UNIVERSITÄT
BAYREUTH

Receding Horizon Control

A Suboptimality–based Approach

Von der Universität Bayreuth
zur Erlangung des akademischen Grades eines

Doktor der Naturwissenschaften (Dr. rer. nat.)

genehmigte Abhandlung

vorgelegt von

Jürgen Pannek

aus Coburg

1. Gutachter:	Prof. Dr. Lars Grüne
2. Gutachter:	Prof. Dr. Anders Rantzer

Tag der Einreichung:	09. Juli 2009
Tag des Kolloquiums:	13. November 2009

Danksagung

Diese Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Lehrstuhl Mathematik V der Universität Bayreuth und als Projektmitarbeiter im DFG Schwerpunktprogramm 1305 “Development of Asynchronous Predictive Control Methods for Digitally Networked Systems”.
In erster Linie möchte ich meinem Betreuer, Prof. Dr. Lars Grüne, für seine Unterstützung und Anleitung während dieser Zeit danken. Seine Ideen waren immer Anregung und Bereicherung für meine Arbeit, und er schuf ein angenehmes und für neue Ideen offenes Arbeitsumfeld, das mir bei meinem wissenschaftlichen Werdegang sehr zugute kam. Weiterhin möchte ich Prof. Dr. Frank Lempio danken, der meine Arbeit durch Bereitstellung einer Doktorandenstelle ermöglicht hat. Für seinen wohlverdienten Ruhestand wünsche ich ihm alles Gute und weiterhin viel Lebens- und Forschungsfreude.
An dieser Stelle möchte ich auch meinen Kollegen am Lehrstuhl danken, insbesondere Karl Worthmann, mit dem ich immer produktiv über Theorie und Praxis diskutieren konnte, aber auch Marcus von Lossow, der nun seit einiger Zeit schon meine Stelle im Lehrbetrieb übernommen und mich dadurch sehr entlastet hat. Ein großer Zugewinn für mich war Thomas Jahn, ohne dessen Mithilfe meine Programmierarbeit kaum in der Form vonstatten hätte gehen können. Besonders wichtig, quasi die Seele des Lehrstuhls, ist Frau Dulleck, die zwar nicht verstehen kann, wie man sich freiwillig mit Mathematik beschäftigen kann, die aber immer mit einem Lächeln zugegen war und viele meiner organisatorischen Arbeiten übernommen hat.
Auch einige meiner Studenten möchte in an dieser Stelle erwähnen, die ich während meiner Tätigkeit an der Universität Bayreuth betreuen durfte und weder dies noch die Zeit mit ihnen missen möchte: Rahel Berkemann, Maria Brauchle, Annika Grötsch, Stefan Trenz, Sabine Weiner (Diplomanden), Tobias Bauerfeind, Michael Bodenschatz, Thomas Höllbacher, Anja Kleinhenz, Matthias Rodler, Harald Voit (Praktikanten) sowie Florian Häberlein und Marleen Stieler (Studenten).

Last but not least möchte ich mich bei meiner zukünftigen Frau Sabina Groth bedanken, die nicht müde wurde mich zu unterstützen und zum Weiterarbeiten zu motivieren.

Bayreuth, den 19. November 2009

Jürgen Pannek

Contents

Deutsche Zusammenfassung

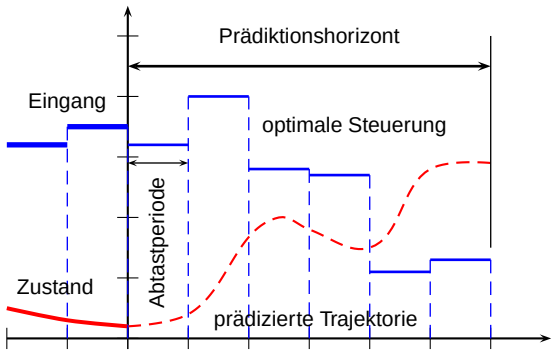
Einführung

Stabilisierung eines vorgegebenen Gleichgewichts oder die Verfolgung einer Referenzbahn ist eine der häufigsten Zielsetzungen in der Regelungstheorie. Bei vielen der dabei betrachteten Prozesse kommt hierzu noch die Einhaltung verschiedenartigster Beschränkungen sowie die Forderung, das gegebene Ziel möglichst gut zu erreichen und mit Hilfe von digitaler Technik umzusetzen. Eine Methode, die diese Vorgaben erfüllen kann und mittlerweile auch weit verbreitet ist, siehe z.B. [16], ist die sogenannte modellprädiktive Regelung, die im Englischen auch *model predictive control* (MPC) oder *receding horizon control* (RHC) genannt wird.

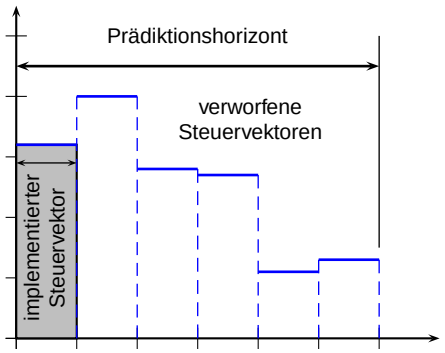
Der Grundgedanke dieser Regelungsmethode ist auch im menschlichen Handeln wiederzufinden, beispielsweise bei der Lebensplanung oder einer Autofahrt zum Supermarkt. Beide Probleme sollen dabei bezüglich einer individuellen Beurteilung, etwa möglichst sicher oder möglichst schnell, geplant und umgesetzt werden. Zudem sollen aktuelle Umstände und auftretende Hindernisse beachtet werden. Die Umsetzung der ursprünglichen Ziele erfolgt aber in der Regel nicht oder zumindest nicht genau so, wie es geplant war. Grund hierfür ist, dass jeder Mensch einen individuellen Planungshorizont beachtet, frühere Planungen von Zeit zu Zeit überdenkt und die verfolgte Strategie anpasst. So kann es z.B. vorkommen, dass man bei der Neuplanung feststellt, dass auf Grund der aktuellen Umstände die alte Planung nicht mehr umsetzbar oder auch nicht mehr optimal ist, in den betrachteten Beispielen etwa durch sich ändernde Lebensumstände oder eine Baustelle, die man zuvor nicht bedacht hat. Dieser Prozess der Ausführung der zum Planungszeitpunkt optimalen Strategie und die Anpassung dieser Strategie wiederholt sich während der gesamten Dauer des Problems.

Abstrahiert man dies von den angegebenen Beispielen, so ergibt sich die Grundstruktur eines derartigen Reglers, die aus den drei folgenden Schritten besteht:

Schritt 1: Auf Basis eines Modells des zu kontrollierenden Prozesses wird über einen endlichen Zeithorizont eine Vorhersage der Entwicklung des Zustandes dieses Systems getroffen. Mit Hilfe eines sogenannten Ziel- oder auch Kostenfunktionals wird dieser Entwicklung, auch Trajektorie genannt, sowie der verwendeten Steuerung ein bestimmter Kostenwert zugeordnet. Nun wird für dieses Funktional unter Einhaltung problemin-

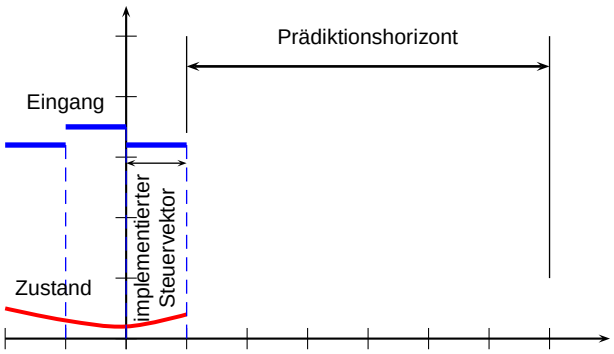


härenter Beschränkungen eine Steuerung bestimmt, die auf dem betrachteten Zeithorizont minimale Kosten verursacht.



Schritt 2: Die resultierende Steuerfunktion liegt bereits in digitaler Form vor, das heißt es handelt sich um eine stückweise konstante Funktion, die zu den vorgegebenen digitalen Schaltpunkten Sprünge aufweisen kann. Diese Funktion ist allerdings nur für den betrachteten Optimierungshorizont definiert, somit also für eine Implementierung auf unendlichem Horizont ungeeignet. Nun wird lediglich das erste Steuerelement dieser Funktion, die sich auch als Folge von Steuervektoren darstellen lässt, am zu kontrollierenden Prozess implementiert. Alle weiteren Steuervektoren werden verworfen.

Schritt 3: Anschließend wird der sich durch die Systemdynamik und des implementierten Steuerelements ergebende Zustand des Prozesses zum darauffolgenden Abtastzeitpunkt gemessen und an den Regler übergeben. Durch Verschiebung des internen Prädiktionshorizonts um ein Abtastintervall kann somit der beschriebene Prozess wiederholt werden. Eine iterative Anwendung dieser Schritte ergibt eine sogenannte geschlossene Regelkette.



Damit gehört die modellprädiktive Regelung zur Klasse der modell–basierten Regelungsverfahren, die im Gegensatz zu herkömmlichen Verfahren wie etwa PID [147, 148, 222, 223] oder adaptiven Reglern [72, 129, 132] das Regelgesetz nicht ausschließlich auf Basis des aktuellen oder vergangener Zustände entwirft.

In der Literatur unterscheidet man lineare und nichtlineare modellprädiktive Regelung. Im linearen Fall wird dabei versucht, die Lösung des linearen Modells der Systemdynamik so zu manipulieren, dass zum einen die linearen Beschränkungen erfüllt werden, zum anderen aber auch ein gewähltes quadratisches Kostenfunktional minimiert wird. Die theoretischen Grundlagen solcher Regler gelten als weitestgehend erschlossen [160, 167] und auch in der industriellen Anwendung sind Implementierungen mittlerweile weit verbreitet [16, 54].

Im Gegensatz zum linearen Fall sind viele der theoretischen Grundlagen nichtlinearer modellprädiktiver Regler noch nicht ausreichend erforscht. Hierzu zählen unter anderem die Robustheitsanalyse derartiger Verfahren gegenüber Störungen sowie die Entwicklung ausgangsbasierter modellprädiktiver Regler. Ein Überblick zu bisherigen Ergebnissen in diesem Bereich findet sich in [5, 39, 49, 160, 167]. Da das Hauptaugenmerk dieser Arbeit jedoch auf den Stabilitäts– und Suboptimalitätsaspekt sowie einer effizienten Implementierung liegt, sei insbesondere auf die Arbeiten [4, 39, 87, 102, 126] und [53, 58] verwiesen.

Betrachtete Systeme

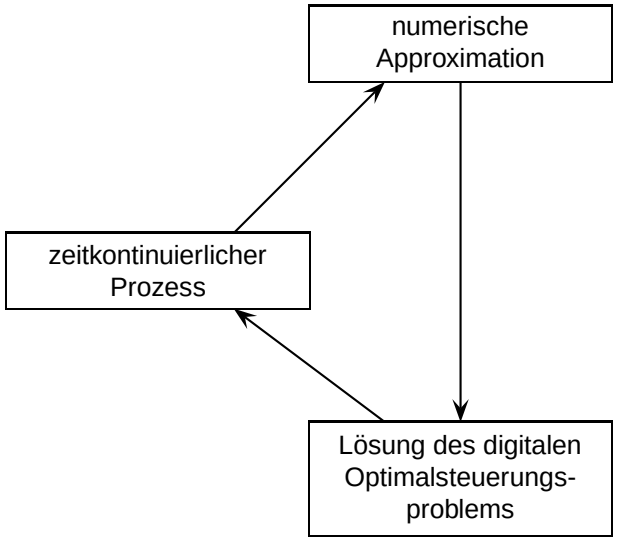
In dieser Arbeit werden unterschiedliche Arten von Kontrollsystemen betrachtet. In der theoretischen Analyse des modellprädiktiven Regelungsverfahrens werden zeitdiskrete Systeme der Form

$$x(n+1) = f(x(n), u(n))$$

verwendet, wobei der Zustand x und die Steuerung u Elemente aus beliebigen metrischen Räumen $\mathbb{X}$ bzw. $\mathbb{U}$ sind. Jedoch sind die betrachteten Beispiele – wie auch reale Anwendungen des Reglers – in Form von zeitkontinuierlichen Modellen

$$\dot{x}(t) = f(x(t), u(t))$$

gegeben. Da die Aufgabenstellung eine digitale Umsetzung der Steuerung verlangt, d.h. dass eine Umsetzung eines Steuerwertes lediglich zu fest vorgegebenen äquidistanten Zeitpunkten möglich ist, sind lediglich stückweise konstante Steuerfunktionen implementierbar. Durch die äquidistanten Umschaltpunkte wird das sogenannte Abtastgitter $\mathbb{T}$ definiert.



Um auch zeitkontinuierliche Systeme mit Hilfe des für zeitdiskrete Systeme entworfenen Reglers stabilisieren zu können, wird die Lösung des zeitkontinuierlichen Systems auf das Abtastgitter $\mathbb{T}$ restringiert. Dadurch erhält man ein sogenanntes digitales Kontroll- oder Abtastsystem der Form

$$x_T(n+1) = f(x_T(n), u(n)).$$

In diesem Zusammenhang stellt $f(\cdot,\cdot)$ die Lösung eines zeitkontinuierlichen Modells zum $(n+1)$sten Abtastzeitpunkt dar. Mit $x_T(n)$ und $u(n)$ sind sowohl Anfangswerte für den $n$ten Abtastzeitpunkt als auch die stückweise konstante Steuerung gegeben, wodurch, solange die Bedingungen des Satzes von Caratheodory erfüllt sind, eine Lösung des zeitkontinuierlichen Systems definiert ist, siehe z.B. [210]. Diese Umwandlung eines zeitkontinuierlichen in einen zeitdiskreten Prozess erlaubt es, die benötigte Steuerung ohne Stabilitätsverlust für das zeitdiskrete System zu berechnen [170], es aber in einem zeitkontinuierlichen Prozess zu implementieren [171].

Beitrag der Arbeit

Die Herleitung hinreichender *a posteriori* und *a priori* Kriterien für die Stabilität der resultierenden geschlossenen Regelkette stellt das Kernelement dieser Arbeit dar. Hierbei werden die aus der Literatur bekannten Modifikationen des Ursprungsproblems wie zusätzliche Endpunktbeschränkungen [126] oder Erweiterungen des Kostenfunktionals um lokale Endgewichtsfunktionen [38, 39, 160] explizit nicht verwendet, sondern direkt die industriell genutzte Klasse modellprädiktiver Regler betrachtet, vgl. [16].
Diese Kriterien erlauben zudem eine Abschätzung der Regelgüte im Vergleich zum bestmöglichen (aber praktisch kaum berechenbaren) Regler mit unendlichem Zeithorizont. Die zugehörige Kenngröße wird als *Suboptimalitätsgrad* bezeichnet. Zwar sind Suboptimalitätsabschätzungen aus der Literatur bekannt, siehe etwa [22, 150, 160, 173], jedoch erlaubt die hergeleitete Abschätzung eine quantifizierende Aussage über den Verlust der Regelgüte, die durch die Endlichkeit des betrachteten Zeithorizonts entsteht.
Zur Berechnung des Suboptimalitätsgrades werden zudem Algorithmen vorgestellt, die *zur Laufzeit des modellprädiktiven Regelungsalgorithmus auswertbar* sind und keinen oder

vergleichsweise geringen Mehraufwand aufweisen. Die dargestellten Resultate wurden in den bereits veröffentlichten Artikeln [97, 98] beschrieben. In der vorliegenden Arbeit werden diese jedoch detailliert bewiesen und um entsprechende algorithmische Umsetzungen sowie das Konzept verschiedener Suboptimalitätsgrade erweitert. Zudem wird die Anwendbarkeit der vorgestellten Algorithmen in numerischen Beispielen nachgewiesen.

Im Weiteren werden diese Abschätzungen verwendet, um *adaptive modellprädiktive Regelungsverfahren* zu entwickeln. Hierzu werden die zur Rechenzeit auswertbaren Suboptimalitätsabschätzungen herangezogen, um iterativ eine Horizontlänge zu bestimmen, die eine untere Schranke für die Regelgüte garantiert. Ziel der hierzu entwickelten Algorithmen ist dabei sowohl eine schnelle, aber auch mit möglichst geringem Mehraufwand behaftete Anpassung des gewöhnlichen modellprädiktiven Regelungsverfahrens. Bei der numerischen Untersuchung dieser Algorithmen zeigt sich, dass dieser Ansatz deutliche Verbesserungen der Rechenzeit mit sich bringt.

Der praktische Teil dieser Arbeit umfasst das Softwarepaket *PCC2*[1] (**P**redictive **C**omputed **C**ontrol 2), das sowohl eine numerisch effizient gestaltete *Implementierung* eines gewöhnlichen wie auch eines adaptiven modellprädiktiven Regler enthält. Hierzu wird das modulare Konzept dieser Implementierung vorgestellt sowie Interaktion und Problemanpassungen der Teilalgorithmen von theoretischer und praktischer Seite her analysiert. Die präsentierte Implementierung wurde in den bereits veröffentlichten Artikeln [93–96, 99] zur Lösung von numerischen Beispielen verwendet, wird im Rahmen dieser Arbeit allerdings zum ersten Mal mit allen Erweiterungen vorgestellt.

Gliederung der Arbeit

Entsprechend der Dreiteilung der Beiträge dieser Arbeit gliedert sich auch deren Darstellung in drei Abschnitte. Hierbei beinhalten die Kapitel 1 – 4 Grundlagen und Konzept der modellprädiktiven Regelung sowie die theoretischen Resultate. In den Kapiteln 5 und 6 werden genutzte numerische Algorithmen vorgestellt und die entwickelte Software detailliert beschrieben. Der letzte Teil der Arbeit, zusammengefasst in Kapitel 8, beinhaltet zum einen die numerischen Untersuchungen der Implementierung selbst, aber auch des Zusammenspiels der verschiedenen Teilkomponenten sowie numerische Ergebnisse der vorgestellten theoretischen Resultate. Hierzu werden die in Kapitel 7 angegebenen Beispiele verwendet, die den Anforderungen der numerischen Untersuchungen entsprechend gewählt sind.
Im Einzelnen beinhalten die Kapitel dabei Folgendes:

- In Kapitel 1 werden Grundbegriffe der Systemtheorie eingeführt und die betrachteten zeitdiskreten und zeitkontinuierlichen Systeme formal definiert. Da der theoretische Teil der Arbeit auf der Verwendung zeitdiskreter Kontrollsysteme basiert, wird zudem die Verbindung der zeitkontinuierlichen Systeme mit der digitalen Implementierung der zu berechnenden Regelung zu dem Begriff eines digitalen Kontrollsystems verschmolzen. Im weiteren Verlauf wird der Begriffe der Stabilität eines dynamischen Systems auf Stabilisierbarkeit und semiglobal praktische Stabilisierbarkeit von Kontroll- und digitalen Kontrollsystemen erweitert. Hierfür wird die

[1] `http://www.nonlinearmpc.com`

Äquivalenz der drei gebräuchlichen Charakterisierungen, d.h. der ε-δ Definition sowie der Darstellung mittels Vergleichsfunktionen und Kontroll–Lyapunov Funktionen, gezeigt. Abschließend wird zudem nachgewiesen, dass unter entsprechenden Voraussetzungen ein für ein approximiertes zeitdiskretes Kontrollsystem berechnetes Rückkopplungsgesetz im zugrunde liegenden kontinuierlichen Prozess umgesetzt und dennoch die Stabilität des geschlossenen Regelkreises garantiert werden kann.

- Im folgenden Kapitel 2 wird einleitend ein kurzer Überblick über die Entwicklung der Steuerungs- und Regelungstheorie gegeben. Dies wird genutzt, um ausgehend von einem kontinuierlichen optimalen Steuerungsproblem auf unendlichem Zeithorizont zunächst das digitale Gegenstück mit unendlichem Zeithorizont zu definieren. Dieses Problem ist zugleich der Maßstab, an dem wir in Kapitel 3 die Güte des modellprädiktiven Regler messen. Zunächst wird jedoch die Problemstellung eines modellprädiktiven Reglers sowie die Lösungsbegriffe der offenen und geschlossenen Regelkette formal definiert. Am Ende dieses Kapitels wird zudem ein Vergleich zwischen dem modellprädiktiven Regler und den Alternativen des PID Reglers, des Lyapunov basierten Reglers sowie des adaptiven Reglers gezogen.

- In Kapitel 3 werden diverse a posteriori und a priori Suboptimalitätsabschätzungen für modellprädiktive Regler entwickelt und entsprechende Berechnungsalgorithmen vorgestellt. In diesem Zusammenhang wird außerdem das Konzept verschiedener Suboptimalitätsgrade eingeführt. Die Hauptvorteile dieser gegenüber aus der Literatur bekannter Abschätzungen sind, dass sie zur Laufzeit des Algorithmus ausgewertet werden können und zudem eine quantifizierende Abschätzung des maximalen Verlusts gegenüber dem Regler mit unendlichem Zeithorizont, also dem bestmöglichen Regler, erlauben. Zudem wird gezeigt, dass diese Abschätzungen auf den Fall der praktischen Stabilität erweiterbar sind. Weiterhin ermöglichen diese Abschätzungen eine Stabilitätanalyse industriell gebräuchlicher modellprädiktiver Regler, da sie ohne die aus der Literatur bekannten Modifikationen der Problemstellung auskommen. Ein entsprechender Vergleich mit älteren Stabilitäts- und Suboptimalitätsresultaten bildet dabei den Abschluss dieses Kapitels.

