Model-Predictive Control and Machine Learning

Summary
Fabian Damken
March 4, 2022

TECHNISCHE UNIVERSITÄT DARMSTADT

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

This summary covers *model predictive control* (MPC) combined with *machine learning* (ML). In MPC, a model of a dynamical system is used to find inputs that steer the system optimally (in some sense). ML, on the other hand, can be used to build such models from data. This document focuses primarily on the MPC part, featuring nominal, robust, and stochastic MPC. Subsequently, connections to and applications of ML are drawn as these fields get more and more interconnected. The key topics are understanding MPC basics, identifying benefits and drawbacks of MPC, understanding the role of ML in control, understanding the basic concepts of ML-supported MPC as well as its benefits and drawbacks.

1.1 What is Model Predictive Control?

In general, *control* is concerned with influencing a dynamical system such that it exhibits a wanted behavior. Usually, this involves incorporating feedback (e.g., the actual state of the system) into the control law (feedback control). In *optimal* control, the inputs shall be optimal in some sense (e.g., minimal energy consumption, avoidance of states, ...).

In *model* predictive control, a model of the system is used to predict the influence of inputs. This has the advantage of the controller actually understanding what it is doing, potentially increasing the performance and yielding a structured design process of the controller. On the other hand, MPC needs a model that can be hard to obtain[1]. Also, it is computationally expensive and its performance is highly influenced by the model quality and accuracy.

An MPC controller performs *prediction* using the model to assess the influence of certain actions. This can be used for finding the optimal inputs by minimizing a cost function on the predicted states. This optimal input can then be applied to the system. Hence, MPC is *optimization-based* control: the optimal input is retrieved by minimizing a cost function with the dynamical system as a constraint on the states. This allows to incorporate additional constraints (e.g., min/max actions or unsafe states) directly into the optimizer. To incorporate feedback from the actual system, this optimization problem is solved repeatedly during execution (see Figure 1.1).

Model predictive control is covered in detail in chapter 3, 4, and 5.

[1] A common way to obtain a model aside from deriving it using first principles are gathering data of the system, fixing a model structure, and fitting the parameters.

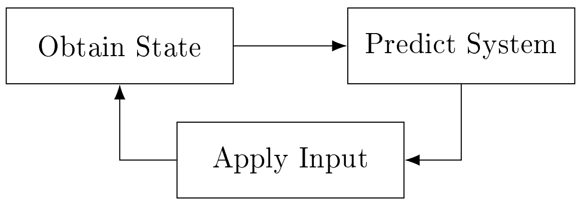

Figure 1.1: Illustration of the model predictive control cycle.

1.2 What is Machine Learning?

While there is no unified definition of machine learning, it is often used to describe systems that use a bunch of data to find relations/patterns/connections/…in them and to extract general rules. Typical methods are neural networks, Gaussian processes, support vector machines, and many more. In control, the applications of machine learning are twofold: first, it can be used to find a model and use the model in a model-based controller (supervised learning and regression); second, this step can also be skipped and ML can be applied directly as the controller (usually covered in reinforcement learning).

Machine learning for MPC is covered in detail in chapter 6.

2 Preliminaries

This chapter covers some preliminaries required to understand the upcoming chapters.

2.1 System Theory

System theory describes the study of all kinds of dynamical systems, their stability, controllability, and various other properties. This section introduces the most important concepts like the different kinds of representations and stability. In MPC, system theory is both used to study the behavior of the actual system as well as the model.

2.1.1 Types of Dynamical Systems

On a high level, dynamical systems separate into two classes: time-continuous and time-discrete. By Shannon's sampling theorem, it is always possible to turn a continuous model into a discrete one with an appropriate sample rate.

Time-Continuous

A time-continuous nonlinear system is represented by an (ordinary) initial value problem

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}; t) \qquad \boldsymbol{y} = \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u}; t) \qquad \boldsymbol{x}(t_0) = \boldsymbol{x}_0$$

where $\boldsymbol{x}$ are the states, $\boldsymbol{u}$ is the control input, $\boldsymbol{y}$ are the observations, and t is the time. If $\boldsymbol{f}$ or $\boldsymbol{h}$ is t-dependent, the system is called *time-variant*, otherwise it is called *time-invariant*. If $\boldsymbol{f}$ and $\boldsymbol{h}$ are linear functions $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}; t) = \mathbf{A}(t)\boldsymbol{x} + \mathbf{B}(t)\boldsymbol{u}$ and $\boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u}; t) = \mathbf{C}(t)\boldsymbol{x} + \mathbf{D}(t)\boldsymbol{u}$, the system is called *linear* with state dynamics matrix $\mathbf{A}$, control matrix $\mathbf{B}$, output/observation matrix $\mathbf{C}$, and control influence matrix $\mathbf{D}$.

If the system matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{D}$ are time-independent, a major advantage of linear systems is that they exhibit an analytical solution:

$$\boldsymbol{x}(t) = \exp\{\mathbf{A}(t - t_0)\}\boldsymbol{x}_0 + \int_{t_0}^{t} \exp\{\mathbf{A}(t - \tau)\}\mathbf{B}\boldsymbol{u}(\tau)\,\mathrm{d}\tau\,.$$

Note that here, $\mathbf{A}(t - t_0)$ does *not* correspond to an invocation and time-dependence, is is simply a multiplication with $t - t_0$. However, the vast majority of dynamical systems are not linear! Hence, these models are often approximated locally (around an operation point $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}})$) using the Taylor series of $\boldsymbol{f}$:

$$\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}; t) \approx \boldsymbol{f}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}; t) + \left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\bigg|_{\boldsymbol{x}=\bar{\boldsymbol{x}}} \right)(\boldsymbol{x} - \bar{\boldsymbol{x}}) + \left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\bigg|_{\boldsymbol{u}=\bar{\boldsymbol{u}}} \right)(\boldsymbol{u} - \bar{\boldsymbol{u}})$$

By cutting off the higher-order terms, this yields a linear approximation.

Time-Discrete

A *time-discrete* nonlinear system is represented by a dynamics equation

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k; k) \qquad\qquad \boldsymbol{y}_k = \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k; k)$$

with an initial value $\boldsymbol{x}_0$. Time-variant and -invariant systems as well as linear models are defined analogous to time-continuous systems. Again, linear time-invariant models can be solved in closed form:

$$\boldsymbol{x}_k = \mathbf{A}^k \boldsymbol{x}_0 + \sum_{j=0}^{k-1} \mathbf{A}^{k-j-1} \mathbf{B}\boldsymbol{u}_j \tag{2.1}$$

However, while discrete systems are easier to handle than continuous systems (e.g., computers work discretely), the world is inherently continuous. Hence, systems are often *discretized* by using discrete indices $\cdot_k$ corresponding to the value at time $t_k = kh$, where h is the *sampling time*. To apply a discrete control signal $\boldsymbol{u}_k$ to a continuous system, it is usually applied using a step function, i.e., $u(t) = u(t_k)$ for $t \in [t_k, t_{k+1})$. To compute a discrete system from a continuous system, the difference quotient can be used:

$$\dot{x} \approx \frac{\boldsymbol{x}_{k+1} - \boldsymbol{x}_k}{h} \qquad \Longrightarrow \qquad \boldsymbol{x}_{k+1} \approx \boldsymbol{x}_k + h\dot{x}$$

This is, in fact, equivalent to Euler's method for solving an initial value problem.

2.1.2 Stability

One of the fundamental properties studied in dynamical systems theory is *stability*. Stability describes the asymptotic behavior of a system: a system is either asymptotically stable, instable, or marginally stable. All of these variants can also occur in a *ringing* configuration where the system oscillates between different values (see Figure 2.1). Usually, the goal of control is to stabilize an unstable system.

Definition 1 (Global Asymptotic Stability). A dynamical system with state $\boldsymbol{x}(t)$ is globally asymptotically stable in an equilibrium point $\bar{\boldsymbol{x}}$ iff $\lim_{t\to\infty} \boldsymbol{x}(t) = \bar{\boldsymbol{x}}$ for all $\boldsymbol{x}_0 \in \mathbb{R}^n$.

Theorem 1 (Global Asymptotic Stability of Linear, Discrete-Time, Time-Invariant Systems). The system $\boldsymbol{x}_{k+1} = \mathbf{A}\boldsymbol{x}_k$ is globally asymptotically stable for $\bar{\boldsymbol{x}} = \mathbf{0}$ iff $|\lambda_i| < 1$ for all $i = 1, 2, \dots, n$, where λ_i is the i-th eigenvalue of $\mathbf{A}$.

