gap in this chapter, we shift our focus to the task of estimating the state $\mathbf{x}$ based on the output $\mathbf{y}$. Similar to the LQR approach from Section 2.1, the aim is to derive a method that provides us with an optimal state estimation $\hat{\mathbf{x}}(t) \approx \mathbf{x}(t)$ and can be applied in realtime. The latter requirement rules out all aposteriori methods minimizing over given data sets, but instead forces a recursive approach. Recursive means that estimates from previous time instances are re-used and are updated using newly acquired output data. Such methods are typically referred to as observers or filters.

3.1. Recursive estimation

A typical estimation problem is given by set of data, a model of a system and a set of parameters which shall be estimated. To illustrate the impact of the realtime requirement, we consider the following example.

Task 3.1 (Mean value computation)
Suppose outputs $\mathbf{y}(j)$, $j = 1, \ldots, N$ *to be given. Calculate the mean of the outputs.*

Solution to Task 3.1: The estimate of the mean $\hat{\mathbf{y}}$ based on N outputs is given by

$$\hat{\mathbf{y}}(N) = \frac{1}{N}\sum_{j=1}^{N}\mathbf{y}(j).$$

The difficulty now arises if another output is available and the mean computation shall be updated.

Task 3.2 (Mean value update)
Consider the result from Task 3.1 to be given and a output $\mathbf{y}(N+1)$ *to be available. Compute the mean of the outputs.*

Solution to Task 3.2: Again, the mean is given by

$$\hat{\mathbf{y}}(N+1) = \frac{1}{N+1}\sum_{j=1}^{N+1}\mathbf{y}(j).$$

In this solution, the previous result from Solution 3.1 is not used. While such an approach is numerically robust and requires no further insight, it may be computationally expensive depending on the number of samples and the complexity of the computation process. Hence, reformulating the problem such that only the newly required calculations are made, recuperating all the previous results, may allow us to generate a more efficient solution method.

Task 3.3 (Real mean value update)
Consider the setting of Task 3.2. Reuse the results from Solution 3.1 to compute the mean value.

Solution to Task 3.3: To recuperate the previous sum, we can equivalently evaluate

$$\begin{aligned}\hat{\mathbf{y}}(N+1) &= \frac{1}{N+1}\sum_{j=1}^{N}\mathbf{y}(j) + \frac{1}{N+1}\mathbf{y}(N+1)\\ &= \frac{N}{N+1}\hat{\mathbf{y}}(N) + \frac{1}{N+1}\mathbf{y}(N+1).\end{aligned}$$

Although this form already meets our requirements of reusing previous computations, it is possible to rearrange it to a more suitable expression:

$$\hat{\mathbf{y}}(N+1) = \hat{\mathbf{y}}(N) + \frac{1}{N+1}\left(\mathbf{y}(N+1) - \hat{\mathbf{y}}(N)\right)$$

Although this expression is very simple, it is very informative because almost every recursive algorithm can be reduced to a similar form. Based on the latter, the following observations can be made:

- The new estimate $\hat{\mathbf{y}}(N+1)$ equals the old estimate $\hat{\mathbf{y}}(N)$ plus a correction term, that is $\frac{1}{N+1}\left(\mathbf{y}(N+1) - \hat{\mathbf{y}}(N)\right)$.
- The correction term consists of two terms by itself: a gain factor $\frac{1}{N+1}$ and an error term.
- The gain factor decreases towards zero as more outputs are already accumulated in the previous estimate. This means that in the beginning of the experiment, less importance is given to the old estimate $\hat{\mathbf{y}}(N)$, and more attention is paid to the new incoming outputs. When N starts to grow, the error term becomes small compared to the old estimate. The algorithm relies more and more on the accumulated information in the old estimate $\hat{\mathbf{y}}(N)$ and it does not vary it that much for accidental variations of the new outputs. The additional bit of information in the new output becomes small compared with the information that is accumulated in the old estimate.
- The second term $\mathbf{y}(N+1) - \hat{\mathbf{y}}(N)$ is an error term. It incorporates the difference between the predicted value of the next output on the basis of the model and the output $\mathbf{y}(N+1)$.
- When properly initiated, i.e. $\hat{\mathbf{y}}(1) = \mathbf{y}(1)$, this recursive result is exactly equal to the non recursive implementation. However, from a numerical point of view, it is a very robust procedure as calculation errors etc. are compensated in each step.

3.2. Transformation of dynamics

To derive the general optimal observation problem, we consider the nonlinear system

$$\begin{aligned} \dot{\mathbf{x}}(t) &= f(\mathbf{x}(t), \mathbf{u}(t), t), \quad \mathbf{x}(t_0) = \mathbf{x}_0 \\ \mathbf{y}(t) &= h(\mathbf{x}(t), \mathbf{u}(t), t). \end{aligned} \tag{1.2}$$

together with the known control $\mathbf{u}(t)$, $t \geq 0$, given outputs $\mathbf{y}(t)$, $t \geq 0$ and an estimate $\hat{\mathbf{x}}_0$ of the *unknown* initial state $\mathbf{x}_0$.
Depending on the time instant of interest, we can classify the following problem classes:

Definition 3.4 (Filtering).
Consider $\mathbf{x}(\cdot)$ to be a state trajectory of a system. Given a specific time instant t, we call the problem of computing

- $\mathbf{x}(\tau)$ with $\tau < t$ an interpolation problem,
- $\mathbf{x}(\tau)$ with $\tau = t$ a filtering problem, and
- $\mathbf{x}(\tau)$ with $\tau > t$ an prediction (or extrapolation) problem.

Within this chapter, we are interested in computing realtime estimates, i.e. $\tau = t$ and therefore work in the area of filtering problems. To solve the latter we apply the ansatz using the so called estimator dynamics:

Definition 3.5 (Estimator dynamics).
Given a system (1.2), we call

$$\dot{\hat{\mathbf{x}}}(t) = f(\hat{\mathbf{x}}(t), \mathbf{u}(t), t) + \mathbf{d}(t), \qquad \hat{\mathbf{x}}(0) = \hat{\mathbf{x}}_0 \tag{3.1}$$

estimator dynamics where $\mathbf{d} : \mathbb{R} \to \mathbb{R}^{n_x}$.

Based on the latter, we can quantify the mismatch between estimator and true system:

Definition 3.6 (Error function).
Consider a system (1.2) and an estimator (3.1). Then we call $e : \mathbb{R} \times \mathcal{X} \to \mathbb{R}^{n_x}$ with

$$e(t, \hat{\mathbf{x}}_0) := \hat{\mathbf{x}}(t, \hat{\mathbf{x}}_0, \mathbf{u}) - \mathbf{x}(t, \mathbf{x}_0, \mathbf{u}) \tag{3.2}$$

error function of the estimator.

Similar to the optimal control problem, we can now define the optimal estimation problem. Yet, in contrast of finding an optimal input $\mathbf{u}(\cdot)$, we aim to find an estimator $\hat{\mathbf{x}}(\cdot)$ such that the estimated error (3.2) becomes as small as possible in the sense of a key performance indicator. Moreover, at time t the estimator shall be computable based on outputs $\mathbf{y}(\tau)$, $0 \leq \tau \leq t$ known at time t only.

Similar to the cost function for the control problem where the idea of the cost is to induce stability via null-controlling, we formulate a cost function for the estimator using the error function. Here, the idea is to use the null-controlling property to enforce stability of the error function and thereby convergence of the estimator.

Definition 3.7 (Cost functional).
Consider a key performance criterion $\ell : \mathcal{X} \times \mathcal{U} \to \mathbb{R}_0^+$. Then we call

$$J(\hat{\mathbf{x}}_0, \mathbf{u}) := \int_0^\infty \ell(e(t, \hat{\mathbf{x}}_0), \mathbf{u}(t))dt \tag{3.3}$$

cost functional.

This gives us

Definition 3.8 (Optimal estimation problem).
Consider a system (1.2) and a cost functional (2.1). Then we call

$$\begin{aligned} \min\ J(\hat{\mathbf{x}}_0, \mathbf{u}) = \int_0^\infty \ell(e(t, \hat{\mathbf{x}}_0), \mathbf{u}(t))dt \quad \text{over all } \mathbf{u} \in \mathcal{U} \\ \text{subject to } e(t, \hat{\mathbf{x}}_0) := \hat{\mathbf{x}}(t, \hat{\mathbf{x}}_0, \mathbf{u}) - \mathbf{x}(t, \mathbf{x}_0, \mathbf{u}) \\ \dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t), t), \quad \mathbf{x}(0) = \mathbf{x}_0 \\ \dot{\hat{\mathbf{x}}}(t) = f(\hat{\mathbf{x}}(t), \mathbf{u}(t), t) + \mathbf{d}(t), \qquad \hat{\mathbf{x}}(0) = \hat{\mathbf{x}}_0 \end{aligned} \tag{3.4}$$

an *optimal estimation problem*.

Note that we can use this problem to directly transfer the null controlling property from Corollary 2.6 for stability to observability. In this case, not the system but the error function of the estimation is stabilized.

Corollary 3.9 (Null controlling observability).
If a optimal estimation problem is null controlling, then the system is observable.

The latter result suggests that the solution of the optimal estimation problem from Definition 3.8 could be identical to the optimal control problem from Definition 2.4. Unfortunately, there are some slight differences:

1. In the optimal control problem, we consider the state to be stabilized, while in the optimal estimation problem the error needs to be stabilized.

2. The solution computed by the optimal control problem is the control strategy, which in the LTI case can be evaluated by a linear feedback law. For the optimal estimation problem, we aim to compute the current state of the problem.

3. Last, the given data for the optimal estimation problem stems from past measurements, which cannot be used in the formulation of the optimal control problem.

In the following, we will address the integration problem of measurements from the past by converting the optimal estimation problem. Then, similar to LQR, our aim now is to derive a problem, for which the null controlling property can be shown.

3.3. Kalman filter

We now focus on the LTI case, where not only the dynamics are much more simple, but we can also derive an explicit dynamics for the error function of the estimator. More precisely, for the LTI case

$$\dot{\mathbf{x}}(t) = A \cdot \mathbf{x}(t) + B \cdot \mathbf{u}(t) \tag{3.5a}$$
$$\mathbf{y}(t) = C \cdot \mathbf{x}(t) + D \cdot \mathbf{u}(t) \tag{3.5b}$$

with estimator

$$\dot{\hat{\mathbf{x}}}(t) = A \cdot \hat{\mathbf{x}}(t) + B \cdot \mathbf{u}(t) + \mathbf{d}(t), \qquad \hat{\mathbf{x}}(0) = \hat{\mathbf{x}}_0 \tag{3.6}$$

we obtain:

Definition 3.10 (Error dynamics).
Given an LTI system (1.5) with estimator dynamics (3.6) we call

$$\dot{e}(t) = A \cdot e(t) + \mathbf{d}(t) \tag{3.7a}$$
$$\mathbf{y}_e(t) = C \cdot e(t) \tag{3.7b}$$

error dynamics.

Remark 3.11
The error dynamics are the dual wrt. the LTI system, cf. Definition 1.31. Hence, controlability of the dual system gives us observability of the primal system.
As a consequence, all the following computations can only be executed if the system (A, C) is observable. Otherwise, no solution can be computed.

Based on the error dynamics, we can integrate the measurements, which are available for past time instances. Hence, the cost functional we design aims to drive the error to zero but operates on a time frame, which leads up to the current time instant.

Definition 3.12 (Quadratic cost functional for observability).
We call

$$J(\hat{\mathbf{x}}_0, \mathbf{d}) := \int_{-\infty}^{\tau} (C \cdot e(t) - \mathbf{y}_e(t))^\top \cdot \hat{Q} \cdot (C \cdot e(t) - \mathbf{y}_e(t)) + \mathbf{d}(t)^\top \cdot R \cdot \mathbf{d}(t) dt \tag{3.8}$$

quadratic cost functional for observability where $\hat{Q} \in \mathbb{R}^{n_y \times n_y}$ and $R \in \mathbb{R}^{n_u \times n_u}$ are (semi)positive definite matrices.

In order to convert the cost functional (3.8) to be in the form (3.3), we apply the following:

Theorem 3.13 (Time transformation).
Consider an LTI system (3.7) *with cost functional* (3.8) *to be given. Given the transformation*

$$\mathbf{x}^\tau(t, \mathbf{x}_0, \mathbf{d}) := \mathbf{x}(\tau - t, \mathbf{x}_0, \mathbf{d}) \tag{3.9}$$

$$\mathbf{y}_e^\tau(t) := \mathbf{y}_e(\tau - t) \tag{3.10}$$

the cost function (3.8) *is equivalent to*

$$J^\tau(\hat{\mathbf{x}}_0, \mathbf{d}) := \int_{0}^{\infty} (C \cdot e^\tau(t) - \mathbf{y}_e^\tau(t))^\top \cdot \hat{Q} \cdot (C \cdot e^\tau(t) - \mathbf{y}_e^\tau(t)) + \mathbf{d}(t)^\top \cdot R \cdot \mathbf{d}(t) dt \tag{3.11}$$

and the respective error dynamics is equivalent to

$$\dot{e}^\tau(t) = -A \cdot e^\tau(t) - \mathbf{d}(\tau - t) \tag{3.12a}$$

$$\mathbf{y}_e^\tau(t) = C \cdot e^\tau(t). \tag{3.12b}$$

Definition 3.14 (Kalman filter problem).
Consider an LTI system (3.7) and outputs $\mathbf{y}(t)$, $t \in (-\infty, \tau]$ to be given. Then we call

$$\begin{aligned} &\min\ J^\tau(\hat{\mathbf{x}}_0, \mathbf{d}) := \int_0^\infty (C \cdot e^\tau(t) - \mathbf{y}_e^\tau(t))^\top \cdot \hat{Q} \cdot (C \cdot e^\tau(t) - \mathbf{y}_e^\tau(t)) + \mathbf{d}(t)^\top \cdot R \cdot \mathbf{d}(t) dt \\ &\text{over all } \mathbf{x}_0 \in \mathcal{X} \\ &\text{subject to } \dot{e}^\tau(t) = -A \cdot e^\tau(t) - \mathbf{d}(\tau - t), \quad e^\tau(0) = \hat{\mathbf{x}}_0 - \mathbf{x}_0 \\ &\qquad\qquad \mathbf{y}_e^\tau(t) = C \cdot e^\tau(t) \end{aligned} \tag{3.13}$$

Kalman filter problem.

Now, we can impose the identical ansatz

$$V(e) = e^{\tau\top} \cdot P \cdot e^\tau \tag{3.14}$$

for $P \in \mathbb{R}^{n_x \times n_x}$. If this ansatz is right, we obtain the following:

Theorem 3.15 (Kalman filter).
Consider an LTI system (3.7) *with cost functional* (3.8) *to be given. Then the solution of the optimal estimation problem is given by*

$$\dot{e^\tau}(\tau) = A \cdot e^\tau(\tau) + L \cdot (C \cdot e^\tau(\tau) - \mathbf{y}_e(\tau)) \tag{3.15}$$

where the gain matrix

$$L := -S \cdot C^\top \cdot \hat{Q} \tag{3.16}$$

is solution of the dual Riccati equation

$$A \cdot S + S \cdot A^\top - S \cdot C^\top \cdot \hat{Q} \cdot C \cdot S + D \cdot R^{-1} \cdot D^\top = 0 \tag{3.17}$$

and the value function of the optimal estimation problem is given by (3.14) *with* $P := S^{-1}$.

Remark 3.16
In (3.15) *we obtain the identical structure of the observer, which we designed in Task 3.3 for the mean value update. For this reason, L is also called gain matrix.*

Note that again a solution P of the dual Riccati equation (3.17) is not unique, yet there exists at most one semi positive definite S. Combining the latter results, we obtain the following procedure to compute the Kalman filter:

Algorithm 3.17 (Computation of Kalman filter)
Consider an Kalman filter problem (3.13) to be given. Then we obtain the solution via

1. Compute a semipositive definite solution S of the dual Riccati equation (3.17)

$$A \cdot S + S \cdot A^\top - S \cdot C^\top \cdot \hat{Q} \cdot C \cdot S + D \cdot R^{-1} \cdot D^\top = 0$$

2. Compute the gain matrix L via (3.16)

$$L := -S \cdot C^\top \cdot \hat{Q}.$$

In practice, a Kalman filter is typically updated periodically, i.e. a dynamic for computing the ansatz matrix S is applied to integrate newly obtained knowledge of outputs. S is also called covariance matrix of the system. In the literature, the dynamic of this matrix is split into an apriori and an aposteriori covariance update as well as an prediction and an correction step of the error dynamics, cf., e.g., [6]. As we focus on continuous time dynamics, this separation is beyond the scope of the lecture.

Part II.

Nonlinear systems

CHAPTER 4

DIGITALIZATION

Generated with chatgpt.com

> If you automate chaos, you get faster chaos.
>
> Bill Gates

To deal with nonlinear systems, we follow a so called direct approach, which is quite different from the direct approach we considered in *Control engineering 2*. Instead of analytically or structurally dealing with the system or its solution, we first transfer the problem into the sphere of digital control problems and than apply optimization to compute a control strategy.
In the present chapter, we focus on the first step and digitize the control system. Here, we follow the most simple approach and consider a so called zero order hold. At this point, we already like to stress that by definition such a control is not Lipschitz continuous. Hence, the feedback will be very different from the ones we considered in *Control engineering 2* and in particular will not be in the form of a function. Moreover, we don't aim to compute a feedback which is stabilizing for all possible digitizations. Instead, we suppose a sampling to be given and then derive a stabilizing controller.
To conclude stability of the original system, in the present chapter we additionally discuss how stability of the digital feedback can be guaranteed for the original system as well. Throughout the nonlinear part of the lecture, we focus on systems of the form

$$\begin{aligned} \dot{\mathbf{x}}(t) &= f(\mathbf{x}(t), \mathbf{u}(t), t), \quad \mathbf{x}(t_0) = \mathbf{x}_0 \\ \mathbf{y}(t) &= h(\mathbf{x}(t), \mathbf{u}(t), t). \end{aligned} \tag{1.2}$$

In the upcoming chapters, we will then design methods to compute and evaluate control laws, which provide us with a stabilizing feedback for the digitized system.

4.1. Zero order hold

The most simple case of a discontinuous feedback is given by the so called zero order hold. The idea is to sample the input, i.e. to fix a time grid $\mathcal{T} := \{t_k\} \subset \mathbb{R}$ and define the input to be constant in between two sampling instances t_k and t_{k+1}. Here, we further simplify the setting by introducing a sampling period T and define the sampling instances to be equidistant, which we already discussed in

$$\mathcal{T} := \{t_k \mid t_k := t_0 + k \cdot T\} \subset \mathbb{R}. \tag{1.4}$$

Remark 4.1
There are two more general cases: For one, the sampling times may be defined by a function of time, or secondly, the sampling times can be defined by a function of states. The first one is common in prediction and prescription of systems where action is the far future are significantly less important. Hence, one typically chooses between exactness of the prediction and computational complexity. The latter case is referred to a event driven control.

We still like to stress that in applications, the choice of T is not fixed right from the beginning, but depends on the obtainable solution and stability properties. Note that the result of sampling the control is not a discrete time system (see Definition 1.7), but a continuous time system (see Definition 1.5) where the input $\mathbf{u}$ is of zero order hold. More formally, we formulate zero order hold input and solution as a parametrization of operators with respect to T.

Definition 4.2 (Zero order hold).
Consider a nonlinear control system (1.2) and a feedback $\mathbf{u} : \mathcal{X} \to \mathcal{U}$ such that $\|\mathbf{u}(\mathbf{x})\| \leq \gamma(\mathbf{x})$ holds for all $\mathbf{x} \in \mathcal{X}$ and some continuous function $\gamma : \mathcal{X} \to \mathbb{R}$. Moreover suppose a *sampling period* $T > 0$ to be given, which defines the sampling times $t_k = k \cdot T$. Then we call the piecewise constant function

$$\mathbf{u}_T(t) \equiv \mathbf{u}(\mathbf{x}(t_k)), \qquad t \in [t_k, t_{k+1}) \tag{4.1}$$

zero order hold.

Remark 4.3
We like to point out that higher order holds are possible as well. In practice, however, such higher order holds are not defined using a polynomial approximation but via additional differential equations for the control itself.

As a consequence of the latter definition, the input $\mathbf{u}_T$ is not continuous but instead exhibits jumps at the sampling times t_k, cf. Figure 4.1.
Still, the function is integrable, which is a requirement for existence of a solution of (1.2) for such an input. This insertion directly leads to the following:

Definition 4.4 (Zero order hold solution).
Given a nonlinear control system (1.2) and a zero order hold input $\mathbf{u}_T : \mathcal{T} \to \mathcal{U}$. Then we call the function $\mathbf{x}_T : \mathcal{T} \to \mathcal{X}$ satisfying

$$\dot{\mathbf{x}}_T(t) = f(\mathbf{x}_T(t), \mathbf{u}_T(t)) \tag{4.2}$$

zero order hold solution.

In order to compute such a solution, we can simply concatenate solutions of subsequent sampling intervals $[t_k, t_{k+1})$. Here, we can use the endpoint of the solution on one sampling interval to be the initial point on the following one. Hence, the computation of $\mathbf{x}_T$ is well defined, cf. Figure 4.2 for an illustration.