- Kapitel 4 widmet sich der Nutzung der Suboptimalitätsabschätzungen aus Kapitel 3, um die starre Problemformulierung des modellprädiktiven Reglers anzupassen. Hierbei werden zwei komplementäre Ziele verfolgt, die aus praktischer Sicht jedoch eine eindeutige Reihenfolge aufweisen: Stabilität und Laufzeit. Der freie Parameter ist dabei die Horizontlänge, die als Vielfaches der Abtastzeit des digitalen Systems auf beide Ziele maßgeblichen Einfluss hat. Mit Hilfe der Suboptimalitätsabschätzungen werden Algorithmen entwickelt und bewiesen, die die Stabilität des geschlossenen Regelkreises garantieren und gleichzeitig den benötigten Rechenaufwand möglichst minimal halten. Zudem wird in diesem Zusammenhang das Konzept der Suboptimalitätsgrade auf den adaptiven modellprädiktiven Regler erweitert und entsprechende Abschätzungen für Stabilität und praktische Stabilität bewiesen.

- Kapitel 5 widmet sich der Theorie der Diskretisierungs- und Optimierungstechniken, die in der Implementierung eines im Verlauf dieser Arbeit entstandenen modellprädiktiven Reglers Verwendung finden. Hierzu werden die vollständige und rekursive Diskretisierung sowie die rekursive Diskretisierung mit Mehrzielknoten formal definiert und die Konsequenzen dieser Methoden für den modellprädiktiven Regler analysiert. Desweiteren werden Grundlagen der nichtlineare Optimierung vorge-

stellt, sowie Modifikationen und Unterschiede der verwendeten Routinen diskutiert und Auswirkungen auf den modellprädiktiven Regelalgorithmus aufgezeigt.

- In Kapitel 6 wird die Implementierung des entwickelten modellprädiktiven Reglers beschrieben und untersucht. Das Kapitel kann als grundlegende Einführung in das Programmpaket verstanden werden. Dabei bietet es nicht nur eine Übersicht über die wichtigsten enthaltenen Funktionen, sondern zeigt auch das Zusammenspiel der verschiedenen Algorithmen. Hierbei wird insbesondere auf die Optimierungsalgorithmen, die Methoden zur Lösung der zugrunde liegenden Systemdynamik und die notwendigen Verbindungskomponenten in dem hierarchisch und modular gestaltete Implementierungskonzept des Programmpakets eingegangen.

- Die abschließenden Kapitel 7 und Kapitel 8 beinhalten Untersuchungsbeispiele und Ergebnisse des implementierten Regelungsverfahrens. Dabei sind die Beispiele in Kapitel 7 so gewählt, dass hiermit drei grundlegende Fragen analysiert werden können. Das erste Beispiel, eine eindimensionale Wärmeleitungsgleichung, dient dazu, den Einfluss von Systemeigenschaften wie Größe und Steifheit auf die Leistungsfähigkeit einzelner Komponenten des Regelalgorithmus zu testen. Weiter wird das bekannte invertierte Pendel betrachtet, das es erlaubt die Parameter des modellprädiktiven Reglers und deren Wechselwirkungen zu veranschaulichen. Diese Untersuchung des Regelungsproblems, die aufgrund der Komplexität des Algorithmus nicht vollständig darstellbar sein kann, versteht sich als Anleitung zur Anpassung eines modellprädiktiven Reglers für andere Probleme. Zuletzt wird ein Folgeproblem verwendet, um Standardsituationen eines modellprädiktiven Reglers zu generieren. Dies erlaubt eine genaue Untersuchung der theoretischen Ergebnisse aus den Kapiteln 3 und 4. Hierbei wird die Anwendbarkeit der Suboptimalitätsabschätzungen gezeigt sowie ein Vergleich zwischen einem gewöhnlichen modellprädiktiven Regler und einem adaptiven modellprädiktiven Regler gezogen, wobei die Vorteile der vorgestellten Algorithmen deutlich werden. Abschließend werden die Ergebnisse dieser Arbeit kurz zusammengefasst und ein Ausblick auf mögliche weitere Forschungen gegeben.

Summary

Within the proposed work we consider analytical, conceptional and implementational issues of so called receding horizon controllers in a sampled–data setting. The principle of such a controller is simple: Given the current state of a system we compute an open–loop control which is optimal for a given costfunctional over a fixed prediction horizon. Then, the control is implemented on the first sampling interval and the basic open–loop optimal control problem is shifted forward in time which allows for a repeated evaluation.
The contribution of this thesis is threefold: First, we prove estimates for the performance of a receding horizon control, a concept which we call *suboptimality degree*. These estimate are online computable and can be applied for stabilizing as well as practically stabilizing receding horizon control laws. Moreover, they not only allow for guaranteeing stability of the closed–loop but also for quantifying the loss of performance of the receding horizon control law compared to the infinite horizon control law. Based on these estimates, we introduce *adaptation strategies* to modify the underlying receding horizon controller in order to guarantee a certain lower bound on the suboptimality degree while reducing the computing cost/time necessary to solve this problem. Within this analysis, the length of the optimization horizon is the parameter we wish to adapt. To this end, we develop and proof several shortening and prolongation strategies which also allow for an effective implementation. Moreover, extensions of our suboptimality estimates to receding horizon controllers with varying optimization horizon are shown. Last, we present details on our implementation of a receding horizon controller *PCC2*[2] (**P**redictive **C**omputed **C**ontrol 2) which is on the one hand computationally efficient but also allows for easily incorporating our theoretical results. Since a full analysis of such a controller would exceed the scope of this work, we focus on the main aspects of this algorithm using different examples. In particular, we concentrate on the impact of certain choices of parameters on the computing time. We also consider interactions between these parameters to give a guideline to effectively implement and solve further examples. Moreover, we show applicability and effectiveness of our theoretical results using simulations of standard problems for receding horizon controllers.

[2] `http://www.nonlinearmpc.com`

Chapter 1

Mathematical Control Theory

Mathematical control theory is an application–oriented area of mathematics which deals with the basic principles underlying the analysis and design of *control systems*. In this context the term *control system* is used in a very general way:

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

The term *control* is used for the influence a certain external action — the *input* to a system — has on the behavior of a system in order to achieve a certain goal. The state of a system which can be sensed in any way from outside is called the *output* of a system. Coming back to the input, the information obtained from the output can be used to check whether the objective is accomplished.
Commonly such a system is visualized using a block diagram. Within such a diagram processing units are represented by boxes while assignments are shown as arrows indicating the direction of the signal.

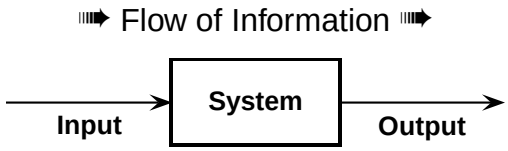
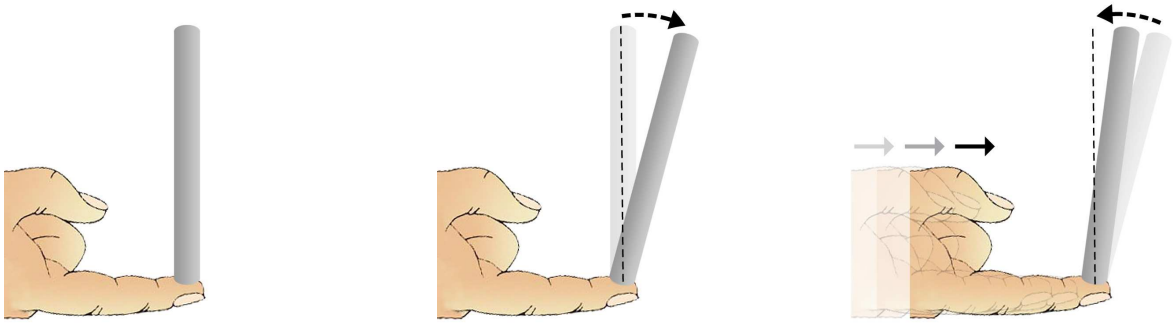


Figure 1.1: Schematic representation of a control system with input and output parameter

A standard example in control theory is the inverted pendulum, i.e. a pendulum which one seeks to stabilize in the (unstable) upright position, see also Figures 1.2a–1.2c. Since it is simple enough to intuitively understand its behaviour, we refer to this example throughout this thesis. Additionally, we can check subjectively/heuristically whether a control law is reasonable and accurate for this example using our physical intuition. Unlike Figures 1.2a–1.2c may indicate, we aim at developing a control law which stabilizes the underlying example not only *locally*, but for a large set of *initial values*. That is, for the inverted pendulum, we consider initial positions far away from the upright position, e.g. the stable downward position.
For general systems, however, we focus on the following aspects:

- Developing a notation describing the long–term behavior and properties of a system
- Designing methods to calculate control laws depending on this analysis
- Giving insight to the acchievable goals using these methods



(a) Inverted Pendulum in the upright position which shall be stabilized

(b) Inverted Pendulum deviating away from the upright position

(c) Counteraction to bring the Pendulum back into the upright position

In particular, we focus on (sub–)optimal digital control by studying the so called *receding horizon control* approach and its interaction with the system under control. By now, theory of receding horizon control has grown rather mature at least for the linear case, see e.g. [69, 140, 160, 167]. As shown in [15, 16], methods to compute such a control are widely used. However, this field is still very active as it is not always clear why these methods actually work out fine. Moreover, the problem of finding a (sub–)optimal digital control is not artificial but application–oriented and therefore has to be solved according to fixed technological bounds. Using E.D. Sontags words [210]:

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

In control theory the questions to be asked are clear. However, this does not hold for the mathematical model which shall be used in the context of digital control, that is whether to use *differential* or *difference equations*. In the literature, both models are used at present, see e.g. [4, 39, 50, 59, 68, 117] and [49, 87, 91, 102, 160, 167] for receding horizon control settings using differential and difference equations respectively. Here, we consider so called *sampled–data systems* — a mixture of both concepts. In particular, our examples are given as differential equations, that is in *continuous–time*, whereas the numerical implementation as well as the analysis of our controller design relies on *discrete–time* systems, i.e. using difference equations. The aim of this chapter is to rigorously define these two models.

To this end, we introduce fundamental concepts and terminology which both discrete and continuous–time systems have in common in Section 1.1. In particular we give a general definition of a control system, its inputs and its outputs. In Section 1.2 these terms are specified for the control systems in both continuous and discrete–time. Moreover we show their relation in terms of digital control. Last, in Section 1.3 we characterize the stability concept which we consider to be the desireable property of the system under control. The development of such a control law will be outlined in the following Chapter 2.

1.1 Basic Definitions

The fundamental difference between discrete–time and continuous–times systems is characterized by the treatment of time. To handle both within one concept we first introduce the notion of a time set.

Definition 1.1 (Time set)
A *time set* $\mathbb{T}$ is a subgroup of $(\mathbb{R}, +)$.

Later we shall either consider $\mathbb{T} = \mathbb{Z}$ or $\mathbb{T} = \mathbb{R}$ for discrete and continuous–time systems respectively. Using this time concept we think of a state of a system at a certain time in the time set $\mathbb{T}$ as an element of some set which may change to another element according to some internal scheme and external force. To state this formally we require some more definitions:

Definition 1.2 (State and Control Value Space)
The set of all maps from an interval $I \subset \mathbb{T}$ to a set $\mathbb{U}$ is denoted by $\mathbb{U}^I = \{u \mid u : I \to \mathbb{U}\}$ and called the *set of control functions*. We refer to $\mathbb{U}$ as the *control value* or *input value space*.
Moreover, the set $\mathbb{X}$ denotes the *state space*.

For our later purposes we think of $\mathbb{X}$ being a subset of some metric space.

Definition 1.3 (Transition map)
A *transition map* is a map $\varphi : \mathbb{D}_\varphi \to \mathbb{X}$ where

$$\mathbb{D}_\varphi \subset \left\{(\tau, \sigma, x, u) \mid \sigma, \tau \in \mathbb{T}, \sigma \leq \tau, x \in \mathbb{X}, u \in \mathbb{U}^{[\sigma,\tau)}\right\}$$

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

Definition 1.4 (Admissability)
Given time instances $\tau, \sigma \in \mathbb{T}$, $\sigma < \tau$, a *control* $u \in \mathbb{U}^{[\sigma,\tau)}$ is called *admissible* for a *state* $x \in \mathbb{X}$ if $(\tau, \sigma, x, u) \in \mathbb{D}_\varphi$.

Using these definitions we introduce a system we aim to analyze.

Definition 1.5 (System)
A tupel $\Sigma = (\mathbb{T}, \mathbb{X}, \mathbb{U}, \varphi)$ is called *system* if the following conditions hold:

- For each state $x \in \mathbb{X}$ there exists at least two elements $\sigma, \tau \in \mathbb{T}$, $\sigma < \tau$, and some $u \in \mathbb{U}^{[\sigma,\tau)}$ such that u is admissible for x. *(Nontriviality)*
- If $u \in \mathbb{U}^{[\sigma,\mu)}$ is admissible for x then for each $\tau \in [\sigma, \mu)$ the restriction $u_1 := u|_{[\sigma,\tau)}$ of u to the subinterval $[\sigma, \tau)$ is also admissible for x and the restriction $u_2 := u|_{[\tau,\mu)}$ is admissible for $\varphi(\tau, \sigma, x, u_1)$. *(Restriction)*
- Consider $\sigma, \tau, \mu \in \mathbb{T}$, $\sigma < \tau < \mu$. If $u_1 \in \mathbb{U}^{[\sigma,\tau)}$ and $u_2 \in \mathbb{U}^{[\tau,\mu)}$ are admissible and x is a state such that $\varphi(\tau, \sigma, x, u_1) = x_1$, $\varphi(\mu, \tau, x_1, u_2) = x_2$, then the *concatenation*

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

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

Definition 1.6 (System with Outputs)
If Σ is a system and additionally there exist a set $\mathbb{Y}$ called *measurement–value* or *output–value space* and a map $h : \mathbb{X} \to \mathbb{Y}$ called *measurement* or *output map*, then $\Sigma = (\mathbb{T}, \mathbb{X}, \mathbb{U}, \varphi, \mathbb{Y}, h)$ is called a *system with outputs*.

Comparing this definition of a system to the reality of a plant, we identify $x \in \mathbb{X}$ as the *state* of the plant, e.g. the angle of the inverted pendulum and its velocity. Note that the state is only a snapshot. The transition map allows us to predict future states $x(t) \in \mathbb{X}$ for all $t \in \mathbb{T}$ telling us how the system evolves. The *control* or *input values* $u \in \mathbb{U}$ are exogenous variables and can be used to manipulate the future development of the plant. Last, $y \in \mathbb{Y}$ represent *measurement* or *output values*. Throughout this thesis, we deal with the case of all states being measurable, that is $\mathbb{Y} = \mathbb{X}$ and h is bijective.
Definition 1.6 also allows for undefined transitions, i.e. when the input u is not admissible for the given state. While for differential/difference equations this phenomenon is called a finite escape time, it might correspond to a blowup of the plant in reality. Hence, we only consider those tupel (τ, σ, x, u) such that u is admissible for x. Whenever the time instances τ and σ are clear from the context we also refer to the pair (x, u) as the *admissible pair*.
Given an initial state $x(\sigma) = x_0$ and a control function $u(\cdot) \in \mathbb{U}^{[\sigma,\tau)}$ we can fully describe the development of the state in time and call the resulting solution a trajectory.

Definition 1.7 (Trajectory)
Given a system Σ, an interval $I \subseteq \mathbb{T}$ and a control $u(\cdot) \in \mathbb{U}^I$ we call $x \in \mathbb{X}^I$ a *trajectory* on the interval I if it satisfies

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

In order to analyze the long–time behaviour of a system, we need to consider time tending to infinity.

Definition 1.8 (Infinite Admissability)
Given a system Σ and a state $x \in \mathbb{X}$, an element of $\mathbb{U}^{[\sigma,\infty)}$ is called *admissible* for x if every restriction $u|_{[\sigma,\tau)}$ is admissible for x and each $\tau > \sigma$.

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

1.2 Control Systems

The following section deals with continuous–time and discrete–time systems. Throughout this thesis we use different time sets, in particular we consider $\mathbb{T} = \mathbb{N}_0$ for analytical purposes while all our examples are continuous in time, that is $\mathbb{T} = \mathbb{R}$. Since there exist fundamental differences between these two settings, we introduce *continuous–time* and *discrete–time systems* separately. Moreover, we define the notion of a *sampled–data system*. For the receding horizon control scheme stated in Chapter 2, the sampled–data concept allows us to treat continuous–time examples in a discrete–time setting.

Remark 1.9
In the following, the set $\mathcal{U} := \{u : \mathbb{T} \to \mathbb{U}\}$ *denotes the set of all controls. Moreover, we consider* $\mathbb{X}$ *and* $\mathbb{U}$ *to be subsets of* $\mathbb{R}^n$ *and* $\mathbb{R}^m$, $m, n \in \mathbb{N}$, *respectively.*

Definition 1.10 (Discrete–time Control System)
Consider a function $f : \mathbb{X} \times \mathbb{U} \to \mathbb{X}$. A system of n difference equations

$$x_u(i+1) := f(x_u(i), u(i)), \qquad i \in \mathbb{N}_0 \tag{1.1}$$

is called a *discrete–time control system*. Moreover $x_u(i) \in \mathbb{X}$ is called *state vector* and $u(i) \in \mathbb{U}$ control vector.

Existence and uniqueness of a solution of (1.1) is shown fairly easily. Using an induction, we obtain a unique solution in positive time direction for a certain maximal existence interval I if (x_0, u) with $x_0 \in \mathbb{X}$ and $u \in \mathbb{U}^I$ is an admissible pair.
If the system (1.1) is independent of the control u then it is called dynamical system:

Definition 1.11 (Discrete–time Dynamical System)
Consider a function $f : \mathbb{X} \to \mathbb{X}$. The system of n difference equations

$$x(i+1) := f(x(i)), \qquad i \in \mathbb{N}_0 \tag{1.2}$$

is called *discrete–time dynamical system*.

Despite of the lack of a control, we are interested in discrete–time dynamical systems and its properties. In particular, the control law u resulting from the receding horizon control setting of Chapter 2 is a function of the state x, a so called *(state) feedback*. Applying this control to (1.1) we obtain a system of type (1.2). The aim of the receding horizon control law (or any other control law) is to induce certain properties like stability for the resulting dynamical system.
Before discussing properties of solutions of (1.1) or (1.2) we explain the context in which the continuous–time examples need to be seen. To this end, we define a control system in continuous–time and the corresponding dynamical system.

Definition 1.12 (Continuous–time Control System)
Consider a function $f : \mathbb{X} \times \mathbb{U} \to \mathbb{X}$. A system of n first order ordinary differential equations

$$\dot{x}_u(t) := \frac{d}{dt} x_u(t) = f(x_u(t), u(t)), \qquad t \in \mathbb{R} \tag{1.3}$$

is called a *continuous–time control system* or *dynamic of the continuous–time control system*.

Definition 1.13 (Continuous–time Dynamical System)
Consider a function $f : \mathbb{X} \to \mathbb{X}$. A system of n first order ordinary differential equations

$$\dot{x}(t) := \frac{d}{dt} x(t) = f(x(t)), \qquad t \in \mathbb{R} \tag{1.4}$$

is called a *continuous–time dynamical system*.

Example 1.14
The mentioned pendulum example is given by

$$\begin{aligned} \dot{x}(t) &= y(t) \\ \dot{y}(t) &= -\frac{g}{l} \cdot sin(x(t)) - d \cdot y(t)^2 \cdot \operatorname{atan}(1000.0 \cdot y(t)) \\ &\quad - \left(\frac{4.0 \cdot y(t)}{1.0 + 4.0 \cdot y(t)^2.0} + \frac{2.0 \cdot \operatorname{atan}(2.0 \cdot y(t))}{\pi} \right) \cdot m \end{aligned}$$

with gravitational constant $g = 9.81$, length $l = 1.25$, drag $d = 0.007$ and moment $m = 0.197$.

In order to be able to talk about a trajectory of (1.1), (1.2), (1.3) and (1.4) according to Definition 1.7, the additional information on the starting point is needed:

Definition 1.15 (Initial Value Condition)
Consider a point $x_0 \in \mathbb{X}$. Then the equation

$$x(0) = x_0 \tag{1.5}$$

is called the *initial value condition*.

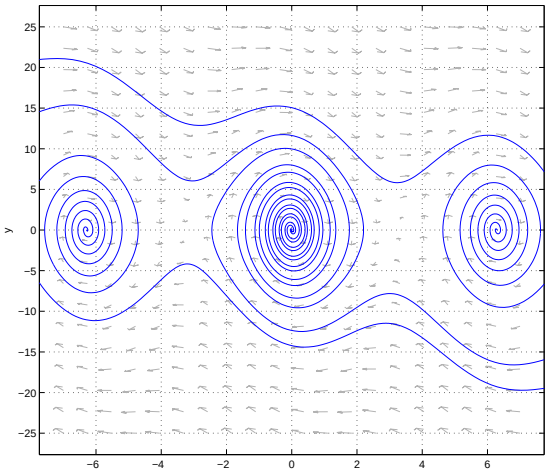


Figure 1.2: Vector field of the pendulum and some solutions given by Example 1.14

If in the case of (1.4) $f(\cdot)$ satisfies the so called *Lipschitz condition*

$$\|f(x) - f(y)\| \leq L\|x - y\| \tag{1.6}$$

for some constant $L \in \mathbb{R}^+$ for all x and y in some neighborhood of x_0, then we can guarantee existence and uniqueness of a solution on some interval, see e.g. Chapter 10 of [228] or Chapter 3 in [129].