Theorem 2 (Global Asymptotic Stability of Linear, Continuous-Time, Time-Invariant Systems). The system $\dot{\boldsymbol{x}} = \mathbf{A}\boldsymbol{x}$ is globally asymptotically stable for $\bar{\boldsymbol{x}} = \mathbf{0}$ iff $\mathrm{Re}(\lambda_i) < 0$ for all $i = 1, 2, \dots, n$, where λ_i is the i-th eigenvalue of $\mathbf{A}$.

State-Feedback Controllers

By introducing feedback-control $\boldsymbol{u}_k = -\mathbf{K}\boldsymbol{x}$ with a *gain matrix* $\mathbf{K}$, the eigenvalues of a dynamical systems are characterized by

$$\boldsymbol{x}_{k+1} = \mathbf{A}\boldsymbol{x}_k + \mathbf{B}\boldsymbol{u}_k = \mathbf{A}\boldsymbol{x}_k - \mathbf{B}\mathbf{K}\boldsymbol{x}_k = \underbrace{(\mathbf{A} - \mathbf{B}\mathbf{K})}_{=:\tilde{\mathbf{A}}}\boldsymbol{x}_k.$$

Hence, the eigenvalues (also called *poles*) can be placed arbitrarily by modifying $\mathbf{K}$ and henceforth $\tilde{\mathbf{A}}$ which defines stability.

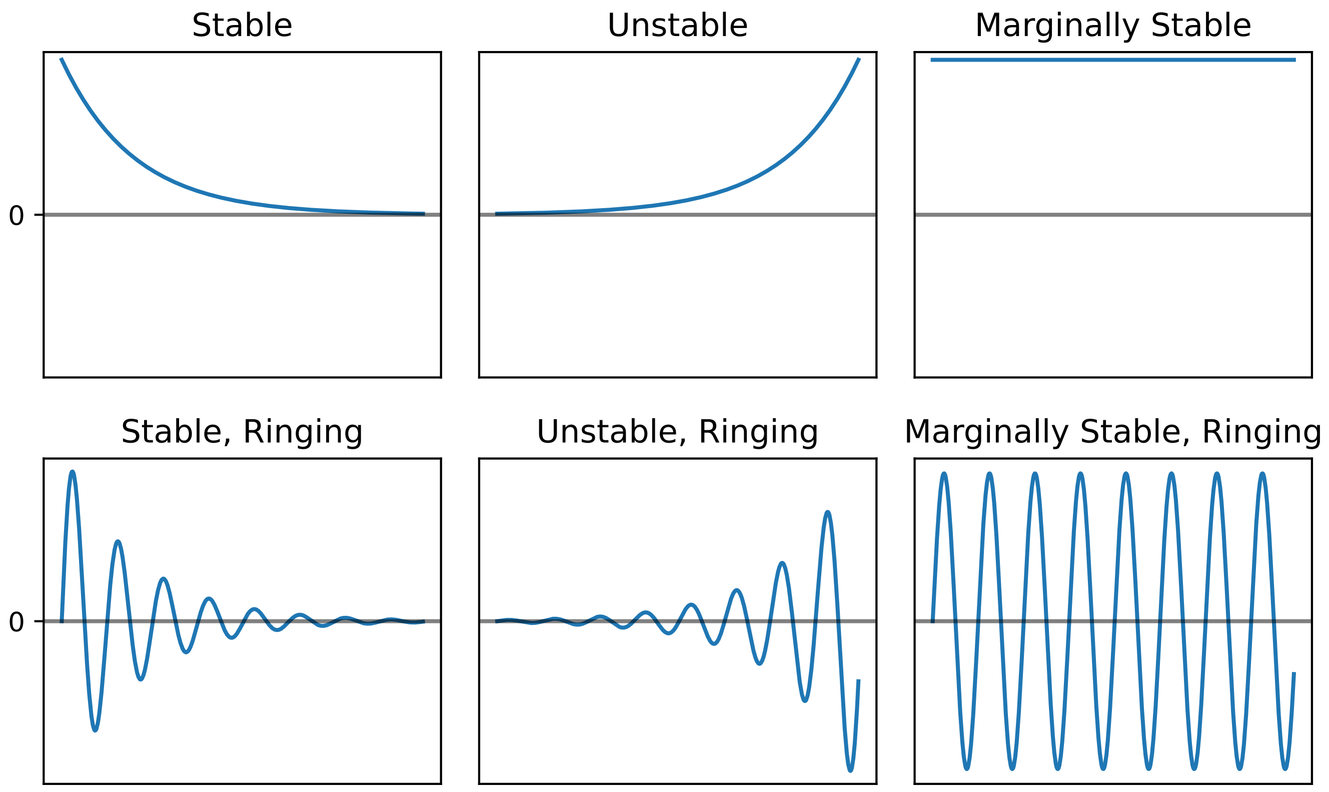

Figure 2.1: Stability Characteristics

Lyapunov Stability and Lyapunov Function

For nonlinear systems, multiple or even infinite or no equilibrium points might exist, making characterization of stability difficult. One option is *Lyapunov stability*:

Definition 2 (Lyapunov Stability). An equilibrium point $\bar{\boldsymbol{x}}$ is Lyapunov stable iff for all t and for all $\epsilon > 0$ there exists a $\delta(\epsilon) > 0$ such that $\|\boldsymbol{x}(t) - \bar{\boldsymbol{x}}\| < \epsilon$ if $\|\boldsymbol{x}(0) - \bar{\boldsymbol{x}}\| > \delta(\epsilon)$.

This builds on the intuition that stability causes the system to stay close to an equilibrium point of the system starts close to it. Lyapunov stability can be further extended to asymptotic stability of nonlinear systems by requiring that the equilibrium is attractive:

Definition 3 ((Global) Asymptotic Lyapunov Stability). An equilibrium point $\bar{\boldsymbol{x}} \in D$ is asymptotically stable in $D \subseteq \mathbb{R}^n$ if it is Lyapunov stable and attractive, i.e., $\lim_{t\to\infty}\|\boldsymbol{x}(t) - \bar{\boldsymbol{x}}\| = 0$ for all $\boldsymbol{x}_0 \in D$. If additionally $D = \mathbb{R}^n$, it is called globally asymptotically stable.

However, while these definitions are quote straightforward, checking stability for an arbitrary nonlinear system is still an open challenge. One option is to linearize the system around an equilibrium point and subsequently analyze the stability of the linear system. Another option is the usage of *Lyapunov functions*:

Definition 4 (Discrete Lyapunov Functions). A continuous function $V : D \to \mathbb{R}$, $D \subseteq \mathbb{R}^n$ is a Lyapunov function for a system $\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k)$ if all of the following hold:

1. $V(\mathbf{0}) = 0$
2. $V(\boldsymbol{x}) > 0$ for all $\boldsymbol{x} \in D \setminus \{\mathbf{0}\}$
3. $V\big(\boldsymbol{f}(\boldsymbol{x})\big) - V(\boldsymbol{x}) \leq 0$ for all $\boldsymbol{x} \in D$

If additionally $V\big(\boldsymbol{f}(\boldsymbol{x})\big) - V(\boldsymbol{x}) > 0$ holds for all $\boldsymbol{x} \in D \setminus \{\mathbf{0}\}$, V is a strict Lyapunov function.

Definition 5 (Continuous Lyapunov Functions). A continuously differentiable function $V : D \to \mathbb{R}$, $D \subseteq \mathbb{R}^n$ is a Lyapunov function for a system $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x})$ if all of the following hold:

1. $V(\mathbf{0}) = 0$
2. $V(\boldsymbol{x}) > 0$ for all $\boldsymbol{x} \in D \setminus \{\mathbf{0}\}$
3. $\dot{V}(\boldsymbol{x}) \leq 0$ for all $\boldsymbol{x} \in D$

If additionally $\dot{V}(\boldsymbol{x}) > 0$ holds for all $\boldsymbol{x} \in D \setminus \{\mathbf{0}\}$, V is a strict Lyapunov function.

Theorem 3 (Lyapunov Functions for Stability). If a Lyapunov functions exists for a dynamical system, it is locally stable in $\boldsymbol{x} = \mathbf{0}$. If the Lyapunov function is strict, the equilibrium is locally asymptotically stable.