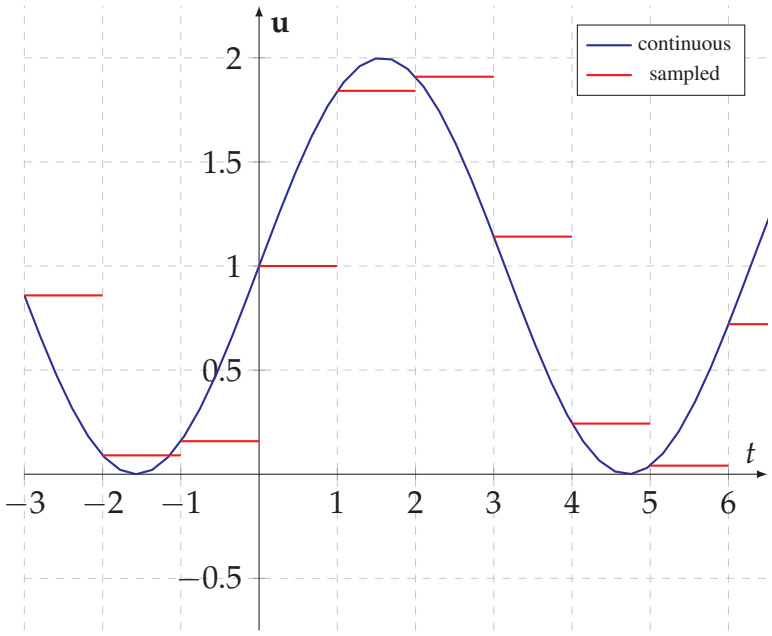


Figure 4.1.: Zero order hold sampling

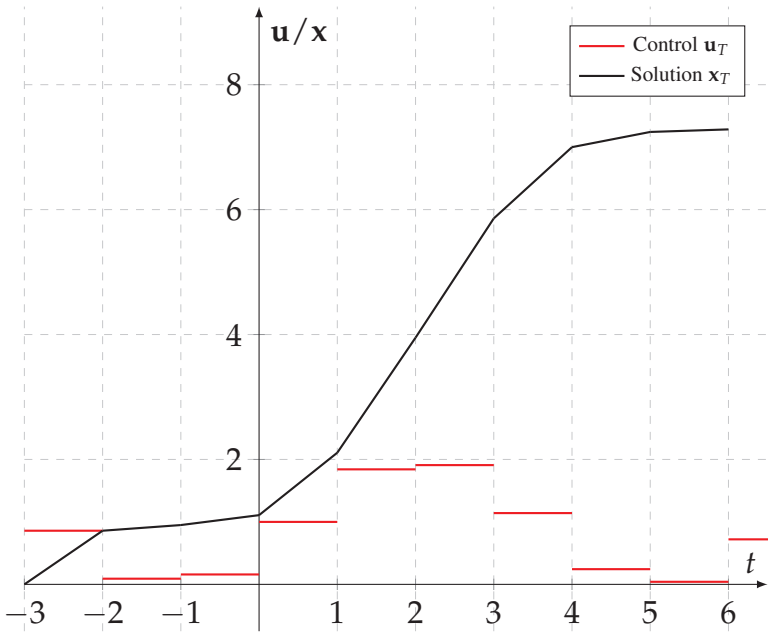


Figure 4.2.: Zero order hold solution

Remark 4.5
Since the system is Lipschitz continuous on each interval $[t_k, t_{k+1})$*, the solution is also unique. Hence, identifying endpoint and initial point of subsequent sampling intervals is sufficient to show that the zero order hold solution is unique. Yet, as a consequence of this concatenation, the*

solution is not differentiable at the sampling points t_k.

Remark 4.6
Note that despite $\mathbf{u}_T$ to be piecewise constant, the zero order hold solution does not exhibit jumps and shows nonlinear behavior.

4.2. Practical stability

We next introduce the concept of stability, which is equivalent to Definition 1.13. To this end, we utilize the so called practical $\mathcal{KL}$ notation, which extends the standard $\mathcal{KL}$ concept using comparison functions to cases where convergence can only be guaranteed to a certain neighborhood. For the stability concept, we use the same simplification as in Chapter 1 to shift the operating point $(\mathbf{x}^\star, \mathbf{u}^\star)$ to the origin.

Definition 4.7 (Practical stability/controllability).
Consider a nonlinear control system (1.2) with $f(0,0) = 0$ and $T > 0$. Then we call a feedback $\mathbf{u}_T$ to *semiglobally practically asymptotically stabilize* the operating point $(\mathbf{x}^\star, \mathbf{u}^\star) = (0,0)$ if there exists a function $\beta \in \mathcal{KL}$ and constants $R > \varepsilon > 0$ such that

$$\|\mathbf{x}_T(t)\| \leq \max\{\beta(\|\mathbf{x}_0\|, t), \varepsilon\} \tag{4.3}$$

holds for all $t > 0$ and all initial value satisfying $\|\mathbf{x}_0\| \leq R$.

Again, the main difference between our setting here and in *Control engineering 2* is that we don't aim to compute a feedback which is stabilizing for all $T \in (0, T^\star]$. Instead, we suppose a sampling to be given and then derive a stabilizing controller.

Remark 4.8
The term „semiglobal“ refers to the constant R, which limits the range of the initial states for which stability can be concluded. The term „practical“ refers to the constant ε, which is a measure on how close the solution can be driven towards the operating point before oscillations as in the case of the bang bang controller occur.

Different from the linear case where existence of a feedback and a feed forward control are equivalent, in the nonlinear case we only have the following:

Lemma 4.9 (Existence of feed forward).
Consider a system (1.2) *and let* $(\mathbf{x}^\star, \mathbf{u}^\star)$ *be an operating point. If a feedback* $\mathbf{u} : \mathcal{X} \to \mathcal{U}$ *exists such that the closed loop is asymptotically stable and additionally both the feedback and the closed loop are Lipschitz, then there exists a feed forward* $\mathbf{u} : \mathcal{T} \to \mathcal{U}$ *such that the system is asymptotically controllable.*

As a direct conclusion of Definition 4.7, we can apply Lemma 4.9 and obtain:

Corollary 4.10 (Existence of practically stabilizing feed forward).
Consider a nonlinear control system (1.2) *with* $f(0,0) = 0$ *and suppose a feedback* $\mathbf{u}_T$, $T > 0$ *to exist, which semiglobally practically asymptotically stabilizes the operating point* $(\mathbf{x}^\star, \mathbf{u}^\star) = (0,0)$. *Then there exists a feed forward* $\mathbf{u} : \mathcal{T} \to \mathcal{U}$ *such that the system is practically asymptotically controllable.*

Definition 4.7 also shows the dilemma of digital control using fixed sampling periods: Both close to the desired operating point and for initial values far away from it, the discontinuous evaluation of the feedback $\mathbf{u}_T$ leads to performance degradation. Close to the operating point, a slow evaluation leads to overshoots even if the dynamics is typically rather slow. Far away from the operating point, the dynamics is too fast to be captured in between two sampling points which leads to unstable behavior.
Still, it may even be possible to obtain asymptotic stability (not only practical asymptotic stability) using fixed sampling periods T as shown in the following task:

Task 4.11
Consider the system

$$\begin{aligned} \dot{x}_1(t) &= \left(-x_1(t)^2 + x_2(t)^2\right) \cdot u(t) \\ \dot{x}_2(t) &= (-2 \cdot x_1(t) \cdot x_2(t)) \cdot u(t). \end{aligned}$$

Design a zero order hold control such that the system is practically asymptotically stable.

Solution to Task 4.11: We set

$$\mathbf{u}_T(t) = \begin{cases} 1, & x_1 \geq 0 \\ -1, & x_1 < 0 \end{cases}.$$

For this choice, the system is globally asymptotically stable for all $T > 0$ and even independent from T. The reason for the latter is that the solutions never cross the switching line $x_1 = 0$, i.e. the input to be applied is always constant, which leads to independence of the feedback from T.

As described before, the behavior observed in Task 4.11 is the exception. In practice, the limitations of semiglobality and practicality is typically the best we can expect in zero order hold input of nonlinear system.

4.3. Existence of stabilizing feedback

In order to show that a stabilizing zero order hold input exists, we utilize the concept of Control-Lyapunov functions, which extend the standard Lyapunov approach.

Definition 4.12 (Practical Control-Lyapunov functions).
Consider a nonlinear control system (1.2) with operating point $(\mathbf{x}^\star, \mathbf{u}^\star) = (0,0)$ such that $f(\mathbf{x}^\star, \mathbf{u}^\star) = 0$ and a neighborhood $\mathcal{N}(\mathbf{x}^\star)$. Then the continuous function $V_T : \mathbb{R}^{n_x} \to \mathbb{R}_0^+$ is called a *semiglobal practical Control-Lyapunov function* if there exist constants $\hat{R} > \hat{\varepsilon} > 0$ as well as functions $\alpha_1, \alpha_2, \in \mathcal{K}_\infty$ and a continuous function $W : \mathcal{X} \to \mathbb{R}^+ \setminus \{0\}$ such that there exists a control function $\mathbf{u}$ satisfying the inequalities

$$\alpha_1(\|\mathbf{x}\|) \leq V_T(\mathbf{x}) \leq \alpha_2(\|\mathbf{x}\|) \tag{4.4}$$

$$\inf_{\mathbf{u}\in\mathcal{U}} V_T(\mathbf{x}_T(t_{k+1})) \leq \max\{V_T(\mathbf{x}_T(t_k) - T \cdot W(\mathbf{x}_T(t_k)), \hat{\varepsilon})\} \tag{4.5}$$

for all $\mathbf{x} \in \mathcal{N} \setminus \{\mathbf{x}^\star\}$ with $V_T(\mathbf{x}) \leq \hat{R}$.

The latter definition extends the concepts of a Control-Lyapunov function is various ways. For one, as the zero order hold solution is not differentiable, we can no longer assume V_T to be differentiable. Hence, the formulation of decrease in energy in inequality (4.5) is given along a solution instead of its vector field. Moreover, the ideas of semiglobality and practicality are integrated.

Remark 4.13
Comparing Definition 4.12 to Definition 4.7, we can identify the similarity of semiglobality between the constants R and $\hat{R}$ as well as ε and $\hat{\varepsilon}$. The difference between these two pairs lies in their interpretation: For $\mathcal{KL}$ function, we utilize the state space, whereas for Control-Lyapunov

functions the energy space is used. Hence, both values are a transformation of one another using the Control-Lyapunov function V_T.

Now, supposingly that a practical Control-Lyapunov function exists, we can directly derive the existence of a zero order hold control.

Theorem 4.14 (Existence of feedback).
Consider a nonlinear control system (1.2) *with operating point* $(\mathbf{x}^\star, \mathbf{u}^\star) = (0,0)$ *such that* $f(\mathbf{x}^\star, \mathbf{u}^\star) = 0$ *and* $T > 0$. *Let* V_T *to be a semiglobal practical Control-Lyapunov function. Then the minimizer*

$$\mathbf{u}_T(t) := \underset{\mathbf{u}\in\mathcal{U}}{argmin}\, V_T(\mathbf{x}_T(t_{k+1})) \tag{4.6}$$

is a semiglobally practially asymptotically stabilizing feedback.

Note that in (4.6), the right hand side depends on $\mathbf{u}$ implicitly as $\mathbf{x}_T(t_{k+1})$ is defined using the initial value $\mathbf{x}_T(t_k)$ and the zero order hold input $\mathbf{u}$. Hence, the definition (4.6) is proper.

Remark 4.15
The transfer from infimum in (4.5) *to minimum in* (4.6) *is only possible as the input is constant in between two sampling instances t_k and t_{k+1} and therefore the solution $\mathbf{x}_T(\cdot)$ is continuous with respect to* $\mathbf{u}$.

Unfortunately, the pure existence of a feedback does not help us in computing it. Additionally, we still require the existence of a practical Control-Lyapunov function to conclude existence of such a feedback. Here, we first address existence of a Control-Lyapunov function, for which the following is known from the literature:

Theorem 4.16 (Existence of practical Control-Lyapunov function).
Consider a nonlinear control system (1.2) *with operating point* $(\mathbf{x}^\star, \mathbf{u}^\star) = (0,0)$ *such that* $f(\mathbf{x}^\star, \mathbf{u}^\star) = 0$. *If the system is asymptotic controllable, then there exists a semiglobal practical Control-Lyapunov function.*

The most important aspect of Theorem 4.16 is the requirement regarding the control system. The result does only require the system to be asymptotically controllable, i.e. without digitalization.

4.4. Intersample behavior

Unfortunately, the results only hold true for the digitized system, i.e. only for time instances $t_k \in \mathcal{T}$. The behavior of the system between these instances is called intersample behavior and can be estimated using properties of the system dynamics. The main tool is the so called uniform boundedness.

Definition 4.17 (Uniform boundedness).
Consider a nonlinear control system (1.2) with operating point $(\mathbf{x}^\star, \mathbf{u}^\star) = (0,0)$ together with a input $\mathbf{u}_T : \mathcal{T} \to \mathcal{U}$. If there exists a function $\gamma \in \mathcal{K}$ and a constant $\eta > 0$ such that for all $\mathbf{x} \in \mathcal{X}$ with $\|\mathbf{x}\| \leq \eta$, the solutions exist on $[0,T]$ and the solutions satisfy

$$\|\mathbf{x}_T(t)\| \leq \gamma(\|\mathbf{x}\|) \tag{4.7}$$

for all $t \in [0,T]$ then the solutions are called *uniformly bounded over* T.

Using boundedness, it can be shown that the system will stay bounded in between sampling instances.

Theorem 4.18 (Asymptotic stability and uniform boundedness over T).
Consider nonlinear control system (1.2) *with operating point* $(\mathbf{x}^\star, \mathbf{u}^\star) = (0,0)$ *together with a input* $\mathbf{u}_T : \mathcal{T} \to \mathcal{U}$. *Then the system is semiglobally practically asymptotically stable iff there exists a semiglobally practically asymptotically stabilizing feedback* $\mathbf{u}_T : \mathcal{T} \to \mathcal{U}$ *and the solutions* $\mathbf{x}_T : \mathcal{T} \to \mathcal{X}$ *are uniformly bounded over* T.

Concluding, if we can compute an semiglobally practically asymptotically stabilizing feedback law for the discrete time system induced by the sampled data system, then the digitizes continuous time closed loop is also semiglobally practically asymptotically stable provided its solutions are uniformly bounded over T.

In practice, however, the two tasks of deriving feedback $\mathbf{u}_T$ and Control-Lyapunov function V_T are often done in the inverse sequence. To this end, first a feedback $\mathbf{u}_T$ is derived, and then the inequality (4.5) is shown to hold for this feedback

$$V_T(\mathbf{x}_T(t_{k+1})) \leq \max\left\{V_T(\mathbf{x}_T(t_k) - T \cdot W(\mathbf{x}_T(t_k)), \hat{\varepsilon}\right\}.$$

The reason for using such a procedure is that Theorem 4.14 only requires a Control-Lyapunov function for fixed $\hat{R}$, $\hat{\varepsilon}$ to exists for some $T_0 > 0$ in order to conclude existence also for all

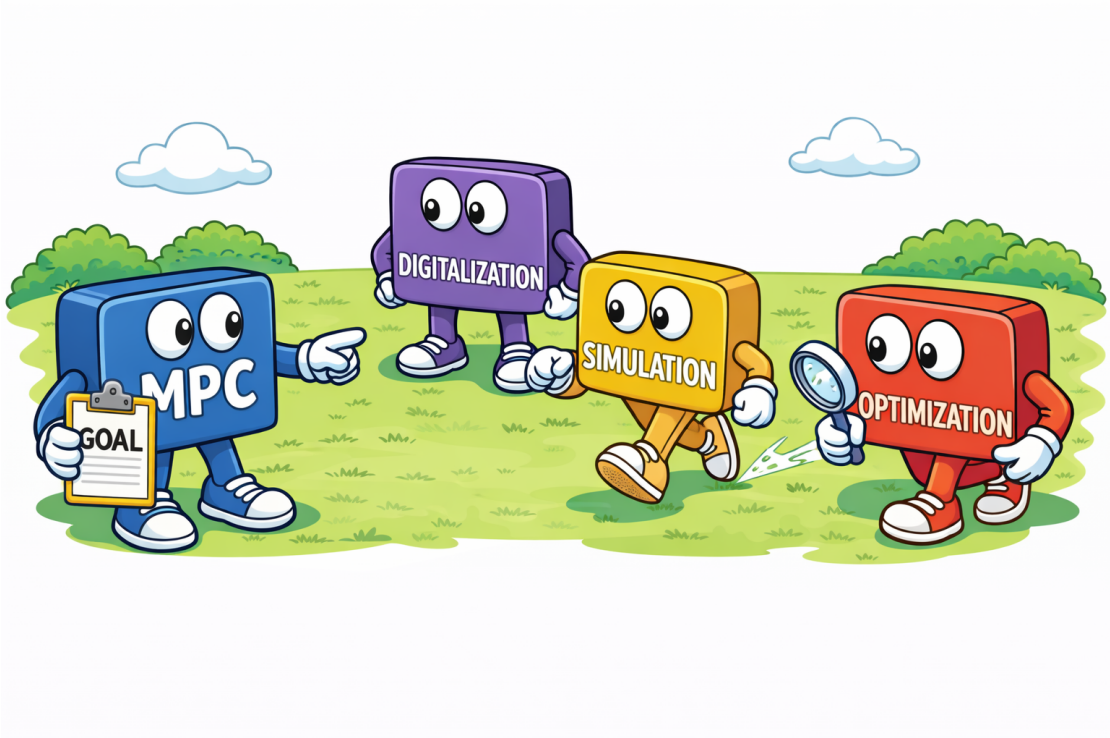
smaller sampling periods. Hence, if we find a constructive way to derive a feedback, then a practical Control-Lyapunov function can be derived and stability properties of this feedback can be concluded.
In the following chapters, we now focus on constructing such a feedback. To simplify the respective notation, we utilize the discrete time notation

$$\begin{aligned} \mathbf{x}(k+1) &= f(\mathbf{x}(k), \mathbf{u}(k), k), \quad \mathbf{x}(0) = \mathbf{x}_0 \\ \mathbf{y}(k) &= h(\mathbf{x}(k), \mathbf{u}(k), k). \end{aligned} \tag{1.3}$$

introduced in Definition 1.7. To this end, we assume that the differential equation is solved to compute the state $\mathbf{x}(k+1)$ based on the continuous time dynamics (1.2) and the zero order hold control $\mathbf{u}(t) := \mathbf{u}_T(t) =: \mathbf{u}(k)$.

CHAPTER 5

MODEL PREDICTIVE CONTROL



Generated with chatgpt.com

> The best way to predict the future is to create it.
>
> Abraham Lincoln

Based on the previous Chapter 4 on digitalization, we now discuss one approach to compute a zero order hold feedback for a nonlinear system. The approaches we considered so far are based on the analytical solution of an optimal control problem using the Riccati approach for a quadratic optimal value function ansatz $V(\mathbf{x}) = \mathbf{x} \cdot P \cdot \mathbf{x}$. However, as soon as the cost is nonquadratic, the dynamics nonlinear or is state and control constraints are introduced, the value function V is no longer quadratic and the approach in general no longer possible. The same holds for the optimal feedback law, which is typically a rather complicated function for which already the storage poses problems and limits such approaches to low dimensions. Moreover, the approach is only capable to compute a Lipschitz continuous feedback. Yet if no continuous feedback exists, by controllability we know that some kind of control exists, for which stability can be shown, e.g. a discontinuous one.
The model predictive control approach takes a step back from optimality over an infinite horizon by approximating it via a series of finite horizon problems. 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 as outlined in Figure 5.1 Thereafter, we show how a feedback can be constructed

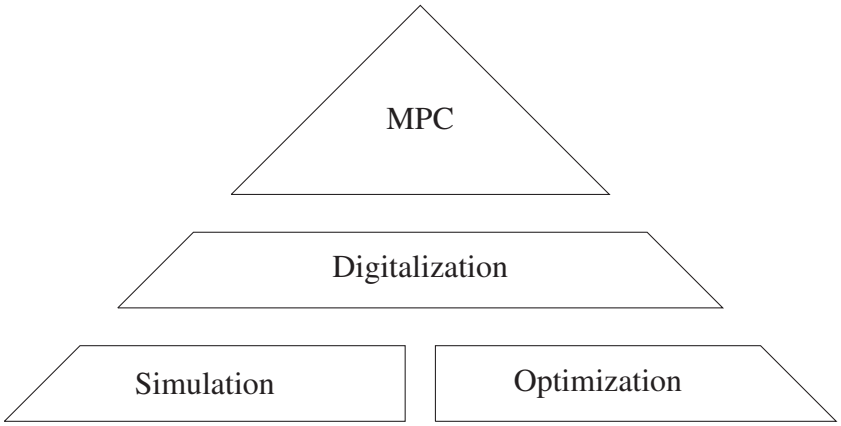


Figure 5.1.: Building blocks within the MPC Algorithm 5.9

from such an approach and how stability of the closed loop can be guaranteed.

5.1. Introduction of constraints

In the previous chapters, we considered systems operating in sets such as the state set $\mathcal{X}$, the control set $\mathcal{U}$ and the output set $\mathcal{Y}$. We then refined this general class of systems given in Definition 1.1 for continuous time systems (1.2) and discrete time systems (1.3) which led us to the term state space, control space and output space.
For designing the LQR, H_2 and H_∞ controllers, we implicitly assumed that these spaces are

unbounded. In practical applications, however, we often face the problem that requirements need to be met. To illustrate this point, we consider the following:

Task 5.1

Consider a supply chain as multi stage network driven by the dynamics

$$\begin{aligned} \dot{s}^p(t) &= f_s(a^p(t), \ell^p(t)) && \textit{(Stock)} \\ \dot{o}_u^p(t) &= f_o(o^p(t), a^p(t)) && \textit{(Unfulfilled order to stock)} \\ \dot{b}^p(t) &= f_b(d^p(t), \ell^p(t)) && \textit{(Backlog from stock)} \end{aligned}$$

where $p \in \mathcal{S} = \{S, M, R\}$ denotes the stages, cf. Figure 5.2. Typically, the stage set contains supplier (S), manufacturer (M) and retailer (R). Moreover, a^p, ℓ^p, o^p and d^p denote the arriving and leaving as well as the order and demand rates. Formulate the basic constraints such a system needs to obey in order to be physically meaningful.