Theorem 1.16 (Local Existence and Uniqueness)
Let $f(\cdot)$ satisfy the Lipschitz condition (1.6) *for all $x, y \in B_r(x_0) := \{x \in \mathbb{R}^n \mid \|x - x_0\| < r\}$. Then there exists some $\delta > 0$ such that the equation* (1.4) *together with* (1.5) *has a unique solution on $[0, \delta]$.*

In the context of continuous–time control problems, one can show that even for rather simple problems the optimal control function is discontinuous. Hence, considering only the set of continuous control functions appears to be too strict for our purposes, see Chapter 10 in [210]. In the case of the previously mentioned inverted pendulum such a control is obtained if one considers the pendulum to point downwards with angular speed zero and wants to start the swing–up. If the control is bounded, then one starts with maximal acceleration and at some point one has to change the sign of the acceleration to avoid overshooting the upright position.
Moreover, the *semigroup property* stated in Definition 1.5 is violated by a *concatenation* of two continuous functions if $\mathcal{U}$ is restricted to the class of continuous functions. The class of measureable function, however, meets the described requirements.

Definition 1.17 (Measureable Functions)
Consider a closed interval $I = [a, b] \subset \mathbb{R}$.

(i) A function $g : I \to \mathbb{R}^m$ is called *piecewise constant* if there exists a finite partition of subintervals I_j, $j = 1, \ldots, n$, such that $g(\cdot)$ is constant on I_j for all $j = 1, \ldots, n$.

(ii) A function $g : I \to \mathbb{R}^m$ is called *(Lebesque-)measureable* if there exists a sequence of piecewise constant functions $g_i : I \to \mathbb{R}^m$, $i \in \mathbb{N}$, such that $\lim_{i\to\infty} g_i(x) = g(x)$ for almost all $x \in I$.

(iii) A function $g : \mathbb{R} \to \mathbb{R}^m$ is called *(Lebesque-)measureable* if for every closed subinterval $I = [a, b] \subset \mathbb{R}$ the restricted function $g|_I$ is (Lebesque-)measureable.

(iv) A function $g : \mathbb{R} \to \mathbb{R}^m$ is called *locally essentially bounded* if for every compact interval $I \subset \mathbb{R}$ there exists a constant $C \in \mathbb{R}$, $C > 0$, such that $\|u(t)\| \leq C$ for almost all $x \in I$.

According to the changes within the right hand side of a control system we have to adapt our theory concerning the existence and uniqueness of solutions. To this end we refer to the theorem of Caratheodory, see [210] for details.

Theorem 1.18 (Caratheodory)
Consider the setting of Definition 1.12 and a control system satisfying (1.5) *and the following conditions:*

(i) $\mathcal{U} := \{u : \mathbb{R} \to \mathbb{U} | u$ *is measureable and locally essentially bounded*$\}$

(ii) $f : \mathbb{X} \times \mathbb{U} \to \mathbb{X}$ *is continuous.*

(iii) *For all* $R \in \mathbb{R}$, $R > 0$ *there exists a constant* $M_R \in \mathbb{R}$, $M_R > 0$ *such that* $\|f(x, u)\| \leq M_R$ *holds for all* $x \in \mathbb{R}^n$ *and all* $u \in \mathbb{U}$ *satisfying* $\|x\| \leq R$ *and* $\|u\| \leq R$.

(iv) *For all* $R \in \mathbb{R}$, $R > 0$ *there exists a constant* $L_R \in \mathbb{R}$, $L_R > 0$ *such that*

$$\|f(x_1, u) - f(x_2, u)\| \leq L_R \|x_1 - x_2\|$$

holds for all $x_1, x_2 \in \mathbb{R}^n$ *and all* $u \in \mathbb{U}$ *satisfying* $\|x_1\| \leq R$, $\|x_2\| \leq R$, $\|u\| \leq R$.

Then, there exists a maximal interval $J = (\tau_{min}, \tau_{max}) \subset \mathbb{R}$, $0 \in J$, *for all* $x_0 \in \mathbb{X}$ *and all* $u \in \mathcal{U}$ *such that there exists a unique and absolutely continuous function* $x_u(t, x_0)$ *satisfying*

$$x_u(t, x_0) = x_0 + \int_0^t f(x_u(\tau, x_0), u(\tau))d\tau \tag{1.7}$$

for all $t \in J$.

So far we have shown the existence and uniqueness of solutions for all four types of systems. Here, we use the following notation for solutions of these systems:

Definition 1.19 (Solution)
The unique function $x_u(t, x_0)$ of (1.1) (or (1.3)) emanating from initial value $x_0 \in \mathbb{X}$ is called *solution* of (1.1) (or (1.3)) for $t \in \mathbb{T}$.
If $f(\cdot)$ is independent of the control u then the unique solution of (1.2) (or (1.4)) with initial value $x_0 \in \mathbb{X}$ is denoted by $x(t, x_0)$ for $t \in \mathbb{T}$.

In this thesis, we consider discrete–time control systems to compute a control strategy while the underlying examples are continuous in time. The interconnection between these two settings is called *sampling*.

Definition 1.20 (Sampling)
Consider a control system (1.3) and a fixed time grid $\mathbb{T} = \{t_0, t_1, \ldots\}$ with $t_0 = 0$ and $t_i < t_{i+1}$ $\forall i \in \mathbb{N}_0$. Moreover, we assume that the conditions of Caratheodory's Theorem 1.18 hold and u is a concatenation of control functions $u_i \in \mathcal{U}$ with $u(t) = u_i(t)$ $\forall t \in [t_i, t_{i+1})$. Then we define the *sampling solution* of (1.3), (1.5) inductively via

$$x_T(t, x_0, u) := x_{u_i}(t - t_i, x_T(t_i, x_0, u)) \qquad \forall t \in [t_i, t_{i+1}). \tag{1.8}$$

using (1.7). The time distance $\Delta_i := t_{i+1} - t_i$ is called *sampling period* and its reciprocal Δ_i^{-1} is called *sampling rate*.

Note that considering (1.8) instead of the class of control problems (1.3), (1.5) is not a restriction since the concatenation of control functions $u_i \in \mathcal{U}$ can be any element of $\mathcal{U}$.
Upon implementation of controllers, digital computers are nowadays used to compute and implement a control action. Since these computers work at a finite sampling rate and cannot change the control signal during the corresponding sampling period, technology influences the mathematical modelling of the problem. Considering digital controllers, it appears natural to use piecewise constant control functions. This gives us the following:

Definition 1.21 (Sampling with zero–order hold)
Consider the situation of Definition 1.20 with constant control functions $u_i(t) \equiv c_i$ $\forall t \in [t_i, t_{i+1})$. Then $x_T(t, x_0, u)$ is called *sampling solution with zero–order hold*.

Remark 1.22
Note that other implementations are possible and have also been discussed in the receding horizon control literature, see e.g. [59]. Throughout this thesis, however, we consider our continuous–time examples to be treated in a sampling with zero–order hold fashion, see also Section 2.4.

Now we can derive a discrete–time system (1.1) from a continuous–time system (1.3).

Definition 1.23 (Sampled–data System)
Consider the sampling solution with zero–order hold x_T as given by Definition 1.21. Then we call the discrete–time system

$$x_u(i, x_0) := x_T(t_i, x_0, u) \tag{1.9}$$

with $u(i) := u_i$ and $f(x_u(i, x_0), u(i)) := x_{u(i)}(\Delta_i, x_u(i, x_0))$ *sampled–data system* or *digital control system* and $x_u(i, x_0)$ is called *sampled–data solution* for all $i \in \mathbb{N}_0$.

Note that by now we have defined six kinds of systems, i.e.

	discrete–time	continuous–time
Control system	Definition 1.10	Definition 1.12
Dynamical system	Definition 1.11	Definition 1.13
Sampled–data system	Definition 1.23	Definition 1.21

Table 1.1: Schematic presentation of the systems under consideration

By definition, every dynamical system can be considered as sampled–data system and also every sampled–data system can be seen as a continuous–time control system. Hence, if a dynamical system shows certain properties, then there exists a sampled–data system and a continuous–time control system with $u \equiv 0$ having identical properties.
Here, we are looking for the converse, i.e. we seek control laws inducing certain properties. In control theory, so called *open–loop* and *closed–loop control laws* are considered.

Definition 1.24 (Open–loop Control Law)
Consider the setting of Definition 1.10, 1.12, 1.21 or 1.23. A function $u : \mathbb{T} \to \mathbb{U}$ based on some initial condition x_0 is called an *open–loop* or *feedforward* control law.

Definition 1.25 (Closed–loop or Feedback Control Law)
Consider the setting of Definition 1.10, 1.12, 1.21 or 1.23. A function $F : \mathbb{X} \to \mathbb{U}$ is called a *closed–loop* or *feedback* control law and is applied by setting $u(\cdot) := F(x(\cdot))$.

Definitions 1.24 and 1.25 also show the two main lines of work in control theory which sometimes have seemed to proceed in different directions but which are in fact complementary.
The open–loop control is based on the assumption that a good model of the object to be controlled is available and we wish to modify/optimize its behaviour. The corresponding techniques have emerged from the classical calculus of variations and from other areas of optimization theory. This approach typically leads to a control law $u(\cdot)$ which has been computed *offline* before the start of the system like a preprogrammed flight plan.

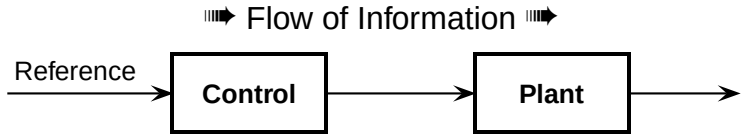


Figure 1.3: Schematic representation of an open–loop control system

In particular, one computes the function $u(\cdot)$ based on the initial conditions x_0 and the vector field $f(\cdot, \cdot)$ and applies it blindly without taking available measurements into account. The result is the so called *open–loop solution*. Considering the discrete–time case exemplarily, the resulting trajectory

$$x(n+1) = f\left(x(n), u(n)\right) \tag{1.10}$$

emanating from the initial value $x(0) = x_0$ with open–loop control $u(\cdot)$ is called *open–loop solution*, see also Figure 1.3 showing the block diagramm for the open–loop setting.
In reality, unknown perturbations and uncertainties may occur which are not accounted for in the mathematical model used in the open–loop approach. If this is the case, then applying an open–loop control law $u(\cdot)$ over a long time horizon may lead to large deviations of the state trajectory.
The second line of work is the attempt to integrate these aspects about the model or about the operating environment of the system into the control law. The central tool is the use of *feedback* correcting deviations from the desired behavior, i.e. we implement a control $u(\cdot)$ depending on the actual state of the system, i.e.

$$x(n+1) = f\left(x(n), u(x(n))\right). \tag{1.11}$$

Using the closed–loop control $u(\cdot)$, we call the trajectory (1.11) emanating from the initial value $x(0) = x_0$ *closed–loop solution*. The closed–loop situation is visualized as shown in Figure 1.4. Note that implementing the feedback controller requires the states of the system to be continuously monitored and the control law to be evaluated *online*.

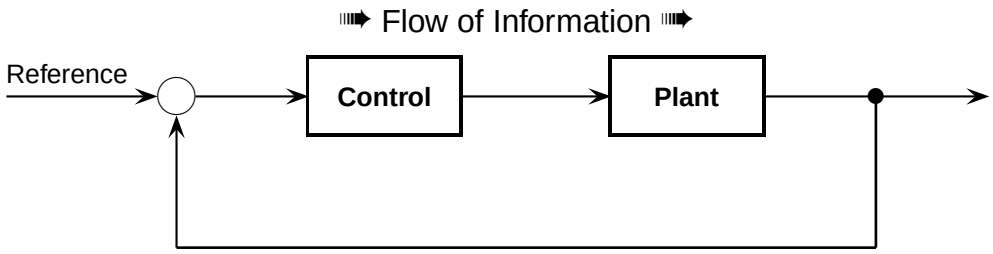


Figure 1.4: Schematic representation of an closed–loop control system

Today, it is widely recognized that these two broad lines deal with different aspects of the same problem. In particular, the open–loop methods may be used to a priori compute a "guiding path" to the desired target whereas closed–loop methods are applied online to prevent deviations from a path.
Considering a closed–loop control law, mathematically the underlying control system turns into a dynamical systems since the feedback law $F(\cdot)$ is a function of the state x. Hence, we start by defining properties of dynamical systems (1.2), (1.4) and giving necessary and sufficient criteria for these properties. Next, we extend these properties and criteria to control systems (1.1), (1.3). In the last step, we consider sampled–data systems according to Definitions 1.21 and 1.23. At the same time, we relax the considered properties by allowing for a small deviation. This is necessary since due to the technical lower bound on the sampling rate we cannot expect a piecewise constant control to exist such that the system under control exhibits the standard stability property. Since small deviations need to be acceptable in reality as well, this extension is also reasonable.
In the following, we focus on certain properties of solutions of such systems, that is *stability* and *controllability*.

1.3 Stability and Controllability

When treating control problems it is our central task to steer a system into a certain state and keep it there. Hence, we are interested in the long term behaviour of solutions.
As mentioned before we start by introducing the so called *stability property*. There exist a lot of references on this property, most of them in the classical dynamical systems literature, see e.g. [30, 107] for a bibliographical survey. Usually, one treats equilibrium points and characterizes their stability properties in the sense of Lyapunov.
Roughly speaking an equilibrium point is considered to be *stable* if all solutions starting at nearby points stay close to this point, otherwise it is called unstable. Moreover, it is called *asymptotically stable* if all solutions starting at nearby points not only stay nearby but also tend to the equilibrium point as time tends to infinity.
These notions are defined properly in Section 1.3.1. Additionally, the concept of comparison functions is introduced and it is shown how stability can be expressed using these functions. Moreover, Lyapunov's method is presented to test whether stability or asymptotic stability can be guaranteed for a given system.
In Section 1.3.2, we extend the stability property by the controllability property which essentially says that there exists at least one control for which stability of an equilibrium can be guaranteed. Again, this is shown in the context of comparison and Lyapunov functions.
Finally, we modify the stability and controllability property by some ε-ball in Section 1.3.3. This is necessary for our analysis of the receding horizon controller in Chapter 3 since the

stability property may hold or be established for an equilibrium of the continuous–time control system but may be lost by digitalization.

1.3.1 Stability of Dynamical Systems

Within this section, we first define and characterize stability properties of a given system. Motivated by our aim to examine digital control systems, that is sampled–data systems of the form (1.9), where we are interested in control sequences which can be applied as piecewise constant function in the sampled–data setup, we consider the discrete–time case (1.2) only. For stability results concerning the continuous–time problem (1.3) we refer, among others, to [127, 154, 214].
We start off by defining stability of an equilibrium for discrete–time dynamical systems (1.2) which naturally arise if we consider closed–loop systems.

Definition 1.26 (Equilibrium)
A point $x^\star \in \mathbb{R}^n$ is called an *equilibrium* of a dynamical system (1.2) if $x(i, x^\star) \equiv x^\star$ holds for all $i \in \mathbb{N}_0$.

For reasons of simplicity we assume $x^\star = 0$ in the following. This is no loss of generality since for $x^\star \neq 0$ one can use the transformed system $\overline{f}(x) := f(x + x^\star)$ which is a shift of the solution but does not affect its long term behaviour.

Definition 1.27 (Stability)
The equilibrium point $x^\star = 0$ of a dynamical system (1.2) is called

- *stable* if, for each $\varepsilon > 0$, there exists a real number $\delta = \delta(\varepsilon) > 0$ such that

$$\|x_0\| \leq \delta \implies \|x(i, x_0)\| \leq \varepsilon \qquad \forall i \in \mathbb{N}_0$$

- *asymptotically stable* if it is stable and there exists a positive real constant r such that

$$\lim_{i \to \infty} x(i, x_0) = 0$$

  for all initial values x_0 satisfying $\|x_0\| \leq r$. If additionally r can be chosen arbitrary large, then $x^\star$ is called *globally asymptotically stable*.

- *unstable* if it is not stable.

E.D. Sontag (re)introduced a different but intuitive approach to characterize stability properties in [208] which is by now a standard formalism in control theory, see also [106, 107] for earlier references. To this end we define so called *comparison functions*:

Definition 1.28 (Comparison Functions)
- A continuous non-decreasing function $\gamma : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ satisfying $\gamma(0) = 0$ is called *class $\mathcal{G}$ function*.

- A function $\gamma : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is of *class $\mathcal{K}$* if it is continuous, zero at zero and strictly increasing.

- A function is of *class $\mathcal{K}_\infty$* if it is of class $\mathcal{K}$ and also unbounded.

- A function is of *class $\mathcal{L}$* if it is strictly positive and it is strictly decreasing to zero as its argument tends to infinity.

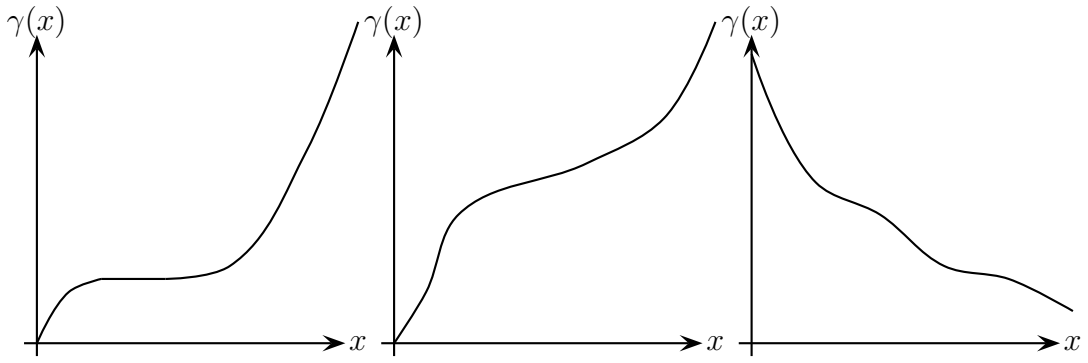


Figure 1.5: Comparison functions of class $\mathcal{G}$, $\mathcal{K}$ and $\mathcal{L}$

- A function $\beta : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is of *class* $\mathcal{KL}$ if for every fixed $t \geq 0$ the function $\beta(\cdot, t)$ is of class $\mathcal{K}$ and for each fixed $s > 0$ the function $\beta(s, \cdot)$ is of class $\mathcal{L}$.

Using comparison functions we can characterize our previous stability concepts in a different way. In particular, these functions allows us a geometrical inclusion of the solution emanating from a given initial value (1.5) by terms of an upper bound for the worst case. For a proof of the following theorem stating this property we refer to [129].

Theorem 1.29 (Stability)
Consider $x(i, x_0)$ to be the solution of (1.2) and to exist for all $i \in \mathbb{N}_0$.

(i) An equilibrium $x^\star = 0$ is stable if and only if there exists a neighborhood $\mathcal{N}(x^\star)$ and a function $\alpha \in \mathcal{K}_\infty$ such that

$$\|x(i, x_0)\| \leq \alpha(\|x_0\|)$$

holds for all $x_0 \in \mathcal{N}(x^\star)$ and all $i \in \mathbb{N}_0$.

(ii) An equilibrium $x^\star = 0$ is asymptotically stable if and only if there exists a neighborhood $\mathcal{N}(x^\star)$ and a function $\beta \in \mathcal{KL}$ such that

$$\|x(i, x_0)\| \leq \beta(\|x_0\|, i)$$

holds for all $x_0 \in \mathcal{N}(x^\star)$ and all $i \in \mathbb{N}_0$. Moreover $x^\star$ is called globally asymptotically stable if and only if $\mathcal{N}(x^\star) = \mathbb{R}^n$.

If a dynamical system possesses more than one equilibrium, then we are also interested in sets of initial values such that the solution emanating from any point in this set tends to a unique equilibrium.

Definition 1.30 (Basin of Attraction)
We define the set

$$\mathcal{D}(x^\star) := \{x_0 \in \mathbb{R}^n \mid x(i, x_0) \to x^\star \text{ as } i \to \infty\}$$

to be the *basin of attraction* of an equilibrium of a dynamical system.

Remark 1.31
The concept of the basin of attraction is of particular interest for the inverted pendulum. Due to its 2π–periodicity there exist infinitely many equilibria and, as we will see in Section 8.3.2 considering the resulting closed–loop solution (1.11) of the receding horizon controller, the equilibrium to be stabilized depends massively on the initial value and other parameter of the controller.

In order to analyze the stability of an equilibrium of a dynamical system (1.2) A.M. Lyapunov published the idea to examine an auxilliary function in 1892. In particular, this function allows us to analyze the development of a dynamical system along the vector field defining the differential equation instead of solving the differential equation itself, see [149].
According to the discrete–time case under consideration, we define the so called *Lyapunov function* for dynamical systems of type (1.2):

Definition 1.32 (Lyapunov Function)
Let $x^\star = 0$ be an equilibrium point for (1.2) and $\mathcal{N} \subset \mathbb{R}^n$ be a neighborhood of $x^\star$. Let $V : \mathcal{N} \to \mathbb{R}$ be continuous. If there exist functions $\alpha_1, \alpha_2 \in \mathcal{K}_\infty$ and a function $W : \mathcal{N} \to \mathbb{R}$ which is locally Lipschitz satisfying $W(x) > 0$ for all $x > 0$, $W(x) = 0$ for $x \leq 0$ and

$$\alpha_1(\|x\|) \leq V(x) \leq \alpha_2(\|x\|) \tag{1.12}$$
$$V(f(x)) \leq V(x) - W(V(x)) \tag{1.13}$$

for all $x \in \mathcal{N}$. Then $V(\cdot)$ is called a *local Lyapunov function.* Moreover, if $\mathcal{N} = \mathbb{R}^n$, then $V(\cdot)$ is called *global Lyapunov function.*

Using a physical interpretation, one can think of this auxilliary function as a positive measure of the energy in the system with a minimum at the equilibrium point $x^\star$. Mathematically this function replaces the Euclidean distance used in Definition 1.27 and Theorem 1.29 by a nonlinear distance. Hence, one can demonstrate stability of the equilibrium if this distance is strictly decreasing along the trajectories of the system.