However, finding these Lyapunov functions is generally hard. For systems derived from first order principles, the energy of the system is generally a good candidate for a Lyapunov function worth checking. Other methods for checking stability are, for example, Nyquist and Routh-Hurwitz stability.

2.1.3 Detectability, Observability, Controllability, and Stabilizability

As seen before, the eigenvalues of linear systems can be placed using a linear control law. However, being able to control a system like this has some requirements: first, the system must be *observable* (i.e., the states must be known, either by observing them directly of by reconstructing them from the measurements). Second, the system must be *controllable* (i.e., the states must be directly or indirectly influenced by the control inputs). As milder condition to stabilize a system is stabilizability requiring that at least the unstable states must be influenceable[1].

Definition 6 (Observability). A system is observable iff there exists an N such that for every initial state $\boldsymbol{x}_0$, the measurements $\boldsymbol{y}_0, \boldsymbol{y}_1, \dots, \boldsymbol{y}_{N-1}$ uniquely determine $\boldsymbol{x}_0$.

Definition 7 (Controllability). A system is controllable iff for every initial state $\boldsymbol{x}_0$ and desired state $\boldsymbol{x}_d$ there exists an input sequence $\boldsymbol{u}_0, \boldsymbol{u}_1, \dots, \boldsymbol{u}_{N-1}$ such that $\boldsymbol{x}_N = \boldsymbol{x}_d$.

Theorem 4 (Observability of Linear Time-Invariant Systems). A system $\boldsymbol{x}_{k+1} = \mathbf{A}\boldsymbol{x}_k$ is observable iff

$$\operatorname{rank}\begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \\ \mathbf{CA}^2 \\ \vdots \\ \mathbf{CA}^{n-2} \\ \mathbf{CA}^{n-1} \end{bmatrix} = n.$$

The system is detectable iff $\operatorname{rank}\begin{bmatrix}\lambda\mathbf{I} - \mathbf{A} & \mathbf{C}\end{bmatrix} = n$.

Theorem 5 (Controllability of Linear Time-Invariant Systems). A system $\boldsymbol{x}_{k+1} = \mathbf{A}\boldsymbol{x}_k$ is controllable iff

$$\operatorname{rank}\begin{bmatrix}\mathbf{B} & \mathbf{AB} & \mathbf{A}^2\mathbf{B} & \cdots & \mathbf{A}^{n-2}\mathbf{B} & \mathbf{A}^{n-1}\mathbf{B}\end{bmatrix} = n.$$

The system is stabilizable iff $\operatorname{rank}\begin{bmatrix}\lambda\mathbf{I} - \mathbf{A} & \mathbf{b}\end{bmatrix} = n$.

2.1.4 Outlook

2.2 Linear Quadratic Regulator

This section introduces the linear quadratic regulator (LQR), an unconstrained optimal control method for simple linear systems. Of course, to perform optimal control, a notion of *optimality* has to be defined. This definition also defines the overall objective of the controller. Some notions are:

- *Terminal Control Problem:* the system shall be as close to a given terminal state as possible within a given period of time
- *Minimum Time:* reach the terminal state in minimum time
- *Minimum Energy:* reach the terminal state with minimum expenditure

All of these goals are subsumed in the *cost function* J of an optimal control problem.

[1]Note that stabilizability is milder as it does not allow to steer the system to arbitrary states.

2.2.1 Cost Functions

As seen already, the cost function is the core component of an optimal control problem defining *optimality*. Some examples for cost functions are

$$J = E(\boldsymbol{x}(t_e)) \qquad J = t_e,\ \boldsymbol{x}(t_e) = \boldsymbol{x}_d \qquad J = \sum_{k=1}^{N-1} L(\boldsymbol{u}_k) \simeq \int_{t_0}^{t_e} L(\boldsymbol{u}(\tau))\,\mathrm{d}\tau$$

encoding a terminal cost, minimum time, and minimum energy, respectively (from left to right). The last cost has to be augmented for continuous problems by replacing the sum with an integral, indicated by $\simeq$. Note that these functions can be combined, e.g., by defining an energy cost and a terminal cost. Also note that L in the energy cost is pretty general and might even represent quantities aside from energy.

In a *regulation*, the desired setpoint $\boldsymbol{x}_d$ is constant (and usually zero by shifting the coordinates appropriately) and does not depend on time. The goal is therefore to stabilize the system at the desired state. With *tracking*, the setpoint is time-variant and the goal is to steer the system to follow the trajectory (trajectory tracking).

2.2.2 LQR Formulation

In the LQR setting, a linear time-discrete system along with a quadratic cost function

$$J = \frac{1}{2}\sum_{k=1}^{N-1} \boldsymbol{x}_k^\top \mathbf{Q}\boldsymbol{x}_k + \boldsymbol{u}_k^\top \mathbf{R}\boldsymbol{u}_k + \boldsymbol{x}_N^\top \mathbf{S}\boldsymbol{x}_N. \tag{2.2}$$

with[2] $\mathbf{Q} \succeq 0$, $\mathbf{R} \succ 0$, $\mathbf{S} \succ 0$. Besides the dynamics $\boldsymbol{x}_{k+1} = \mathbf{A}_k\boldsymbol{x}_k + \mathbf{B}_k\boldsymbol{u}$, no further constraints are considered. The optimal control problem is now framed as follows (with $\tilde{\boldsymbol{u}} = \boldsymbol{u}_{1:N-1} = \{\boldsymbol{u}_1, \boldsymbol{u}_2, \dots, \boldsymbol{u}_{N-1}\}$):

$$\begin{aligned} &\min_{\tilde{\boldsymbol{u}}} \frac{1}{2}\sum_{k=1}^{N-1} \boldsymbol{x}_k^\top \mathbf{Q}\boldsymbol{x}_k + \boldsymbol{u}_k^\top \mathbf{R}\boldsymbol{u}_k + \boldsymbol{x}_N^\top \mathbf{S}\boldsymbol{x}_N \\ \text{s.t.} \quad &\boldsymbol{x}_{k+1} = \mathbf{A}_k\boldsymbol{x}_k + \mathbf{B}_k\boldsymbol{u}_k \end{aligned} \tag{2.3}$$

In the upcoming sections, this problem is solved in two different fashions.

2.2.3 Batch Optimization

The straightforward approach for solving (2.3) in the time-invariant case is to use batch optimization over the variables $\tilde{\boldsymbol{u}}$ by treating them as a function of the initial state $\boldsymbol{x}_0$ by exploiting the closed form solution (2.1) and writing it in vector form:

$$\underbrace{\begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_{N-1} \\ \boldsymbol{x}_N \end{bmatrix}}_{\tilde{\boldsymbol{x}} :=} = \underbrace{\begin{bmatrix} \mathbf{I} \\ \mathbf{A} \\ \mathbf{A}^2 \\ \vdots \\ \mathbf{A}^{N-1} \\ \mathbf{A}^N \end{bmatrix}}_{\tilde{\mathbf{A}} :=} \boldsymbol{x}_0 + \underbrace{\begin{bmatrix} \mathbf{O} & \cdots & \cdots & \cdots & \mathbf{O} \\ \mathbf{B} & \mathbf{O} & \cdots & \cdots & \mathbf{O} \\ \mathbf{AB} & \mathbf{B} & \mathbf{O} & \cdots & \mathbf{O} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \mathbf{A}^{N-2}\mathbf{B} & \mathbf{A}^{N-3}\mathbf{B} & \ddots & \mathbf{B} & \mathbf{O} \\ \mathbf{A}^{N-1}\mathbf{B} & \mathbf{A}^{N-2}\mathbf{B} & \mathbf{A}^{N-3}\mathbf{B} & \cdots & \mathbf{B} \end{bmatrix}}_{\tilde{\mathbf{B}} :=} \underbrace{\begin{bmatrix} \boldsymbol{u}_0 \\ \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \\ \vdots \\ \boldsymbol{u}_{N-1} \end{bmatrix}}_{\tilde{\boldsymbol{u}} :=}$$

[2]For a matrix $\mathbf{M}$, $\mathbf{M} \succeq 0$ and $\mathbf{M} \succ 0$ mean that $\mathbf{M}$ is (semi) positive definite.