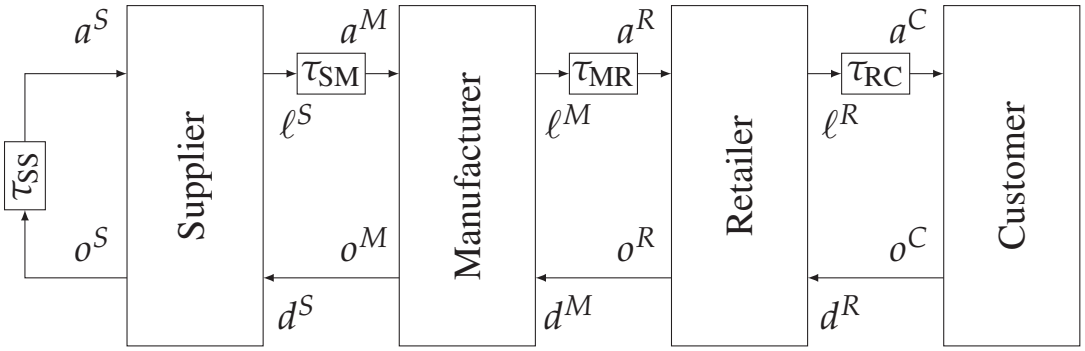


Figure 5.2.: Sketch of a three stage supply network

Solution to Task 5.1: For all times $t \geq 0$ and stages $p \in \mathcal{S}$, the system is subject to the constraints

$$\begin{aligned} 0 \leq o^p(t) &\leq o^p_{\max} & 0 \leq s^p(t) &\leq s^p_{\max} \\ 0 \leq o^p_u(t) &\leq o^p_{u,\max} & 0 \leq b^p(t) &\leq b^p_{\max} \end{aligned}$$

as well as unknown costumer orders o^C and fixed delivery delays τ_{ij}, where $i, j \in \mathcal{S}$ represent consecutive stages. 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 $i, j \in \mathcal{S}$ and $a^i(\tau_{ii}) = o^i(t)$ for the supplier to define these connections. The state for each stage can be defined via $\mathbf{x}^p := (s^p, o^p_u, b^p)^\top$.

Hence, constraints arise naturally in practical problems as states need to be bounded, e.g. to prevent the system from collapsing or hitting physical barriers, or the controls need to be bounded, e.g., for energy reasons or actuator limitations, or outputs need to be bounded, e.g., due to sensor limitations. To address these requirements formally, we define constraints for our system as follows.

Definition 5.2 (Constraints).
Given the state, control and output sets $\mathcal{X}, \mathcal{U}$ and $\mathcal{Y}$, we call $\mathbb{X} \subset \mathcal{X}$ state constraints, $\mathbb{U} \subset \mathcal{U}$ control constraints and $\mathbb{Y} \subset \mathcal{Y}$ output constraints.

We like to stress that constraints are always causing trouble in numerical computations. For this reason, in many applications constraints are not formulated „hard“, that is as constraints that must be satisfied, but instead as „soft“ by adding them as KPI to the cost function by penalizing the violation of constraints.

Remark 5.3
Note that by definition soft constraints may be violated. Hence, such an approach is not applicable for safety critical constraints.

Alternatively, modelers can focus on circumventing the usage of constraints as outlined in the following task:

Task 5.4
Model cars going from an initial point $\mathbf{x}_0 \in \mathbb{R}^2$ to a target point $\mathbf{x}^\star \in R^2$ via routing points $\mathbf{x}_j \in \mathbb{R}^2$, $j = 1, \ldots, M$ as illustrated in Figure 5.3 using a one dimensional system only.

Solution to Task 5.4: Define the route of each vehicle via routing points via interpolation by splines. The car is then controlled along the arc of the spline. Then, we create a one dimensional dynamics via the velocity along the arc length as a control.

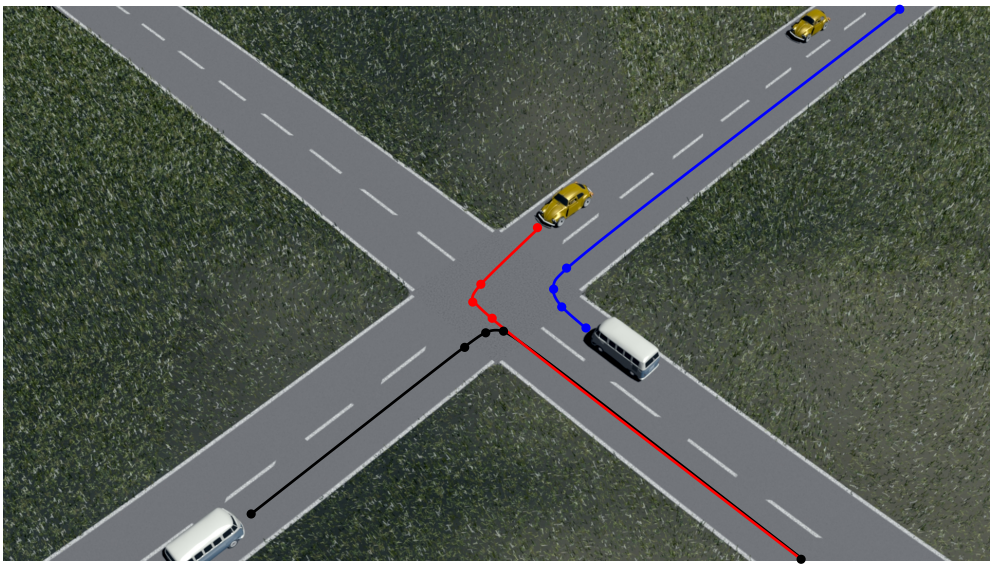


Figure 5.3.: Definition of the driving path via splines for given routing points

To formalize this approach, we call $M \in \mathbb{N}$ the number of routine points. Denoting the entire arc length by L, the routing points are interpolated via the cubic spline

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

which is parametrized by ℓ representing the position on the arc. The arc length is approximated by

$$\ell_0 := 0, \quad \ell_{j+1} := \ell_j + \sqrt{(x_{j+1} - x_j)^2 + (y_{j+1} - y_j)^2}, \quad L := \ell_M.$$

Last, we re-obtain the parametrized driving route via

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

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

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

where t denotes time and $\mathbf{u}(t)$ represents the velocity of the car at time instant t. By choosing

the velocity $\mathbf{u} \in \mathbb{U}$ 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 5.5

- *Note that deriving the routing points in Task 5.4 is a different and 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.*
- *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.*
- *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.*

5.2. MPC approach

Having defined constraints, we can now generalize the setting from Chapter 2 to a nonlinear constrained optimal control problem. Note that in Definition 2.4, we used the general nonlinear form, which we later specified to LTI systems to discuss the LQR, H_2 and H_∞ controller.
Formally, we obtain

Definition 5.6 (Constrained optimal control problem).
Consider a system (1.2) and a cost functional (2.1). Then we call

$$\begin{aligned} \min\ J(\mathbf{x}_0, \mathbf{u}) &= \int_0^\infty \ell(\mathbf{x}(t, \mathbf{x}_0, \mathbf{u}), \mathbf{u}(t))dt \quad \text{over all } \mathbf{u} \in \mathbb{U}^\infty \\ \text{subject to } \dot{\mathbf{x}}(t) &= f(\mathbf{x}(t), \mathbf{u}(t), t), \quad \mathbf{x}(t_0) = \mathbf{x}_0 \\ \mathbf{x}(t) &\in \mathbb{X}, \quad t \in [0, \infty) \end{aligned} \tag{5.1}$$

an *constrained optimal control problem*. The function

$$V(\mathbf{x}_0) := \inf_{\mathbf{u}\in\mathcal{U}} J(\mathbf{x}_0, \mathbf{u}) \tag{5.2}$$

is called *optimal value function*.

Since the continuous time formulation allows for infinitely many control changes, it is not only computationally difficult or intractable to solve. Additionally, actuators work in a sampled manner, hence such a control is practically also not usable. To address these issues, we apply the following adaptations:

- By applying digitalization, we can shift the problem to the discrete time formulation solving the sampling issue. Moreover, digitalization allows us to decouple optimization and simulation.
- Cutting the infinite horizon to a finite one allows us to address the computational issue. For one, simulation techniques to digitalized or discrete time systems are very effective, and secondly, optimization methods for finitely many inputs are well developed.

These are the ingredients linked in Figure 5.1, which allow us to divide the control problem (5.1) accordingly. To formalize this procedure, we first introduce the following:

Notation 5.7 (Open/closed loop index)
In the context of MPC we denote the closed loop time index by n and the open loop time index by k. Moreover, we denote the open loop horizon by N.

Now, the subproblems to solve take the following form:

Definition 5.8 (Digital constrained optimal control problem).
Consider a constrained optimal control problem (5.1). Applying digitalization, zero order hold and horizon cutting, we call

$$\begin{aligned} \min\ J(\mathbf{x}_0, \mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}(k, \mathbf{x}_0, \mathbf{u}), \mathbf{u}(k)) \quad \text{over all } \mathbf{u} \in \mathbb{U}^N \\ \text{subject to } \mathbf{x}(k+1) = f(\mathbf{x}(k), \mathbf{u}(k), k), \quad \mathbf{x}(0) = \mathbf{x}_0 \\ \mathbf{x}(k) \in \mathbb{X}, \quad k \in [0, N] \end{aligned} \tag{5.3}$$

a *digital finite constrained optimal control problem* or *MPC open loop problem*.

While the problem is solvable now, it does not give us a solution of the original problem. To still be able to at least approximate such a solution, MPC can be used. The idea of MPC is split up the problem over time and only consider time windows, for which the problem is to be solved. This goes hand in hand with the digitalization idea and the time windows are constructed such that each window starts at a sampling instant. To capture long term system behavior, the length of the time windows is longer than one sampling period and measured in multiples of the sampling period. As the time windows solution is longer than required, only a fraction of the solution is applied.
Combined, MPC is a three step scheme:

Algorithm 5.9 (Basic MPC Algorithm)
For each closed loop time index $n = 0, 1, 2 \ldots$:

(1) Obtain the state $\mathbf{x}(n) \in \mathbb{X}$ of the system.

(2) Set $\mathbf{x}_0 := \mathbf{x}(n)$, solve the digital finite optimal control problem (5.3) and denote the obtained optimal control sequence by $\mathbf{u}^\star(\cdot) \in \mathbb{U}^N$.

(3) Define the MPC feedback $\mu_N(\mathbf{x}(n)) := \mathbf{u}^\star(0)$.

While easily accessible and adaptable, the method behind Algorithm 5.9 exhibits some severe flaws that need to be considered before putting it into practice:

1. Cutting the horizon to $N < \infty$ may result in infeasibility of the problem at closed loop time indexes $n > 0$. A simple example is a car driving towards a wall. If the prediction horizon is too small, the car is unable to stop before hitting the wall. Mathematically speaking, no solution can be found satisfying all constraints. We address this issue in Section 5.3 and show how feasibility can be guaranteed recursively.

2. Cutting the horizon may also result in destabilizing the system. Again, we can use the car/wall example and put the target point behind the wall, i.e. the car needs to go around the wall. If the wall is long compared to the prediction horizon, the car will not be able to „see“ a possibility of going around the wall and stop in front of it. Hence the system is not asymptotically stable. In Section 5.4, we address this issue using three different strategies.

5.3. Recursive feasibility

From the discussing above on existence of a solution throughout the MPC iterations we obtain that we require for each n

- existence of a solution for problem (5.3) at closed loop index n and
- guarantee that the subsequent problem (5.3) at closed loop index $n+1$ exhibits a solution.

Remark 5.10
At this point, we want to stress the fact that loss of feasibility is due to the method of MPC, i.e. the cutting of the horizon. This problem does not exist for the original constrained optimal control problem (5.1)*. However, if the latter does not exhibit a solution, then it is not possible to approximate such a non-existing solution using MPC.*

The first property is referred to as feasibility, the second as recursive feasibility. To formalize these properties, we first introduce the following:

Definition 5.11 (Admissibility).
Consider a discrete time control system (1.3) with state and input constraints $\mathbb{X} \subset \mathcal{X}$ and $\mathbb{U} \subset \mathcal{U}$.

- The states $\mathbf{x} \in \mathbb{X}$ are called *admissible states* and the inputs $\mathbf{u} \in \mathbb{U}(\mathbf{x})$ are called *admissible inputs for* $\mathbf{x}$. The elements of the set $\{(\mathbf{x}, \mathbf{u}) \mid \mathbf{x} \in \mathbb{X}, \mathbf{u} \in \mathbb{U}(\mathbf{x})\}$ are called *admissible pairs*.
- For $N \in \mathbb{N}$ and initial value $\mathbf{x}_0 \in \mathbb{X}$ we call an input sequence $\mathbf{u} \in U^N$ and the corresponding trajectory $\mathbf{x}_\mathbf{u}(k, \mathbf{x}_0)$ *admissible for* $\mathbf{x}_0$ *up to time* N if
  - the running time constraint
  $$(\mathbf{x}_\mathbf{u}(k, \mathbf{x}_0), \mathbf{u}(k)) \text{ are admissible pairs } \forall k = 0, \dots, N-1$$
  - and the terminal constraint
  $$\mathbf{x}_\mathbf{u}(N, \mathbf{x}_0) \in \mathbb{X}$$

  hold. We denote the respective set of admissible sequences by $\mathbb{U}_\mathbb{X}^N(\mathbf{x}_0)$.
- An input sequence $\mathbf{u} \in U^\infty$ are the corresponding trajectory $\mathbf{x}_\mathbf{u}(k, \mathbf{x}_0)$ are called *admissible for* $\mathbf{x}_0$ if they are admissible for $\mathbf{x}_0$ up to every time $N \in \mathbb{N}$. We denote the set of admissible input sequences for $\mathbf{x}_0$ by $\mathbb{U}_\mathbb{X}^\infty(\mathbf{x}_0)$.
- A feedback $\mu : \mathcal{X} \to \mathcal{U}$ is called *admissible* if $\mu(\mathbf{x}) \in \mathbb{U}_\mathbb{X}^1(\mathbf{x})$ holds for all $\mathbf{x} \in \mathbb{X}$.

We like to note the slight difference between $\mathbb{U}$ and $\mathbb{U}^1(\mathbf{x})$: By definition of admissibility for $\mathbf{x}$

up to time 1, we have that

$$\mathbb{U}_{\mathbb{X}}^{1}(\mathbf{x}) := \{\mathbf{u} \in \mathbb{U}(\mathbf{x}) \mid f(\mathbf{x}_0, \mathbf{u}) \in \mathbb{X}\} \subset \mathbb{U}(\mathbf{x}).$$

This is essential especially for our definition of an admissible feedback, which ensures exactly that.

Remark 5.12
Note that even if $\mathbb{U}(\mathbf{x}) = \mathbb{U}$ *is independent of the actual state* $\mathbf{x}$*, the set* $\mathbb{U}^N(\mathbf{x})$ *may still depend on* $\mathbf{x}$ *for some or all* $N \in \mathbb{N}$.

The property of admissibility is defined on sequences of states and inputs, yet not on the problem. We now use admissibility to formalize the problem property of feasibility:

Definition 5.13 (Feasibility).
Consider a digital finite constrained optimal control problem (5.3).

- We call an initial condition $\mathbf{x}_0 \in \mathbb{X}$ *feasible* for (5.3) if $\mathbb{U}^N(\mathbf{x}_0) \neq \emptyset$.
- The MPC Algorithm 5.9 is called *recursively feasible* on a set $A \subset \mathbb{X}$ if each $\mathbf{x} \in A$ is feasible for (5.3) and $\mathbf{x} \in A$ implies $f(\mathbf{x}, \mu_N(\mathbf{x})) \in A$.

In order to guarantee that Algorithm 5.9 is recursively feasible, the so called *viability assumption* can be used.

Theorem 5.14 (Recursive feasibility and admissibility).
Consider the MPC Algorithm 5.9. If the viability assumption

$$\forall \mathbf{x} \in A \subset \mathbb{X} : \ \exists \mathbf{u} \in \mathbb{U}(\mathbf{x}) \text{ such that } f(\mathbf{x}, \mathbf{u}) \in A \subset \mathbb{X} \tag{5.4}$$

holds, then the MPC Algorithm 5.9 is recursively feasible on A *and the pairs* $(\mathbf{x}_{\mu_N}(n), \mu_N(\mathbf{x}_{\mu_N}(n)))$ *as well as the feedback* μ_N *are admissible for all* $n \in \mathbb{N}$.

We like to point out that the viability assumption (5.4) looks simple, yet in practice it is rather difficult to identify the set A.

Task 5.15
Consider sampled data model of a car

$$\mathbf{x}(k+1) = \begin{pmatrix} \mathbf{x}_1(k) + \mathbf{x}_2(k) + \mathbf{u}(k)/2 \\ \mathbf{x}_2(k) + \mathbf{u}(k) \end{pmatrix}$$

on a one dimensional road with position $\mathbf{x}_1$*, speed* $\mathbf{x}_2$ *and piecewise constant acceleration* $\mathbf{u}$*. Assume all variables to be constrained to* $[-1,1]$*. Compute the set* A*.*

Solution to Task 5.15: Using the dynamics and the extreme values $\mathbf{x}_1 = \mathbf{x}_2 = 1$ we obtain

$$\mathbf{x}_1(k+1) = \mathbf{x}_1(k) + \mathbf{x}_2(k) + \mathbf{u}(k)/2 \geq 3/2 > 1$$

for any $\mathbf{u} \in \mathbb{U} = [-1,1]$. Hence, such a state is not recursively feasible. Via elementary computations, we can define

$$A := \left\{ \mathbf{x} \in \mathbb{R}^2 \mid \mathbf{x}_1 \in [-1,1],\ \mathbf{x}_2 \in [-1,1] \cap [-3/2 - \mathbf{x}_1, 3/2 - \mathbf{x}_1] \right\}$$

for which the choice

$$\mathbf{u} := \begin{cases} 1, & \mathbf{x}_2 < -1/2 \\ -2\mathbf{x}_2, & \mathbf{x}_2 \in [-1/2, 1/2] \\ -1, & \mathbf{x}_2 > 1/2 \end{cases}$$

satisfies $\mathbf{u} \in [-1,1]$ and $f(\mathbf{x},\mathbf{u}) \in A$.

Figure 5.4 illustrates the viability condition (blue) in comparison to the state constraints (black), where the difference occurs in the encircled regions (red).
In practice, we are interested to compute a feedback which is not only admissible, but also asymptotically stabilizes our system.

5.4. Stability conditions

To guarantee stability of the closed loop using the MPC feedback computed via Algorithm 5.9, there are three different ideas in the literature. Two of them include the usage of so called terminal conditions, that is conditions imposed to the end point of the open loop prediction horizon used

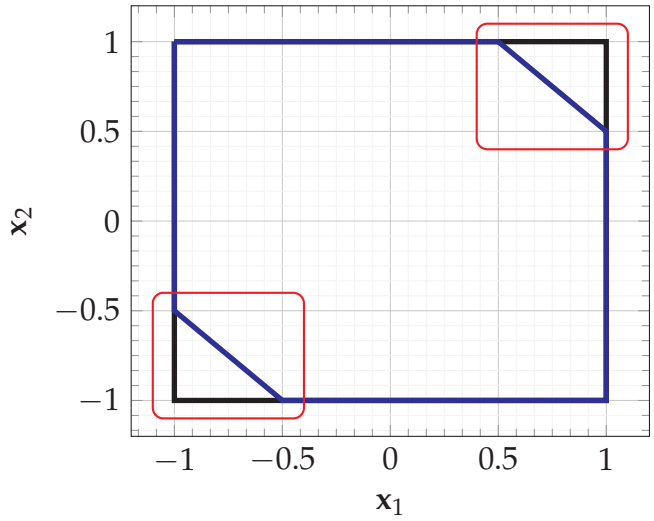


Figure 5.4.: Sketch of a viability set

within MPC, and one based on Lyapunov functions. Here, we will not go into details regarding the specifics of these methods, but discuss them from an application point of view.
Terminal conditions are conditions, which are added to the problem (5.3) at open loop time instant $k = N$.

Remark 5.16
Note that as terminal conditions alter the problem, the solutions of the problem are in general different.

The first approach uses so called *terminal constraints*:

Definition 5.17 (Terminal constraints).
Consider a digital finite constrained optimal control problem (5.3). Then we call

$$\mathbf{x_u}(N, \mathbf{x}_0) \in \mathbb{X}_0 \tag{5.5}$$

terminal constraint and $\mathbb{X}_0 \subset \mathbb{X}$ *terminal constraint set*.

The idea of terminal constraints is straightforward: By imposing a terminal constraint set, the set of admissible pairs is limited, i.e. the set of initial values and controls to be chosen are reduced. Hence, it is no longer necessary to compute the set A from the viability conditions, but it is implicitly imposed using the right terminal conditions.

Remark 5.18
The right choice for terminal conditions can be made using ideas such as linearization around the operating point $\mathbf{x}^\star$. From Control engineering 2 we then now that there exists a linear feedback such that the terminal constraint set is recursively feasible.