Theorem 1.33 (Asymptotic Stability)
An equilibrium $x^\star = 0$ is asymptotically stable if and only if there exists a function $V(\cdot)$ satisfying the conditions of Definition 1.32.

For a proof of this theorem we refer to Chapter 2 of [20] or Chapter 1 in [214] in the discrete–time case, for the continuous–time case a proof is given in Chapter 4 of [129] respectively.
In the literature, one often assumes $V(\cdot)$ to be differentiable, see e.g. [43, 105, 129, 155]. Differentiability, however, is too strigent if we consider the dynamical system to be the outcome of a control system with discontinuous feedback. In this case, we cannot expect the Lyapunov function to be smooth, cf. [41, 207].

1.3.2 Stability of Control Systems

Until now we have only considered dynamical systems. Now we extend the stability concepts stated in Definition 1.27 and Theorems 1.29, 1.33 to discrete–time control systems (1.1) assuming the control function to exist for any initial value. Moreover, we assume the solution of a control system (1.1) to exist for all time. Note that the control is not expected to be unique, and hence the solution $x_u(t, x_0)$ may not be uniquely defined as well. Thus, we have to consider the case of more than one solution emanating from an initial value x_0.
Again, we are interested in the long time behaviour of (1.1), i.e. equilibrium points and stability properties, cf. Definitions 1.26 and 1.27. Since an additional parameter can be set arbitrarily, we distinguish between independent and induced properties. Similar to the previous Section 1.3.1, we present all definitions and results in the discrete–time form of system (1.1). Therefore, we consider the set of controls $\mathcal{U} = \mathbb{U}^{\mathbb{N}}$.

Definition 1.34 (Stability Concepts for Equilibrium Points)
Consider a control system (1.1).

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

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

This definition naturally induces two concepts of stability and asymptotical stability, robustness and controllability. Both concepts depend on the interpretation of u within the difference equation (1.1), i.e. to be an influencable external control or some disturbance within the system.

Definition 1.35
The equilibrium point $x^\star = 0$ of a control system (1.1) is

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

$$\|x_0\| \leq \delta \implies \|x_u(i, x_0)\| \leq \varepsilon \qquad \forall i \in \mathbb{N}_0 \tag{1.14}$$

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

$$\lim_{i \to \infty} x_u(i, x_0) = 0 \tag{1.15}$$

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

- *weakly stable* or *controllable* if, for each $\varepsilon > 0$, there exists a real number $\delta = \delta(\varepsilon) > 0$ such that for each x_0 there exists a control $u \in \mathcal{U}$ guaranteeing

$$\|x_0\| \leq \delta \implies \|x_u(i, x_0)\| \leq \varepsilon \qquad \forall i \in \mathbb{N}_0. \tag{1.16}$$

- *weakly asymptotically stable* or *asymptotically controllable* if there exists a control $u \in \mathcal{U}$ depending on x_0 such that (1.16) holds and there exists a positive constant r such that

$$\lim_{i \to \infty} x_u(i, x_0) = 0 \qquad \forall \|x_0\| \leq r. \tag{1.17}$$

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

Remark 1.36
In the case of strong asymptotic stability, the implemented control does not affect the stability of the system. Still, one in general obtains better results imposing a control method such as, e.g., receding horzizon control. From the computational point of view, strong asymptotic stability is interesting if one has to consider significant measurement or discretization errors. In the context of our receding horizon control scheme, such a property allows for using rather large tolerances in the optimization as well as in the solution of the differential equation system which may significantly reduce the required

computing time, see also Section 8.3.
The concept of weak stability or weak asymptotical stability, on the other hand, is more general. It naturally leads to the question how to compute a control law such that $x^\star$ is weakly stable, and, in particular, how to characterize the quality of a control law which is our main goal in Chapters 2 and 3.
Within our implementation of the receding horizon controller we consider a compromise which is problem dependent, i.e. we allow for errors during the computation but limit the range of these errors to obtain stability of the closed–loop, see Section 8.3 for details.

Here, we consider u in the control context and focus on the question how to find controls such that stability can be guaranteed. In order to develop sufficient conditions for stability we first forward the equivalent concepts from the previous section, that is the characterization via comparison functions and the use of Lyapunov functions.

Theorem 1.37 (Stability Concepts)
Consider a control system (1.1).

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

$$\|x_u(i, x_0)\| \leq \beta(\|x_0\|, i)$$

holds for all $x_0 \in \mathcal{N}$, $u \in \mathcal{U}$ and all $i \in \mathbb{N}_0$.

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

$$\|x_u(i, x_0)\| \leq \beta(\|x_0\|, i)$$

holds for all $i \in \mathbb{N}_0$.

In the continuous–time case a proof can be found, e.g., in Chapter 4 of [129]. Since the proof is similar in the discrete–time setting we omit it here. Moreover, this characterization can be extended if one takes disturbances into account. This leads to the ISS and ISDS concept, see, e.g., [208] and [89, 118] respectively.
Similar to the extension of the concept of stability towards controllability in Definition 1.35, Lyapunov functions can be defined for control systems. The main difference lies in considering a minimizing control in the neighborhood of the considered state.

Definition 1.38 (Control–Lyapunov–Function)
Consider an equilibrium $x^\star = 0$, a control system (1.1) with $f(0, 0) = 0$ and $\mathcal{N} \subset \mathbb{R}^n$ to be a neighborhood of $x^\star$. Then a continuous function $V : \mathcal{N} \to \mathbb{R}$ is called a *control–Lyapunov function* if there exist functions $\alpha, \alpha_1, \alpha_2 \in \mathcal{K}_\infty$ such that for all $x \in \mathcal{N}$ there exists a control function $u \in \mathcal{U}$ such that the inequalities

$$\alpha_1(\|x\|) \leq V(x) \leq \alpha_2(\|x\|) \tag{1.18}$$

$$\inf_{u \in \mathbb{U}} V(x_u(1, x_0)) - V(x_0) \leq -\alpha(x_0) \tag{1.19}$$

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

Using Definition 1.38 the characterization of the stability concepts is similar to dynamical systems, see Proposition 1.33. For a proof of the following stability theorem we refer to Chapter 7 in [127] in the discrete–time case, for the continuous–time case see Chapter 5 of [210].

Theorem 1.39 (Asymptotic Stability)
Consider a control system (1.10) *where* $f(0,0) = 0$, $\mathcal{N} \subset \mathbb{R}^n$ *to be a neighborhood of* $x^\star$ *and a continuous function* $V : \mathcal{N} \to \mathbb{R}$.

(i) An equilibrium $x^\star = 0$ *is strongly asymptotically stable or robustly asymptotically stable if* (1.18) *and*

$$\sup_{u \in \mathbb{U}} V(x_u(1, x_0)) - V(x_0) \leq -\alpha(x_0)$$

hold for all $x_0 \in \mathcal{N}$.

(ii) An equilibrium $x^\star = 0$ *is weakly asymptotically stable or asymptotically controllable if for all* $x_0 \in \mathcal{N}$ *there exists a control* $u \in \mathcal{U}$ *such that* (1.18), (1.19) *hold.*

The converse of this theorem as shown in [118] is more delicate and related to the existence of stabilizing feedbacks, see [11, 41, 209]. In the continuous–time case, (1.19) is sometimes replaced by $DV(x)f(x,u) < 0$ for some $u \in \mathcal{U}$ where $DV(\cdot)$ denotes the derivative of $V(\cdot)$, see, e.g., [42, 127, 132, 207, 210]. Here, however, we do not want to assume that $V(\cdot)$ is differentiable. Apart from this difference, on can also treat disturbances in the context of control–Lyapunov functions, see, e.g., Chapter 3 in [72] or Chapter 3 in [89].
In any case, all converse Lyapunov results are computationally difficult. A systematic way to build a control–Lyapunov function is called *backstepping* and pursued, among others, in Chapter 3 of [132] and Chapter 5 of [72]. Yet, there exists no generally applicable method which can be used as a guideline in the search for a control–Lyapunov function or even a Lyapunov–function. For many systems, physical insight plus a good amount of trial and error is typically the only way to handle this matter. Still, there are many heuristics that help in this context, see, for instance, [198].

Remark 1.40
The stability property is referred to as Feedback–stabilization when the reference signal is constant. Here, we suppose the equilibrium $x^\star = 0$ *to be this reference signal. However, one can also consider time–varying signals* $x_{ref}(\cdot) \in \mathbb{R}^n$ *evolving with respect to a law* $g : \mathbb{R} \to \mathbb{R}^n$ *which is then called tracking. As a result, the corresponding Lyapunov function is time–varying as well. For further details on tracking we refer to [72, 132, 135, 136, 233] and references therein.*
In the following, we only consider the case of constant reference functions. Results, however, are expected to be generalizable to the time–varying case.

1.3.3 Stability of Digital Control Systems

Now, we focus on digital control, i.e. sampling with zero–order hold as stated in Definition 1.21. We already mentioned that this setup naturally occurs if one wishes to implement a control law using a digital computer, see the end of Section 1.2.
Nowadays, the use of a digital computers has become standard in control applications. Reasons for that are the flexibility in programming, in particular compared to analogous circuits, but also the availability of preimplemented controllers and cost reasons. Yet, we are forced to consider the following setup:

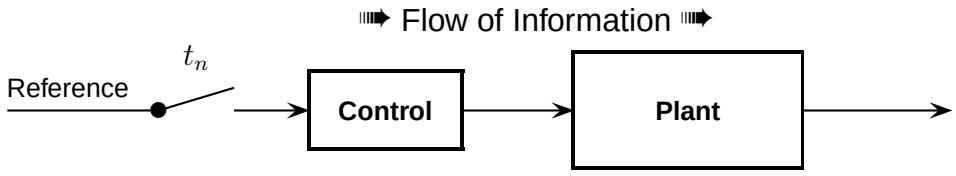


Figure 1.6: Schematic representation of an open–loop digital control system

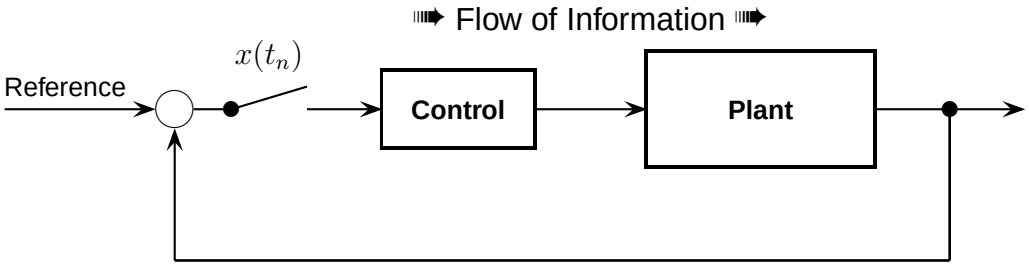


Figure 1.7: Schematic representation of an closed–loop digital control system

As shown in Figures 1.6 and 1.7, the control u is updated by some law only at the sampling instants t_n, $n \in \mathbb{N}_0$. The output of this law, the vector $u(t_n)$, is then fed into the system as a control and held constant on the interval $[t_n, t_{n+1})$. In the following chapters, our aim is to develop such a law inducing certain properties of the resulting closed–loop system. Here, we focus on defining properties which suit our control task of stabilizing a given point $x^\star$.
Note that according to Definitions 1.21 and 1.23 the control u can be described as a piecewise constant function or as a sequence of control values.

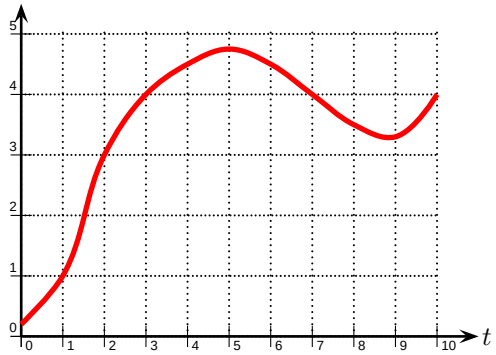
Remark 1.41
(i) As usual, we assume the premises of Caratheodory's Theorem 1.18 to be fulfilled which hence guarantees a unique solution x_u to exist on each sampling interval $[t_n, t_{n+1})$.
(ii) If u is in feedback form, we assume that the computation of u can be done quickly relative to the length of the sampling intervals. Otherwise, the model must be modified to account for the extra delay.

The easiest way to construct a digital controller is to first design a continuous–time controller for the continuous–time plant ignoring sampling, and then discretize the obtained controller for digital implementation [8, 40, 70, 134].
The brute force approach to do this is called *Emulation* where one implements the controller

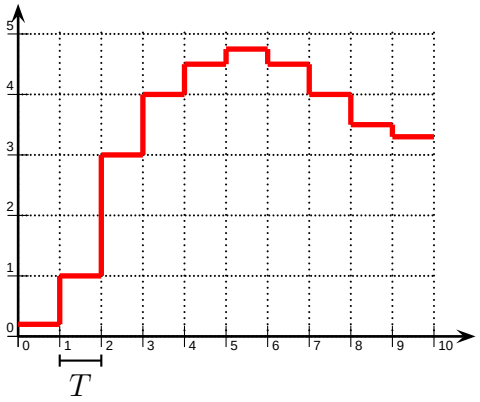
$$u_T(t) = u(t_n), \quad \forall t \in [t_n, t_{n+1}),\ t_n = n \cdot T,\ n \in \mathbb{N}_0 \tag{1.20}$$

in (1.3) and then samples as fast as possible, i.e. taking T to be as small as possible. Resulting controls may look as shown in Figures 1.8(a)–(c).

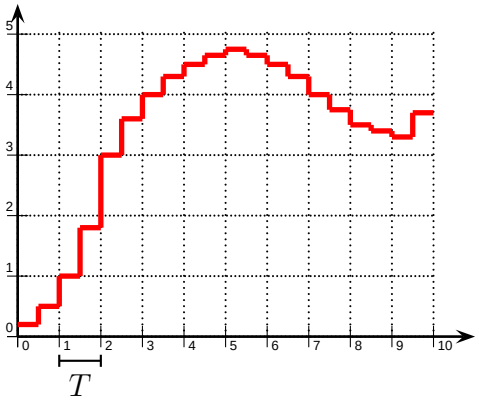
Remark 1.42
Designs such as the mentioned Emulation, the Tustin or matched pole-zero discretizations [134] are fairly simple which is the reason why this concept has also been extended to other areas of interest, e.g. networked control systems, see [168, 169].
Yet, emulated controllers are required to have appropriate robustness with respect to the



(a) Continuous–time control without sampling



(b) Control using sampling with $T = 1$



(c) Control using sampling with $T = 1/2$

Figure 1.8: Continuous–time control and corresponding emulation

sample and zero order hold implementation in order to preserve stability [117]. Typically, stability is obtained by sufficiently reducing the sampling period. However, due to hardware limitations on the minimum achievable T this approach is often not feasible even in the linear case [7, 125]. Therefore, the sampling error needs to be considered in the design of the controller.

Remark 1.43
There also exist advanced controller discretization concepts which ease the fixed discretization by an optimized one. Here, one computes "the best discretization" of the continuous–time controller in some sense, see e.g. [7, 40].

In this work, however, we focus on equidistant time grids $\mathbb{T} = T\mathbb{N}_0$, i.e. $\Delta_i = T$ for all $i \in \mathbb{N}_0$ in Definition 1.23.
Due to this restriction to piecewise constant functions u_T on an equidistant time grid $\mathbb{T}$, we cannot assume that there exists a control revealing the stability property stated in Definition 1.35, see also Remark 1.42. Hence, we ease the considered stability concept to allow for a stabilized set instead of a stabilized point. Moreover, we consider weak stability exclusively, see Definition 1.34.

Definition 1.44 (Semi–global practically asymptotic Stability)
Consider a digital control system (1.9) where $f(0, 0) = 0$. Assume that for given pair of

real–valued numbers (σ, ρ), $0 < \rho < \sigma < \infty$, there exists a constant $\varepsilon^\star > 0$ and a *family of piecewise constant functions* $u_T : \mathbb{R} \to \mathbb{U}$ for $T \in (0, T^\star(\sigma, \rho)]$ depending on the initial state x_0 such that for all $\varepsilon \in (0, \varepsilon^\star)$ the following conditions hold:

(1) Stability:
For all $\varepsilon > \rho$ there exists a $\delta(\varepsilon) > 0$ such that

$$\|x_0\| \leq \delta(\varepsilon) \quad \Rightarrow \quad \|x_u(i, x_0)\| \leq \varepsilon \ \ \forall i \in \mathbb{N}_0. \tag{1.21}$$

(2) Boundedness:
For all $r \in (0, \sigma)$ there exists a finite real number $\nu(r) > 0$ such that

$$\|x_0\| \leq r \quad \Rightarrow \quad \|x_u(i, x_0)\| \leq \nu(r) \ \ \forall i \in \mathbb{N}_0. \tag{1.22}$$

(3) Convergence:
For all $r \in (0, \sigma)$ and $\varepsilon > \rho$ there exists a finite time $T(r, \varepsilon) \in \mathbb{N}$ such that

$$\|x_0\| \leq r \quad \Rightarrow \quad \|x_u(i, x_0)\| \leq \varepsilon \ \ \forall i \geq T(r, \varepsilon) \tag{1.23}$$

Then an equilibrium $x^\star = 0$ is called (σ, ρ) *semi–globally practically asymptotically stable.*

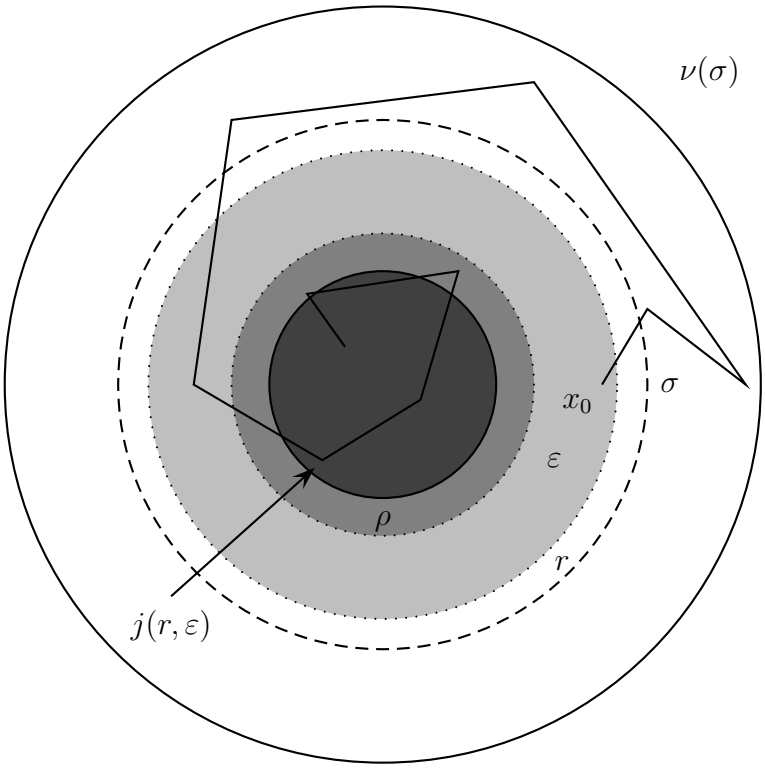


Figure 1.9: Semiglobal practically asymptotic stability in the sense of Definition 1.44

Next we forward the characterization concepts from the previous sections to the semi–global practically asymptotic case.

Proposition 1.45 (Stability Characterization using Comparison Functions)
Consider a control system (1.9) *where* $f(0, 0) = 0$ *and a pair* (Δ_1, Δ_2) *of real–valued numbers such that* $0 < \Delta_1 < \Delta_2 < \infty$ *holds. An equilibrium* $x^\star = 0$ *is* (σ, ρ) *semi–globally*

practically asymptotically stable with $\sigma := \Delta_1$ *and* $\rho := \Delta_2$ *for a family of piecewise constant functions* $u_T : \mathbb{R} \to \mathbb{U}$ *for* $T \in (0, T^\star]$ *depending on the initial value* x_0 *if there exists a function* $\beta \in \mathcal{KL}$ *such that the inequality*

$$\|x_u(i, x_0)\| \leq \max\{\beta(\|x_0\|, i), \Delta_2\} \tag{1.24}$$

holds for all initial values x_0 *in the open ball* $B^0_{\Delta_1}(x^\star)$ *and for all* $i \in \mathbb{N}_0$.