Using the matrices defined above, this can be written shortly as $\tilde{\boldsymbol{x}} = \tilde{\mathbf{A}}\boldsymbol{x}_0 + \tilde{\mathbf{B}}\tilde{\boldsymbol{u}}$. Similarly, the cost function (2.2) can be reformulated as $J(\boldsymbol{x}_0, \tilde{\boldsymbol{u}}) \propto \tilde{\boldsymbol{x}}^\top \tilde{\mathbf{Q}}\tilde{\boldsymbol{x}} + \tilde{\boldsymbol{u}}^\top \tilde{\mathbf{R}}\tilde{\boldsymbol{u}}$ with

$$\tilde{\mathbf{Q}} = \operatorname{diag}(\underbrace{\mathbf{Q}, \mathbf{Q}, \dots, \mathbf{Q}}_{N \text{ times}}, \mathbf{S}) \qquad \tilde{\mathbf{R}} = \operatorname{diag}(\underbrace{\mathbf{R}, \mathbf{R}, \cdots, \mathbf{R}}_{N \text{ times}}).$$

By plugging $\tilde{\boldsymbol{x}}$ into this cost function, it can be solved in closed form, yielding the optimal control inputs

$$\tilde{\boldsymbol{u}}^* = -\mathbf{H}^{-1}\mathbf{F}^\top \boldsymbol{x}_0,$$

with $\mathbf{H} = \tilde{\mathbf{B}}^\top \tilde{\mathbf{Q}}\tilde{\mathbf{B}} + \tilde{\mathbf{R}}$ and $\mathbf{F} = \tilde{\mathbf{A}}^\top \tilde{\mathbf{Q}}\tilde{\mathbf{B}}$.

However, while this approach is simplistic, it a major caveat: no feedback is involved (the control signals just depend on the initial value, i.e., it is an open-look controller). Hence, if the real system deviates from the model, the control inputs might be suboptimal or even harmful. This problem is addressed by the next solution method which also exploits the special structure of the problem.

2.2.4 Dynamic Programming

As seen before, applying batch optimization—while being straightforward—is not ideal as the solution does not incorporate the system's feedback. An alternative approach is to use *dynamic programming*. The underlying idea of dynamic programming is that partial trajectories of trajectories are optimal, too. In other words: a trajectory composed of partial optimal ones is optimal. A relevant quantity for solving LQR with this principle is the optimal *cost to go*, also called the *value function*:

$$J_j^*(\boldsymbol{x}_j) = \min_{\boldsymbol{u}_{j:N-1}} \frac{1}{2} \sum_{k=j}^{N-1} \boldsymbol{x}_k^\top \mathbf{Q}\boldsymbol{x}_k + \boldsymbol{u}_k^\top \mathbf{R}\boldsymbol{u}_k + \boldsymbol{x}_N^\top \mathbf{S}\boldsymbol{x}_N.$$

This function quantifies the optimal cost when starting from state $\boldsymbol{x}_j$ at time j. By solving this problem[3] recursively starting from $j = N$, the optimal control input at time step k is found to be

$$\boldsymbol{u}_k^* = \mathbf{K}_k \boldsymbol{x}_k$$

with $\mathbf{K}_k = -\big(\mathbf{B}^\top \mathbf{P}_{k+1}\mathbf{B} + \mathbf{R}\big)^{-1}\mathbf{B}^\top \mathbf{P}_{k+1}\mathbf{A}$ and optimal cost $J_k^*(\boldsymbol{x}_j) = \boldsymbol{x}_k^\top \mathbf{P}_k \boldsymbol{x}_k / 2$. Here, $\mathbf{P}_k$ is given by the *discrete time-variant algebraic Riccati equation*

$$\mathbf{P}_k = \mathbf{Q} + \mathbf{A}^\top \mathbf{P}_{k+1}\mathbf{A} - \mathbf{A}^\top \mathbf{P}_{k+1}\mathbf{B}\big(\mathbf{R} + \mathbf{B}^\top \mathbf{P}_{k+1}\mathbf{B}\big)^{-1}\mathbf{B}^\top \mathbf{P}_{k+1}\mathbf{A}$$

which has to be calculated from $k = N, N-1, \dots, 1$, starting with $\mathbf{P}_N = \mathbf{S}$. With $N \to \infty$, $\mathbf{P}_k$ becomes k-independent and the Riccati equation becomes an implicit algebraic equation, the *discrete time algebraic Riccati equation* (DARE):

$$\mathbf{P} = \mathbf{Q} + \mathbf{A}^\top \mathbf{P}\mathbf{A} - \mathbf{A}^\top \mathbf{P}\mathbf{B}\big(\mathbf{R} + \mathbf{B}^\top \mathbf{P}\mathbf{B}\big)^{-1}\mathbf{B}^\top \mathbf{P}\mathbf{A}. \tag{DARE}$$

Thus, also $\mathbf{K}$ become time-invariant and the feedback control law becomes time-invariant, too. Note that the Riccati equation only converges to the above value if $(\mathbf{A}, \mathbf{B})$ is stabilizable and $(\mathbf{Q}^{1/2}, \mathbf{A})$ is detectable.

For continuous dynamics and cost, the optimal input is given as $\boldsymbol{u}^*(t) = \mathbf{K}(t)\boldsymbol{x}(t)$ with $\mathbf{K}(t) = -\mathbf{R}^{-1}\mathbf{B}^\top \mathbf{P}(t)$ and the solution of the *continuous time Riccati equation* (CARE):

$$-\dot{\mathbf{P}}(t) = \mathbf{P}(t)\mathbf{A} + \mathbf{A}^\top \mathbf{P}(t) - \mathbf{P}(t)\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^\top \mathbf{P}(t) + \mathbf{Q} \tag{CARE}$$

The optimal cost is again $J_t^*\big(\boldsymbol{x}(t)\big) = \boldsymbol{x}^\top(t)\mathbf{P}(t)\boldsymbol{x}(t)/2$. For an infinite horizon, all of this becomes again time-invariant with $\dot{\mathbf{P}} = \mathbf{O}$, turning (CARE) again into al algebraic equation.

[3] See "Robot Learning" (`https://fabian.damken.net/summaries/cs/elective/ce/role/role-summary.pdf`) for a more thorough (stochastic) treatment.

2.2.5 Stability

Theorem 6 (Stability of the Time-Discrete LQR with Infinite Horizon). Consider a time-discrete linear system with quadratic cost where $(\mathbf{A}, \mathbf{B})$ is stabilizable and $(\mathbf{A}, \mathbf{Q}^{1/2})$ is detectable. Then the solution

$$\begin{aligned} \boldsymbol{u}_k^* &= \mathbf{K}\boldsymbol{x}_k \\ \mathbf{K} &= -\big(\mathbf{B}^\top \mathbf{P}\mathbf{B} + \mathbf{R}\big)^{-1} \mathbf{B}^\top \mathbf{P}\mathbf{A} \\ \mathbf{P} &= \mathbf{Q} + \mathbf{A}^\top \mathbf{P}\mathbf{A} - \mathbf{A}^\top \mathbf{P}\mathbf{B}\big(\mathbf{R} + \mathbf{B}^\top \mathbf{P}\mathbf{B}\big)^{-1} \mathbf{B}^\top \mathbf{P}\mathbf{A} \end{aligned}$$

of the optimal control problem asymptotically stabilizes the system $\boldsymbol{x}_{k+1} = \mathbf{A}\boldsymbol{x}_k + \mathbf{B}\boldsymbol{u}_k$.

Proof Sketch. Showing asymptotic stability can be done by proofing that the infinite horizon cost $J^*(\boldsymbol{x}_k) = \boldsymbol{x}_k^\top \mathbf{P}\boldsymbol{x}_k$ is a strict Lyapunov function of the (controlled) system. □

Note that for finite-horizon LQR, the optimal input is not necessarily stabilizing!

2.3 Constrained Static Optimization

So far, optimal control has been considered without any constraints. However, most real systems have constraints (e.g., a car shall stay on the road, forces are limited, ...). This section focuses on optimization with constraints in a *static* setting, i.e., where the system does not evolve (it is not a *dynamic* system). The general form of such an optimization problem is

$$\begin{aligned} \min_{\boldsymbol{u} \in \mathbb{R}^n} \quad & F(\boldsymbol{u}) \\ \text{s.t.} \quad & \boldsymbol{G}(\boldsymbol{u}) = \mathbf{0} \\ & \boldsymbol{H}(\boldsymbol{u}) \leq \mathbf{0} \end{aligned}$$

where $F : \mathbb{R}^n \to \mathbb{R}$, $\boldsymbol{G} : \mathbb{R}^n \to \mathbb{R}^m$, and $\boldsymbol{H} : \mathbb{R}^n \to \mathbb{R}^p$ are the objective function, equality, and inequality constraints, respectively. The $=$ and $\leq$ in the constraints are element-wise (with a slight abuse of notation). A constraint is called *active* if the solution $\boldsymbol{u}^*$ causes the constraint to vanish (thus, equality constraints are always active and inequality constraints may be inactive).