In fact, we obtain the following restriction:

Definition 5.19 (Feasibility set).
Consider a digital finite constrained optimal control problem (5.3) together with terminal constraint (5.5). Then we call

$$\mathbb{X}_N := \left\{ \mathbf{x}_0 \in \mathbb{X} \mid \exists \mathbf{u} \in \mathbb{U}^N(\mathbf{x}_0) : \ \mathbf{x}_\mathbf{u}(N, \mathbf{x}_0) \in \mathbb{X}_0 \right\} \tag{5.6}$$

feasible set for horizon N and

$$\mathbb{U}^N_{\mathbb{X}_N}(\mathbf{x}_0) := \left\{ \mathbf{u} \in \mathbb{U}^N(\mathbf{x}_0) \mid \mathbf{x}_\mathbf{u}(N, \mathbf{x}_0) \in \mathbb{X}_0 \right\} \tag{5.7}$$

set of admissible control sequences for horizon N.

Combining terminal constraints and the MPC algorithm, we obtain the following:

Corollary 5.20 (Feasibility).
Consider the MPC Algorithm 5.9. For each $\mathbf{x}_0 \in \mathbb{X}_N$ we have

$$f(\mathbf{x}, \mu_N(\mathbf{x})) \in \mathbb{X}_{N-1}. \tag{5.8}$$

Based on the latter, we directly obtain:

Theorem 5.21 (Recursive feasibility using terminal constraints).
Consider the MPC Algorithm 5.9 with terminal constraint (5.5)*. Then the MPC Algorithm is recursively feasible.*

If we additionally know that for the region defined by the terminal constraint (5.5) there exists an asymptotically stabilizing feedback, then the following can be concluded:

Theorem 5.22 (Asymptotical stability using terminal constraints).
Consider the MPC Algorithm 5.9. Suppose a terminal constraint (5.5) *to be imposed on problem* (5.3) *and furthermore an asymptotically stabilizing feedback to exist for all* $\mathbf{x} \in \mathbb{X}_0$. *Then the MPC Algorithm is asymptotically stabilizing the system* (1.3).

While being simple in usage, the limitations of terminal constraints are the reduction of admissible controls, which can only be reduced by enlarging the prediction horizon N. Since the latter induces high computing times, it would be much simpler to increase the size of the terminal constraints, which stand at the center of the second approach.
Different from terminal constraints, the second approach appends a terminal cost to the cost function in problem (5.3). The intention is to enlarge the terminal constraints by including costs arising for the cutoff horizon $[N, \infty)$. These terminal costs are defined as follows:

Definition 5.23 (Terminal costs).
Consider a digital finite constrained optimal control problem (5.3). Then we call a function $L : \mathbf{x} \to \mathbb{R}_0^+$ *terminal cost* if it is added to the cost function of problem (5.3)

$$\min \; J(\mathbf{x}_0, \mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}(k, \mathbf{x}_0, \mathbf{u}), \mathbf{u}(k)) + L(\mathbf{x}_\mathbf{u}(N, \mathbf{x}_0). \tag{5.9}$$

Again, we obtain asymptotic stability using the existence of an asymptotically stabilizing feedback in the terminal constraint set:

Theorem 5.24 (Asymptotical stability using terminal costs).
Consider the MPC Algorithm 5.9. Suppose a terminal constraint $\mathbb{X}_0$ *and terminal costs* $L(\cdot)$ *to be imposed on problem* (5.3) *and furthermore an asymptotically stabilizing feedback to exist for all* $\mathbf{x} \in \mathbb{X}_0$. *Then the MPC Algorithm is recursively feasible and asymptotically stabilizes the system* (1.3).

The last idea to guarantee asymptotic stability of the MPC closed loop utilizes a control-Lyapunov function based approach. Here, we can directly utilize the MPC formulation to check the requirements of Definition 4.12 for practical control-Lyapunov function:

Theorem 5.25 (Asymptotical stability using suboptimality).
Consider the MPC Algorithm 5.9 and suppose the viability condition 5.4 to hold. If there exists

a function $V : \mathcal{X} \to \mathbb{R}_0^+$ *such that there exist functions* $\alpha_1, \alpha_2 \in \mathcal{K}_\infty$ *and a constant* $\alpha \in (0,1]$ *such that*

$$\alpha_1(\|\mathbf{x} - \mathbf{x}^\star\|) \leq V(\mathbf{x}) \leq \alpha_2(\|\mathbf{x} - \mathbf{x}^\star\|) \tag{5.10}$$

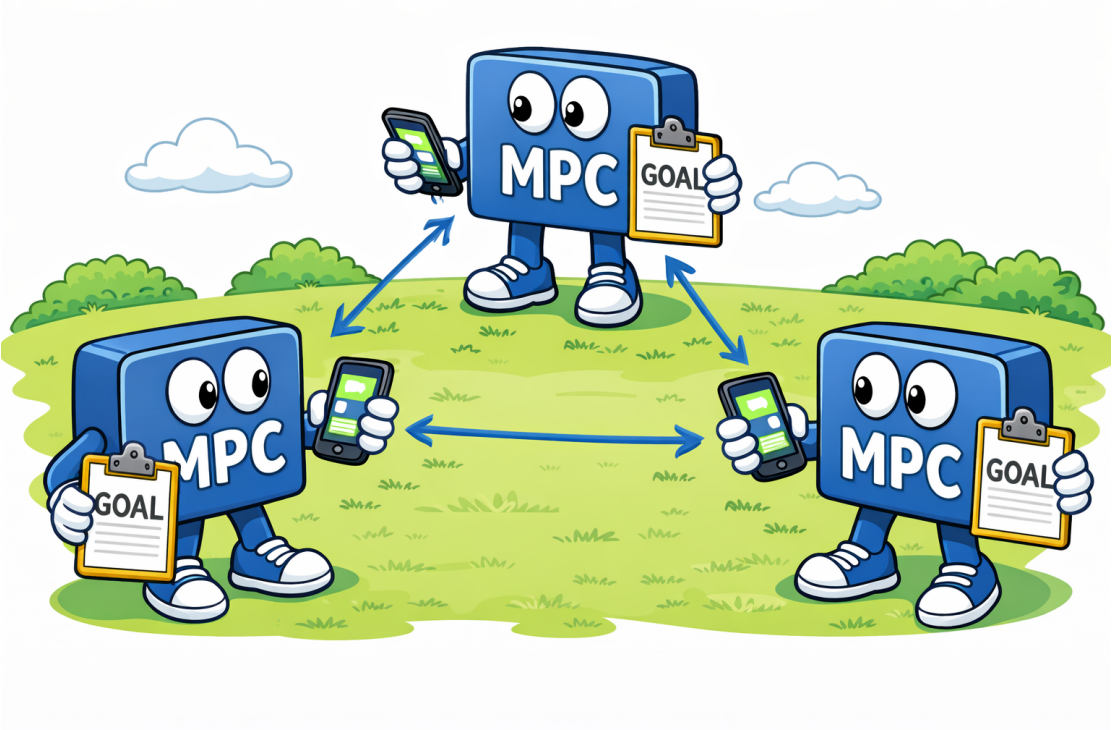
$$V(\mathbf{x}) \geq \alpha\ell(\mathbf{x}, \mu_N(\mathbf{x})) + V(f(\mathbf{x}, \mu_N(\mathbf{x}))) \tag{5.11}$$

holds, then the MPC Algorithm is recursively feasible and asymptotically stabilizes the system (1.3).

The intention of the last approach is to avoid constructing terminal constraints or costs and to also avoid alteration of the original control problem. While being technically simple to monitor, conditions (5.10)–(5.11) are very hard to check analytically. For further details, we refer to [2]. From an energy point of view, the conditions of Theorem 5.25 state that energy is continuously drawn from the system, hence any trajectory is driven towards the operating point $\mathbf{x}^\star$. Yet, it is not equivalent to the standard notation of Lyapunov, which uses $\alpha = 1$. The latter parameter can be interpreted as a measure of suboptimality, i.e. the tradeoff in optimality we have to accept for cutting the horizon and making the problem to be computationally tractable.

CHAPTER 6

DISTRIBUTED CONTROL



Generated with chatgpt.com

> Avoiding complexity reduces bugs.
>
> Linus Torwalds

So far, we have considered systems and processes that can be governed by a single control unit. In practical applications, however, this assumption is often violated. One reason is that systems may be too large or too complex to be handled centrally, making it necessary to decompose them into smaller, yet possibly interconnected, subsystems. Typical examples include chemical plants, supply chains, and production lines. Similar challenges also arise on a purely software level, for instance in robotic process automation, where large workflows must be coordinated across multiple agents or services. A second class of problems is characterized by an inherent structural decomposition. Such situations arise when multiple decision-making units operate within a shared environment and must coordinate their actions. Representative examples range from autonomous vehicles sharing a common traffic space to robots collaborating in a workspace, or economic agents interacting in buyer–seller scenarios. In this chapter, we focus on approaches based on model predictive control (MPC) to address such multi-unit and decomposed control problems. In particular, we concentrate on three fundamental ideas that allow the control problem to be split—or, respectively, to preserve an existing decomposition—while still accounting for the overall system objectives and constraints. As illustrated in Figure 6.1, the coupling between subsystems is introduced at the MPC level of the problem.

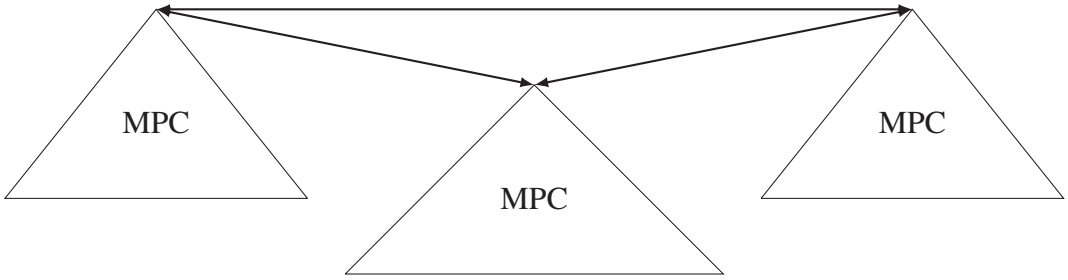


Figure 6.1.: Building blocks within the MPC Algorithm 5.9

To this end, we distinguish three basic approaches. The first approach follows a first-come–first-serve principle, in which one controller determines its strategy first, and the remaining controllers adapt their decisions based on the residual degrees of freedom. The second approach exploits structural properties of the system by identifying subsystems that do not mutually interfere, allowing them to operate in parallel, while explicitly coordinating only those subsystems that do interact. The third and most general approach assumes full parallelization, where all subsystems solve their MPC problems simultaneously. As will become apparent, these approaches differ significantly in their communication requirements as well as in the amount and type of information that must be exchanged between controllers.

Remark 6.1
At this point, we emphasize that the present discussion focuses on decompositions in the spatial or control domain, where the system is split according to states, inputs, or subsystems. An alternative approach to managing complexity is based on a temporal decomposition, in which the prediction horizon is partitioned rather than the state space. While these decompositions share conceptual similarities from a systems-theoretic perspective, temporal splitting typically requires a formulation in terms of partial differential equations.

6.1. Separation of systems

Instead of considering a single system of the form

$$\mathbf{x}(k+1) = f\big(\mathbf{x}(k), \mathbf{u}(k), k\big), \quad \mathbf{x}(0) = \mathbf{x}_0, \ \mathbf{y}(k) \ = h\big(\mathbf{x}(k), \mathbf{u}(k), k\big), \tag{1.3}$$

we now turn to a setting consisting of multiple interacting subsystems. For the sake of clarity and without loss of generality for the subsequent developments, we omit explicit time dependence of the dynamics as well as system outputs, and consider a family of discrete-time systems of the form

$$\mathbf{x}^p(k+1) = f^p\big(\mathbf{x}^p(k), \mathbf{u}^p(k), i^p(k)\big), \quad \mathbf{x}^p(0) = \mathbf{x}_0^p. \tag{6.1}$$

Here, $p \in \mathcal{P} := \{1, \dots, P\}$ denotes the index of the corresponding subsystem. The state and control variables satisfy $\mathbf{x}^p(k) \in \mathcal{X}^p$ and $\mathbf{u}^p(k) \in \mathcal{U}^p$ where the sets $\mathcal{X}^p$ and $\mathcal{U}^p$ may differ between subsystems. The variable $i^p(k) \in I^p$ represents the neighboring data associated with subsystem p. It serves as the interface through which coupling between subsystems is introduced. Depending on the application, $i^p(k)$ may encode, for instance, states or predicted trajectories of neighboring subsystems, shared resources, or coordination variables. Both the neighboring data and the corresponding set I^p may depend on the subsystem index p and, in general, on time.
Throughout this chapter, we restrict our attention to decompositions that arise from splitting the system dynamics. To illustrate the idea, we first consider the following example:

Task 6.2
Reconsider the Example from Task 5.15 with dynamics

$$\begin{pmatrix} \mathbf{x}_1(k+1) \\ \mathbf{x}_2(k+1) \end{pmatrix} = \begin{pmatrix} \mathbf{x}_1(k) + \mathbf{x}_2(k) + \mathbf{u}(k)/2 \\ \mathbf{x}_2(k) + \mathbf{u}(k) \end{pmatrix}$$

and split the system into two subsystems using $\mathbf{x}^1 = \mathbf{x}_1$, $\mathbf{x}^2 = \mathbf{x}_2$ *and* $\mathbf{u}^2 = \mathbf{u}$.

Solution to Task 6.2: Setting $\mathbf{x}^1 = \mathbf{x}_1$, $\mathbf{x}^2 = \mathbf{x}_2$ and $\mathbf{u}^2 = \mathbf{u}$ and leaving $\mathbf{u}^1$ undefined, we obtain

$$\begin{aligned}\mathbf{x}^1(k+1) &= \mathbf{x}^1(k) + \overbrace{\mathbf{x}^2(k) + \mathbf{u}^2(k)}^{\text{from subsystem 2}}/2\\ \mathbf{x}^2(k+1) &= \mathbf{x}^2(k) + \mathbf{u}^2(k).\end{aligned}$$

For that choice, subsystem 2 is independent from subsystem 1. However, to evaluate subsystem 1 the information $i^1(k)$ is required to evaluate $\mathbf{x}^2(k)$ and $\mathbf{u}^2(k)$ from subsystem 2. Note that the connection depends on how the control input from the overall system is assigned to the subsystems. Setting $\mathbf{u}^1 = \mathbf{u}$ and leaving $\mathbf{u}^2$ undefined, both subsystems depend on each other.

The objective of a system decomposition is to ensure that, upon recombination of the subsystems defined in (6.1), the original overall system dynamics are recovered. More precisely, the subsystems are required to jointly represent an overall system of the form

$$\mathbf{x}(k+1) = f\big(\mathbf{x}(k), \mathbf{u}(k)\big), \tag{6.2}$$

where the global state and control vectors are obtained by concatenation of the local variables $\mathbf{x}(k) = (\mathbf{x}^1(k)^\top, \ldots, \mathbf{x}^P(k)^\top)^\top \in \mathcal{X} = \mathcal{X}^1 \times \ldots \times \mathcal{X}^P$ and $\mathbf{u}(k) = (\mathbf{u}^1(k)^\top, \ldots, \mathbf{u}^P(k)^\top)^\top \in \mathcal{U} = \mathcal{U}^1 \times \ldots \times \mathcal{U}^P$. Throughout this chapter, we refer to (6.2) as the overall system, to the family of systems in (6.1) as the set of subsystems, and to each index $p \in \mathcal{P}$ as a subsystem. The neighboring data variables $i^p(k)$ introduced earlier serve to ensure that the local subsystem dynamics are compatible with the global dynamics when recombined.

As observed in Task 6.2, a consistent decomposition generally requires not only a partitioning of the state space $\mathcal{X}$, but also of the control space $\mathcal{U}$. In order to formalize such coordinated splits in a mathematically precise manner, we introduce the notion of projections and decompositions.

Definition 6.3 (Projection).
Let S be a vector space. A linear map $\pi : S \to S$ is called a *projection* if it is idempotent, that is $\pi \circ \pi = \pi$. In this case, S can be written as the direct sum $S = \mathrm{Im}(\pi) \oplus \mathrm{Ker}(\pi)$ of the image and the kernel of π and we say that π projects S onto $\mathrm{Im}(\pi)$ along $\mathrm{Ker}(\pi)$.

Projections allow us to isolate subspaces of interest while ignoring complementary components. Using a family of such projections, we can define a structured splitting of a vector space.

Definition 6.4 (Decomposition).
Let S be a vector space and let $\mathcal{P} = \{1, \dots, P\}$ where $P \in \mathbb{N}$. Consider a family of projections $(\pi^p)_{p\in\mathcal{P}}$ on S where $S^p := \mathrm{Im}(\pi^p)$ is a subset of S for all $p \in \mathcal{P}$ to be given. If we have that

$$\langle (S^p)_{p\in\mathcal{P}} \rangle = S \text{ and } S^q \cap \langle (S^p)_{p\in\mathcal{P}, p\neq q} \rangle = \{0\} \text{ for all } q \in \mathcal{P}$$

hold, then we call the family $(S^p)_{p\in\mathcal{P}}$ a *decomposition* of S.

These conditions ensure that every element of S admits a unique representation as a sum of components belonging to the individual subspaces S^p. In the context of distributed model predictive control, such decompositions provide the mathematical foundation for assigning distinct state and control components to individual subsystems while preserving a well-defined global system representation.
Using the notion of decomposition introduced above, we can now rewrite the overall system (6.2) as a collection of subsystems defined on suitable subspaces. To this end, we require two families of projections, one acting on the state space and one acting on the control space. More precisely, for each subsystem index $p \in \mathcal{P}$, we introduce

- a state projection $\pi^p_{\mathcal{X}} : \mathcal{X} \to \mathcal{X}$ to split the state set such that $\mathrm{Im}(\pi^p_{\mathcal{X}}) = \mathcal{X}^p$, and
- a control projection $\pi^p_{\mathcal{U}} : \mathcal{U} \to \mathcal{U}$ to split the control set such that $\mathrm{Im}(\pi^p_{\mathcal{U}}) = \mathcal{U}^p$.

These projections formalize the assignment of global state and control components to individual subsystems.
In general, however, these projections do not yield a strict separation of the system dynamics. As already observed in Task 6.2, the evolution of a given subsystem may depend on state or control variables that are assigned—via the projections—to other subsystems. As a consequence, the projected dynamics $\pi^p_{\mathcal{X}} \circ f$ typically depend on additional variables beyond the local state and control. To capture this structure, the projection induces, for each subsystem, a classification of state and control variables into three distinct components.
In particular, the state space $\mathcal{X}$ can be decomposed into $\left(\mathcal{X}^p, \widetilde{\mathcal{X}}^p, \overline{\mathcal{X}}^p\right)$ where

- $\mathbf{x}^p \in \mathcal{X}^p$ denotes the *local state* of subsystem p, i.e. the primary state variables governed and predicted by the corresponding local controller.
- $\widetilde{\mathbf{x}}^p \in \widetilde{\mathcal{X}}^p$ represents the *neighboring states*, that is, state components of other subsystems that are required to correctly evaluate the projected dynamics $\pi^p_{\mathcal{X}} \circ f$.

- $\overline{\mathbf{x}}^p \in \overline{\mathcal{X}}^p$ collects all remaining state components that do not influence the projected dynamics of subsystem p.

Analogously, the control space $\mathcal{U}$ decomposes into $\left(\mathcal{U}^p, \widetilde{\mathcal{U}}^p, \overline{\mathcal{U}}^p\right)$ where

- $\mathbf{u}^p \in \mathcal{U}^p$ denotes the *local control input* computed by subsystem p.
- $\widetilde{\mathbf{u}}^p \in \widetilde{\mathcal{U}}^p$ contains the *neighboring control inputs*, that is, control variables generated by other subsystems that influence the projected dynamics $\pi^p_{\mathcal{X}} \circ f$.
- $\overline{\mathbf{u}}^p \in \overline{\mathcal{U}}^p$ consists of control components that are irrelevant for the projected dynamics of subsystem p.

Figure 6.2 illustrates these splits.