Proof. The proof is similar to the one given in Section 3 of [142] for disturbed dynamical systems. Suppose there exists a function $\beta \in \mathcal{KL}$ such that (1.24) holds. Hence, by (1.24), there exists a lower bound Δ_2 for ε and an upper bound Δ_1 for $\delta(\varepsilon)$. This allows us to choose a $\mathcal{K}_\infty$–function $\delta_1(\cdot)$ satisfying

$$\delta_1(\varepsilon) \leq \overline{\beta}_1^{\,-1}(\varepsilon) := \beta^{\,-1}(\varepsilon, 0)$$

for all $\Delta_2 \leq \varepsilon \leq c_1 := \sup \beta(\cdot, 0)$ $(\leq \infty)$. Consequently, we define $\sigma =: \Delta_1$, $\rho =: \Delta_2 < \varepsilon$ and $\delta(\varepsilon) := \max(\delta_2(\varepsilon), \Delta_1)$. Using these definitions we now show that the conditions of Definition 1.44 hold.
Stability condition (1) is satisfied since $\beta(\|x_0\|, \cdot) \in \mathcal{L}$ for fixed initial values $(x_0, 0)$ and hence for all $i \in \mathbb{N}_0$ we have

$$\lim_{i \to \infty} \beta(\|x_0\|, i) = 0.$$

Moreover, the continuity of β and the boundedness of $r \in (0, \sigma) = (0, \Delta_1)$ implies

$$\max\{\beta(\|r\|, i), \Delta_2\} \qquad \forall i \in \mathbb{N}_0.$$

to be bounded. Hence, we can conclude that a solution of (1.3) is also bounded giving condition (2).
Condition (3) follows from conditions (1) and (2). To show this suppose that there exists no finite time instant $T(r, \varepsilon)$ for $r \in (0, \sigma) = (0, \Delta_1)$, $\varepsilon > \rho := \Delta_2$. Since r is bounded and therefore $\max\{\beta(\|r\|, i), \Delta_2\}$ is bounded as well, it follows that

$$\lim_{t \to \infty} \beta(\|r\|, i) > \varepsilon > \Delta_2 = \rho \geq 0 \quad \forall i \in \mathbb{N}_0,$$

holds which contradicts the $\mathcal{KL}$–property of β. □

Similarly, the semi–globally practically asymptotic stability can be described using Lyapunov–functions. In the considered context, however, Definition 1.38 for control–Lyapunov–functions is not appropriate. While in (1.19) strict monotonicity along a solution is assumed, the convergence condition (3) in Definition 1.44 may be violated in the target area. This leads to the following relaxation of Definition 1.38:

Definition 1.46 (Semi–global practical Lyapunov–function)
Consider a digital control system (1.8) where $f(0, 0) = 0$. A family of continuous functions $V_T : \mathbb{R}^n \to \mathbb{R}_0^+$ for $T \in (0, T^\star]$ is called *semi–global practical family of sampling Lyapunov–functions* if there exist functions $\alpha, \alpha_1, \alpha_2 \in \mathcal{K}_\infty$ such that

$$\alpha_1(\|x\|) \leq V_T(x) \leq \alpha_2(\|x\|) \tag{1.25}$$

holds for all $T \in (0, T^\star]$ and all $x \in \mathbb{R}^n \setminus \{0\}$. Moreover, for a given pair of real–valueed constants (δ_1, δ_2), $\delta_1 > \delta_2 > 0$, there exists a constant $T_0(\delta_1, \delta_2) > 0$ such that

$$\inf_{u \in \mathbb{U}} V_T(x_T(T, x_0, u)) - V_T(x_0) \leq -T\alpha(x_0) + \delta_2 \tag{1.26}$$

holds for all $T \in (0, T_0]$ and all $x_0 \in B^0_{\delta_1}(0)$.

Within this definition the left hand side of (1.26) is depending on the sampling parameter T. For this reason, we consider representation (1.8) instead of (1.9) of the digital control system.

Proposition 1.47 (Stability Characterization using Lyapunov functions)
Consider a control system (1.8) *where* $f(0,0)=0$. *An equilibrium* $x^\star = 0$ *is semi–globally practically asymptotically stable for a family of piecewise constant functions* $u_T : \mathbb{R} \to \mathbb{U}$ *for* $T \in (0, T^\star]$ *depending on the initial value* x_0 *if there exists a family of semi–global practical Lyapunov–functions* $V_T : \mathbb{R}^n \to \mathbb{R}_0^+$ *for all* $T \in (0, T^\star]$ *for the point* $x^\star$.

Proof. Consider $x(\cdot)$ to be the trajectory of (1.8) with initial value $x_0 \in \mathbb{R}$. Therefore by Definition 1.46

$$\inf_{u\in\mathbb{U}} V_T(x_T(T,x_0,u)) - V_T(x_0) \leq -T\alpha(x_0) + \delta_2$$

holds for all $x(t_0) \in B_{\delta_1}^0(0)$ with $\delta_1, \delta_2 > 0$. Moreover, we have

$$\inf_{u\in\mathbb{U}} V_T(x_T((i+1)T, x_T(iT,x_0,u_T),u)) - V(x_T(iT,x_0,u_T)) \leq -T\alpha\left(x_T(iT,x_0,u_T)\right) + \delta_2$$

where $x_T(iT,x_0,u_T)$ is given recursively by the initial value $x(t_0)=x_0$ and the minimizer sequence $u_T(i) = \operatorname{arginf}_{u\in\mathbb{U}} V(x_T((i+1)T, x_T(iT,x_0,u_T),u))$. Then, using (1.25), we obtain

$$\begin{aligned}
\inf_{u\in\mathbb{U}} V_T(x_T(T,x_0,u)) &\leq V_T(x_0) - T\alpha\left(\alpha_2^{-1}(V_T(x_0))\right) + \delta_2 \\
\inf_{u\in\mathbb{U}} V_T(x_T(T,x_T(T,x_0,u_T),u)) &\leq V_T(x_T(T,x_0,u_T)) - T\alpha\left(\alpha_2^{-1}(V_T(x_T(T,x_0,u_T)))\right) + \delta_2 \\
&\leq V_T(x_0) - T\alpha\left(\alpha_2^{-1}(V_T(x_0))\right) + \delta_2 \\
&\quad -T\alpha\left(\alpha_2^{-1}(V_T(x_0) - T\alpha\left(\alpha_2^{-1}(V_T(x_0))\right) + \delta_2)\right) + \delta_2
\end{aligned}$$

Hence, we can recursively define

$$\begin{gathered}
v_0 := V_T(x_0)\,, \qquad v_i := -T\tilde{\alpha}\left(v_{i-1}\right) + v_{i-1} + \delta_2 \\
\Rightarrow V_T(x_T(iT,x_0,u_T)) \leq v_{i+1}
\end{gathered}$$

with $\tilde{\alpha}(\cdot) := \alpha(\alpha_2^{-1}(\cdot))$ and $T \in (0,1]$.
Now, our aim is to prove the existence of a mapping $g : \mathbb{R}\times\mathbb{R}\times\mathbb{R} \to \mathbb{R}$ such that

$$V_T(x_T(iT,x_0,u_T)) \leq g\left(V_T(x_0), iT, \delta_2\right)$$

and moreover that

$$g\left(V_T(x_0), iT, \delta_2\right) = \max\left\{\overline{\beta}(V_T(x_0), iT), \delta_2\right\}$$

holds with some function $\overline{\beta} \in \mathcal{KL}$.
To show the existence of g, we identify

$$g\left(V(x_0), iT, \delta_2\right) = v_{i+1} = -T\tilde{\alpha}\left(v_i\right) + v_i + \delta_2.$$

Since $\tilde{\alpha}(\cdot)$ is a $\mathcal{K}_\infty$ function, i.e. continuous, zero at zero and strictly increasing, and v_i can be written as self–concatenation of $\tilde{\alpha}(\cdot)$ which is applied to a fixed value $V(x_0)$ and modified by linear additive terms, g is a continuous function. Additionally, we know that, due to the positive semidefinitness of $V(\cdot)$, zero is a lower bound of the sequence $(v_i)_{i\in\mathbb{N}_0}$.

From the $\mathcal{K}$ property of $\tilde{\alpha}(\cdot)$ we can also conclude that the sequence $(v_i)_{i\in\mathbb{N}_0}$ is bounded from above, i.e.

$$\lim_{i\to\infty} v_i \leq M < \infty,$$

and obtain the sequence to be well–defined and to exist for all $i \in \mathbb{N}_0$.
Due to the $\mathcal{K}$–property of $\tilde{\alpha}(\cdot)$ there exists a $\overline{T} > 0$ such that $v_i - T\tilde{\alpha}(v_i) \in \mathcal{K}$ for all $0 < T < \overline{T}$ and all $i \in \mathbb{N}_0$. Now we define

$$\overline{\beta}(V_T(x_0), iT) := \begin{cases} v_{i+1} - \delta_2, & \|V_T(x_T(iT, x_0, u_T))\| \geq \tilde{\alpha}^{-1}\left(\frac{\delta_2}{T}\right) \\ 0, & \text{else} \end{cases}$$

Hence, by the previous argumentation, we obtain $\overline{\beta}(\cdot, t) \in \mathcal{K}$.
Moreover, if $\|V_T(x_T(iT, x_0, u_T))\| > \tilde{\alpha}^{-1}\left(\frac{\delta_2}{T}\right)$, we get

$$\begin{aligned} \overline{\beta}(V_T(x_0), (i+1)T) - \overline{\beta}(V_T(x_0), iT) &= v_{i+1} - v_i = v_i - T\tilde{\alpha}(v_i) + \delta_2 - v_i \\ &= \delta_2 - T\tilde{\alpha}(v_i) < 0 \end{aligned}$$

using (1.26). This shows that $\overline{\beta}(r, \cdot)$ is strictly monotonely decreasing. This property is not lost if we choose

$$\overline{\beta}(V_T(x_0), iT) := \begin{cases} \sup\limits_{j\geq i} v_{j+1} - \delta_2, & \|V_T(x_T(iT, x_0, u_T))\| \geq \tilde{\alpha}^{-1}\left(\frac{\delta_2}{T}\right) \\ 0, & \text{else} \end{cases}$$

Additionally, using the definition, we have $\overline{\beta}(0, iT) \equiv 0$. Adding a $\mathcal{KL}$–function $\tilde{\beta}$, i.e. $\hat{\beta} := \overline{\beta} + \tilde{\beta}$, the previously stated properties are preserved and we get $\lim_{t\to\infty} \hat{\beta}(\cdot, t) = 0$ and $\hat{\beta} \in \mathcal{KL}$.
In the context of Proposition 1.45, we set $\Delta_1 := \delta_1$ and $\Delta_2 := \alpha^{-1}\left(\frac{\delta_2}{T}\right)$. For this choice we get

$$\left\{ x_0 \in \mathbb{R}^n \,\middle|\, \inf_{u\in\mathbb{U}} V_T(x_T(T, x_0, u)) - V_T(x_0) \geq 0 \right\} \subset B_{\Delta_2}(0)$$

and forward invariance of the set $B_{\Delta_2}(0)$. The corresponding $\mathcal{KL}$–function β for the state is given by $\beta := \alpha_1^{-1} \circ \hat{\beta}$ and hence Proposition 1.45 shows the assertion. □

Having proved these conditions for semiglobally practically asymptotic stability we now embed our examples into this setting. To this end, we present results from [170, 171] showing that the stability property of Definition 1.44 (and Propositions 1.45, 1.47) can be extended from $\mathbb{T} = \mathbb{N}_0 T$, $T > 0$ fixed, to the full time axis $\mathbb{T} = \mathbb{R}$, see Theorems 1.50 and 1.51 below. As a result, we obtain that the system does not exhibit unbounded oscillations in between two sampling points $t_i, t_{i+1} \in \mathbb{T}$.

Considering an example one may face the problem of using approximations of the digital control system (1.9) instead of the exact system. To cover this issue, we have to show that properties which are derived for such an approximation also hold for the exact system. Such errors can be caused by the discretization of the system, modelling uncertainties, delays, inaccurate implementation of the control or many other factors. Since we consider examples given as continuous–time control systems and solve these problems using numerical solution methods for integration and optimization, we inevitably induce such an approximation error.

Here, we denote the approximation of the digital control system by

$$x_u^{(a)}(i+1, x_0) = f^{(a)}(x_u(i, x_0), u(i)) \tag{1.27}$$

according to Definition 1.23 and show that if the approximated system exhibits the desired semiglobally practically asymptotic stability property and satisfies certain consistency conditions, then the exact digital control system is also semiglobally practically asymptotically stable. This has been shown in [170] using the following consistency definitions.

Definition 1.48 (One–step Consistency)
The family $(u_T, f^{(a)})$ is said to be one–step consistent with (u_T, f) if, for each compact set $A \subset \mathbb{R}^n$, there exists a class $\mathcal{K}_\infty$–function ρ and a real–valued constant $T^\star > 0$ such that

$$\|f(x_u(i, x_0), u(i)) - f^{(a)}(x_u(i, x_0), u(i))\| \leq T\rho(T) \tag{1.28}$$

holds for all $x \in A$ and all $T \in (0, T^\star)$.

Definition 1.49 (Multi–step Consistency)
The family $(u_T, f^{(a)})$ is said to be multi–step consistent with (u_T, f) if, for each real–valued constants $\delta, L, \eta > 0$ and each compact set $A \subset \mathbb{R}^n$, there exists a function $\alpha : \mathbb{R}_0^+ \times R_0^+ \to \mathbb{R}_0^+ \cup \{\infty\}$ and a real–valued constant $T^\star > 0$ such that

$$\|f(x_u(i, x_0), u(i)) - f^{(a)}(x_u(i, z_0), u(i))\| \leq \alpha(\delta, T) \tag{1.29}$$

holds for all $x_0, z_0 \in A$ satisfying $\|x_0 - z_0\| \leq \delta$ and

$$\alpha^k(0, T) \leq \eta$$

holds for all $k \leq \frac{K}{T}$ and all $T \in (0, T^\star)$.

Then, according to [170], the following holds:

Theorem 1.50 (Stability of the Exact Digital Control System)
Consider $\beta \in \mathcal{KL}$ and (Δ_1, Δ_2) to be the bound and constants stated in Proposition 1.45 for an approximation (1.27) *of a digital control system* (1.9) *and a family of piecewise constant functions $u_T : \mathbb{R} \to \mathbb{U}$ for $T \in (0, T^\star]$ depending on the initial value x_0 and $\mathcal{N} \subset \mathbb{R}^n$ to be a neighborhood of the origin.*

(1) Then the exact digital control system (1.9) *is semiglobally practically asymptotically stable for the family $(u_T, f^{(a)})$ if the following holds:*

- *(a) The family $(u_T, f^{(a)})$ is multi–step consistent with (u_T, f).*
- *(b) There exists a real–valued constant $\hat{T} > 0$ such that the approximated digital control system* (1.27) *is semiglobally practically asymptotically stable for the family $(u_T, f^{(a)})$.*

(2) Suppose the following conditions hold:

- *(a) The family $(u_T, f^{(a)})$ is multi–step consistent with (u_T, f).*

(b) There exists a semiglobal practical family of Lyapunov functions V_T according to Definition 1.46 for the family $(u_T, f^{(a)})$ and for all compact sets $A \subset \mathbb{R}^n \backslash \{0\}$ there exist real–valued constants $M, \hat{T} > 0$ such that the Lipschitz condition

$$|V_T(x) - V_T(y)| \leq M\|x - y\|$$

holds for all $x, y \in A$ and all $T \in (0, \hat{T})$.

Then the exact digital control system (1.9) *is semiglobally practically asymptotically stable for the family $(u_T, f^{(a)})$.*

In order to show semiglobal practically asymptotic stability of the sampling solution with zero order hold, see Definition 1.21, we can utilize results of [171] showing equivalence of the mentioned stability property of the exact sampled–data system and the sampling solution with zero order hold. Although this is not shown for the semiglobal practical case, the stated proof in [171] can be extended to cover this issue.

Theorem 1.51 (Stability of the Sampling Solution with Zero Order Hold)
Consider a sampling system with zero order hold according to Definition 1.21 and the corresponding exact sampled–data system (1.9). *The sampling system with zero order hold is (σ, ρ) semiglobally practically asymptotically stable for all $t \geq 0$ if and only if the following holds:*

(1) The exact sampled–data system is semiglobally practically asymptotically stable.

(2) The sampling solution is bounded over T, that is, there exists a function $\gamma \in \mathcal{K}_\infty$ such that

$$\|x_T(t, x_0, u)\| \leq \gamma(\|x_T(t_i, x_0, u)\|) \tag{1.30}$$

holds for all $t \in [t_i, t_i + T)$ and all $x_0 \in B_\sigma^0(0)$.

Now we have shown how our numerical examples are embedded within the theoretical background of digital control systems in Theorem 1.50. Together with Theorem 1.51 we have seen that, given stated conditions, if the stability property can be generated for the approximated digital control system, then the exact system (1.9) and the sampling system in continuous–time (1.8) inherit this property. Our aim in the following chapters is to derive a method to compute a control law which introduces the desired stability property. Moreover, we develop and discuss conditions for this method guaranteeing semiglobal practically asymptotic stability.

Chapter 2

Model Predictive Control

In this chapter we focus on design methods of control laws for given nonlinear control systems. Again, we distinguish between open–loop and closed–loop controls, see Definitions 1.24 and 1.25 and the explanations thereafter. The design method we aim to describe lies somewhere in the middle of these two concepts, i.e. it is based on an open–loop control computation but implements it in a closed–loop fashion. Our approach to introduce this technique is close to the conceptual ideas which led to its development.
We start with a historical review on control in Section 2.1. Thereafter, we introduce the concept of *optimal control* for continuous–time control problems and the definitions of *constraints* and *cost functionals* in Section 2.2. In particular, we discuss how the stability property of the underlying system is replaced by the implicit construction of the cost functional. In the following Section 2.3 we use these terms in the context of digital control. For the considered setup we derive an initial characterization of a control which exhibits desired properties, that is to minimize a user defined cost and to satisfy given restrictions. Since the calculation of the mentioned control is computationally expensive or even unsolvable, we focus on the so called *receding horizon control* approach in Section 2.4. The resulting control problem, its properties and an implementation of a solution method are the main issue of this thesis. We conclude this chapter with a short classification of other important control design techniques in Section 2.5 and discuss advantages and disadvantages of the receding horizon control method.

2.1 Historical Background

First of all, we have to mention that the stability concept cannot only be found in control theory and its applications, but it also occurs in nature. To give an example, we refer to predator–prey–relationships between two or more species [71, 221] which occasionally exhibit convergent development towards an equilibrium such that birth and death rate are equal. In this thesis, we focus on controller design concepts for artificial applications which aim at establishing such a property.

The evolution of control theory can be divided into four major periods:
The first period dates back to 6000 BC when irrigation was used for the probably first time in Mesopotamia and Egypt. At that time, barley was grown in areas where the natural rainfall was insufficient to support such a crop [1] and the control mechanism was handled by humans. In contrast to that, one can find automatic closed–loop controls in roman aqueducts where water levels are kept constant through the use of various combinations of valves [166] where no human action is necessary during the operation process.

Back then, control was more try–and–error than actually understood. This was still the case in modern control theory which started during the 17th century when Christiaan Huygens (1629–1695) and Robert Hooke (1635–1703) designed pendulum clocks. Huygens' contributions arose out of his interest in developing accurate clocks for use in navigation. Huygens idea and main contribution was to avoid the jumps in the motion caused by the escapement mechanism of a clock. To this end, he developed a conical pendulum (1673). The resulting speed control design was (probably) the first published design of a control which was intended to eliminate the offset. Since he was working prior to the invention of the calculus, he did not exactly understand the way it worked at least in a mathematical sense.
Hooke also invented a conical pendulum (1666–1667), but he had not published much detail before Huygens' book on clocks appeared in 1673; he also worked on applications of this pendulum to govern the speed of an astronomical telescope (1674) and on spring–restored flyballs (1677). During the 18th century, these were used for speed control of windmills. The idea is to synchronize the flyball with the windmill. Then the centrifugal force due to the angular velocity causes the attached balls to rise. These balls are linked to the sails such that an upward movement affects the positions of the sails.
Yet, Huygens' and Hooke's idea of speed control did not become popular until James Watt (1736–1819) adapted it to flyball governors of steam engines. Here, the speed of the engine is to be regulated despite of a variable load. In particular, a steady–state error can appear which leads to variations of the integral feedback idea in order to deal with this problem.
The first attempt to systematically analyze the governor was made by George Biddell Airy (1801–1892) in his works on dynamical control theory [2, 3] in which he gave a "good" governor for an astronomical telescope. The first complete mathematical analysis of a governor and its properties was published by James Clerk Maxwell (1831–1879) in 1868, see [157]. Because of this work which deals with erratic behaviour as well as the effects of integral control, he is called the founder of control theory. Additionally, it gave rise to the first wave of theoretical research in control.
For more details about this period we refer to [74, 75, 161].

The second period began in the early 1900s and was characterized by a strict mathematical treatment of control processes and properties in many applications. In particular, one has to mention the work of Aurel Boreslav Stodola (1859–1942) and Max Tolle (1846–1945) about controlling turbines and piston force machines [211, 212, 217]. Tolle's book can be seen as the first systematic textbook in control theory. Also at the same time Adolf Hurwitz (1859–1919) and Edward John Routh (1831–1907) developed characterizations of stability of linear systems, see [199].
In the upcoming 1930s Karl Küpfmüller (1897–1977), Harry Nyquist (1889–1976) and Hendrik Wade Bode (1905-1982) (among others) analyzed electrical circuits and raised the theory of feedback amplifiers to guarantee stability and appropriate responses of their devices [25, 26, 133, 176]. Their work is up till now the basis of frequency design. Moreover, analog computers appeared almost at the same time and were immediately used for controlling purposes.

Around the year 1940 the third period started. At that time, Adolf Leonhard (1899–1995) and Winfried Oppelt (1912–1999) published their work which made control theory a uniform, autonomous and systematic engineering science [141]. Additionally, Rudolf Oldenbourg (*1911) and Hans Sartorius (*1913) as well as Hendrik Wade Bode gave the

first complete mathematical analysis of the dynamics of automatic controls in [27, 177]. These so-called classical approaches were limited for the most part by their restriction to linear time–invariant systems with scalar inputs and outputs. Only during the 1950s, control theory began to develop powerful general techniques that allowed treating multivariable time–varying systems, as well as many nonlinear problems. At the end of World War II, Norbert Wiener (1894–1964) introduced the term "cybernetics" for control theory and related areas [230]. These years were dominated by statistical control treated by Norbert Wiener [229, 231], digital control [219] by John Groff Truxal (1923–2007) and nonlinear control by Robert Lien Cosgriff [46]. Moreover, the techniques of control units were widely unified and the concepts for pneumatic and electronic PID controller were developed [23].