The optimization problem is *feasible* if the feasible set $\mathcal{U} = \{\boldsymbol{u} : \boldsymbol{G}(\boldsymbol{u}) = \mathbf{0}, \boldsymbol{H}(\boldsymbol{u}) \leq \mathbf{0}\}$ is non-empty. With this set, the following notions of optimality can be defined:

Definition 8 (Local Optimality). A point $\boldsymbol{u}^*$ is locally optimal if $F(\boldsymbol{u}) \geq F(\boldsymbol{u}^*)$ for all $\boldsymbol{u} \in \mathcal{U}$ with $\|\boldsymbol{u} - \boldsymbol{u}^*\| \leq R$ for some $R > 0$, i.e., for all points in a vicinity of $\boldsymbol{u}^*$.

Definition 9 (Global Optimality). A point $\boldsymbol{u}^*$ is globally optimal if $F(\boldsymbol{u}) \geq F(\boldsymbol{u}^*)$ for all $\boldsymbol{u} \in \mathcal{U}$.

In both cases, the optimal cost is $p^* = F(\boldsymbol{u}^*)$. If the problem is infeasible, it is said to have optimal cost $p^* = \infty$.

2.3.1 Convexity

It turns out that *convex* optimization problems are especially easy: every local optimum is a global optimum. To discuss this further, first convexity and a few other nomenclature have to be defined:

Definition 10 (Convex Sets). A set $\mathcal{U}$ is convex iff $\lambda\boldsymbol{u} + (1 - \lambda)\boldsymbol{v} \in \mathcal{U}$ holds for all $\boldsymbol{u}, \boldsymbol{v} \in \mathcal{U}$ and $\lambda \in [0, 1]$. That is, all points on a line between two arbitrary points lie in the set, too.

Definition 11 (Convex Functions). A function $F : \mathcal{U} \to \mathbb{R}$ is convex iff $F\big(\lambda \boldsymbol{u} + (1-\lambda)\boldsymbol{v}\big) \leq \lambda F(\boldsymbol{u}) + (1-\lambda)F(\boldsymbol{v})$ holds for all $\boldsymbol{u}, \boldsymbol{v} \in \mathcal{U}$ and $\lambda \in [0, 1]$. A function is strictly convex iff the relaxed inequality is never equal.

For convex functions, it is possible to use the derivatives of the function (if it is differentiable) to check its convexity:

Theorem 7 (First Order Condition for Convexity). A differentiable function $F : \mathcal{U} \to \mathbb{R}$ is convex iff $F(\boldsymbol{v}) \geq F(\boldsymbol{u}) + \boldsymbol{\nabla} F^{\top}(\boldsymbol{u})(\boldsymbol{v} - \boldsymbol{u})$ holds for all $\boldsymbol{u}, \boldsymbol{v} \in \mathcal{U}$ where $\boldsymbol{\nabla} F$ is the gradient of F.

Theorem 8 (Second Order Condition for Convexity). A twice differentiable function $F : \mathcal{U} \to \mathbb{R}$ is convex iff $\boldsymbol{\nabla}^2 F(\boldsymbol{u}) \succeq 0$ holds for all $\boldsymbol{u} \in \mathcal{U}$ where $\boldsymbol{\nabla}^2 F$ is the Hessian of F.

Additionally, the further discussion needs the notion of *(sub-)level sets:*

Definition 12 (Level Sets). The level set L_c of a function $F : \mathcal{U} \to \mathbb{R}$ is the set of values for which F takes on a constant value c, i.e., $L_c := \{\boldsymbol{u} : F(\boldsymbol{u}) = c\}$.

Definition 13 (Sublevel Sets). The sublevel set L_c^- of a function $F : \mathcal{U} \to \mathbb{R}$ is the set of values for which F takes on a value equal or less than a constant c, i.e., $L_c^- := \{\boldsymbol{u} : F(\boldsymbol{u}) \leq c\}$. Note that if F is convex or concave, L_c^- is convex.

Theorem 9 (Convexity of Level Sets). Let $F : \mathcal{U} \to \mathbb{R}$ be a convex function. Then the level set L_c of F for some constant c is convex.

Proof. Let L_c^- and L_{-c}^- be the sublevel sets of F and $-F$, respectively. As F is convex, $-F$ is concave and hence both L_c^- and L_{-c}^- are convex. By definition, $L_c^- \cap L_c^+ = \{\boldsymbol{u} : F(\boldsymbol{u}) \leq c\} \cap \{\boldsymbol{u} : -F(\boldsymbol{u}) \leq -c\} = \{\boldsymbol{u} : F(\boldsymbol{u}) \leq c\} \cap \{\boldsymbol{u} : F(\boldsymbol{u}) \geq c\} = \{\boldsymbol{u} : F(\boldsymbol{u}) = c\} = L_c$ is the level set of F for c. Hence, as the intersection of convex sets is convex again, L_c is convex. $\square$

Convex Optimization Problems and Optimality

Theorem 10 (Global Optimality of Convex Optimization Problems). For an optimization problem with a convex feasibility set $\mathcal{U}$ and a convex objective function, a convex optimization problem, every locally optimal solution is globally optimal.

To check convexity of an optimization problem, it usually suffices to look at the objective and constraints separately due to the properties of convex sets, namely that the intersection of two convex sets is still convex. If all constraints are convex, their respective (sub-) level sets are convex too, and hence the feasibility set is convex.

2.3.2 Quadratic Programming

If the objective function is a quadratic function $J(\boldsymbol{u}) = \boldsymbol{u}^{\top}\mathbf{Q}\boldsymbol{u} + \boldsymbol{q}^{\top}\boldsymbol{u} + r$ with $\mathbf{Q} \in \mathbb{R}^{n \times n}$, $\mathbf{Q} \succeq 0$, $\boldsymbol{q} \in \mathbb{R}^n$, and $r \in \mathbb{R}$ and the constraints are affine, the problem is called q *quadratic program* (QP). For QPs, the optimal $\boldsymbol{u}^*$ is either inside the feasible set or at its boundary. If the inequality constraints are quadratic and convex, too, the problem is called a *quadratically constrained quadratic program* (QCQP) and the feasible set is an intersection of the ellipsoids defined by the constraints. Quadratic programs are a friendly class of problems for which many efficient solvers exist.

2.3.3 Optimality Conditions for Constrained Optimization Problems

The optimality conditions for constrained optimization problems use the (generalized) Lagrangian:

Definition 14 ((Generalized) Lagrangian). For an optimization problem with objective $F(\boldsymbol{u})$, equality constraints $\boldsymbol{G}(\boldsymbol{u}) = \boldsymbol{0}$, and inequality constraints $\boldsymbol{H}(\boldsymbol{u}) \leq \boldsymbol{0}$, the generalized Lagrangian is

$$\mathcal{L}(\boldsymbol{u}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = F(\boldsymbol{u}) + \boldsymbol{\lambda}^\top \boldsymbol{G}(\boldsymbol{u}) + \boldsymbol{\mu}^\top \boldsymbol{H}(\boldsymbol{u})$$

with the Lagrange multipliers $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$. If no inequality constraints are present, this is called the Lagrangian and with inequality constraints it is the generalized Lagrangian.