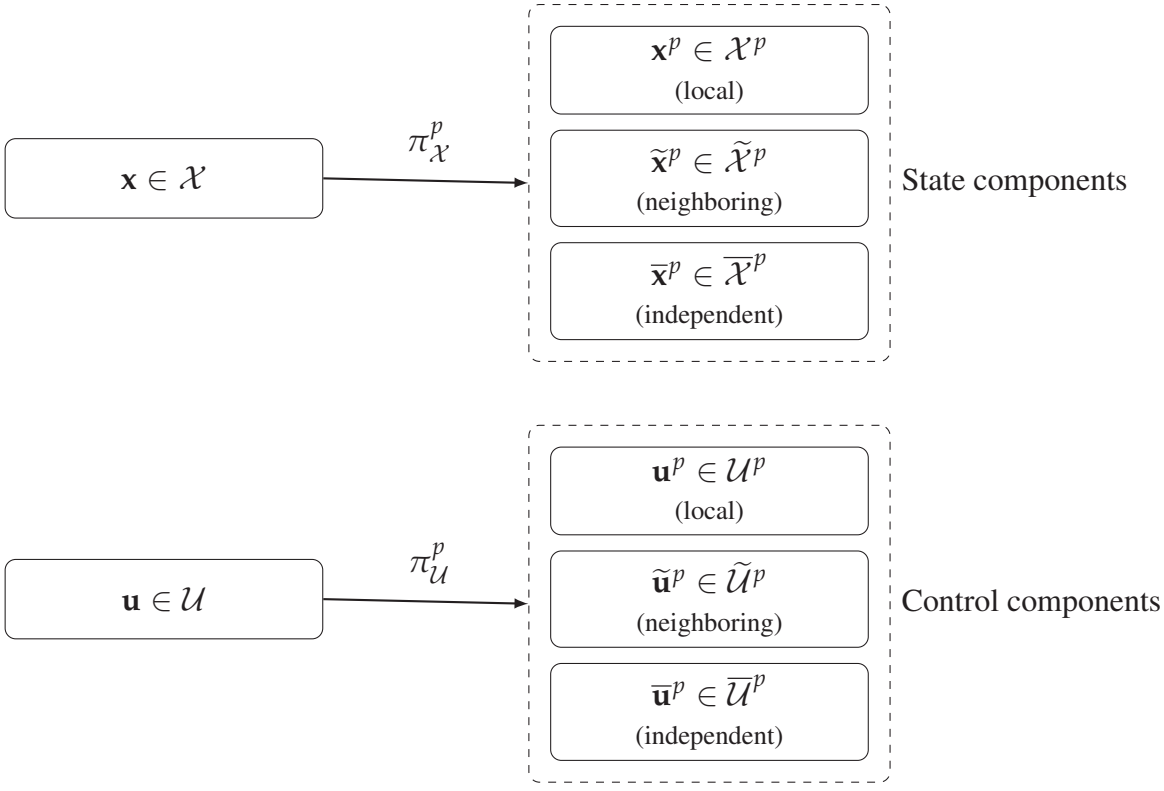


Figure 6.2.: Decomposition of global state and control variables by the projections $\pi^p_{\mathcal{X}}$ and $\pi^p_{\mathcal{U}}$

Remark 6.5
The control variables $\widetilde{\mathbf{u}}^p \in \widetilde{\mathcal{U}}^p$ are determined by controllers associated with other subsystems. Consequently, their inclusion in the evaluation of the projected dynamics of subsystem p requires explicit communication of the corresponding control information.

By construction, the projected dynamics $\pi^p_{\mathcal{X}} \circ f$ are independent of the variables $\overline{\mathbf{x}}^p \in \overline{\mathcal{X}}^p$ and $\overline{\mathbf{u}}^p \in \overline{\mathcal{U}}^p$. For this reason, these variables are referred to as *independent states* and *independent controls*, respectively.

Remark 6.6
From a software and implementation perspective, the variables $\mathbf{x}^p(k) \in \mathcal{X}^p$ and $\mathbf{u}^p(k) \in \mathcal{U}^p$ are commonly referred to as local or private variables. In contrast, the variables $\widetilde{\mathbf{x}}^p(k) \in \widetilde{\mathcal{X}}^p$, $\widetilde{\mathbf{u}}^p(k) \in \widetilde{\mathcal{U}}^p$, as well as the independent state and control components, are typically treated as interface or public variables.

The sets $\widetilde{\mathcal{X}}^p$ and $\widetilde{\mathcal{U}}^p$ explicitly characterize the information dependencies between subsystems. In particular, they allow us to identify which subsystems must exchange information in order to evaluate their respective projected dynamics.
This leads to the following definition.

Definition 6.7 (Neighboring index set).
Consider a decomposition of system (6.2). Then we call $\mathcal{I}^p = \{p_1, \ldots, p_m\} \subset \mathcal{P} \setminus \{p\}$ neighboring index set if it satisfies

$$(\mathcal{X}^{p_1} \times \ldots \times \mathcal{X}^{p_m}) \times (\mathcal{U}^{p_1} \times \ldots \times \mathcal{U}^{p_m}) \supset (\widetilde{\mathcal{X}}^p \times \widetilde{\mathcal{U}}^p). \tag{6.3}$$

In other words, the neighboring index set $\mathcal{I}^p$ contains exactly those subsystems whose state and control variables are required to evaluate the local dynamics of subsystem p.
We emphasize that Definition 6.7 permits, in principle, to choose the neighboring index set as large as $\mathcal{I}^p(k) = \mathcal{P}$, that is, subsystem p may request information from all other subsystems. While this choice is always admissible from a modeling point of view, it is typically undesirable in practice: communication bandwidth, latency, and reliability constraints motivate to keep the neighboring index sets as small as possible. In particular, only those subsystems should be included whose states and/or controls actually influence the projected dynamics of subsystem p. The information exchanged from these subsystems will be referred to as *neighboring data*. Conceptually, neighboring data are precisely the variables that are not locally available to subsystem p, but are required to evaluate its projected dynamics and its MPC prediction model.

Definition 6.8 (Neighboring data).
Let $p \in \mathcal{P}$ and let $\mathcal{I}^p(k) \subseteq \mathcal{P} \setminus \{p\}$ be a neighboring index set of subsystem p at time k. The

neighboring data of subsystem p at time k is the set

$$i^p(k) := \left\{ (q,\, k_q,\, \mathbf{x}^q(\cdot),\, \mathbf{u}^q(\cdot)) \;\middle|\; q \in \mathcal{I}^p(k) \right\}. \tag{6.4}$$

Here $k_q \in \mathbb{N}_0$ is a time stamp and the trajectories satisfy

$$\mathbf{x}^q(\cdot) \in (\mathcal{X}^q)^{N+1}, \qquad \mathbf{u}^q(\cdot) \in (\mathcal{U}^q)^N.$$

The associated *neighboring data set* is given by

$$I^p = I^p = 2^Q \qquad Q = (\mathcal{P} \setminus \{p\}) \times \mathbb{N}_0 \times \mathcal{X}^{N+1} \times \mathcal{U}^N,$$

where 2^Q denotes the power set of Q.

To illustrate the latter, Figure 6.3 captures the concept of neighboring data for a fixed subsystem p at time k.

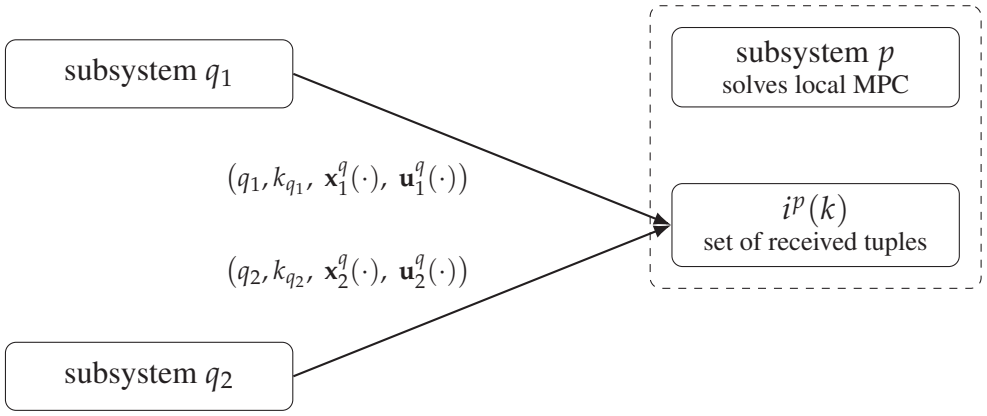


Figure 6.3.: Illustration of collecting neighboring data from those subsystems $q \in \mathcal{I}^p(k)$.

Subsystem p shall solve a local MPC problem based on its own state and control variables. However, due to coupling in the system dynamics, the evaluation of the projected dynamics $\pi^p_{\mathcal{X}} \circ f$ requires additional information from other subsystems.

Using the neighboring index set $\mathcal{I}^p(k)$, subsystem p identifies exactly those subsystems whose predicted behavior influences subsystem p. Each neighboring subsystem $q \in \mathcal{I}^p(k)$ transmits its data tuple. The collection of all such tuples forms the neighboring data set $i^p(k)$. This set constitutes the sole external information required by subsystem p to compute consistent predictions and control actions. In particular, subsystems not contained in $\mathcal{I}^p(k)$ do not contribute to the neighboring data and therefore do not need to communicate with subsystem p. Consequently, the size of the neighboring index set directly determines the communication requirements of the distributed MPC scheme.

Task 6.9
Reconsider Task 6.2 and compute neighboring index set and neighboring data.

Solution to Task 6.9: For the chosen split of state and control variables, subsystem 1 depends on variables of subsystem 2, whereas subsystem 2 does not depend on any other subsystem. Consequently, the neighboring index sets are given by $\mathcal{I}^1(k) = \{2\}$ and $\mathcal{I}^2(k) = \emptyset$. As shown in the solution of Task 6.2, the evaluation of the projected dynamics of subsystem 1 requires access to the state and control variables of subsystem 2. Hence, the neighboring data of subsystem 1 at time k is given by $i^1(k) = \{(2, k, \mathbf{x}^2(k), \mathbf{u}^2(k))\}$ while $i^2(k) = \emptyset$.

This example illustrates an important distinction: the neighboring data required to evaluate the *instantaneous* system dynamics differs from the information required to solve a model predictive control problem. While a single time instance may suffice to compute the right-hand side of the dynamics, MPC requires predictions over an entire horizon.
In particular, to generate a predicted state trajectory for subsystem p, the predicted state and control trajectories of all subsystems in the neighboring index set must be available.

Remark 6.10
For simplicity, we assume throughout this chapter that all subsystems use the same prediction horizon length N. Under this assumption, the horizon length does not need to be transmitted as part of the neighboring data. If subsystem-dependent horizons are allowed, the neighboring data must be extended to include this information, leading to a definition of the form

$$i^p(k) = \{(q, k^q, N^q, \mathbf{x}^q(\cdot), \mathbf{u}^q(\cdot)) \mid q \in \mathcal{I}^p(k)\} \in I^p. \tag{6.5}$$

Although extending the transmitted data is conceptually straightforward, the resulting algorithmic modifications — such as horizon alignment and synchronization — as well as the corresponding stability analysis become considerably more involved.

Using the concept of neighboring data, we can now formally relate the overall system dynamics to the set of subsystem dynamics.

Corollary 6.11 (Equivalent subsystem split).
Suppose an overall system of the form (6.2) *is given together with an index set $\mathcal{P} = \{1, \dots, P\}$ and families of projections $\left(\pi_{\mathcal{X}}^p\right)_{p\in\mathcal{P}}$, $\left(\pi_{\mathcal{U}}^p\right)_{p\in\mathcal{P}}$, which induce decompositions of state and control spaces $\langle(\mathcal{X}^p)_{p\in\mathcal{P}}\rangle$ and $\langle(\mathcal{U}^p)_{p\in\mathcal{P}}\rangle$.*

Then the overall system (6.2) *is equivalent to a set of subsystems* (6.1)*, where the dynamics of subsystem p are given by*

$$\begin{aligned} f^p(\mathbf{x}^p, \mathbf{u}^p, (\widetilde{\mathbf{x}}^p, \widetilde{\mathbf{u}}^p)) :=& [Id^{n_x^p \times n_x^p}; 0^{(n_x - n_x^p) \times n_x^p}] \circ \pi_{\mathcal{X}}^p \\ & \circ f(\sigma_{\mathcal{X}^p}{}^{-1}(\mathbf{x}^p, \widetilde{\mathbf{x}}^p, 0), \sigma_{\mathcal{U}^p}{}^{-1}(\mathbf{u}^p, \widetilde{\mathbf{u}}^p, 0)) \end{aligned} \tag{6.6}$$

for permutations $\sigma_{\mathcal{X}^p} : \mathcal{X} \to \mathcal{X}^p \times \widetilde{\mathcal{X}}^p \times \overline{\mathcal{X}}^p$ satisfying $\sigma_{\mathcal{X}^p}(\mathbf{x}) = (\mathbf{x}^p, \widetilde{\mathbf{x}}^p, \overline{\mathbf{x}}^p)$ and $\sigma_{\mathcal{U}^p} : \mathcal{U} \to \mathcal{U}^p \times \widetilde{\mathcal{U}}^p \times \overline{\mathcal{U}}^p$ with $\sigma_{\mathcal{U}^p}(\mathbf{u}) = (\mathbf{u}^p, \widetilde{\mathbf{u}}^p, \overline{\mathbf{u}}^p)$ for all $p \in \mathcal{P}$.

This result shows that, once neighboring data are introduced, the overall system dynamics can be represented exactly by a set of coupled subsystems. The coupling is captured by the costate and cocontrol variables, which correspond to the information exchanged via neighboring data.
Returning to the definition of the neighboring index sets, we observe that the projections are not uniquely determined. Nevertheless, from both a computational and a communication perspective, it is advisable to choose them such that the neighboring index sets are as small as possible. At the same time, the subsystems do not depend on the independent state and control subspaces $\overline{\mathcal{X}}^p$ and $\overline{\mathcal{U}}^p$. Maximizing these independent subspaces therefore directly reduces the computational burden of the local MPC problems.
As outlined at the beginning of this section, the projection-based approach is not limited to the system dynamics, but can be applied analogously to the cost functionals and constraints of the MPC problem. In the presence of state or control constraints, the projections defining costate and independent components may depend on the current overall system state $\mathbf{x} \in \mathcal{X}$. This state dependence must be taken into account when defining admissible neighboring data and feasible local optimization problems. Using these projections, we are now in a position to formulate a *projected digital constrained optimal control problem*, which serves as the local optimization problem solved by each subsystem within a distributed MPC scheme.

Definition 6.12 (Projected digital constrained optimal control problem).
Consider a digital constrained optimal control problem (5.3), a set $\mathcal{P} = \{1, \dots, P\}$ as well as projections $\left(\pi_{\mathcal{X}}^p\right)_{p \in \mathcal{P}}$, $\left(\pi_{\mathcal{U}}^p\right)_{p \in \mathcal{P}}$ inducing a decomposition. Then we call

$$\begin{aligned} &\min\ J^p(\mathbf{x}_0^p, \mathbf{u}^p) = \sum_{k=0}^{N-1} \ell^p(\mathbf{x}^p(k, \mathbf{x}_0^p, \mathbf{u}^p), \mathbf{u}^p(k)) \quad \text{over all } \mathbf{u}^p \in \mathbb{U}_{\mathbb{X}_0^p}^{p,N} \\ &\text{subject to } \mathbf{x}^p(k+1) = f^p(\mathbf{x}^p(k), \mathbf{u}^p(k)), \quad \mathbf{x}^p(0) = \mathbf{x}_0^p \\ &\qquad\qquad \mathbf{x}^p(k) \in \mathbb{X}^p, \quad k \in [0, N] \end{aligned} \tag{6.7}$$

a *projected digital finite constrained optimal control problem.*

This definition mirrors the standard digital constrained optimal control problem from Definition5.8 used in standard (or centralized) MPC, with the crucial difference that all quantities are defined on the projected subsystem dynamics. The state $\mathbf{x}^p(\cdot)$ and control $\mathbf{u}^p(\cdot)$ represent only the local components assigned to subsystem p. Coupling to other subsystems enters implicitly through the dynamics f^p, costs ℓ^p and constraints $\mathbb{X}^p$, which depend on neighboring data. Consequently, each subsystem optimizes its own performance criterion while respecting local constraints, but remains consistent with the overall system through exchanged neighboring data. The projected optimal control problem defined above forms the computational core of a distributed MPC scheme. A basic realization of such a scheme is given in Algorithm 6.13. It follows the standard MPC loop — measure, optimize, apply — but augments it by communication and coordination steps required to handle subsystem coupling.

Algorithm 6.13 (Basic MPC Algorithm)
For each closed loop time index $n = 0, 1, 2 \ldots$:

(1) For each subsystem $p \in \mathcal{P}$

Obtain the state $\mathbf{x}^p(n) \in \mathbb{X}^p$ of the system.

(2) For each subsystem $p \in \mathcal{P}$

a) Obtain neighboring index set $\mathcal{I}^p(n)$ and collect data i^p.

b) Set $\mathbf{x}_0^p := \mathbf{x}^p(n)$, solve the projected digital finite optimal control problem (6.7) and denote the obtained optimal control sequence by $\mathbf{u}^{p,\star}(\cdot) \in \mathbb{U}_{\mathbb{X}_0^p}^{p,N}(\mathbf{x}_0^p, i^p)$.

c) Send data $(p, \mathbf{x}^p(\cdot), \mathbf{u}^p(\cdot))$ to all subsystems $q \in \mathcal{P} \setminus \{p\}$.

until $\mathbf{u}^p(\cdot)$ and i^p has converged for all $p \in \mathcal{P}$.

(3) For each subsystem $p \in \mathcal{P}$

Define the MPC feedback $\mu_N^p(\mathbf{x}^p(n)) := \mathbf{u}^{p,\star}(0)$.

Within Algorithm 6.13, Step (1) corresponds to the standard measurement or state-estimation step of MPC, performed independently by each subsystem. In Step (2), each subsystem identifies the set of neighboring subsystems that influence its dynamics and collects the corresponding neighboring data. Based on this information, the projected optimal control problem is solved locally. Since the dynamics depend on neighboring predictions, the optimization problems are generally coupled; the inner loop therefore iterates until the exchanged trajectories and neighboring data are consistent across all subsystems. Finally, in Step (3), each subsystem applies only the first element of its optimal control sequence, yielding a distributed MPC feedback law in receding-horizon fashion.

Algorithm 6.13 serves as a conceptual baseline. The central question addressed in the following sections is how Step (2) — in particular the exchange and reconciliation of neighboring data — can be organized efficiently. This leads to different coordination strategies, ranging from sequential and prioritized schemes to fully parallel and iterative methods.

Remark 6.14
Distributed optimal control problems span a wide range of architectures. At one extreme, a centralized formulation treats all subsystems as a single large system. At the other extreme, a decentralized formulation assumes complete independence, i.e., all variables of other subsystems are independent and no communication is required. Between these extremes lie cooperative settings, where subsystems share identical performance objectives, and noncooperative settings, where objectives may differ and coordination is required to resolve conflicts.

As in standard model predictive control, all distributed MPC schemes rely on a basic feasibility assumption. In the present setting, feasibility does not only concern the existence of a locally admissible control sequence for a given state, but also the existence of consistent neighboring data such that all local projected optimal control problems can be solved and coordinated successfully. This requirement is formalized in the following assumption.

Assumption 6.15 (Feasibility)
Given Algorithm 6.13 suppose that for each $\mathbf{x}(n) = \left(\mathbf{x}^1(n), \dots, \mathbf{x}^p(n)\right)$ obtained in Step 1 there exists $i^p(n)$ for all $p \in \mathcal{P}$ such that $\mathbb{U}^{p,N}_{\mathbb{X}^p_0}(\mathbf{x}^p_0, i^p) \neq \emptyset$ and Step 2 terminates successfully.

Assumption 6.15 ensures that, at every closed-loop time step, each subsystem can construct a feasible local MPC problem once appropriate neighboring data are available. Moreover, it requires that the coordination procedure in Step 2 of Algorithm 6.13 — including data exchange and possible iterative reconciliation — terminates in finite time. In other words, feasibility here is a joint property of the subsystem dynamics, the projection-based decomposition, and the coordination mechanism.
This assumption is the distributed analogue of the feasibility assumption which we introduced in Definition 5.13 for standard (or centralized) MPC, where one assumes that, for every admissible state encountered in closed loop, the finite-horizon optimal control problem admits a feasible solution.
Under the feasibility assumption, the distributed MPC scheme inherits a fundamental property known from standard MPC.

Theorem 6.16 (Recursive feasibility of distributed NMPC).
Consider Algorithm 6.13 and suppose Assumption 6.15 to hold. Then the closed loop is recursively feasible.

Recursive feasibility means that, once Algorithm 6.13 is initialized with a feasible state, feasibility is preserved for all subsequent closed-loop time steps. The proof follows the same high-level argument as in centralized MPC: at each time step, only the first control input of a feasible sequence is applied, and a new feasible solution is assumed to exist at the next time step.
The essential difference to standard MPC lies in the source of feasibility. In centralized MPC, recursive feasibility is typically established by suitable terminal constraints, terminal costs, or invariant sets. In the distributed setting, recursive feasibility additionally depends on the existence of consistent neighboring data and on the successful coordination of subsystem optimizations. Thus, feasibility is no longer purely a property of the system dynamics and constraints, but also of the communication and coordination structure.
Unfortunately, Assumption 6.15 cannot be guaranteed in full generality. In particular, the existence of suitable neighboring data and the convergence of the coordination step may fail if the coupling between subsystems is strong, if communication is restricted, or if the local constraint sets are incompatible. For this reason, the remainder of this chapter focuses on specific classes of distributed MPC schemes and coordination strategies for which recursive feasibility can be established under additional structural assumptions.

6.2. Sequential approach

The first distributed MPC approach we discuss is based on a temporal decoupling of the coordination problem. Instead of solving all subsystem optimal control problems simultaneously, the subsystems are ordered and solved sequentially within each sampling instant. This approach is commonly referred to as the Richards and How algorithm introduced in [8].
The key idea is to impose a strict order on the subsystems and to propagate information along this order. Subsystem 1 computes its control first and transmits its predicted trajectory to all remaining subsystems. Subsystem 2 then computes its control using this information, and so on, until the last subsystem P has solved its local optimal control problem. The resulting communication structure is illustrated in Figure 6.4.
Note that the sequence of subsystems is not prescribed by the method, but instead to be chosen freely. Hence, a good choice of such a sequencing is of great importance and to some extend an open question. In practice, it is often advantageous to place strongly coupled or safety-critical subsystems early in the sequence.