The last period began by 1960 when "modern" control theory started to evolve. It is characterized by the use of electronic calculating machines to solve complex problems. The use of machines made it possible to introduce optimal control methods such as the *maximum principle* by Lew Semjonowitsch Pontryagin (1908–1988) et al. [181] in the Soviet Union as well as the *dynamic programming principle* by Richard Bellman (1920–1984) [17] in the United States. The concept of these modern optimal control methods required a new notation, the so called *state space notation*, which was (re)introduced by Rudolf Kalman (*1930) in [122] and particularly useful in his work on filtering [123]. Moreover, the state space notation allows for handling multivariable systems. These, in turn, caused mathematically rather complex problems to be considered which resulted in high computational and numerical costs.
Also in the 1960s, digital computers were used in digital control systems [137, 138]. These so called process computers allowed the handling of large measurement data and were able to take over the optimal guidance of the whole process. Soon, they were standard in monitoring, logging and controlling processes. However, centralizing attempts to integrate all these functions in just one unit stayed unsatisfactory and risky.
From 1960 onwards, also mathematical control theory evolved massively entering many different fields such as control of partial differential equations, see e.g. the works of Hector Fattorini (*1938), Jacques–Louis Lions (1928–2001) [145] and Fredi Tröltzsch (*1951) [57, 218], which is computationally extremely demanding and many analytical questions are still unanswered. Also distributed systems as well as delayed systems are considered using tools from functional analysis, see e.g. the works of Lions [146] or Jack Hale (*1928) [110] respectively.
In [31, 115, 119], Roger Brockett (*1939), Alberto Isidori (*1942) and Héctor Sussmann (*1946), among others, treated nonlinear control systems geometrically using Lie groups and identified control problems in both the Lagrangian and Hamiltonian framework. Moreover, passivity methods for stabilization as well as the controllability problem were considered by Brockett. Starting in the 1990s, a more concentrated interest in developing the theory of mechanical control systems occured which was coming from two directions. On the one hand, the interest in understanding the role of external forces and constraints arose in the theoretical research community, see e.g. [24, 174]. And secondly, coming from the application side, a focus was put on stabilization and passivity techniques for mechanical systems [9, 178]. For further details on the historical development of mathematical control theory see also [32, 153].
Mathematical control theory also influenced engineering sciences, for example the algebraization of frequency–domain analysis [122] is by now standard. The frequency–domain approach was and is still extensively used in powerful decentralized processes

which are controlled by microprocessors. This was possible from 1975 onwards due to their cheap availability. Yet, for security reasons, these systems were closed so no user could implement controllers which are different from the installed classical *PID controller* [147, 148, 222–224]. This was overcome around 1998 and since then many extensions have been implemented. Among these are *Fuzzy controllers*, see e.g. [124, 236], which were first introduced 1965 by Lotfi Zadeh (*1921) in his work [235], *neuronal networks* as seen in [55, 113, 120, 197], *Lyapunov function based* and *adaptive controllers*, cf. [12,72,129,132,165,220], and *model predictive controllers*, see e.g. [4,16,76,140,160,167].

In the following, we focus on the last mentioned controllers, so called *model predictive* or *receding horizon controllers*. Since this design technique is derived from optimal control, we start by defining necessary terms in the context of continuous–time control. Then, we show the equivalent for our considered setting of digital control systems before we state the main idea and mathematical description of the receding horizon control approach. Concluding, we compare the presented receding horizon control idea to its main competitors, the PID controller as well as the Lyapunov function based controller and the adaptive controller.

2.2 Continuous–time Optimal Control

From the historical development in control theory we can see that, prior to the last period, the goal for designing control laws is to obtain desired properties of the system, e.g. stability. For these laws, the effectiveness is a matter of the experience of the developer and one does not know whether the chosen implementation is actually the best one.
To overcome this issue, we introduce the so called *cost functional* and (possibly nonlinear) *constraints*. Note that the constraints are *additional* conditions which may depend on any of the parameters of the system. Yet, they implicitly exist within the prior controller designs where the controller is constructed to avoid violating the now explicitly considered boundaries which allows to neglect them during the process run.
Using the cost functional instead of the experience of a developer, we are not longer in the position to choose tuning parameters of the control law. As a result, we cannot avoid considering the mentioned bounds within the control law.
The cost functional, on the other hand, allows us to abstractly implement the task we want to achieve, e.g. stabilizing a given point or a trajectory or minimizing energy consumption. In particular, we do not model the control itself as for example in the PID approach, but implicitly define it via our goal. The cost functional itself corresponds to introducing a new, higher layer within the control system which governs the problem and simultaneously eases the mathematical formulation of our objective.
Hence, we traded the ease of designing the control law for considering constraints. Since these constraints already existed before the change of the problem formulation and required a good knowledge of the systems in order to design an appropriate control law, the incurred trade–off appears to be fair.
Here, we define constraints via the set of admissible states.

Definition 2.1 (Admissible set)
We refer to the nonempty set $\mathbb{X} \subset \mathbb{R}^n$ as the *set of admissible states*. Moreover, we call a solution to be *admissible* if $x_u(t, x_0) \in \mathbb{X}$ holds for all $t \in \mathbb{T}$.

Note that similarly $\mathbb{U} \subset \mathbb{R}^m$ implies the control to be constrained. For numerical purposes, we always think of the set $\mathbb{X}$ as a set being bounded by some functions which may depend on the actual state and/or the control.

Definition 2.2 (Vector of Constraints)
Consider $s \in \mathbb{N}$ with $s \geq 1$.

(1) Given a set of mappings $c_k : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, $k \in \{1, \ldots, s\}$, we call the inequalities

$$c_k(x_u(t, x_0), u(t)) \geq 0, \qquad \forall t \in \mathbb{T} \tag{2.1}$$

mixed constraints.

(2) For a set of mappings $c_k : \mathbb{R}^n \to \mathbb{R}$, $k \in \{1, \ldots, s\}$, the inequalities

$$c_k(x_u(t, x_0)) \geq 0, \qquad \forall t \in \mathbb{T} \tag{2.2}$$

are called *pure state constraints.*

(3) Given $s_m, s_z, s \in \mathbb{N}$, $s := s_m + s_z$ and $c : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^s$ with

$$c_k(x_u(t, x_0), u(t)) := \begin{cases} c_k(x_u(t, x_0), u(t)), & k = 1, \ldots, s_m, \\ c_k(x_u(t, x_0)), & k = s_m + 1, \ldots, s, \end{cases}$$

we call

$$c(x_u(t, x_0), u(t)) \geq 0 \qquad \forall t \in \mathbb{T} \tag{2.3}$$

the *vector of constraints.*

Here, $\mathbb{T}$ can be any set, hence the admissible set and the constrained vector are defined for discrete–time systems

$$x_u(n+1) := f(x_u(n), u(n)), \quad n \in \mathbb{N}, \qquad x_u(0) = x_0$$

as well as for continuous–time systems

$$\dot{x}_u(t) := \frac{d}{dt} x_u(t) = f(x_u(t), u(t)), \quad t \in \mathbb{R}, \qquad x_u(0) = x_0$$

We now focus on the continuous–time case $\mathbb{T} = \mathbb{R}$.
The first aspect, the cost functional, allows us to incorporate the desired property of the system directly into the problem. Typical goals are to stabilize a given *setpoint* $x^\star$ which is also called *regulation*, or *track a provided reference trajectory* $x_{\text{ref}}(\cdot)$.
This functional points out solutions, i.e. control functions $u \in \mathcal{U}$, which fulfill this task at best in the sense of the cost functional. Hence, it can be seen as a measure of the quality or performance of a solution. Since we utilize regulating as well as tracking type examples, we define the cost functional in the more general tracking form.

Definition 2.3 (Costfunctional)
Consider $L : \mathbb{R}^n \times \mathbb{U} \to \mathbb{R}_0^+$ to be a continuous nonnegative function and $u \in \mathcal{U}$ to

be a given control function which uniquely defines a solution $x_u(\cdot,\cdot)$ of (1.3), (1.5) and $x_{\text{ref}} : \mathbb{R} \to \mathbb{R}^n$ to be a given function. The functional $J : \mathbb{R}^n \times \mathcal{U} \to \mathbb{R}$ given by

$$J_\infty(x_0, u) := \int_0^\infty L\left(x_u(t, x_0) - x_{\text{ref}}(t), u(t)\right) dt \tag{2.4}$$

is called *infinite horizon cost functional*. In this context, the function $L(\cdot,\cdot)$ is called *running cost* and $x_{\text{ref}}(\cdot)$ denotes the *reference function*. If there exist constraints and the solution $x_u(\cdot, x_0)$ violates one of these at some point in time, then we set $J_\infty(x_0, u) := \infty$.

Remark 2.4
By setting $x_{ref}(\cdot) \equiv x^\star$ we obtain the setpoint form of the functional. In the following Chapters 3 and 4, the setpoint form with $x^\star = 0$ will be the basis of our analysis, see also Remark 1.40.

Given this functional we seek a minimizing function u and obtain the following problem:

$$\begin{aligned}
\text{Find } u(x_0,\cdot) &:= \operatorname*{argmin}_{u\in\mathcal{U}} J_\infty(x_0,u) \\
\text{ST. } J_\infty(x_0,u) &= \int_0^\infty L\left(x_u(t,x_0) - x_{\text{ref}}(t), u(t)\right) dt \\
\dot{x}_u(t) &= f(x_u(t,x_0),u(t)), \qquad \forall t \in \mathbb{R} \\
x_u(0,x_0) &= x_0 \\
x_u(t,x_0) &\in \mathbb{X} \qquad \forall t \in \mathbb{R} \\
u(t) &\in \mathbb{U} \qquad \forall t \in \mathbb{R}
\end{aligned} \tag{OCP$_\infty$}$$

Since our main focus lies on digital control, we do not go into more detail on solving such a problem. Results concerning this topic can, among others, be found in [77,181,190,191].

2.3 Digital Optimal Control

We now turn our focus on so called *sampled–data systems*

$$x_u(n, x_0) := x_T(t_n, x_0, u)$$

according to Definition 1.23. In contrast to the continuous–time setting of the previous section, we are going to deal with discrete–time control systems and hence we have to adapt the problem description.
Since the general structure of the control is known in advance, i.e. the points of discontinuity are fixed, we can derive the digital optimal control problem directly from problem OCP$_\infty$. In particular, we are looking for a piecewise constant control which minimizes a given cost functional. This control law can be fully characterized by a control sequence, i.e. by an element of $\mathbb{U}^{\mathbb{N}_0}$, instead of a control function. Considering the case of sampled–data systems, this allows for incorporating the sampling aspect into the control design.
Since we can describe a piecewise constant control function using an element of the set $\mathbb{U}^{\mathbb{N}_0}$, we can minimize the cost functional over this set. Moreover, we can make use

of Definition 1.23 to rewrite the solution of the control system as an element in $\mathbb{X}^{\mathbb{N}_0}$. This allows us to map the state trajectory and control function on the discrete–time grid $\mathbb{T} = (t_i)_{i\in\mathbb{N}_0}$ representing all sampling instants.
Now, we modify the considered cost functional from Definition 2.3 and restate it using only the discrete–time representation of the control system.

Definition 2.5 (Costfunctional)
Given a continuous nonnegative function $l : \mathbb{R}^n \times \mathbb{U} \to \mathbb{R}_0^+$, a reference function $x_{\text{ref}} : \mathbb{N}_0 \to \mathbb{R}^n$ and a given control sequence $u \in \mathbb{U}^{\mathbb{N}_0}$ which uniquely defines a solution $x_u(\cdot,\cdot)$ of (1.10), the functional $J : \mathbb{R}^n \times \mathbb{U}^{\mathbb{N}_0} \to \mathbb{R}$ given by

$$J_\infty(x_0, u) := \sum_{i=0}^{\infty} l(x_u(i, x_0) - x_{\text{ref}}(i), u(i)) \tag{2.5}$$

is called the *infinite horizon cost functional*. Moreover, the function $l(\cdot,\cdot)$ is called *stage cost*.

Remark 2.6
The cost functional (2.4) *is a special case of* (2.5) *since we can choose*

$$l(x_u(i, x_0) - x_{ref}(i), u(i)) = \int_{t_i}^{t_{i+1}} L(x_T(t, x_0, u) - x_{ref}(t), u(t_i))dt.$$

where $x_{ref}(\cdot)$ represents a continuous–time version of the reference signal.

Given the cost functional (2.5), we seek a sequence $u \in \mathbb{U}^{\mathbb{N}_0}$ which minimizes $J_\infty(x_0,\cdot)$ given an initial value x_0. This defines a nonlinear distance which is also called the optimal value function.

Definition 2.7 (Optimal Value Function)
We define the optimal value function via

$$V_\infty(x_0) := \inf_{u\in\mathbb{U}^{\mathbb{N}_0}} J_\infty(x_0, u). \tag{2.6}$$

Remark 2.8
For reasons of simplicity, we assume that the minimum with respect to $u \in \mathbb{U}^{\mathbb{N}_0}$ is attained in (2.6)*. Moreover, we consider the initial time to be fixed which allows us to consider the optimal value function* (2.6) *to be autonomous, see also Remark 1.40.*

The function (2.6) can be used to formulate the so called *Optimality Principle* which was first stated by Bellman in 1957, see [17]. Geometrically it says that one can split up the distance along an optimal state trajectory. Additionally, we have that endpieces of optimal trajectories are again optimal.

Theorem 2.9 (Bellman's Optimality Principle)
Suppose $u^\star$ minimizes the cost functional (2.5) *for a given initial value $x_0 \in \mathbb{X}$ and a reference $x_{ref}(\cdot)$ such that all constraints are satisfied. Then, for all $m \in \mathbb{N}_0$ we have*

$$V_\infty(x_0) = \sum_{i=0}^{m} l(x_{u^\star}(i, x_0) - x_{ref}(i), u^\star(i)) + V_\infty(x_{u^\star}(m, x_0)). \tag{2.7}$$

Using this principle we can characterize an optimal control. Note that this control depends on the choice of the cost functional and hence we can only guarantee optimality for this particular choice.

Theorem 2.10 (Optimal Feedback Law)
Given an initial value $x_0 \in \mathbb{X}$ and a reference $x_{ref}(\cdot)$, the mapping μ defined via

$$u^\star(n) := \mu(x_u(n, x_0)) := \operatorname*{argmin}_{v \in \mathbb{U}} \{V_\infty(x_v(1, x_u(n, x_0))) + l(x_u(n, x_0) - x_{ref}(n), v)\} \quad (2.8)$$

minimizes (2.5) *and is called optimal feedback or infinite horizon control law.*

Proofs of Theorems 2.9 and 2.10 can be found in Chapter 8.1 of [210] using a discrete–time setting which is equivalent to the sampled–data setting considered here.
Similar to the continuous–time optimal control problem OCP_∞, we can summarize the digital or sampled–data optimal control problem SDOCP_∞.

$$\begin{aligned}
\text{Find } u(x_0, \cdot) &:= \operatorname*{argmin}_{u \in \mathbb{U}^{\mathbb{N}_0}} J_\infty(x_0, u) \\
\text{ST. } J_\infty(x_0, u) &= \sum_{i=0}^{\infty} l(x_u(i, x_0) - x_{\mathrm{ref}}(i), u(i)) \\
x_u(i+1, x_0) &= f(x_u(i, x_0), u(i)) \qquad \forall i \in \mathbb{N}_0 \\
x_u(0, x_0) &= x_0 \\
x_u(i, x_0) &\in \mathbb{X} \qquad \forall i \in \mathbb{N}_0 \\
u(i) &\in \mathbb{U} \qquad \forall i \in \mathbb{N}_0
\end{aligned} \quad (\mathrm{SDOCP}_\infty)$$

Note that Theorem 2.10 offers a possible solution method for the digital stabilization problem. However, we do not follow this path since it is often hard, if not impossible, to compute the value function $V_\infty(\cdot)$ which is necessary to compute the feedback. To avoid the burden of solving this problem, we circumvent the computation of a closed solution and consider the so called receding horizon control approach instead.

2.4 Receding Horizon Control

The concept of receding horizon control (RHC), also called model predictive control (MPC) or moving horizon control (MHC), is not a very new one. In 1963 Anatoli Propoĭ described such a controller for the linear case [189] and in 1967 Ernest Lee and Lawrence Markus stated the RHC procedure as it widely used today, see [139].
The fundamental idea of RHC is to solve the optimal control problem SDOCP_∞ for the current system state only on a finite horizon of N sampling instants. Such a control can be computed using e.g. methods from nonlinear optimization, see Chapter 5, but the solution is unfeasible for the infinite horizon control problem since it is not an infinite sequence. For this reason it needs to be prolongated, e.g. by using a *default value* $u(n) \in \mathbb{U}$ for all sampling instances $n > N$. Within our implementation of the receding horizon controller we use the default value

$$u(n) := 0 \qquad \forall n > N. \quad (2.9)$$

The resulting infinite control sequence, however, does not minimize the infinite cost functional (2.5) and, in particular if the system under control is unstable, it may not even be stabilizing.
To solve this problem, only the first part of the resulting input signal is applied open–loop to the system until the next recalculation instant. For this point in time the optimal control problem is solved for the new state of the system. Hence, an often intractable problem is replaced by a series of tractable ones.
In [139], this method is described as follows:

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

Within the literature, one usually distinguishes between linear and nonlinear receding horizon control. Linear RHC refers to control schemes based on linear dynamics of the system, a linear or quadratic cost functional and linear constraints of the state and control variables. In contrast to that nonlinear model dynamics, nonlinear contraints on process variables and a non–quadratic cost functional are characteristics of nonlinear RHC. Linear receding horizon control is by now widely used in industrial applications, see [15,16,54,73, 76,167]. Moreover, much is known about theoretical and implementation issues as shown in the survey articles [140, 160, 167] and references therein.

For nonlinear receding horizon control, the situation is fundamentally different. Similar to linear RHC its development was driven by industry, not by academia. Reasons for the great need in industry are in particular tighter environmental regulations, quality specifications and higher productivity demands which are in general nonlinear. Since operating points close to those constraints offer the highest output in terms of the chosen cost functional, linear models cannot be used instead. Still, there exists a large amount of software and applications of this method, see e.g. [15, 16, 54], although its theoretical background is not completely understood. Especially the compensation of naturally occuring delays and the design of robust nonlinear RHC are great open questions. On the other hand, there exists quite a range of results for certain setups, for more details we refer to [5, 6, 59, 60, 62, 68, 69] for the continuous–time case, [49, 91, 160, 193] for the discrete–time case. Stability and robustness issues of both cases are also discussed in [4, 39, 50, 61, 87, 95, 102, 126].

This thesis extends results for nonlinear receding horizon control by introducing a performance index for the quality of the closed–loop solution. For this performance index, several online evaluatable methods are derived and their effectiveness has been shown using numerical experiments. In a second step, the standard receding horizon control concept itself is extended via adaptation algorithms which aim at guaranteeing a certain performance index. These procedures have also been implemented and numerical simulations have shown that a significant decrease in the required computing time can be expected by using these methods.

Now, we introduce the concept of receding horizon control more formally and thereafter we schematically compare the general implementation of such a controller and other conventional ones. The intention of this section is to provide an intuitively understandable

basis for the stability and suboptimality analysis in Chapter 3 and its extension towards adaptive receding horizon control in Chapter 4. For overviews on RHC results we refer to [5, 39, 49, 160, 167].
As stated before we now modify the digital control problem SDOCP_∞ by "cutting off" the end of the infinite horizon. In particular, we consider the cost functional (2.5) but truncate it after a certain point N in time. In order to define this functional properly, we first need to define the state trajectories and possible control sequences which we are going to consider. Since both are computed in an open–loop manner, see Definition 1.24, we refer to them as open–loop trajectories and open–loop controls.

Definition 2.11 (Finite Open–loop Control)
Consider the time set $\mathcal{I}_u := \{0, \dots, N-1\}$. A function $u_N : \mathcal{I}_u \to \mathbb{U}$ based on some initial condition x_0 is called a *finite open–loop control law*.

For notational purposes, we use $u_N(x_0, i)$ to represent the i-th control value within the open–loop control sequence corresponding to the initial value x_0 when it is necessary to distinguish between two or more different open–loop controls.

Definition 2.12 (Finite Open–loop State Trajectory)
Consider the time set $\mathcal{I}_x := \{0, \dots, N\}$. A solution of (1.9)

$$x_{u_N}(i+1, x_0) = f\left(x_{u_N}(i, x_0), u_N(x_0, i)\right) \tag{2.10}$$

emanating from the initial condition $x_{u_N}(0, x_0) = x_0$ using a finite open–loop control law u_N according to Definion 2.11, is called an *finite open–loop state trajectory* or *open–loop solution*.

The interval of interest is given by $\mathcal{I}_x = \{0, \dots, N\} \subset \mathbb{N}_0$ for the state trajectory. Note that the sequence of control values cannot be defined as a mapping from $\mathcal{I}_x$ to $\mathbb{U}$ since it only contains N and not $N+1$ elements. To this end, we introduced the set $\mathcal{I}_u$.