The Lagrangian is a weighted sum of the objective and the constraints. It can now be used to characterize the optimality conditions for the optimization problem:

Theorem 11 (Necessary First Order Conditions (Karush-Kuhn-Tucker)). Let $\boldsymbol{u}^*$ be a (local) extrema of F subject to $\boldsymbol{G}(\boldsymbol{u}^*) = \boldsymbol{0}$ and $\boldsymbol{H}(\boldsymbol{u}^*) \leq \boldsymbol{0}$, then there exist Lagrangian multipliers $\boldsymbol{\lambda}^*$ and $\boldsymbol{\mu}^*$ such that all of the following hold:

$$\begin{aligned} \boldsymbol{\nabla}_{\boldsymbol{u}} \mathcal{L}(\boldsymbol{u}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) &= \boldsymbol{0} \\ \boldsymbol{\nabla}_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{u}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) &= \boldsymbol{0} \\ \boldsymbol{\nabla}_{\boldsymbol{\mu}} \mathcal{L}(\boldsymbol{u}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) &\leq \boldsymbol{0} \\ \left(\boldsymbol{\mu}^*\right)^\top \boldsymbol{H}(\boldsymbol{u}^*) &= \boldsymbol{0} \\ \boldsymbol{\mu}^* &\geq \boldsymbol{0} \end{aligned} \tag{KKT}$$

For problems with strong duality, these conditions are also sufficient.

Theorem 12 (Necessary Second Oder Conditions). Let $\boldsymbol{u}^*$ be such that the KKT-conditions hold. If $\boldsymbol{u}^*$ is a minimum, the following holds for all $\boldsymbol{p}$ with $\boldsymbol{p}^\top \boldsymbol{\nabla}_{\boldsymbol{u}} \boldsymbol{G}(\boldsymbol{u}^*) = 0$:

$$\boldsymbol{p}^\top \left(\boldsymbol{\nabla}^2_{\boldsymbol{u}} \mathcal{L}(\boldsymbol{u}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)\right) \boldsymbol{p} \geq 0$$

If this inequality is strict, this condition is also sufficient.

2.3.4 Numerical Solvers

A variety of numerical solvers exist for constrained optimization problems, and especially fast solvers exist for QPs (e.g., Matlab's `quadprog` and `fmincon`). Some approaches are interior point, trust region, and active set, and the class of sequential quadratic programming (SQP) methods[4].

SQP methods use a Taylor approximation of the objective (quadratic) and constraints (linear) to solve the optimization problem. By applying this method multiple times, it usually converges to a good local minimum. Another method for dealing with constraints is to reformulate them and integrate them into the objective by using barrier functions and penalty terms: penalty methods add a term $\epsilon(\boldsymbol{u})$ to the objective that has a high values once a constraint is violated, forcing the optimizer to look elsewhere. Common choices for such a barrier term are logarithmic barrier functions and exact penalty functions. An alternative method is to allow the constraints to be violated. This is done by introducing a slack variable $\boldsymbol{H}(\boldsymbol{u}) \leq \epsilon$ and penalize it in the objective with a function $\Theta(\epsilon)$, e.g., a quadratic function in ϵ

[4]See "Optimization of Static and Dynamic Systems" (`https://fabian.damken.net/summaries/cs/elective/ce/opt/opt-summary.pdf`) for a more thorough study of numerical optimization techniques

3 Nominal Model Predictive Control

So far, optimal control without constraints (LQR) and constrained optimization without dynamics (KKT) has been covered. However, in MPC, an optimal control problem should be solved while taking constraints into account. This chapter deals with the simplest case of MPC, *nominal* MPC. In nominal MPC, it is assumed that the model at hand is perfect, i.e., that the real system evolves exactly as the model predicts. Subsequently, after introducing nominal MPN and receding horizon control, certain properties regarding stability as discussed.

The general MPC problem has the form

$$\begin{aligned}\min_{\tilde{\boldsymbol{u}}} \;& \sum_{k=0}^{\infty} F(\boldsymbol{x}_k, \boldsymbol{u}_k) \\ \text{s.t.} \quad & \boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k),\, \boldsymbol{x}_0 \\ & \boldsymbol{x}_k \in \mathcal{X} \\ & \boldsymbol{u}_k \in \mathcal{U}\end{aligned}$$

with stage cost F, the state dynamics $\boldsymbol{f}$ with initial state $\boldsymbol{x}_0$, and state and input constraints $\mathcal{X}$ and $\mathcal{U}$, respectively. With LQR-conformant cost, the optimal controller would be stabilizable in general. However, solving this problem with optimal control is impossible due to the constraints. Also, due to the infinite time horizon, it is not possible to apply batch normalization as the number of variables are infinite. The underlying idea of MPC now is to approximate the infinite horizon problem by a finite horizon problem

$$\begin{aligned}\min_{\tilde{\boldsymbol{u}}} \;& E(\boldsymbol{x}_n) + \sum_{k=0}^{N-1} F(\boldsymbol{x}_k, \boldsymbol{u}_k) \\ \text{s.t.} \quad & \boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k),\, \boldsymbol{x}_0 \\ & \boldsymbol{x}_k \in \mathcal{X} \\ & \boldsymbol{u}_k \in \mathcal{U} \\ & \boldsymbol{x}_N \in \mathcal{E}\end{aligned} \tag{3.1}$$

with a terminal cost E and terminal constraints $\mathcal{E}$. These can be interpreted as follows: the terminal cost approximates the “remainder” of the cost behind the horizon and the terminal constraints approximate the “remainder” of the constraints. However, even this finite-horizon MPC cannot be solved using dynamic programming due to the constraints; but it is possible to apply batch optimization! By repeatedly solving the batch optimization problem, feedback is incorporated by re-computing the batch solution at every time step and applying only a single input. That is, the feedback is given by repeatedly measuring the state and incorporating it as the initial value of the dynamical systems. This method is also called *receding horizon control*.

Then the problem of MPC boils down to two pressing questions:

- How to guarantee that the optimal control problem is feasible in every time step?
- How to guarantee that the optimal solution of the finite horizon approximate actually stabilizes the system?

The remainder of this chapter mostly deals with these two questions in *nominal* settings. As already said, this means that the system model is a perfect representation and the plant will behave exactly as predicted. Note that this is not true in reality (due to modeling errors, noise, etc.) but is a purely theoretical assumption[1]. Incorporating uncertainties into the model and control is covered in chapter 4 and 5.

3.1 Linear MPC

Linear MPC assumes a linear system model and a quadratic stage cost, leaving the following optimization problem (at time instance i):

$$\begin{aligned} \min_{\hat{\boldsymbol{u}}} \ & \boldsymbol{x}_{i+N}^\top \mathbf{S} \boldsymbol{x}_{i+N} \sum_{k=i}^{i+N-1} \boldsymbol{x}_k^\top \mathbf{Q} \boldsymbol{x}_k + \boldsymbol{u}_k^\top \mathbf{R} \boldsymbol{u}_k \\ \text{s.t.} \ & \boldsymbol{x}_{k+1} = \mathbf{A}\boldsymbol{x}_k + \mathbf{B}\boldsymbol{u}_k,\ \boldsymbol{x}_i \\ & \boldsymbol{x}_k \in \mathcal{X} \\ & \boldsymbol{u}_k \in \mathcal{U} \\ & \boldsymbol{x}_{i+N} \in \mathcal{E} \end{aligned} \tag{3.2}$$

where $\mathcal{X}$, $\mathcal{U}$, and $\mathcal{E}$ are convex sets. As the dynamics, constraints, and cost are time-invariant, the notation can be simplified by assuming $i = 0$; cf. (3.1) with the initial state $\boldsymbol{x}_0$ being equal to the i-th measured state.

3.1.1 Feasibility and Stability

Recall from section 2.3 that a static optimization problem is feasible iff the feasible set is non-empty. In dynamic optimization, an optimal control problem is feasible iff the constraints on the states and controls, $\mathcal{X}$ and $\mathcal{U}$, the terminal constraints $\mathcal{E}$, and the dynamics are satisfied. Hence, checking feasibility is more tedious as it has to be checked whether there exists an input sequence satisfying all constraints (a *admissible* input sequence).

Definition 15 (Admissible Input Sequence). An input sequence $U(\boldsymbol{x}_0) \coloneqq (\boldsymbol{u}_0, \boldsymbol{u}_1, \dots, \boldsymbol{u}_{N-1})$ for an initial state $\boldsymbol{x}_0$ is admissible iff all constraints are satisfied:

$$\forall k = 0, 1, \dots, N-1 : \boldsymbol{u}_k \in \mathcal{U} \qquad \forall k = 0, 1, \dots, N-2 : \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k) \in \mathcal{X} \qquad \boldsymbol{f}(\boldsymbol{x}_{N-1}, \boldsymbol{u}_{N-1}) \in \mathcal{E}$$

Definition 16 (Feasible Initial Conditions). Let $\mathcal{X}_0 \coloneqq \{\boldsymbol{x}_0 : \text{exists an admissible } U(\boldsymbol{x}_0)\}$ be the set of feasible initial conditions.