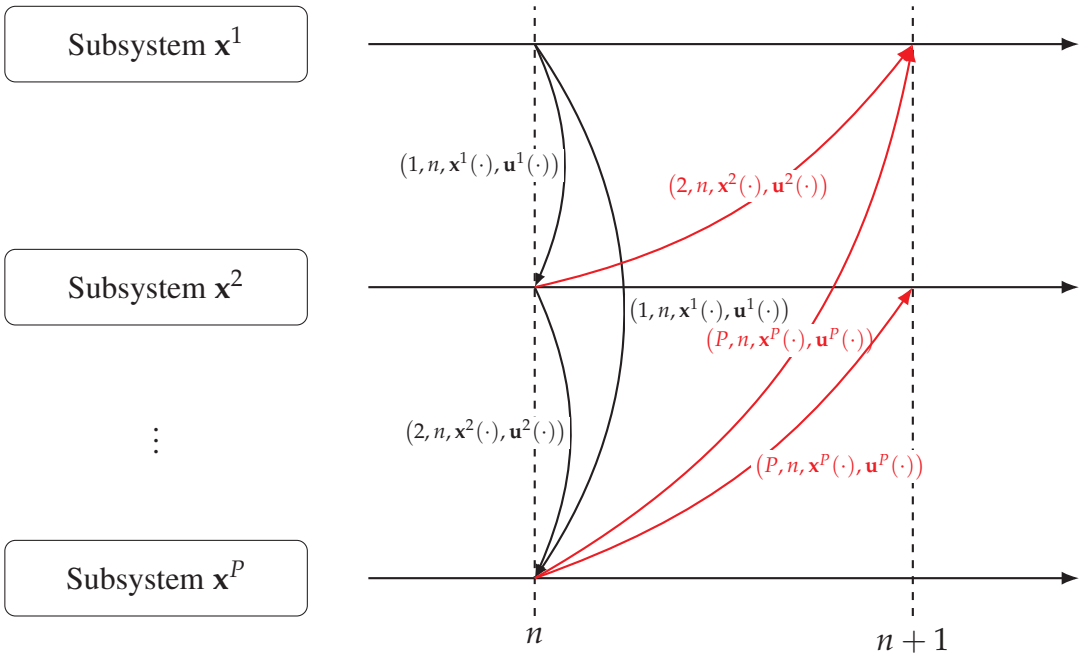


Figure 6.4.: Sequential communication structure of the Richards and How DMPC scheme

From a conceptual point of view, this scheme is closely related to Gauss–Seidel–type iterations known from numerical optimization: later subsystems profit from the most recent information, whereas earlier subsystems must rely on outdated predictions of later ones.

As highlighted by the red arrows in Figure 6.4, the utilized sequence induces an inherent information delay. Information generated by subsystem $\mathbf{x}^p$ at time n can only be exploited by subsystems $\mathbf{x}^q$ with $q < p$ at the next time instant $n+1$. Hence, for some subsystems, the neighboring data available within the current MPC iteration is incomplete at the end of the prediction horizon. Note that we have already seen this problem in standard (centralized) MPC during the iteration step: By shifting the horizon by one step, this lack of information corresponds to the missing time step at the end of the neighboring trajectories. Depending on the horizon shift strategy, this delay may be larger or smaller in practice.

To compensate for this missing information and to preserve feasibility of the local optimal control problems, we introduce the notion of a neighboring data extension.

Definition 6.17 (Neighboring data extension).
Consider a neighboring index set $\mathcal{I}^p(k)$ of subsystem $p \in \mathcal{P}$. We call the set

$$\widetilde{i}^p(k) = \{(q, k_q, \mathbf{x}^q(\cdot), \mathbf{u}^q(\cdot)) \mid q \in \mathcal{I}^p(k)\} \in \widetilde{I}^p(i^p) \tag{6.8}$$

neighboring data extension if

- $\mathbf{x}^q(\cdot)$ and $\mathbf{u}^q(\cdot)$ is defined for $k = 0, \ldots, N-1$ and

- for all undefined $\mathbf{x}^q(\cdot)$ and $\mathbf{u}^q(\cdot)$ an admissible solution is substituted.

The neighboring data extension fills the gap created by the sequential communication structure. Whenever predicted trajectories from neighboring subsystems are not available for the full horizon, an admissible continuation is appended. This continuation does not need to be optimal; it merely has to preserve feasibility of the local projected optimal control problem. There are multiple options on how to target this continuation. The most simple idea is to copy the last neighboring data entry. Note that while simple, such a procedure upholds admissibility not in all cases. Yet we have already discussed the case of terminal conditions. If the system has reached the desired equilibrium point within the prediction horizon, then we can last state/control entry may simply be copied.
More generally, such an extension always exists in MPC schemes based on terminal constraints. The reason for the latter is that the terminal point corresponds to an equilibrium of the subsystem dynamics, hence applying the control corresponding to the equilibrium points results in vanished dynamics of the system. For MPC with terminal costs, existence of an extension can be ensured if the terminal region is chosen such that a Lyapunov based controller can be evaluated for the linearization of the system. In the distributed case, we additionally require that all trajectories within this region are independent variables for all other subsystems.
Combining the sequential communication structure from Figure 6.4, the concept of neighboring data extension, and the basic distributed MPC Algorithm 6.13, we obtain the following scheme.

Algorithm 6.18 (Richards and How Algorithm for Distributed NMPC)
Initialization:

(1) For each subsystem $p \in \mathcal{P}$

Obtain the state $\mathbf{x}^p(0) \in \mathbb{X}^p$ of the system.

(2) For each subsystem $p \in \mathcal{P}$

a) Find control sequences $\mathbf{u}^{p,\star}(\cdot) \in \mathbb{U}^{p,N}_{\mathbb{X}^p_0}(\mathbf{x}^p_0)$ such that the overall system is feasible.

b) Send data $(p, \mathbf{x}^p(\cdot), \mathbf{u}^p(\cdot))$ to all subsystems $q \in \mathcal{P} \setminus \{p\}$.

(3) For each subsystem $p \in \mathcal{P}$

a) Define the MPC feedback $\mu^p_N(\mathbf{x}^p(0)) := \mathbf{u}^{p,\star}(0)$.

Feedback loop: For each closed loop time index $n = 1, 2 \ldots$:

(1) For each subsystem $p \in \mathcal{P}$

Obtain the state $\mathbf{x}^p(n) \in \mathbb{X}^p$ of the system.

(2) For each subsystem $p \in \mathcal{P}$ do sequentially

a) Collect neighboring data i^p for all subsystems and extend neighboring data for all subsystems $j > p$.

b) Set $\mathbf{x}_0^p := \mathbf{x}^p(n)$, solve the projected digital finite optimal control problem (6.7) and denote the obtained optimal control sequence by $\mathbf{u}^{p,\star}(\cdot) \in \mathbb{U}^{p,N}_{\mathbb{X}_0^p}(\mathbf{x}_0^p, i^p)$.

c) Send data $(p, \mathbf{x}^p(\cdot), \mathbf{u}^p(\cdot))$ to all subsystems $q \in \mathcal{P} \setminus \{p\}$.

until $\mathbf{u}^p(\cdot)$ and i^p has converged for all $p \in \mathcal{P}$.

(3) For each subsystem $p \in \mathcal{P}$

a) Define the MPC feedback $\mu_N^p(\mathbf{x}^p(n)) := \mathbf{u}^{p,\star}(0)$.

To illustrate the Richards–How algorithm and, in particular, the role of neighboring data and neighboring data extensions, we now revisit the simple two-dimensional system introduced earlier. Despite its low dimension, this example already exhibits asymmetric coupling between subsystems, one-step information delays induced by the computation order, and the necessity of admissible trajectory extensions.

Task 6.19

Consider the system from Task 6.2. Assume a common prediction horizon $N \in \mathbb{N}$ and local constraints $\mathbf{x}^1(\cdot) \in \mathbb{X}^1$, $\mathbf{x}^2(\cdot) \in \mathbb{X}^2$, $\mathbf{u}^2(\cdot) \in \mathbb{U}^2$.

(a) Revisit the subsystem dynamics f^1, f^2 from Task 6.2 in the form

$$\mathbf{x}^p(k+1) = f^p(\mathbf{x}^p(k), \mathbf{u}^p(k), i^p(k)), \qquad p \in \{1,2\},$$

including the minimal choice of neighboring index sets $\mathcal{I}^p(k)$.

(b) Explain why, under the Richards–How sequential computation order $1 \to 2$, subsystem 1 requires a neighboring data extension at time n.

(c) Construct one admissible neighboring data extension for subsystem 1 at time n using the previously transmitted predicted trajectories of subsystem 2 from time $n-1$. Suppose subsystem 2 to have reached an equilibrium within the prediction horizon.

Solution to Task 6.19: **(a)** By construction, subsystem 2 contains the control input, whereas subsystem 1 does not. Writing $\mathbf{x}^1 = \mathbf{x}_1$, $\mathbf{x}^2 = \mathbf{x}_2$, $\mathbf{u}^2 = \mathbf{u}$ yields:

$$\mathbf{x}^2(k+1) = \mathbf{x}_2(k+1) = \mathbf{x}_2(k) + \mathbf{u}(k) = \mathbf{x}^2(k) + \mathbf{u}^2(k).$$

Hence, subsystem 2 is autonomous w.r.t. other subsystems and we can choose

$$\mathcal{I}^2(k) = \emptyset, \qquad i^2(k) = \emptyset,$$

and

$$f^2(\mathbf{x}^2, \mathbf{u}^2, \emptyset) = \mathbf{x}^2 + \mathbf{u}^2.$$

Subsystem 1 evolves according to

$$\mathbf{x}^1(k+1) = \mathbf{x}_1(k+1) = \mathbf{x}_1(k) + \mathbf{x}_2(k) + \mathbf{u}(k)/2 = \mathbf{x}^1(k) + \mathbf{x}^2(k) + \mathbf{u}^2(k)/2.$$

Thus, subsystem 1 requires the neighbor state and control from subsystem 2. A minimal neighboring index set is therefore

$$\mathcal{I}^1(k) = \{2\}.$$

In the trajectory-based neighboring-data notation (for MPC), subsystem 1 uses

$$i^1(n) = \left\{ \left(2,\ \tau_2,\ \mathbf{x}^2(\cdot),\ \mathbf{u}^2(\cdot)\right) \right\},$$

and we may write

$$f^1(\mathbf{x}^1, \mathbf{u}^1, i^1) = \mathbf{x}^1 + \mathbf{x}^2 + \mathbf{u}^2/2,$$

where $\mathbf{x}^2, \mathbf{u}^2$ are taken from the received neighboring data tuple.
(b) In the Richards–How scheme, at closed-loop time n the subsystem problems are solved sequentially. As subsystem 1 solves first, we obtain that its dynamics depend on the predicted trajectories $\mathbf{x}^2(\cdot)$ and $\mathbf{u}^2(\cdot)$ of subsystem 2 over the horizon. Since subsystem 2 has not been solved yet at time n, subsystem 1 cannot access the current predictions of subsystem 2. Instead, it can only use the most recently transmitted prediction from time $n-1$, which — after the horizon shift — typically leaves a missing point near the end of the horizon. This resembles the „one-step lack“ in neighboring data induced by the sequential order.

(c) Suppose that at time $n-1$ subsystem 2 transmitted the predicted trajectories

$$\mathbf{x}^2_{n-1}(\cdot) = \big(\mathbf{x}^2_{n-1}(0), \dots, \mathbf{x}^2_{n-1}(N)\big), \quad \mathbf{u}^2_{n-1}(\cdot) = \big(\mathbf{u}^2_{n-1}(0), \dots, \mathbf{u}^2_{n-1}(N-1)\big).$$

After the shift from n to $n+1$, we require the input from subsystem 2 at $k = N-1$ (and the corresponding terminal state). To this end, we choose the steady state control $\mathbf{u}^{p,\star}$

$$\tilde{\mathbf{u}}^2(N-1) = \mathbf{u}^{p,\star},$$

and define the final state consistently using subsystem 2 dynamics

$$\mathbf{x}^2(N) = \mathbf{x}^2(N-1).$$

Since the constraint sets $\mathbb{X}^2$ and $\mathbb{U}^2$ are satisfied by this continuation, the resulting trajectories yield an admissible neighboring data extension

$$\widetilde{i}^1(n) = \Big\{ \big(2, n-1, \mathbf{x}^2(\cdot), \mathbf{u}^2(\cdot)\big) \Big\},$$

which allows subsystem 1 to solve its projected finite-horizon problem at time n despite the sequential delay.

This asymmetry shown in Task 6.19 is typical for sequential schemes and already suggests a natural ordering of subsystems. The example also clarifies how recursive feasibility in the Richards–How algorithm is maintained. As long as the neighboring data extension is admissible and the local terminal conditions of the subsystems are chosen appropriately, the feasibility of the projected optimal control problems is preserved. In this sense, the neighboring data extension plays a role analogous to terminal constraints or terminal costs in centralized MPC: it provides a structured mechanism to guarantee the existence of feasible continuations.

Note that the tricky part is the initialization phase, which must guarantee that the distributed MPC loop starts from a globally feasible configuration. Hence, the initial states of all subsystems must lie in the feasible region of the overall system and, at minimum, the initial state should not violate any state or coupling constraints outright. Similarly, any shared resource or coupled input constraint must be satisfied by the initial configuration. While there are several quite involved schemes to address this issue, one common idea to satisfy this initialization, an equilibrium or a known safe configuration inside the terminal sets for all subsystem is chosen as initial state. Hence, the „do nothing“ approach of holding position for all subsystems will do the trick.

During the feedback loop, subsystems are processed sequentially. Each subsystem solves its projected optimal control problem using the most recent information available and propagates its

updated predictions to all other subsystems. The algorithm thus trades parallelism for simplicity and robustness.
Regarding recursive feasibility, we can convert out stability results for centralized MPC problems from Chapter 5 to obtain the following:

Theorem 6.20 (Stability of Richards and How Algorithm).
Consider Algorithm 6.18. If the initialization phase exhibits a solution, then we have

$$\mathbb{U}_{\mathbb{X}_0^p}^{p,N}(\mathbf{x}_0^p, i^p) \neq \emptyset \qquad \forall n \in \mathbb{N}. \tag{6.9}$$

If additionally the stability conditions from either Theorem 5.22, Theorem 5.24 or Theorem 5.25 hold for each subsystem $p \in \mathcal{P}$, then the closed loop of the overall system is asymptotically stable.

This result shows that the Richards–How scheme effectively reduces the distributed MPC problem to a sequence of subproblems each resembling a local standard MPC. Stability is inherited directly from classical MPC theory, provided that each subsystem satisfies the usual terminal conditions. In this sense, the sequential approach represents the closest distributed analogue to centralized MPC.
While the Richards and How algorithm is straightforward to implement and analyze, its sequential nature leads to significant waiting times for subsystems later in the order. This is particularly inefficient when subsystems are weakly coupled or completely independent. These limitations motivate the parallel and partially parallel coordination strategies discussed in the following sections.

6.3. Hierarchical approach

In the previous section, we considered a purely sequential coordination strategy, where subsystems are totally ordered and solved one after another. The hierarchical approach generalizes this idea by exploiting partial orders induced by the dependency structure of the subsystems. Instead of enforcing a single chain, subsystems are arranged in a dependency tree (or forest), allowing subsystems without mutual dependencies to operate in parallel.
The core idea is to decouple communication from dependency. While communication may occur between many subsystems, only those subsystems whose projected dynamics actually depend on one another must be ordered. All others may be grouped into the same hierarchy level and optimized concurrently. This separation is illustrated in Figure 6.5: dashed edges represent potential communication links, while solid directed edges encode true dependencies that impose an order.

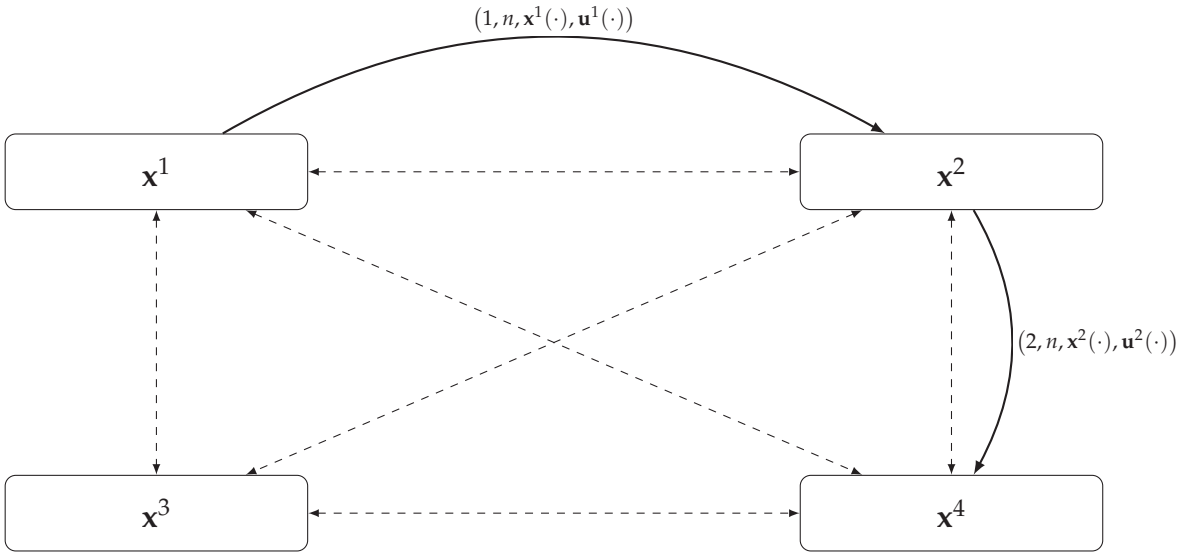


Figure 6.5.: Communication graph (dashed) and dependency graph (solid) in a hierarchical DMPC scheme

The communication graph describes which subsystems may exchange neighboring data. This graph is typically dense, reflecting physical proximity, shared constraints, or communication capabilities. The dependency graph encodes which subsystem must be solved before another because its state or control enters as a costate or cocontrol. This graph is generally much sparser and may change over time.

Here, the most important point already arises: Only the dependency graph determines the hierarchy; communication alone does not enforce ordering.

To make use of this decoupling, we must identify those systems, which are independent from one another. Using the denomination from our projection, we directly obtain:

Corollary 6.21 (Independence of systems).
Consider a decomposition of system (6.2) *using a set of projections* $(\pi^p)_{p\in\mathcal{P}}$. *Given a current state of the overall system* $\mathbf{x} \in \mathcal{X}$, *then subsystems* p *and* q *are independent if*

$$\widetilde{\mathcal{X}}^p = \emptyset, \quad \widetilde{\mathcal{U}}^p = \emptyset \qquad \textit{and} \qquad \widetilde{\mathcal{X}}^q = \emptyset, \quad \widetilde{\mathcal{U}}^q = \emptyset. \tag{6.10}$$

Corollary 6.21 provides a precise system-theoretic criterion for parallelism: two subsystems are independent if neither requires costate nor cocontrol variables from the other. This definition is deliberately strict—it ensures that parallel optimization does not introduce hidden coupling via dynamics or constraints.

Using this independence, we know that certain sets of systems may operate in parallel. The corresponding Definition 6.22 formalizes sets of subsystems that may be optimized in parallel.

Definition 6.22 (List of parallel operational systems).
Consider a decomposition of system (6.2) using a set of projections $(\pi^p)_{p\in\mathcal{P}}$. Then we call the set of sets $\mathcal{L} \in 2^{\mathcal{P}}$ satisfying

$$\mathcal{L} := \{p \in \mathcal{P} \mid (6.10) \text{ holds}\} \tag{6.11}$$

list of parallel operational systems.

Note that multiple such sets may exist, hence additional rules are required to arrive at a unique and well-defined hierarchy.
Since Definition 6.22 is formulated on the power set $2^{\mathcal{P}}$, there is, in general, not a unique "best" choice of a list of parallel operational systems. Indeed, even if independence relations are fixed, multiple maximal parallel groups may exist.
To see this, consider three subsystems with the dependency pattern "2 depends on 3", while subsystem 1 is independent of both. Then subsystem 1 may be placed in the same parallel level either with subsystem 2 or with subsystem 3. Both choices are admissible, but they lead to different execution schedules and different communication patterns. Hence, the power-set formulation reflects the fact that a hierarchy is not uniquely determined by independence alone; an additional selection principle is required.
To obtain a well-defined and reproducible schedule, we introduce two operators: one that resolves ambiguity by selecting an order (priority), and one that removes outdated ordering information when dependencies may change (deordering).

Definition 6.23 (Priority and deordering rule).
We call the operator $\Pi : 2^{\mathcal{P}} \to 2^{\mathcal{P}}$ priority rule and the operator $\Delta : 2^{\mathcal{P}} \to 2^{\mathcal{P}}$ deordering rule.

The priority rule serves to impose a deterministic ordering within a set of subsystems that may operate in parallel. In particular, it resolves ties whenever multiple valid parallel groupings exist and thereby turns the (potentially non-unique) decomposition into a concise hierarchy suitable for implementation.

Task 6.24
Give an example of a priority rule.

Solution to Task 6.24: The lexicographical order $<_{\mathbb{N}}$ is a priority rule that sorts subsystems according to their index. In addition, it implicitly selects that list of parallel operational systems for which subsystems are assigned to the lowest possible hierarchy level compatible with the ordering.

A priority rule resolves the ambiguity arising from multiple admissible parallel groupings. Its role is purely organizational: it does not alter feasibility or stability, but it ensures that the hierarchical structure is uniquely determined and reproducible. In practice, priority rules often encode engineering preferences, such as favoring safety-critical subsystems, subsystems with faster dynamics, or simply lower indices.
The idea of the deordering rule is fundamentally different. Dependencies between subsystems are not static: they may appear or disappear depending on the current system state, the activity of constraints, or changes in the operating regime. Whenever a dependency arises, the affected subsystems must be separated into different hierarchy levels to respect the induced order.
However, when such a dependency ceases to exist, the corresponding ordering should ideally be revoked to re-enable parallel computation. Unfortunately, this cannot be detected reliably from the solutions of the local MPC problems alone. The reason is subtle but important: if a dependency is induced by a constraint, then feasibility of the solution already enforces this dependency implicitly. As a consequence, the absence of constraint violations does not imply the absence of a potential dependency — it merely indicates that the current solution satisfies all constraints.
In other words, we can detect dependencies when they become active, but we cannot safely infer their absence from feasibility alone. For this reason, hierarchical DMPC schemes typically employ simple and conservative deordering rules that periodically remove assumed dependencies and force the hierarchy to be rebuilt.