Definition 2.13 (Finite Costfunctional)
Given a continuous nonnegative *stage cost* function $l : \mathbb{R}^n \times \mathbb{U} \to \mathbb{R}_0^+$ and a given control sequence $u_N \in \mathbb{U}^{\mathcal{I}_u}$ which uniquely defines a solution x_u of (1.10), the functional $J : \mathbb{R}^n \times \mathbb{U}^{\mathcal{I}_u} \to \mathbb{R}$ given by

$$J_N(x_0, u_N) = \sum_{i=0}^{N-1} l(x_{u_N}(i, x_0) - x_{\mathrm{ref}}(i), u_N(i)). \tag{2.11}$$

is called the *reduced* or *finite horizon cost functional*.

Remark 2.14
The form of the cost functional in (2.11) *is also termed Lagrangian form. Within the receding horizon control scheme it is also possible to use the so called Mayer term, i.e.* $J_N(x_0, u_N) = F(x_{u_N}(N, x_0))$*, which represents a function of the end point of the trajectory* x_{u_N} *on the finite horizon. This term is often considered for stability reasons, see [39, 117, 160] and the discussion in Section 3.4. Moreover, a mixture of these two forms can be used which is then called a cost functional of Bolza–type. In the following, we focus on the Lagrangian form only.*

Similar to the infinite horizon problem SDOCP_∞, the cost functional defines a nonlinear distance. In Chapter 3, we use this distance to analyze stability and suboptimality.

Definition 2.15 (Finite Optimal Value Function)
The function

$$V_N(x_0) := \inf_{u_N \in \mathbb{U}^N} J_N(x_0, u_N). \tag{2.12}$$

is called *finite optimal value function.*

Again, we assume that the minimum with respect to $u_N \in \mathbb{U}^{\mathcal{I}_u}$ is attained and the initial time is fixed, see Remark 2.8. Moreover, we obtain Bellman's Optimality Principle to hold in the finite case, cf. [18].

Theorem 2.16 (Bellman's Optimality Principle)
Consider $u^\star$ to minimize (2.11) *for a given initial value $x_0 \in \mathbb{X}$ satisfying all constraints. Then*

$$V_N(x_0) = \sum_{i=0}^{m} l(x_{u^\star}(i, x_0) - x_{ref}(i), u^\star(i)) + V_{N-m}(x_{u^\star}(m, x_0)) \tag{2.13}$$

holds for all $m \in \mathcal{I}_u$.

The finite optimal value function also points out minimizing controls which we are going to use in the receding horizon control setting.

Definition 2.17 (Minimizing open–loop Control)
The function $u_N : \mathcal{I}_u \to \mathbb{U}$ satisfying

$$u_N(x_0, \cdot) = \operatorname*{argmin}_{u_N \in \mathbb{U}^N} J_N(x_0, u_N) \tag{2.14}$$

is called *minimizing open–loop control.*

The computation of (2.14) requires us to solve the following problem:

$$\begin{aligned}
\text{Find } u_N(x_0, \cdot) &= \operatorname*{argmin}_{u_N \in \mathbb{U}^N} J_N(x_0, u_N) \\
\text{ST. } J_N(x_0, u_N) &= \sum_{i=0}^{N-1} l\left(x_{u_N}(i, x_0) - x_{\text{ref}}(i), u_N(x_0, i)\right) \\
x_{u_N}(i+1, x_0) &= f\left(x_{u_N}(i, x_0), u_N(x_0, i)\right) \quad \forall i \in \mathcal{I}_u \\
x_{u_N}(0, x_0) &= x_0 \\
x_{u_N}(i, x_0) &\in \mathbb{X} \qquad \forall i \in \mathcal{I}_x \\
u_N(x_0, i) &\in \mathbb{U} \qquad \forall i \in \mathcal{I}_u
\end{aligned} \tag{SDOCP$_\text{N}$}$$

Schematically this can be described as shown in Figure 2.1.
The underlying digital control system is simulated over a finite horizon using a model of the system under control. The resulting trajectory is used to generate an open–loop control which minimizes the given cost functional. At the same time all constraints of the process have to be satisfied on the simulated finite horizon. Since this internal computation is usually done iteratively, we represent it as an internal feedback loop in Figure 2.1 which illustrates this setup graphically.

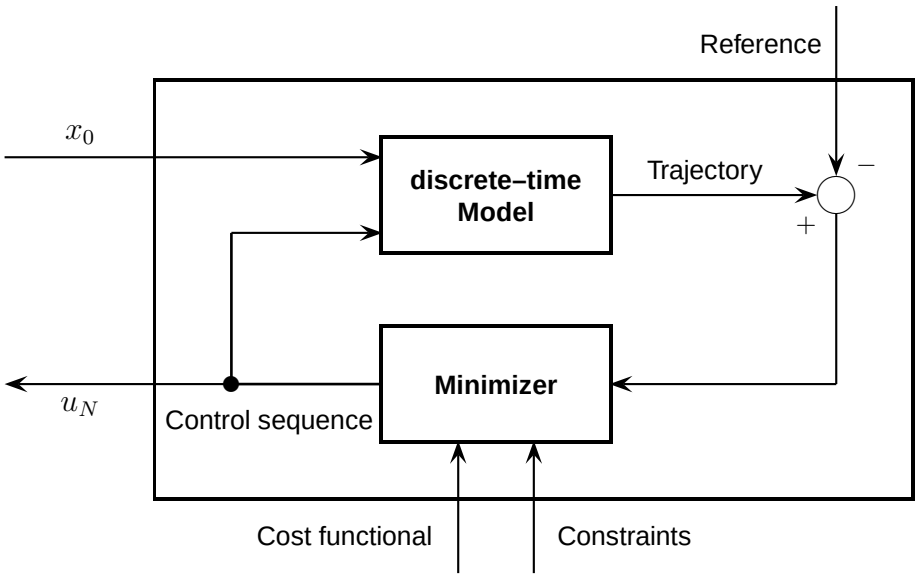


Figure 2.1: Schematic representation a receding horizon controller

In order to obtain an infinite control sequence from this setting we use the following iterative procedure:

Step 1: First, we solve the problem $\mathrm{SDOCP_N}$ for a given initial value x_0^n for the n-th iterate of the procedure. As a result, we obtain a prediction of the trajectory on the considered time horizon and the output of the RHC block in Figure 2.1, the minimizing control sequence $u_N \in \mathbb{U}^{N_u}$.

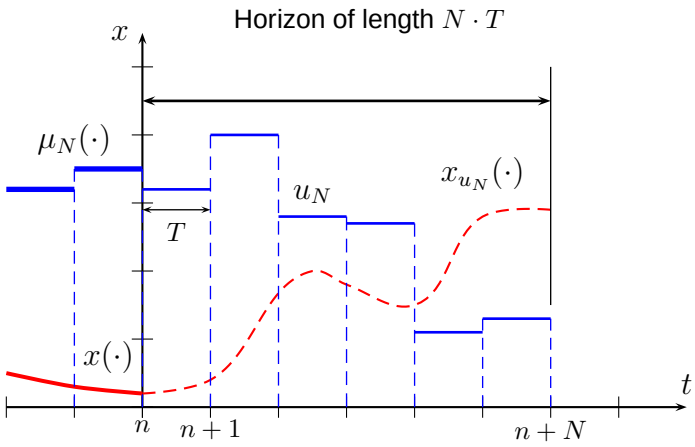


Figure 2.2: Step 1 of the RHC algorithm

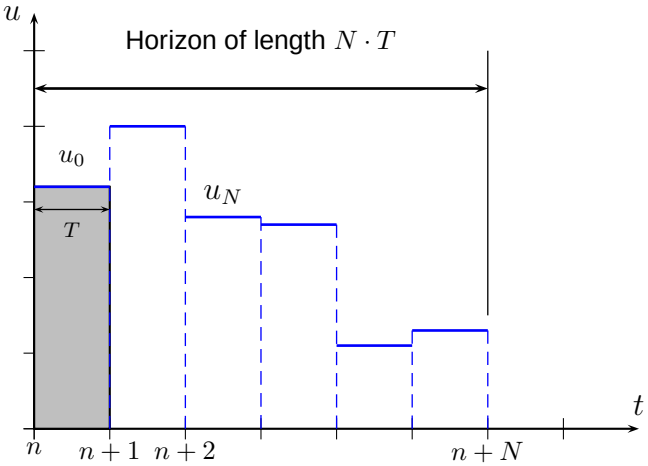


Figure 2.3: Step 2 of the RHC algorithm

Step 2: Now, the first element of the RHC output sequence $u(x_0, \cdot)$ is transmitted to and implemented by the actuator, see also Figure 2.5. The remainder of this control sequence can be dropped or reused as an initial guess for the control in the case that an iterative solver is employed to calculate the solution of SDOCP_N.

Step 3: In the last step, we obtain a new measurement or estimate of the current state x at the end of the sampling period. At this point, a new control value has to be computed and applied. Hence, the state x is transmitted to the receding horizon controller, again see Figure 2.5, and we can define a new problem of type SDOCP$_N$ with a shifted optimization horizon and initial value $x_0^{n+1} = x$.

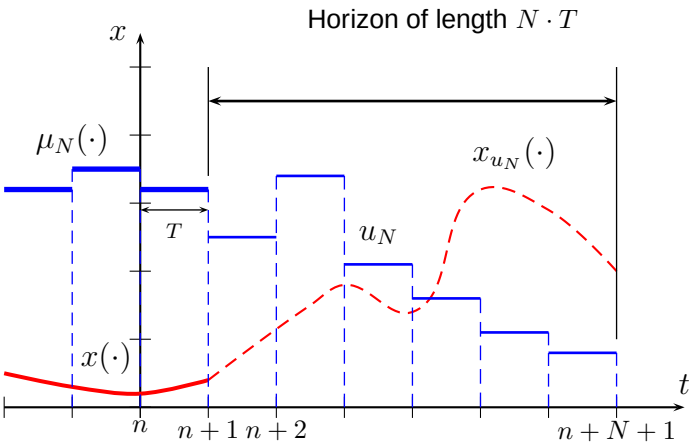


Figure 2.4: Step 3 of the RHC algorithm

Hence, we obtain an infinite control sequence from this setting by defining a feedback law μ_N as the recurrent implemention of the first element of the optimal control sequence u_N. Using Bellman's principle of optimality for the optimal value function $V_N(\cdot)$, we can formally define the following:

Definition 2.18 (Closed–loop Control)
Given an initial value $x_0 \in \mathbb{X}$ and a reference $x_{\text{ref}}(\cdot)$, the function $\mu_N : \mathbb{X} \to \mathbb{U}$ satisfying

$$\mu_N(x_0) := \left[\underset{u_N \in \mathbb{U}}{\operatorname{argmin}} \left\{ V_{N-1}(x_0) + l(x_0 - x_{\text{ref}}(0), u_N(0)) \right\} \right]_{[0]} \tag{2.15}$$

is called *closed–loop* or *receding horizon feedback control* where we use the short notation $u_{[0]} := u_N(x_0, 0)$.

The computation of such a closed–loop control on the infinite time interval can be done by solving the following infinite sequence of finite optimal control problems, the so called *receding horizon control problem*:

$$\begin{aligned}
\text{Find } \mu_N(x(n)) &:= u_{[0]} \\
\text{ST. } u(x(n), \cdot) &= \underset{u_N \in \mathcal{U}_N}{\operatorname{argmin}} J_N(x(n), u_N) \\
J_N(x(n), u_N) &= \sum_{i=0}^{N-1} l\left(x_{u_N}(i, x(n)), u_N(x(n), i)\right) \\
x_{u_N}(i+1, x(n)) &= f\left(x_{u_N}(i, x(n)), u_N(x(n), i)\right) \quad \forall i \in \mathcal{I}_u \\
x_{u_N}(0, x(n)) &= x(n) \\
x_{u_N}(i, x(n)) &\in \mathbb{X} \qquad \forall i \in \mathcal{I}_x \\
u_N(x(n), i) &\in \mathbb{U} \qquad \forall i \in \mathcal{I}_u
\end{aligned} \tag{RHC$_N$}$$

Remark 2.19
Within the problem (RHC$_N$) the initial values $x(n)$ represent external data/measurements which have to be provided to the algorithm, see also Figure 2.1.

Using the control law (2.15), we obtain the following solution:

Definition 2.20 (Closed–loop Solution)
The trajectory

$$x_{\mu_N}(n+1, x_0) = f\left(x_{\mu_N}(n, x_0), \mu_N(x_{\mu_N}(n, x_0))\right) \tag{2.16}$$

emanating from the initial value $x_{\mu_N}(0, x_0) = x_0$ with closed–loop control μ_N according to (2.15) is called *closed–loop solution* of the problem $\mathrm{RHC_N}$.

Remark 2.21
Using the closed–loop control we can rewrite the optimal value function on a finite horizon by

$$V_N(x_0) = \sum_{i=0}^{N-1} l(x(i), \mu_{N-i}(x(i))) \tag{2.17}$$

where $x(0) = x_0$ and $x(i+1) = f\left(x(i), \mu_{N-i}(x(i))\right)$. Note that different to the closed–loop solution (2.16) *we now use the recomputed optimal control μ_{N-i} in a receding horizon fashion. In this case, the closed–loop solution is identical to the open–loop solution $x_{u_N}(\cdot, x_0)$ since by principle of optimality endpieces of optimal solutions are again optimal.*

Note that the optimal control problem $\mathrm{SDOCP_N}$ cannot be solved instanteously in practice. Hence, the initial value used as basis point of the optimization is in general a prediction of the current state or an estimate calculated from available measurements. Due to the time delay caused by the computation, it may be necessary to postpone closing the control loop and to implement more than the first element of the control sequence. Considering Bellman's principle of optimality (2.13), we can also characterize the first m elements of the control sequence u_N for $m \in \mathcal{I}_u$.

Definition 2.22 (m–step closed–loop Control)
The function $\mu_N : \mathbb{N}_0 \to \mathbb{U}^{\mathcal{I}_u^m}$ satisfying

$$\mu_{N,m}(x_0) := \left[\operatorname*{argmin}_{u_N \in \mathbb{U}} \left\{ V_{N-m-1}(x_{\mu_{N,m}}(m, x_0)) + \sum_{j=0}^{m-1} l(x_{\mu_{N,m}}(j, x_0), u_N(j)) \right\} \right]_{[0,m-1]} \tag{2.18}$$

is called *m–step closed–loop* or *m–step receding horizon feedback control*. The parameter m is called the *control horizon*.

Remark 2.23
Such a feedback can be useful for several situations:

(1) The computing time necessary to solve the optimization problem $SDOCP_N$ is larger than the sampling time and parallel computations are not realizable.

(2) If the system is exponentially controllable or finite time controllable, then the performance of the control may be improved using longer control horizons, i.e. $m > 1$. In some cases, one can also obtain stability if the control horizon is enlarged, cf. [99].

(3) In the context of networked control systems, delays or package dropouts naturally lead to situations which force us to utilize larger control horizons. In this case, the implemented control horizon is unknown prior to computation and one has to consider this parameter to be variable along the closed–loop, see e.g. [101].

However, one has to keep in mind that the robustness of the algorithm concerning disturbances may be lost due to update delays within the control law.

Within the continuous–time setup of the plant, we can implement the defined receding horizon controller in the following fashion:

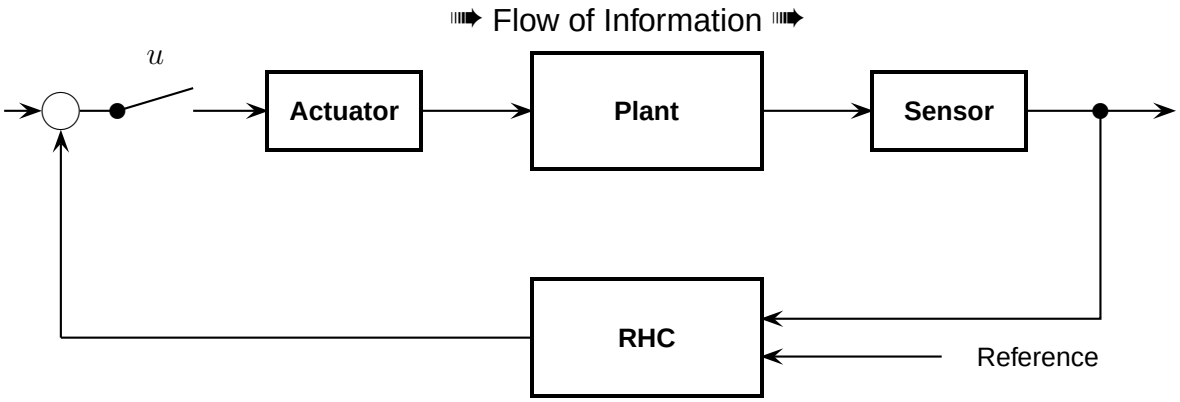


Figure 2.5: Schematic representation of the usage of a receding horizon controller

Using the technical terms we just introduced, we can now state the contribution of this thesis more precisely: First, we prove estimates for the performance of the control in the following Chapter 3, a concept which we call *suboptimality degree*. Within this analysis, we focus on the case $m = 1$, however, we also expect these results to hold for increased control horizon lengths. Secondly, based on these estimates, we introduce strategies to adapt the optimization problem $\mathrm{SDOCP_N}$ in order to guarantee a certain lower bound on the suboptimality degree while reducing the computing cost/time, see Chapter 4. Last, the chosen implementation of the receding horizon controller in the *PCC2*[1] (**P**redictive **C**omputed **C**ontrol 2) package is described in Chapters 5 and 6. The dependency of the required computing time on certain parameters within the numerical implementation is discussed in Chapter 8 using the examples from Chapter 7 and may also be used as a guideline to effectively implement and solve further examples.

Remark 2.24
To simplify our notation in the following chapters, we use no subscript for the used controller and no second argument corresponding to the initial state to denote the state by $x(\cdot) := x_{\mu_N}(\cdot, x_0)$ *if these components are clear from the context or if* $x(\cdot)$ *is used as a basis of our calculations only.*

2.5 Comparing Control Structures

Having introduced the fundamental terms and the strategy of receding horizon control, we now discuss advantages and disadvantages of this method compared to other standard control schemes.

[1] `http://http://www.nonlinearmpc.com`

2.5.1 PID Controller

In contrast to the state space or time domain setting which we considered until now, the *proportional–integral–derivative (PID) controller* is designed in the frequency domain using the Laplace transform (see e.g. [67])

$$F(s) = \int_0^\infty f(t)e^{-st}dt =: \mathcal{L}[f(t)]$$

and retranslated to the time domain using the inverse transform

$$f(t) = \frac{1}{2\pi j}\int_{c-i\infty}^{c+i\infty} F(s)e^{st}ds =: \mathcal{L}^{-1}[F(s)]$$

where $s \in \mathbb{C}$ denotes the state of the system in the frequency domain. Here, we adapt the standard notation used to design controllers in the frequency domain. The function $f(\cdot)$ denotes the development of the deviation of the state of the system for the reference trajectory. Moreover, we only give a short glance at the fundamental setup of PID controllers. For further details we refer to [147, 148, 222–224].

Definition 2.25 (PID Controller)
The function $G_C : \mathbb{C} \to \mathbb{C}$ satisfying

$$G_C(s) = K_R\left(1 + \frac{1}{T_I s} + T_D s\right)$$

is called *PID transfer function* in the frequency domain. The corresponding time domain control is given by

$$u(t) = K_R e(t) + \frac{K_R}{T_I}\int_0^t e(\tau)d\tau + K_R T_d \frac{\partial}{\partial t}e(t)$$

where $e(t) := x(t) - x_{\text{ref}}(t)$ denotes the deviation of the state from the desired reference.

The parameter K_R, T_I and T_D represent the amplification factor, the integral and derivative time respectively and can be used to set the resulting behavior of the system. Hence, an appropriate choice of these parameters causes the closed–loop system to be stable.

Remark 2.26
Due to device–related restrictions, the D–element cannot be implemented exactly. Instead, a retarded DT_1–element is used giving

$$G_C(s) = K_R\left(1 + \frac{1}{T_I s} + T_D\frac{s}{1+Ts}\right)$$

with $T \ll 1$.

In order to obtain the digital control problem a *sample–and–hold element* is used. Mathematically, we first apply an *impulse function* δ to the continuous–time signal where the impulse instants are located at the sampling instants. The resulting signal

$$f_{\text{sample}}(t) = f(t)\sum_{k=0}^{\infty}\delta(t-kT) = \sum_{k=0}^{\infty} f(kT)\delta(t-kT).$$

is called *sampling signal*. Next, we apply a holding element which is a square impulse of width T and height one, i.e. we apply the Emulation design (1.20), that is

$$f_{\text{hold}}(t) = \sigma(t) - \sigma(t-T).$$

Applying the Laplace transformation to both parts we obtain

$$F_{\text{sample}}(s) = \mathcal{L}[f_{\text{sample}}(t)] = \sum_{k=0}^{\infty} f(kT)e^{-kTs},$$
$$F_{\text{hold}}(s) = \mathcal{L}[f_{\text{hold}}(t)] = \frac{1-e^{-Ts}}{s}.$$

Combined, we obtain the Laplace transform of step function, that is

$$F_T(s) = F_{\text{hold}}(s)F_{\text{sample}}(s) = \frac{1-e^{-Ts}}{s}\sum_{k=0}^{\infty} f(kT)e^{-kTs}$$

and can apply the PID method to this input.