With these two definitions, proofing feasibility of linear MPC means to show that $\mathcal{X}_0$ is non-empty for all consecutive time steps, i.e., the states need to stay in $\mathcal{X}_0$ for all time (also called *recursive* feasibility). However, even for nominal MPC, this can (in general) not be guaranteed due to the finite-horizon problem being "shortsighted": the controller might drive the system towards the boundary of the constraints in the long run which it cannot account for as the horizon is too short. Once MPC "sees" the problem, it may be impossible to countersteer due to input bounds.

While recursive feasibility is impossible to achieve for general linear MPC, it is achievable for some (important) special cases using the terminal constraints and cost. These are studied in the upcoming sections.

[1]Although, as discussed in section 4.1, it turns out that even nominal MPC is (to some extend) robust towards modeling errors.

Terminal Equality

By fixing the terminal state to zero, i.e., $\mathcal{E} = \{\mathbf{0}\}$, the terminal cost vanishes. Then recursive feasibility and stability can be shown using the terminal set and Lyapunov functions, respectively. However, requiring that $\boldsymbol{x}_N = \mathbf{0}$ is restrictive and makes the set of feasible initial conditions (for the first control problem) small. This is due to the requirement that the origin is reachable in N steps. While this can be tackled by increasing N, this increases the computational effort as the number of optimization variables increases. Also, it just pushes the problem away a bit.

Theorem 13 (Recursively Feasibility of Linear MPC With Terminal Equality Constraint). Linear MPC with the receding horizon optimal control problem (3.2) is recursively feasible if the initial problem is feasible and $\mathcal{E} = \{\mathbf{0}\}$.

Proof. Let the linear MPC be feasible for the first instantiation with $i = 0$ and the initial condition $\boldsymbol{x}_0$. Let $(\boldsymbol{u}_0^*, \boldsymbol{u}_1^*, \dots, \boldsymbol{u}_{N-1}^*)$ be the optimal admissible input sequence. The corresponding state sequence end, by the terminal constraints, with $\boldsymbol{x}_N^* = \mathbf{0}$. As the MPC is nominal, the system evolves to $\boldsymbol{x}_1 = \boldsymbol{x}_1^* = \mathbf{A}\boldsymbol{x}_0 + \mathbf{B}\boldsymbol{u}_0^*$. For $i = 1$, $\boldsymbol{x}_1^*$ is used as the input sequence. Therefore, the previous input sequence $(\boldsymbol{u}_1^*, \dots, \boldsymbol{u}_{N-1}^*)$ is still admissible and optimal, but one entry too short. This can be fixed by appending $\boldsymbol{u}_N^* = \mathbf{0}$, yielding $\boldsymbol{x}_{N+1}^* = \mathbf{A}\boldsymbol{x}_N^* + \mathbf{B}\boldsymbol{u}_N^* = \mathbf{A}\mathbf{0} + \mathbf{B}\mathbf{0} = \mathbf{0}$. This next terminal state still fulfills the state constraints. Hence, by repeating this procedure, linear MPC with zero terminal state is recursively feasible. $\square$

Theorem 14 (Asymptotic Stability of Linear MPC With Terminal Equality Constraint). Linear MPC with the receding horizon optimal control problem (3.2) is asymptotically stable if the initial problem is feasible and $\mathcal{E} = \{\mathbf{0}\}$.

Proof. $\square$

Terminal Inequality

To fix the problems introduced by a terminal equality constraint (namely that ht feasible set of initial conditions is small), this section extends the terminal set to have more than a single value. For discusses which properties the terminal set has to fulfill, a few definitions are needed beforehand:

Definition 17 (Invariant Set). A set $\mathcal{S}$ is positively invariant for a system $\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k)$ if $\boldsymbol{x}_k \in \mathcal{S}$ holds for all $k \in \mathbb{N}_+$ given that $\boldsymbol{x}_0 \in \mathcal{S}$. In other words: trajectories starting in an invariant set, stay in the invariant set. The unique set that contains all positively invariant sets is called the maximum positively invariant set.

Definition 18 (Control Invariant Set). A set $\mathcal{S}$ is control invariant for a system $\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k)$ if for all $\boldsymbol{x}_k \in \mathcal{S}$ there exists an input $\boldsymbol{u}_k \in \mathcal{U}$ such that $\boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k) \in \mathcal{S}$ holds. In other words: every state in a control invariant set can be steered such that the successor is still in the set. The set containing all other control invariant sets is called the maximum control invariant set and is denoted $\mathcal{S}_\infty$. Let $\boldsymbol{u}_k = \kappa_{\boldsymbol{f}}^{\mathcal{S}}(\boldsymbol{x}_k)$ be the control law ensuring control invariance.

Using these definitions, it can be shown that linear MPC with a control invariant terminal set is recursively feasible and stabilizing. However, finding the terminal constraint set is rather hard and there is no general “recipe” to do it. For linear systems, one approach is to use unconstrained, infinite-horizon LQR, and finding the maximum invariant set under the closed-loop control law.

Theorem 15 (Recursively Feasibility of Linear MPC With Terminal Inequality Constraint). Linear MPC with the receding horizon optimal control problem (3.2) is recursively feasible (i.e., $\mathcal{X}_0$ is positively invariant) if the initial problem $\mathcal{E}$ is control invariant.

Proof. $\square$

Theorem 16 (Asymptotic Stability of Linear MPC With Terminal Inequality Constraint). Linear MPC with the receding horizon optimal control problem (3.2) is asymptotically stable if the initial problem $\mathcal{E}$ is control invariant.

Proof. $\square$

3.2 Solving MPC Problem

Until now, (nominal) MPC was studied from a rather theoretical perspective without actually solving it. This sections deals with two methods for reformulating a (linear) MPC problem in a quadratic program that can be solved efficiently. For nonlinear MPC, the prevalent approach is to approximate the problem locally to get quadratic optimization problems. Reformulating the optimal control problem as a QP can be done in two versions: with and without substituting the dynamics.

3.2.1 Option A: Substituting the Dynamics

This first idea is directly borrowed from batch optimization in LQR (subsection 2.2.3). To incorporate the linear constraints

$$\mathcal{U} = \{\boldsymbol{u} : \mathbf{A}_u \boldsymbol{u} \le \boldsymbol{b}_u\} \qquad \mathcal{X} = \{\boldsymbol{x} : \mathbf{A}_x \boldsymbol{x} \le \boldsymbol{b}_x\} \qquad \mathcal{E} = \{\boldsymbol{x} : \mathbf{A}_f \boldsymbol{x} \le \boldsymbol{b}_f\}$$

they are reformulated into a single constraint $\tilde{\mathbf{H}}\boldsymbol{u} \le \boldsymbol{w} + \mathbf{E}\boldsymbol{x}_0$. In a first step, the constraints

$$\mathbf{A}_u \boldsymbol{u}_k \le \boldsymbol{b}_u \qquad \mathbf{A}_x \boldsymbol{x}_k \le \boldsymbol{b}_x \qquad \mathbf{A}_f \boldsymbol{x}_N \le \boldsymbol{b}_f$$

for $k = 0, 1, \dots, N-1$ are batched using the matrices and vectors

$$\tilde{\mathbf{A}}_u = \operatorname{diag}(\underbrace{\mathbf{A}_u, \mathbf{A}_u, \dots, \mathbf{A}_u}_{N \text{ times}}) \qquad \tilde{\boldsymbol{b}}_u = (\underbrace{\boldsymbol{b}_u^\top, \boldsymbol{b}_u^\top, \dots, \boldsymbol{b}_u^\top}_{N \text{ times}})^\top$$

$$\tilde{\mathbf{A}}_x = \operatorname{diag}(\underbrace{\mathbf{A}_x, \mathbf{A}_x, \dots, \mathbf{A}_x}_{N \text{ times}}, \mathbf{A}_f) \qquad \tilde{\boldsymbol{b}}_x = (\underbrace{\boldsymbol{b}_x^\top, \boldsymbol{b}_x^\top, \dots, \boldsymbol{b}_x^\top}_{N \text{ times}}, \boldsymbol{b}_f^\top)^\top$$

yielding $\tilde{\mathbf{A}}_u \tilde{\boldsymbol{u}} \le \tilde{\boldsymbol{b}}_u$ and $\tilde{\mathbf{A}}_x \tilde{\boldsymbol{x}} \le \tilde{\boldsymbol{b}}_x$ with the usual abuse of notation. Note that the control constraints are already in the wanted form (with $\boldsymbol{w} \triangleq \tilde{\boldsymbol{b}}_u$). Reformulating the state constraint is simply by plugging in the (batched) state dynamics:

$$\tilde{\mathbf{A}}_x \tilde{\boldsymbol{x}} \le \tilde{\boldsymbol{b}}_x \qquad \iff \qquad \tilde{\mathbf{A}}_x(\tilde{\mathbf{A}}\boldsymbol{x}_0 + \tilde{\mathbf{B}}\tilde{\boldsymbol{u}}) \le \tilde{\boldsymbol{b}}_x \qquad \iff \qquad \tilde{\mathbf{A}}_x \tilde{\mathbf{B}}\tilde{\boldsymbol{u}} \le \tilde{\boldsymbol{b}}_x - \tilde{\mathbf{A}}_x \tilde{\mathbf{A}} \boldsymbol{x}_0.$$