Task 6.25
Give an example of a deordering rule.

Solution to Task 6.25: The operator $\Delta(\mathcal{L}) = \emptyset$ is a deordering rule. It removes all previously assumed dependencies, thereby forcing the hierarchy to be reconstructed before the next optimization step.

Note that fully discarding all structure may be counterproductive as cycles could occur such as identical but mirrored hierarchies for opposing subsystem. Hence, a more structure-preserving

alternative may be applied by removing only a single dependency, for example chosen at random. Such partial deordering rules may reduce unnecessary reordering at the expense of slower adaptation to changing dependencies.
Combining the priority and deordering rules with Algorithm 6.13 yields the hierarchical DMPC scheme stated in Algorithm 6.26.

Algorithm 6.26 (Hierarchical DMPC Algorithm)
For each closed loop time index $n = 0, 1, 2 \ldots$:

(1) For each subsystem $p \in \mathcal{P}$

Obtain the state $\mathbf{x}^p(n) \in \mathbb{X}^p$ of the system.

(2a) **Deordering**
For each j from 2 to P

For k from 1 to $\#\mathcal{L}_j$

i. Set $\mathcal{I}^p{}_k(n) := \Delta(\mathcal{I}^p{}_k(n))$

ii. If $\mathcal{I}^p{}_k(n) = \emptyset$ remove p_k from $\mathcal{L}_j$ and set $\mathcal{L}_1 := (\mathcal{L}_1, p_k)$
Else if $\tilde{m} = \min_{k \in \mathcal{L}_m, p_k \in \mathcal{I}^p{}_k(n)} m < j$, remove p_k from $\mathcal{L}_j$ and set $\mathcal{L}_{\tilde{m}} := (\mathcal{L}_{\tilde{m}}, p_k)$

a) Set $\mathbf{x}_0^p := \mathbf{x}^p(n)$, solve the projected digital finite optimal control problem (6.7) and denote the obtained optimal control sequence by $\mathbf{u}^{p,\star}(\cdot) \in \mathbb{U}^{p,N}_{\mathbb{X}_0^p}(\mathbf{x}_0^p, i^p)$.

b) Send data $(p, \mathbf{x}^p(\cdot), \mathbf{u}^p(\cdot))$ to all subsystems $q \in \mathcal{L}_k$ with $k > j$.

(2b) **Priority**
For each j from 1 to P do

a) If $\#\mathcal{L}_j \in \{0, 1\}$ goto Step 3. Else sort index via $\mathcal{L}_j := \Pi(\mathcal{L}_j)$.

b) Collect neighboring data i^p for all subsystems.

c) For k from 2 to $\#\mathcal{L}_j$ do

If p_k exhibits costate/cocontrol of p_k, $k < k$, set $\mathcal{L}_{j+1} := (\mathcal{L}_{j+1}, k)$ and $\mathcal{L}_j := \mathcal{L}_j \setminus \mathcal{L}_{j+1}$

d) Solve the projected digital finite optimal control problem (6.7) and denote the obtained optimal control sequence by $\mathbf{u}^{p,\star}(\cdot) \in \mathbb{U}^{p,N}_{\mathbb{X}_0^p}(\mathbf{x}_0^p, i^p)$.

e) Send data $(p, \mathbf{x}^p(\cdot), \mathbf{u}^p(\cdot))$ to all subsystems $q \in \mathcal{L}_k$ with $k \geq j$.

(3) For each subsystem $p \in \mathcal{P}$

Define the MPC feedback $\mu_N^p(\mathbf{x}^p(n)) := \mathbf{u}^{p,\star}(0)$.

The algorithm alternates between two conceptually distinct phases.

- Step 2a (Deordering): This phase removes outdated or potentially obsolete dependencies. Subsystems are promoted to lower hierarchy levels whenever no current dependency justifies their previous ordering. Information is only transmitted to strictly higher levels, since dependencies within the same level are still under assessment.

- Step 2b (Priority): This phase reconstructs a consistent hierarchy. Subsystems within the same level are ordered using the priority rule, and any newly detected costate or cocontrol dependencies trigger promotion to higher levels. Neighboring data are now transmitted to subsystems on equal or higher levels, reflecting the fact that dependencies within the same level have been resolved by ordering.

We like to stress that in Step 2b, neighboring information is sent to subsystems on equal or higher hierarchy levels, whereas in Step 2a only higher levels are addressed. This distinction is essential: in order to establish the dependency graph correctly, a subsystem must be able to determine whether another subsystem on the same level induces a costate or cocontrol. If such a dependency is detected, the priority rule enforces a promotion to a higher level, thereby restoring a valid hierarchy.
The hierarchical approach aims to increase parallelism by exploiting independence between subsystems. However, its effectiveness is fundamentally limited by the structure of the underlying system. If subsystems remain persistently dependent on one another — due to strongly coupled dynamics or permanently active coupling constraints — the hierarchy may collapse into a chain. In this case, the hierarchical DMPC scheme degenerates to a purely sequential approach, and no parallelism can be achieved.
This observation motivates the consideration of fully parallel coordination strategies, which aim to handle persistent dependencies without imposing a strict hierarchy.

6.4. Parallel approach

A fundamentally different way of decoupling subsystems is to remove constraints and couplings from the problem formulation by incorporating them into the cost functional. The resulting problem is formally unconstrained and can then be decomposed additively. This idea underlies a broad class of methods commonly referred to as dual decomposition or Lagrangian-based distributed optimization.
In contrast to the sequential and hierarchical approaches, where feasibility is enforced explicitly at every step, the parallel approach allows all subsystems to compute their control inputs simultaneously, at the price of introducing an additional coordination variable: the Lagrange multiplier.

This multiplier is updated by a central entity (server) and steers the subsystems toward consensus and constraint satisfaction.

As a consequence, the communication structure is qualitatively different and is no longer acyclic. Instead, several rounds of communication between subsystems and the server are required within a single closed-loop time step. The corresponding structure is illustrated in Figure 6.6.

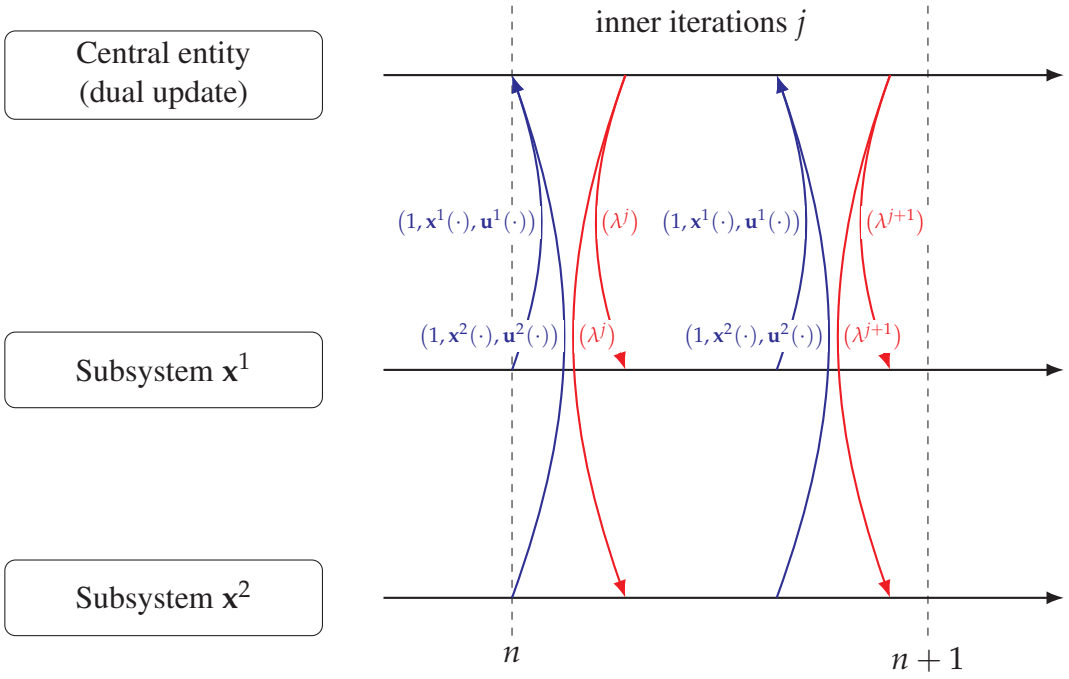


Figure 6.6.: Communication schedule for dual decomposition

Within Figure 6.6, blue arrows represent so-called *primal information*, that is, neighboring data in the form of predicted state and control trajectories exchanged by the subsystems. Red arrows represent the newly introduced *dual information*, namely the Lagrange multipliers that encode global consistency and constraint satisfaction. In contrast to the sequential and hierarchical schemes, all subsystems communicate symmetrically with a central entity, and no direct subsystem-to-subsystem ordering or dependency structure is required.

Beyond the introduction of a central entity, the communication pattern also differs fundamentally in its temporal structure. While sequential and hierarchical approaches typically require exactly one exchange of neighboring data per closed-loop time step, the parallel approach relies on multiple communication rounds within each closed-loop step. In particular, predicted trajectories are repeatedly sent from the subsystems to the server, and updated Lagrange multipliers are broadcast back to the subsystems until a termination criterion is met. Hence, coordination is achieved through iteration rather than ordering.

To formalize this idea, we reinterpret constraints and dynamics as *penalties* in the cost functional. The key ingredient enabling this reformulation is the following operator.

Definition 6.27 (Cost operator).
Consider a control problem (5.3) with n constraints given by state constraints, control constraints and dynamics. Then we call an operator $\Gamma : \mathcal{X} \times \mathcal{U} \to \mathbb{R}^{n_x+n}$ a cost operator if it satisfies

$$\Gamma(\mathbf{x}, \mathbf{u}) = 0 \tag{6.12}$$

iff the conditions

$$\mathbf{x}(k+1) = f(\mathbf{x}(k), \mathbf{u}(k), k), \quad \mathbf{x}(0) = \mathbf{x}_0 \tag{6.13}$$
$$\mathbf{x}(k) \in \mathbb{X}, \quad k \in [0, N] \tag{6.14}$$

hold.

The cost operator provides a unified representation of all constraints of the original optimal control problem. Instead of enforcing these constraints explicitly, the operator measures their violation. In particular, $\Gamma(\mathbf{x}, \mathbf{u}) = 0$ characterizes exactly the set of admissible state–control trajectories.

Task 6.28
Reconsider the system from Task 6.2. Define a cost operator Γ *that encodes these coupling constraints.*

Solution to Task 6.28: A suitable cost operator penalizing violation of the coupling dynamics is

$$\Gamma(\mathbf{x}, \mathbf{u}) = \mathbf{x}^1(k+1) - \mathbf{x}^1(k) - \mathbf{x}^2(k) - \mathbf{u}^2(k)/2.$$

The operator satisfies $\Gamma(\mathbf{x}, \mathbf{u}) = 0$ if and only if the coupling dynamics are fulfilled.

Using this operator, we can apply the Lagrangian principle and obtain the augmented cost functional

$$L(\mathbf{x}_0, \mathbf{u}, \lambda) := J_N(\mathbf{x}_0, \mathbf{u}) + \lambda^\top \cdot \Gamma(\mathbf{x}_0, \mathbf{u}) \tag{6.15}$$

where λ denotes the vector of Lagrange multipliers associated with the constraints. The dual function $g(\lambda) = argmin_{\mathbf{u}\in\mathcal{U}} L(\mathbf{x}_0, \mathbf{u}, \lambda)$ corresponds to the dual of the original constrained optimal control problem (5.3).

The decisive advantage of this reformulation is that the Lagrangian (6.15) decomposes additively with respect to subsystems. As a result, for fixed Lagrange multipliers, each subsystem can solve its own optimal control problem independently. Global consistency and constraint satisfaction are enforced only through updates of the multipliers, which motivates the parallel primal–dual algorithm introduced next.

Task 6.29
Reconsider the system from Task 6.2. Write down the local Lagrangian cost functionals for subsystems 1 and 2 and explain why both subsystems can be optimized in parallel for a fixed Lagrange multiplier.

Solution to Task 6.28: Introducing a Lagrange multiplier $\lambda(j)$, the Lagrangian splits additively into

$$L^1(\mathbf{x}_0, \mathbf{u}, \lambda) = \sum_{k=0}^{N-1} \Big(\|\mathbf{x}^1(k)\|^2 + \lambda(j)\, \mathbf{x}^1(k+1) \Big),$$

and

$$L^2(\mathbf{x}_0, \mathbf{u}, \lambda) = \sum_{k=0}^{N-1} \Big(\|\mathbf{x}^2(k)\|^2 + \|\mathbf{u}^2(k)\|^2 - \lambda(j) \left(\mathbf{x}^2(k) + \mathbf{u}^2(k)/2 \right) \Big).$$

Thus, the multiplier couples the subsystems only through the cost. Hence, for a fixed multiplier sequence $\lambda(\cdot)$, the two Lagrangian subproblems are independent. Hence, subsystems 1 and 2 can be optimized fully in parallel, with consistency enforced only through updates of the Lagrange multiplier by a central entity.

Now, we can additively distribute the Lagrangian problem (6.15), which leads to the following algorithm.

Algorithm 6.30 (Dual decomposition)
For each closed loop time index $n = 0, 1, 2 \ldots$:
At subsystem:

(1) For each subsystem $p \in \mathcal{P}$

Obtain the state $\mathbf{x}^p(n) \in \mathbb{X}^p$ of the system and set $\lambda^0 = 0$ and $j = 0$.

(2) For each subsystem $p \in \mathcal{P}$ do

(2a) Collect data $(0, n, \lambda^j)$.

(2b) Compute a minimizer for the Lagrangian (6.15) and denote the solution by $\mathbf{u}_n^p(\cdot)$.

(2c) Send data $\left(p, n, \mathbf{x}_0^p, \mathbf{u}^{p,j+1}(\cdot)\right)$ to central entity.

At central entity:

(2a) Collect neighboring data i^p for all subsystems.

(2b) Update Lagrange multiplier

$$\lambda^{j+1} := \lambda^j + \rho^j \cdot \Gamma(\mathbf{x}_0, \mathbf{u}^j, \lambda^j)$$

(2c) Send Lagrange multiplier $(0, n, \lambda^{j+1})$ to all subsystems $p \in \mathcal{P}$. Set $j := j + 1$ and go to (2) unless a termination criterion is satisfied.

At subsystem:

(3) For each subsystem $p \in \mathcal{P}$

a) Define the MPC feedback $\mu_N^p(\mathbf{x}^p(n)) := \mathbf{u}^{p,\star}(0)$.

The principal advantage of Algorithm 6.30 lies in its remarkable generality. By reformulating constraints and coupling relations via the Lagrangian, the algorithm can be applied to essentially any finite-horizon optimal control problem, independent of the structure of the system dynamics or the form of the constraints. In particular, the decomposition of the problem into subproblems is no longer tied to a specific partitioning of the state or control variables. Instead, the split may be chosen freely, for example according to computational resources, communication architecture, or organizational boundaries, rather than along dynamical or constraint-based interfaces.

This flexibility distinguishes the parallel approach fundamentally from the sequential and hierarchical schemes discussed earlier. In those approaches, the admissible decompositions are dictated by the dependency structure of the dynamics and constraints, and feasibility must be enforced explicitly at every closed-loop step. In contrast, dual decomposition shifts the burden of coordination entirely to the cost functional: coupling is not eliminated but rather handled implicitly through the Lagrange multipliers.

The price to be paid for this generality is twofold. First, the algorithm requires a central entity that collects primal information from all subsystems, updates the Lagrange multipliers, and broadcasts them back to the subsystems. While this entity may be lightweight from a computational perspective, its existence breaks full decentralization and introduces a potential single point of failure. Moreover, the communication pattern is inherently iterative: within each closed-loop

time step, multiple rounds of message exchange between subsystems and the central entity are typically required.

Second, feasibility and consistency are no longer guaranteed at intermediate iterations. The iteration index j in Algorithm 6.30 emphasizes that convergence toward constraint satisfaction and optimality is achieved only asymptotically. During the intermediate primal–dual iterations, the computed trajectories may violate constraints or coupling relations. Consequently, practical implementations must carefully balance the number of iterations against real-time requirements, often terminating the inner loop early and accepting a suboptimal or partially infeasible solution.

The update of the Lagrange multipliers plays a crucial role in this trade-off. The factor ρ acts as a step size or line-search parameter in the dual ascent method. If chosen too large, the multiplier updates may lead to oscillations or divergence; if chosen too small, convergence becomes prohibitively slow. Adaptive or diminishing step-size strategies are therefore commonly employed to stabilize the iteration and accelerate convergence. The selection of ρ thus directly influences both the numerical performance of the algorithm and its suitability for real-time control applications.

In summary, dual decomposition provides maximal flexibility and parallelism at the expense of guaranteed feasibility, strict decentralization, and predictable convergence speed. It is particularly attractive for large-scale systems with complex coupling structures, where structural decompositions are difficult to identify, but where sufficient communication bandwidth and computational time are available to accommodate iterative coordination.

CHAPTER 7

CONTROL BARRIER FUNCTIONS

Generated with chatgpt.com

> To enjoy freedom we have to control ourselves.
>
> Virginia Woolf

In many control systems, admissibility conditions must be rigorously maintained—particularly in safety-critical applications where specific state constraints must never be violated during operation. These constraints can pertain to states, inputs, or outputs. Representative examples include maintaining a safe following distance in adaptive cruise control systems, ensuring that robotic manipulators remain within their joint limits, or preventing overflow in chemical process tanks.
In previous chapters, such constraints were incorporated into the optimal control framework. While this inclusion is analytically sound, the numerical methods used to solve these problems do not inherently guarantee constraint satisfaction in closed-loop execution, especially in the presence of modeling inaccuracies, disturbances, or numerical solver tolerances.
To address this gap, we introduce the concept of *forward invariance* in Section 7.1, and its systematic enforcement via *Control Barrier Functions (CBFs)* in Section 7.2. CBFs provide a rigorous framework to guarantee that system trajectories remain within predefined *safe sets* over time, by encoding safety requirements as inequality constraints on the control input.
Conceptually related to Lyapunov functions, CBFs act as safety certificates: while Lyapunov functions ensure convergence, barrier functions prevent unsafe behavior by repelling the state from constraint boundaries. Integrating these conditions into the control law provides formal safety guarantees.
Although the structure of CBF resembles MPC in many ways — particularly in the use of constraints – we initially present these functions as a complementary paradigm focused on real-time constraint enforcement. This approach does not rely on the solution of a full trajectory optimization problem, but instead enforces safety at each time step using local information. Later, in Section 7.3, we combine both approaches to balance safety and performance.
Throughout this chapter, we focus on discrete-time nonlinear control-affine systems of the form

$$\mathbf{x}(k+1) = f_0(\mathbf{x}(k)) + f(\mathbf{x}(k))\mathbf{u}(k), \tag{7.1}$$

where $\mathbf{x}(k) \in \mathcal{X}$ is the state and $\mathbf{u}(k) \in \mathcal{U}$ the control input. The control-affine form is general enough to capture many relevant mechanical and robotic systems, while enabling tractable mathematical analysis using Lie derivative techniques introduced in the next sections.

7.1. Forward invariance

As sketched before, we want to generate a setting for an MPC such that constraint satisfaction is always guaranteed. The latter is requires since MPC is fundamentally horizon-limited: constraint satisfaction is only guaranteed along the planned trajectory, not necessarily for all future time. This motivates a control concept that ensures constraint satisfaction independently of horizon length and reference changes.

The notion of *forward invariance* provides exactly this property.

Definition 7.1 (Forward invariance).
A set $\mathcal{H} \subset \mathcal{X}$ is called *forward invariant* for the system (7.1) under a feedback law $\mathbf{u} = \mu(\mathbf{x})$ if

$$\forall \mathbf{x}_0 \in \mathcal{H} : \ \mathbf{x}(k) \in \mathcal{H} \ \forall k \geq 0 \tag{7.2}$$

holds true.

Breaking down the latter definition, a forward invariant set $\mathcal{H}$ states that if a system is evolving from within $\mathcal{H}$, then it will never leave this set given the applied control law. Unlike the Definition 5.13 of MPC feasibility, which is evaluated over a finite horizon, forward invariance is a *time-unbounded property* similar to our Theorem 5.14 on recursive feasibility.

Remark 7.2
Forward invariance differs fundamentally from stability. While stability concerns convergence to a specific set or equilibrium, invariance concerns the impossibility of leaving a prescribed set.

Task 7.3 (Adaptive cruise control)
Consider the system

$$\begin{aligned} \mathbf{x}_1(k+1) &= \mathbf{x}_1(k) + (v - x_2(k)) - \frac{1}{2}\mathbf{u}(k), \\ \mathbf{x}_2(k+1) &= \mathbf{x}_2(k) + \mathbf{u}(k) \end{aligned}$$

resembling a simple adaptive cruise control (ACC) scenario for a vehicle following a lead vehicle. Here, $\mathbf{x}_1(k)$ denotes the gap between the follower and the lead car at time k, and $\mathbf{x}_2(k)$ represents the velocity of the follower car. Assume the lead vehicle travels at a constant velocity v. Then we obtainwith input acceleration $\mathbf{u}(k)$. Define a safety constraint as a forward invariant set.

Solution to Task 7.3: A possible safety constraint is that the gap $\mathbf{x}_1 k$ should never become negative (or zero), to avoid collision. Thus we define $\mathcal{H} = \mathbb{R}^+ \times \mathbb{R}$.

As forward invariance expresses the requirement that we want to enforce, we are now looking for a constructive way to obtain a respective set. To this end, we utilize the concept of *safe sets*.

Definition 7.4 (Safe set).
Let $h : \mathcal{X} \to \mathbb{R}$ be a continuously differentiable function. Then we call a

$$\mathcal{H} := \{\mathbf{x} \in \mathcal{X} \mid h(\mathbf{x}) \geq 0\}. \tag{7.3}$$

safe set and denote its boundary and interior by

$$\partial\mathcal{H} := \{\mathbf{x} \in \mathcal{X} \mid h(\mathbf{x}) = 0\}, \qquad int(\mathcal{H}) := \{\mathbf{x} \in \mathcal{X} \mid h(\mathbf{x}) > 0\} \tag{7.4}$$

respectively.

Now, the idea is to derive the function $h(\cdot)$ from the dynamics (7.1). Forward invariance holds true particularly if the evolution of $h(\mathbf{x})$ is restricted such that it cannot become negative. Here $f_0(\mathbf{x}) + f(\mathbf{x})\mathbf{u}$ denotes the next state $\mathbf{x}(k+1)$ given current state $\mathbf{x} = \mathbf{x}(k)$ and input $\mathbf{u} = \mathbf{u}(k)$. Intuitively, this means for any safe state, there is at least one control option that keeps $h(\mathbf{x}(k+1))$ non-negative at the next step, thus preventing the state from leaving $\mathcal{H}$.

Task 7.5 (Adaptive cruise control)
Given the example from Task 7.3, derive a function $h : \mathcal{X} \to \mathbb{R}$ defining the safe set.

Solution to Task 7.5: We can choose $h(\mathbf{x}) = \mathbf{x}_1$ to describe the forward invariant set.

The key idea is to enforce a one-step condition from $\mathbf{x}(k)$ to $\mathbf{x}(k+1)$ that is sufficient to guarantee forward invariance for all future time. In discrete time, the latter is achieved by enforcing an inequality of the form

$$h(\mathbf{x}(k+1)) - h(\mathbf{x}(k)) \geq -\alpha(h(\mathbf{x}(k))) \tag{7.5}$$

via a comparison function $\alpha \in \mathcal{K}_\infty$. The inclusion of α introduces a safety margin that prevents $h(\mathbf{x})$ from dropping to zero in one step when starting positive. Essentially, $\alpha(h(\mathbf{x}(k))) < h(\mathbf{x}(k)))$ for $h(\mathbf{x}(k)) > 0$, so the next state's value $h(\mathbf{x}(k+1))$ cannot decrease by the full amount of $h(\mathbf{x}(k))$ and therefore no jump from a positive $h(\mathbf{x}(k))$ to a negative $h(\mathbf{x}(k+1))$ is possible. In other words, α limits how fast the system can approach the boundary $\partial\mathcal{H}$, providing a buffer against uncertainties from digitalization, numerics or modeling. Figuratively, including an α function is analogous to requiring a car to slow down as it approaches a stop sign.

7.2. Control barrier functions

We are particularly interested in ensuring that the system state $\mathbf{x}(\cdot)$ does not cross the zero of our function $h(\cdot)$. We can track the latter using the gradient $\nabla h(\mathbf{x})$, but also need to cover for the dynamics of the system. This can be done using so called *Lie derivatives*.

Definition 7.6 (Lie derivative).
Consider a control affine system (7.1) and let $h : \mathcal{X} \to \mathbb{R}$ be continuously differentiable. The *Lie derivative* of h along f_0 is defined as

$$L_{f_0} h(\mathbf{x}) := \nabla h(\mathbf{x})^\top f_0(\mathbf{x}) \in \mathbb{R}. \tag{7.6}$$

Moreover, we call

$$L_f h(\mathbf{x}) := \nabla h(\mathbf{x})^\top f(\mathbf{x}) \in \mathcal{U}^\top \tag{7.7}$$

Lie derivative of $h(\cdot)$ along f.

Now, we can utilize these Lie derivatives to quantify how this margin evolves since

- $L_{f_0} h(\mathbf{x})$ describes the change of $h(\cdot)$ due to the autonomous system behavior, i.e. inertia, damping, gravity, thermal dynamics, or other unforced effects, and
- $L_f h(\mathbf{x})\mathbf{u}$ describes the change of $h(\cdot)$ induced by the control input, i.e. how actuators can increase or decrease the constraint margin.

This allows us to derive so called *control barrier functions*.

Definition 7.7 (Control barrier function).
Consider a control affine system (7.1) and let $\mathcal{H}$ be a safe set as defined in (7.3). Then we call a continuously differentiable function $h : \mathcal{X} \to \mathbb{R}$ a *control barrier function* for system (7.1) on $\mathcal{H}$ if there exists $\alpha \in \mathcal{K}_\infty$ such that

$$\sup_{\mathbf{u} \in \mathcal{U}} \left[L_{f_0} h(\mathbf{x}) + L_f h(\mathbf{x})\mathbf{u} + \alpha(h(\mathbf{x})) \right] \geq 0 \tag{7.8}$$

holds for all $\mathbf{x} \in \mathcal{H}$. In the autonomous case $f(\mathbf{x}) \equiv 0$ the latter simplifies to

$$L_{f_0} h(\mathbf{x}) \geq 0 \tag{7.9}$$

for all $\mathbf{x} \in \mathcal{H}$ and $h(\cdot)$ is called a *barrier function*.

Here, we like to point out that due to continuity the supremum $\sup_{\mathbf{u}\in\mathcal{U}}$ is equivalent to the existence of such a control $\mathbf{u}$.
We like to point out that it is also possible to introduce control barrier functions without the Lie derivatives. We include this to highlight the relation between CBFs and Lyapunov functions.

Definition 7.8 (Control Barrier Function).
Consider a control affine system (7.1) and a continuously differentiable function $h : \mathcal{X} \to \mathbb{R}$ defining the safe set $\mathcal{H}$ according to (7.3). Then we call a continuously differentiable function $B : \mathcal{H} \to \mathbb{R}$ a *control barrier function* for system (7.1) on $\mathcal{H}$ if there exist class $\mathcal{K}_\infty$ functions α_1, α_2, and α_3 such that the conditions

$$\frac{1}{\alpha_1(h(\mathbf{x}))} \leq B(\mathbf{x}) \leq \frac{1}{\alpha_2(h(\mathbf{x}))} \tag{7.10}$$

$$\exists\, \mathbf{u} \in \mathcal{U} :\ B(f(\mathbf{x})\mathbf{u}) - B(\mathbf{x}) \leq \alpha_3(h(\mathbf{x})) \tag{7.11}$$

hold for all $\mathbf{x} \in \mathcal{H}$.

Task 7.9 (Adaptive cruise control)
Given the example from Task 7.3, derive a control barrier function.

Solution to Task 7.9: The one-step change in h is

$$\begin{aligned} h(\mathbf{x}(k+1)) - h(\mathbf{x}(k)) &= \mathbf{x}_1(k+1) - \mathbf{x}_1(k) \\ &= (v - \mathbf{x}_2(k)) - \frac{1}{2}\mathbf{u}(k). \end{aligned}$$

To enforce forward invariance, we impose control barrier function condition

$$(v - \mathbf{x}_2(k)) - \frac{1}{2}\mathbf{u}(k) \geq -\alpha(h(\mathbf{x}(k))),$$

with $\alpha \in \mathcal{K}_\infty$. Rearranging the latter yields an upper bound on the admissible control via

$$\mathbf{u}(k) \leq 2\left[v - \mathbf{x}_2(k) + \alpha(\mathbf{x}_1(k))\right].$$

This constraint ensures that the control input $\mathbf{u}(k)$ prevents the barrier function from decreasing too rapidly, thereby enforcing safety of the inter-vehicle distance over time. Then, a

natural choice for the barrier function is

$$B(\mathbf{x}) := \frac{1}{\mathbf{x}_1}.$$

For this choice, we can identify $\alpha_1(s) = \alpha_2(s) = s$ revealing

$$\frac{1}{\alpha_1(h(\mathbf{x}))} = \frac{1}{\mathbf{x}_1} = B(\mathbf{x}) = \frac{1}{\alpha_2(h(\mathbf{x}))}$$

Moreover, we can choose $\alpha_3(s) = \gamma s$ with some constant $\gamma > 0$ and obtain

$$\frac{1}{\mathbf{x}_1 + v - \mathbf{x}_2 - \frac{1}{2}\mathbf{u}} - \frac{1}{\mathbf{x}_1} \leq \gamma \mathbf{x}_1.$$

Here, condition (7.10) ensures that $B(\mathbf{x})$ grows unbounded as $\mathbf{x}$ approaches the boundary of the safe set $\mathcal{H}$. In particular, since $\alpha_1(0) = 0$ and $\alpha_2(0) = 0$, the inequalities in (7.10) imply that if $h(\mathbf{x}) \to 0^+$ (state $\mathbf{x}$ gets arbitrarily close to $\partial\mathcal{H}$ from the safe side), then $B(\mathbf{x}) \to \infty$. In other words, $B(\mathbf{x})$ acts as a *barrier*.
Condition (7.11) is a *inequality constraint* on the evolution of $B(\mathbf{x})$ under the system dynamics. It requires that for each state $\mathbf{x} \in \mathcal{H}$, there exists at least one control input $\mathbf{u}$ that makes the increment of $B(\mathbf{x}(k))$ from $\mathbf{x}(k)$ to the next state $\mathbf{x}(k+1)$ bounded by $\alpha_3(h(\mathbf{x}(k)))$. Thus, (7.11) ensures that when the state is close to the boundary (small $h(\mathbf{x})$), the controller can be chosen to prevent $B(\mathbf{x})$ from increasing too much (since the right-hand side $\alpha_3(h(\mathbf{x}))$ will be small). This condition effectively limits how much closer to the boundary $\partial\mathcal{H}$ the state can move in one time-step, by restricting the growth of the barrier function B. The existence of such a control input $\mathbf{u}$ for every $\mathbf{x} \in \mathcal{H}$ means we can always make a control choice that satisfies this safety constraint on B.
In the following, we want to make use of the control barrier functions to strengthen our MPC.

7.3. Integration of CBF in MPC

The key idea is to show that that as long as there always exists a control input satisfying this constraint at each state $\mathbf{x}(k) \in \mathcal{H}$, we know that the system remains within a predefined safe set over all time. For our MPC framework, this would allow us to go from feasibility to recursive feasibility also in the distributed context.
As a first step, we utilize the construction of the control barrier function $h(\cdot)$ to re-obtain forward invariance of the set $\mathcal{H}$.

Theorem 7.10 (Forward invariance via CBF).
Suppose a control affine system (7.1) *to be given with control barrier function* $h : \mathcal{X} \to \mathbb{R}$. *If the control input* $\mathbf{u}(\cdot)$ *satisfies*

$$L_{f_0}h(\mathbf{x}(k)) + L_f h(\mathbf{x}(k))\mathbf{u}(k) \geq 0 \qquad \forall k \geq 0, \tag{7.12}$$

then the set $\mathcal{H} \subset \mathcal{X}$ *is forward invariant.*

Proof. If $\mathbf{x}(k)$ reaches the boundary $\partial\mathcal{H}$, then (7.12) implies that $h(\mathbf{x}(k+1)) - h(\mathbf{x}(k)) \geq 0$. Hence, $h(\mathbf{x}(k))$ cannot decrease below zero and trajectories cannot leave $\mathcal{H}$. □

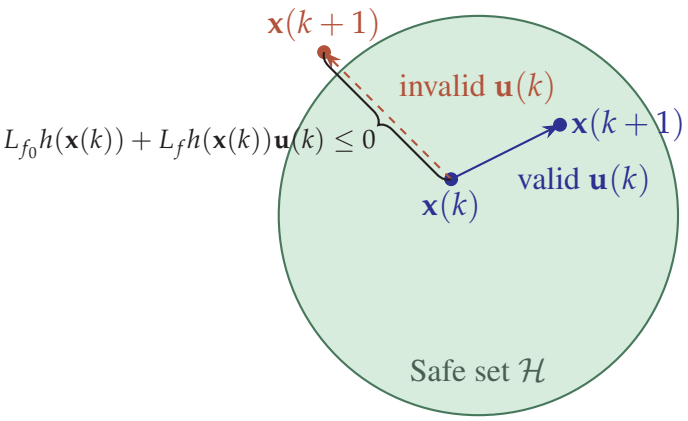


Figure 7.1.: Illustration of forward invariance via CBF

Similarly, Definition 7.8 can be imposed to derive forward invariance.

Theorem 7.11 (Safety Guarantee via CBF).
Suppose $B : \mathcal{X} \to \mathbb{R}$ *is a control barrier function for a control affine system* (7.1) *on the safe set* $\mathcal{H}$ *with corresponding class* $\mathcal{K}_\infty$ *functions* $\alpha_1, \alpha_2, \alpha_3$ *as in Definition 7.8. If* $\mathbf{x}_0 \in int(\mathcal{H})$, *then there exists a sequence of control inputs* $\mathbf{u}(k)$ *such that the state remains in* $\mathcal{H}$ *for all* $k \in \mathbb{N}$ *and* $\mathcal{H}$ *is forward invariant.*

Proof. The proof proceeds by induction on the time step k. At time $k = 0$, we have $\mathbf{x}_0 \in \mathcal{H}$ by assumption. For the induction assume $\mathbf{x}(k) \in \mathcal{H}$. Hence, by (7.11) there exists at least one control $\mathbf{u}(k) \in \mathcal{U}$ satisfying (7.11). Since $h(\mathbf{x}(k)) \geq 0$ we have $\alpha_3(h(\mathbf{x}(k))$ is a finite, non-negative number and hence $B(\mathbf{x}(k+1)) \leq B(\mathbf{x}(k)) + \alpha_3(h(\mathbf{x}(k)))$. Since $B(\mathbf{x}(k))$ is finite, $B(\mathbf{x}(k+1))$ must also be finite.
Now suppose $\mathbf{x}(k+1) \notin \mathcal{H}$, i.e. $h(\mathbf{x}(k+1)) < 0$. Since $B(\mathbf{x})$ would be required to be at least $\frac{1}{\alpha_1(h(\mathbf{x}))}$, it follows that $B(\mathbf{x})$ would tend to $+\infty$ for $\mathbf{x}$ approaching the barrier from within $\mathcal{H}$. As

we already found $B(\mathbf{x}(k+1)$ to be bounded, this contradicts the assumption and the assertion follows. □

Task 7.12 (Adaptive cruise control)
Given the example from the previous Task 7.9 show the safety implications of the CBF.

Solution to Task 7.12: We can solve

$$\frac{1}{\mathbf{x}_1 + v - \mathbf{x}_2 - \frac{1}{2}\mathbf{u}} - \frac{1}{\mathbf{x}_1} \leq \gamma \mathbf{x}_1.$$

for $\mathbf{u}$ to obtain

$$\mathbf{u} \leq \frac{2[\gamma \mathbf{x}_1^3 + \gamma \mathbf{x}_1^2(v - \mathbf{x}_2) - (\mathbf{x}_2 - v)]}{1 + \gamma \mathbf{x}_1^2}$$

From this expression, we observe the following two extreme cases:

- If $\mathbf{x}_1$ is very small (the follower is dangerously close), the numerator $\gamma \mathbf{x}_1^3 + \cdots$ will be dominated by the term $-(\mathbf{x}_2 - v)$ since $\mathbf{x}_1^3$ and $\mathbf{x}_1^2$ terms become negligible. Thus, the control approximately reduces to $\mathbf{u} \lesssim -2(\mathbf{x}_2 - v)$, which is negative if $\mathbf{x}_2 > v$. In other words, if the follower is too close and moving faster than the leader, the condition essentially requires a negative acceleration (braking) proportional to the closing speed. This ensures the follower slows down.
- If $\mathbf{x}_1$ is large (the follower is far behind), the terms with $\mathbf{x}_1$ will dominate. Hence, $\mathbf{u}$ can be positive and large without violating the inequality. In fact, as $\mathbf{x}_1 \to \infty$, the bound tends to $\mathbf{u} \lesssim 2\gamma^{-1}\mathbf{x}_1$, which grows with $\mathbf{x}_1$. Thus, far from the leader, the follower is free to accelerate more aggressively (since safety is not at risk).

Now we are in a position to formally integrate control barrier functions into the digital constrained optimal control framework of Definition 5.8.

Definition 7.13 (MPC problem with Control Barrier Function).
Suppose a control affine system (7.1) and a control barrier function $h : \mathcal{X} \to \mathbb{R}$ to be given with corresponding safe set $\mathcal{H} = \{\mathbf{x} \in \mathcal{X} \mid h(\mathbf{x}) \geq 0\}$. Then we call the problem

$$\min \; J(\mathbf{x}_0, \mathbf{u}) = \sum_{k=0}^{N-1} \ell(\mathbf{x}(k, \mathbf{x}_0, \mathbf{u}), \mathbf{u}(k)) \quad \text{over all } \mathbf{u} \in \mathbb{U}^N \tag{7.13}$$

subject to:

$$\begin{aligned}
&\mathbf{x}(k+1) = f_0(\mathbf{x}(k)) + f(\mathbf{x}(k))\mathbf{u}(k), \quad \mathbf{x}(0) = \mathbf{x}_0 \\
&\mathbf{x}(k) \in \mathbb{X}, \quad \mathbf{u}(k) \in \mathcal{U}, \quad \forall k \in [0, N-1] \\
&L_{f_0} h(\mathbf{x}(k)) + L_f h(\mathbf{x}(k))\mathbf{u}(k) \geq 0 \quad \forall k \in [0, N-1].
\end{aligned}$$

a *digital finite constrained optimal control problem subject to control barrier functions* or *MPC/CBF open loop problem.*

The final constraint in equation 7.13 stipulates the control barrier function condition at each prediction step. Given the condition's affine nature in the control input $\mathbf{u}(k)$, its imposition as a linear inequality in optimization-based controllers is a viable approach.
This local condition guarantees that each predicted state is in accordance with the forward invariance condition of the safe set $\mathcal{H}$. That is to say, provided the CBF constraint is satisfied at each stage of the prediction horizon, all subsequent states will remain within the safety region. Therefore, the safety constraint stays recursively feasible under prediction during optimization.
In the literature, this formulation is often referred to as

- *CBF-constrained MPC*, or
- *safety-augmented MPC*, or
- *constrained CBF-QP (when quadratic cost functions are used).*

The problem structure unifies optimality and safety in a principled fashion, particularly suited to real-time control of safety-critical systems such as autonomous vehicles, industrial manipulators, and collaborative robots.
Control barrier functions are inherently state-local constraints, meaning they are imposed at individual time steps based on the instantaneous system state and model. This time-step-wise nature finds congruence with the structure of MPC, which computes a control sequence based on the system's evolution over a finite prediction horizon. The CBF constraint functions as a stage constraint, analogous to input or state bounds, and can be enforced at each node of the prediction.
Furthermore, in contrast to formulations of global reachability or safety, the CBF condition necessitates only forward invariance over a single step, a property that renders it both computationally tractable and numerically efficient for incorporation into real-time MPC schemes. This local structure renders CBFs particularly attractive and compatible within the MPC framework.

BIBLIOGRAPHY

[1] ANDERSON, B.D.O. ; MOORE, J.B.: *Optimal control: linear quadratic methods*. Courier Corporation, 2007

[2] GRÜNE, L. ; PANNEK, J.: *Nonlinear Model Predictive Control: Theory and Algorithms*. 2. Springer, 2017

[3] HINRICHSEN, D. ; PRITCHARD, A.J.: *Mathematical system theory I: Modeling, state space analysis and robustness*. Springer, 2010

[4] ISIDORI, A.: *Nonlinear Control Systems*. 3rd edition. Springer, 1995

[5] KALMAN, R.E. u. a.: Contributions to the theory of optimal control. In: *Bol. soc. mat. mexicana* 5 (1960), No. 2, pp. 102–119

[6] LJUNG, L.: *System Identification: Theory for the User*. Pearson Education, 1998

[7] RAWLINGS, J.B. ; MAYNE, D.Q. ; DIEHL, M.: *Model predictive control: theory, computation, and design*. Second. Nob Hill Publishing Madison, WI, 2017

[8] RICHARDS, A. ; HOW, J.: A Decentralized Algorithm for Robust Constrained Model Predictive Control. In: *Proceedings of the American Control Conference*, 2004, pp. 4241–4266

[9] SKOGESTAD, S. ; POSTLETHWAITE, I.: *Multivariable feedback control: analysis and design*. John Wiley & Sons, 2005

[10] SONTAG, E.D.: *Mathematical Control Theory: Deterministic Finite Dimensional Systems*. Springer, 1998. – 531 S.

Jürgen Pannek
Institute for Intermodal Transport and Logistic Systems
Hermann-Blenck-Str. 42
38519 Braunschweig

During winter term 2025/26 I give the lecture to the module *Control Engineering 3 (Regelungstechnik 3)* at the Technical University of Braunschweig. To structure the lecture and support my students in their learning process, I prepared these lecture notes. The aim of the lecture notes is to provide participating students with knowledge of advanced control methods, which extend the range of control engineering. The students shall be enabled to list modern control methods and recall their properties. Moreover, students shall be able to apply these methods in simulation experiments and assess respective results.