Plugging it all together, the constraints are compactly represented as $\tilde{\mathbf{H}}\boldsymbol{u} \le \boldsymbol{w} + \mathbf{E}\boldsymbol{x}_0$ with

$$\tilde{\mathbf{H}} = \begin{bmatrix} \tilde{\mathbf{A}}_u \\ \tilde{\mathbf{A}}_x \end{bmatrix} \qquad \boldsymbol{w} = \begin{bmatrix} \tilde{\boldsymbol{b}}_u \\ \tilde{\boldsymbol{b}}_x \end{bmatrix} \qquad \mathbf{E} = \begin{bmatrix} \mathbf{O} \\ \tilde{\mathbf{A}}_x \tilde{\mathbf{A}} \end{bmatrix}$$

where $\mathbf{O}$ are zeros such that $\mathbf{E}$ has the appropriate size. After all of this, the QP to solve is as simple as

$$\begin{aligned} \min_{\tilde{\boldsymbol{u}}} \ & \tilde{\boldsymbol{u}}^\top \mathbf{H} \tilde{\boldsymbol{u}} + 2\boldsymbol{x}_0^\top \mathbf{F} \boldsymbol{u} + \boldsymbol{x}_0^\top \mathbf{G} \boldsymbol{x}_0 \\ \text{s.t.} \ \ & \tilde{\mathbf{H}} \tilde{\boldsymbol{u}} \le \boldsymbol{w} + \mathbf{E}\boldsymbol{x}_0 \end{aligned} \tag{3.3}$$

with $\mathbf{H} = \tilde{\mathbf{B}}^\top \tilde{\mathbf{Q}} \tilde{\mathbf{B}} + \tilde{\mathbf{R}}$, $\mathbf{F} = \tilde{\mathbf{A}}^\top \tilde{\mathbf{Q}} \tilde{\mathbf{B}}$, and $\mathbf{G} = \tilde{\mathbf{A}}^\top \tilde{\mathbf{Q}} \tilde{\mathbf{A}}$.

3.2.2 Option B: Adding the Dynamics as Constraints

The second approach is to extend the optimization variables by the states,

$$\boldsymbol{z} := \begin{bmatrix} \boldsymbol{x}_1^\top & \boldsymbol{x}_2^\top & \cdots & \boldsymbol{x}_N^\top & \boldsymbol{u}_0^\top & \boldsymbol{u}_1^\top & \boldsymbol{u}_{N-1}^\top \end{bmatrix}^\top$$

and adding the dynamics as an equality constraint. With

$$\mathbf{H}_{\text{eq}} = \begin{bmatrix} \mathbf{I} & & & & -\mathbf{B} & & & \\ -\mathbf{A} & \mathbf{I} & & & & -\mathbf{B} & & \\ & -\mathbf{A} & \mathbf{I} & & & & -\mathbf{B} & \\ & & \ddots & \ddots & & & & \ddots \\ & & & -\mathbf{A} & \mathbf{I} & & & & -\mathbf{B} \end{bmatrix} \qquad \mathbf{E}_{\text{eq}} = \begin{bmatrix} \mathbf{A} \\ \mathbf{O} \\ \vdots \\ \mathbf{O} \end{bmatrix}$$

the dynamics can be expressed as $\mathbf{H}_{\text{eq}}\boldsymbol{z} = \mathbf{E}_{\text{eq}}\boldsymbol{x}_0$ in term of the extended optimization variables $\boldsymbol{z}$. Similarly, the state, terminal, and control constraints can be expressed as $\mathbf{H}_{\text{in}}\boldsymbol{z} \leq \boldsymbol{w} + \mathbf{E}_{\text{in}}\boldsymbol{x}_0$ with

$$\mathbf{H}_{\text{in}} = \begin{bmatrix} \mathbf{O} & \mathbf{O} \\ \tilde{\mathbf{A}}_x & \mathbf{O} \\ \mathbf{O}\tilde{\mathbf{A}}_u & \end{bmatrix} \qquad \boldsymbol{w}_{\text{in}} = \begin{bmatrix} \tilde{\boldsymbol{b}}_x \\ \tilde{\boldsymbol{b}}_u \end{bmatrix} \qquad \mathbf{E}_{\text{in}} = \begin{bmatrix} -\mathbf{A}_x \\ \mathbf{O} \\ \vdots \\ \mathbf{O} \end{bmatrix}$$

and $\tilde{\mathbf{A}}_x$ and $\tilde{\mathbf{A}}_u$ defined as in option A. Finally, with $\bar{\mathbf{H}} = \text{diag}(\tilde{\mathbf{Q}}, \tilde{\mathbf{R}})$, the QP is as follows:

$$\begin{aligned} \min_{\boldsymbol{z}} \ & \boldsymbol{z}^\top \bar{\mathbf{H}} \boldsymbol{z} \\ \text{s.t.} \quad & \mathbf{H}_{\text{eq}}\boldsymbol{z} = \mathbf{E}_{\text{eq}}\boldsymbol{x}_0 \\ & \mathbf{H}_{\text{in}}\boldsymbol{z} \leq \mathbf{E}_{\text{in}}\boldsymbol{x}_0 \end{aligned} \tag{3.4}$$

A comparison of the two methods is given in the next section.

3.2.3 Comparison of A and B

Both presented versions, (3.3) and (3.4), can be efficiently solved by QP solvers. The main difference is that option A has Nm variables (with m input dimensionality) and option B has $N(n+m)$ variables (with n being the state dimensionality). While it seems that option A must be more efficient, option B has sparse cost and constraint functions while option A is dense. Note also that option B has the advantage that it can easily be applied to nonlinear MPC.

4 Robust Model Predictive Control

4.1 Inherent Robustness of Nominal MPC

4.2 Uncertain Models

4.2.1 System Evolution

4.3 Cost Functions for Uncertain Systems

4.4 Minimax MPC

4.5 Set Subtraction and Addition

4.6 Robust Open-Loop MPC

4.7 Tube MPC

5 Stochastic Model Predictive Control

5.1 Stochastic Uncertainty

5.2 Uncertain System Evolution

5.3 Chance Constraints

5.4 Stochastic Tube MPC

5.5 Outlook

6 Machine Learning in Model Predictive Control

6.1 Machine Learning for MPC

6.1.1 Modeling Dynamical Systems with ML

Implications

Safe Sets

Robust Learning Supported Tube MPC

6.1.2 Modeling External Signals

6.1.3 Modeling Constraints

6.1.4 Modeling Cost Functions

6.1.5 Learning Control Input: Replacing MPC

6.2 Gaussian Processes

6.2.1 Setup and Definition

6.2.2 GP Regression

Prior Distribution

Mean Function

Covariance Function

Posterior Distribution

Noise Observations

6.2.3 Hyper-Parameter Learning

Influence of the Hyper-Parameters

Automatic Relevance Determination

6.2.4 Dynamic Process Models

6.2.5 Benefits and Drawbacks

6.3 (Artificial) Neural Networks

6.3.1 Setup

6.3.2 Signal Direction

6.3.3 Connections Between Nodes

6.3.4 Activation Functions

6.3.5 Training

Optimization Algorithm

Challenges

6.3.6 Neural Networks in MPOC

6.3.7 Benefits and Drawbacks

6.4 Remarks on Nomenclature

7 Outlook

7.1 Libraries for Machine Learning

7.2 Reinforcement Learning vs. MPC