A typical implementation of PID controllers in applications is done in a very hierarchical setup as shown in Figure 2.6, see also [16]. It also shows how a receding horizon controller fits into this hierarchy and it intuitively explains the advantages of RHC in comparison to PID controllers. This structure is not a new one, in fact is was described in the 1970s/80s, see e.g. [188, 194].
One can see that both concepts reveal the same high level optimization concerning the whole plant as well as plant units, but also an identical local foundation at each machine is used. The latter represents a decentralized control system controlled by local PID controllers. For large plants, this is necessary on the one hand to compensate for disturbances, and one the other hand because the higher level computations still show too long computation latencies. On top of the hierarchy, there is always a plant–wide optimizer which determines the optimal steady–state settings for the single plant units. Note that such a controller could, again, be a receding horizon controller supervising all other controllers. However, this as well as the local optimization is in general done by humans.

In between these layer the designs differ. The main task of the dynamic constraint control part is to compute paths from one constrained steady–state to another one. Such a path should fulfill certain requirements like energy efficiency or other economical criteria. However, during this transition one has to minimize possible violations of constraints. In the classical setting, Figure 2.6 shows the interaction of PID algorithms, lead–lag (L/L) blocks and high/low select logic units to accomplish this task. It is particularly difficult to design an appropriate control structure to satisfy all control requirements. This is exactly the setup RHC has been designed for, so the entire dynamic constraint control part can be done using this algorithm. The possible sampling times of the RHC algorithm, however, are large and the method is computationally demanding compared to the PID controller. Still, its ability to explicitly consider nonlinear dynamics, constraints and the possibility to design ones interests, i.e. the goals of the local optimizer, in a cost functional makes RHC the algorithm of choice.

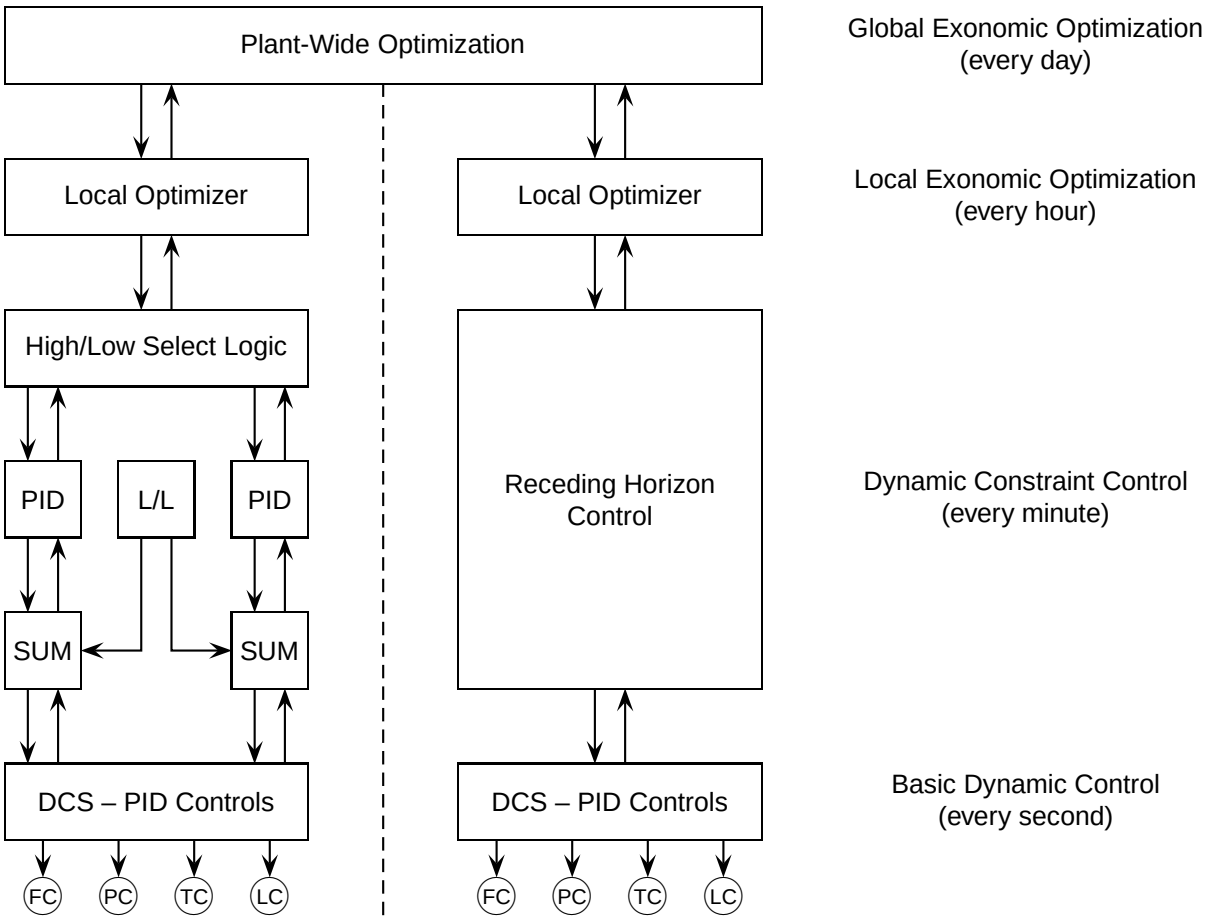


Figure 2.6: Hierarchy of control system functions in a typical processing plant. Conventional structure is shown at the left; RHC structure is shown at the right

2.5.2 Lyapunov Function Based and Adaptive Controllers

Another class are so called *Lyapunov function based controllers*. In contrast to a PID controller, these controllers are designed in the state space by an appropriate choice of a Lyapunov function for the system. For simplicity of exposition, we only consider the continuous–time control–affine case

$$\dot{x}(t) = f_0(x(t)) + \sum_{i=1}^{m} f_i(x(t))u_i(t). \tag{2.19}$$

Note that we can use for example the Emulation technique discussed in Section 1.3.3 to apply a resulting continuous–time controller in the considered digital setting as well. The controller is defined as follows:

Theorem 2.27 (Lyapunov Function based controller / Sontag's formula)
Consider $V : \mathbb{R}^n \to \mathbb{R}_0^+ \in \mathcal{C}^1(\mathbb{R})$ *to be a control Lyapunov function according to Definition 1.38 satisfying*

$$\inf_{u \in \mathbb{U}, \|u\| \leq \gamma(\|x\|)} \frac{\partial V(x)}{\partial t} f(x,u) \leq -\alpha(x) \tag{2.20}$$

for a function $\gamma \in \mathcal{K}$ *and* $x^\star = 0$ *to be the target equilibrium. Moreover, we suppose the derivative* $\frac{\partial V(x)}{\partial t}$ *to be Lipschitz continuous if* $x \neq 0$*. Then, the* i*-th component of the*

asymptotically stabilizing control function $u : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^m$ *is given by*

$$u_i(x(t)) = K_i(x(t)) := \kappa\left(a(x), \|B(x)\|^2\right) B(x)^\top \tag{2.21}$$

where

$$\kappa(a,b) := \begin{cases} -\frac{a+\sqrt{a^2+b^2}}{b(1+\sqrt{1+b})}, & \text{if } b > 0 \\ 0, & \text{if } b = 0 \end{cases} \tag{2.22}$$

and $a(x) := \frac{\partial V(x)}{\partial x} f_0(x)$, $B(x) := \frac{\partial V(x)}{\partial x}[f_1(x), \ldots, f_m(x)]$. *The control function* u *is called Lyapunov function based feedback or minimum norm feedback.*

Implementing this controller, the resulting time derivative $\frac{\partial V(\cdot)}{\partial t}$ is strictly negative along the closed–loop trajectory, hence the stabilizing property of controller (2.21) follows directly according to Lyapunov arguments, see e.g. [143] for a proof.

The class of *adaptive controllers* is closely related to the class of Lyapunov function based controllers. The concept of an adaptive controller is quite simple: It uses identification and parameter estimation algorithms to obtain a process or signal model, i.e. an adaptive controller is able to adapt itself if external signals occur or properties of the process under control change. In a second step, a preprogrammed controller design algorithm is used to recompute the parameter of the controller itself.
The development of this class of controllers began around 1960, the same time as receding horizon control was introduced, see e.g. [12,163,165,220] for some early reviews regarding mainly the continuous–time case without digitalization. Since 1970 adaptive controllers for digital control systems have become popular. This led to a change in the description technique, that is the discrete–time setting was considered almost exclusively. For more details we refer to [13, 232].
Similar to Chapter 1, we distinguish between two main concepts, an *externally influenced (direct) adaptive controller* and an *(indirect) adaptive controller with feedback*, see Figures 2.7 and 2.8 respectively.

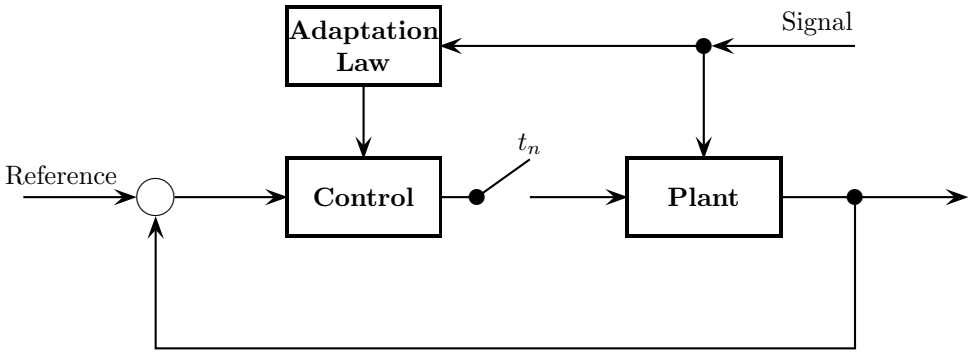


Figure 2.7: Schematic representation of an externally influenced adaptive controller

An adaptive controller is called externally influencable if internal changes of properties can be detected using measurable external signals, the relationship between these signals and the control law is known and this adaptation can be implemented as a control.
The missing backlink between the process output and the adaptation law is the main difference to an *adaptive controller with feedback*. This setup is particularly useful if the

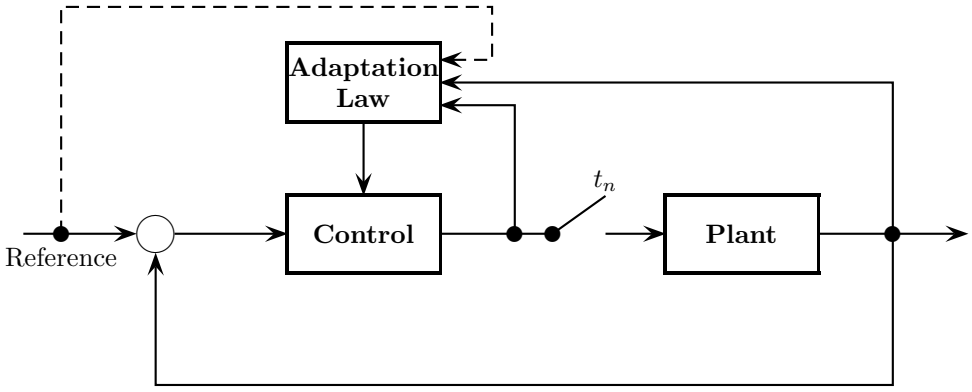


Figure 2.8: Schematic representation of an adaptive controller with feedback

internal changes can indeed not be detected directly and have to be recognized indirectly using the output of the plant. Here, one can utilize process identification algorithms to obtain the changing properties. The distinguishing mark of an adaptive controller with feedback is the second loop which superimposes the closed–loop control.

Since the receding horizon controller is representing a static state feedback law, we focus on the adaptive controller with feedback here. For these, another distinction is made by the *Lyapunov–based* and *estimation–based* designs which is more fundamental.
Within the Lyapunov–based approach, a reference model x_{ref} in continuous– or discrete–time is used to derive a Lyapunov function for the deviation of the state

$$e(t) = x(t) - x_{\text{ref}}(t)$$

and for the deviation of the system parameters. For reasons of simplicity, we restrict ourselves to the linear and continuous–time case. For a larger class of systems, this is done e.g. in [132].

Definition 2.28 (Adaptive Controller)
Consider the reference model

$$\dot{x}_{\text{ref}}(t) = A_M x_{\text{ref}}(t) + B_M u(t)$$

of the time–depending plant

$$\dot{x}(t) = A(t)x(t) + B(t)u(t)$$

and the Lyapunov function

$$V(e(t)) = e(t)^\top P e(t) + [(A_M - A(t))^\top F_A (A_M - A(t))] + [(B_M - B(t))^\top F_B (B_M - B(t))]$$

for this plant where P is the solution of $A_M^\top P + P A_M = -Q$. Then, the plant model is adapted by the *adaptation law*

$$\dot{A}(t) = F_A^{-1} P e(t) x(t)^\top$$
$$\dot{B}(t) = F_B^{-1} P e(t) u(t)^\top$$

and the adaptive controller is computed according to (2.21).

In this case, stability follows by the same Lyapunov arguments as for the Lyapunov based controller since the resulting time derivative $\frac{\partial V(\cdot)}{\partial t}$ is strictly negative along the closed–loop for the adapted linear control system, see [132]. Again, applicability to digital control systems can be obtained using the Emulation technique of Section 1.3.3. For purely discrete–time settings, we again refer to [13, 232].
These design techniques are known for quite some time but stayed mainly unapplicable. This changed during the 1990s when the *backstepping* technique to design the required Lyapunov function was developed, see [72, 132] for an overview.
The estimation–based design leads to so called *modular designs*, cf. [132, 180, 187]. Here, the estimation of the parameter of the plant and the calculation of the parameter within the controller via optimization routines are separated. For further details, we again refer to [132].

Upon implementation, Lyapunov function based controllers as well as adaptive controllers are used on the same level as PID controllers, see Figure 2.6. Compared to a PID, an adaptive controller exhibits better results since it utilizes past information to fit the new situation.
Considering the receding horizon controller, both Lyapunov function based controllers as well as adaptive controllers offer a cheaper computation of the control vector and hence allow for smaller sampling periods T. Moreover, the adaptive controller can be implemented without knowledge of a good model of the system as it is required for the RHC approach. However, to compensate for unexact parameter values, the optimization within the RHC can be extended to cover this issue, see e.g. [62] for output feedbacks using RHC or [204] for general numerical data fitting methods.
Lyapunov function based controllers as well as adaptive controllers also show an applicability disadvantage: Both the adaptation formula and the estimation of the parameter of the plant require significant analytical effort to design the control Lyapunov function or the estimator respectively. Until now, is not clear how to construct these auxiliary functions for general nonlinear systems. Moreover, we expect the receding horizon controller to be superior to the adaptive controller designs if an estimation method of the systems parameters is added to it due to their ability to incorporate past *and* future information within the control law without requiring a control Lyapunov function.

Chapter 3

Stability and Optimality

The presented receding horizon control scheme is based on a sequence of finite time optimal control problems, the stability property defined in Section 1.3.3, however, is stated for an infinite horizon. The central question we are going to consider in this chapter is whether the proposed method is able to establish the stability property stated in Definition 1.44. This manner has been treated before in a number of papers, see e.g. [52,126,205], and some sufficient conditions have been derived to guarantee stability. For most of the commercial RHC settings used in industry, however, these conditions do not apply, see [16]. Moreover, it is not possible to quantify the maximal trade–off between the infinite horizon control (2.8) and the receding horizon control (2.15) in terms of the cost functional (2.5), a parameter which we call *degree of suboptimality*.
Here, we develop techniques to overcome these lacks. In Section 3.1 we derive sufficient conditions for stability conditions for asymptotically stabilizable systems which can be checked *a posteriori*. Additionally, we give an estimate on the degree of suboptimality in this case. In the following Section 3.2 we show how stability can be guaranteed and the degree of suboptimality can be computed *a priori*, that is at runtime of the process. These methods are generalized to practically asymptotically stabilizable systems in Section 3.3. The parameter of interest in this context is the horizon length N of the sequence of subproblems. On the one hand, we expect better results, i.e. a higher degree of suboptimality, if the parameter N is enlarged. On the other hand, since the resulting computing cost rises significantly, we want to choose it as small as possible. Techniques to appropriately adapt this parameter will be the aim of the following Chapter 4.
In the remaining Section 3.4 of this Chapter we present the mentioned previous works on this topic and classify all results.

3.1 A posteriori Suboptimality Estimation

According to our definition of the receding horizon control problem $\mathrm{RHC_N}$ in Section 2.4, we deal with discrete–time nonlinear systems

$$x_u(n+1) = f(x_u(n), u(n)), \quad x_u(0) = x_0$$

on arbitrary metric spaces $\mathbb{X}$ and use finite horizon optimal control problems without terminal costs or terminal constraints in our RHC setup, that is we consider only cost functionals of the form

$$J_N(x_0, u) = \sum_{i=0}^{N-1} l(x_u(i, x_0), u(i)) \tag{3.1}$$

which is commonly used in industrial receding horizon controllers.
For these schemes, we present techniques for estimating the degree of suboptimality *online* and *along the closed–loop trajectory*. Like in [92, 102], our approach is based on relaxed dynamic programming but relies on the computation of a "characteristic value" α at each time instant n along the closed–loop trajectory $x(n)$ and the actual estimate can then be computed from the collection of all these α-values.

Remark 3.1
Since we deal with NMPC schemes without terminal costs, we can use the observation

$$V_M(x) \leq V_N(x) \leq V_\infty(x) \tag{3.2}$$

for all M, $N \in \mathbb{N}$ with $M \leq N$.

The motivation for this work is twofold: on the one hand, we expect the trajectory based estimates to be less conservative than the global estimates derived, e.g., in [92, 102, 205], because in these references the worst case over the whole state space is estimated. Here, however, we only use those points of the state space which are actually visited by the closed–loop trajectory. On the other hand, these trajectory based estimates can be used as a building block for RHC schemes in which the optimization horizon is tuned adaptively, similar to adaptive step size control in numerical schemes for differential equations, see Chapter 4 for details.
The main idea of our task is to reduce the computing time necessary to obtain

$$u_N(x_0, \cdot) := \operatorname*{argmin}_{u_N \in \mathcal{U}_N} J_N(x_0, u_N)$$

in every step of this RHC setup. At the same time, we want to be able to guarantee a certain degree of suboptimality — and implicitly stability — of the closed–loop solution

$$\begin{aligned} x(n+1) &= f\left(x(n), \mu_N(x(n))\right), \quad x(0) = x_0,\ n \in \mathbb{N}_0 && (3.3)\\ \mu_N(x(n)) &:= \operatorname*{argmin}_{u_N \in \mathbb{U}} \left\{V_{N-1}(x(n+1)) + l(x(n), u_N)\right\} && (3.4) \end{aligned}$$

compared to the infinite horizon solution (1.1), (2.8) with $u(n) = \mu(x(n))$. Within this work, the parameter of choice is the horizon length N since computational costs grow more than linear in this variable and, as known from many applications, large N induce better performance and stability.

In order to develop such an adaption strategy, the most important ingredient we require is an estimate on the suboptimality of the feedback $\mu_N(\cdot)$ for the infinite horizon problem which can be evaluated without significant additional computational costs. More precisely, if we define the infinite horizon cost corresponding to $\mu_N(\cdot)$ by

$$V_\infty^{\mu_N}(x_0) := \sum_{n=0}^{\infty} l\left(x(n), \mu_N(x(n))\right), \tag{3.5}$$

then we are interested in upper bounds for this infinite horizon value, either in terms of the finite horizon optimal value function $V_N(\cdot)$ or in terms of the infinite horizon optimal value function $V_\infty(\cdot)$. In particular, the latter gives us estimates about the degree of suboptimality of the controller $\mu_N(\cdot)$ in the actual step of the RHC process.

The main tool we are going to use for this purpose is a rather straightforward and easily proved "relaxed" version of the dynamic programming principle. This fact has been used implicitly in many papers on dynamic programming techniques during the last decades. Recently, it has been studied by Lincoln and Rantzer in [144, 192].
Here, we use the estimate given in Proposition 2.2 of [102]:

Proposition 3.2 (Global a posteriori Estimate)
Consider a feedback law $\mu_N : \mathbb{X} \to U$ and a function $V_N : \mathbb{X} \to \mathbb{R}_0^+$ satisfying the inequality

$$V_N(x) \geq V_N(f(x, \mu_N(x))) + \alpha l(x, \mu_N(x)) \tag{3.6}$$

for some $\alpha \in [0, 1]$ and all $x \in \mathbb{X}$. Then for all $x \in \mathbb{X}$ the estimate

$$\alpha V_\infty(x) \leq \alpha V_\infty^{\mu_N}(x) \leq V_\infty(x) \tag{3.7}$$

holds.

The drawback of this Proposition is the fact that we require (3.6) to hold for all $x \in \mathbb{X}$. To avoid this computational burden, we consider only those points in the state space $\mathbb{X}$ which are visited by a trajectory (2.15), (2.16). As a result, we obtain the following adaption of Proposition 3.2:

Proposition 3.3 (Trajectory based a posteriori Estimate)
Consider a feedback law $\mu_N : \mathbb{X} \to U$ and its associated trajectory $x(\cdot)$ according to (2.16) *with initial value $x(0) = x_0 \in \mathbb{X}$. If there exists a function $V_N : \mathbb{X} \to \mathbb{R}_0^+$ satisfying*

$$V_N(x(n)) \geq V_N(x(n+1)) + \alpha l(x(n), \mu_N(x(n))) \tag{3.8}$$

for some $\alpha \in [0, 1]$ and all $n \in \mathbb{N}_0$ then

$$\alpha V_\infty(x(n)) \leq \alpha V_\infty^{\mu_N}(x(n)) \leq V_\infty(x(n)) \tag{3.9}$$

holds for all $n \in \mathbb{N}_0$.

Proof. The proof is similar to that of [192, Proposition 3] and [102, Proposition 2.2]. Rearranging (3.8) and summing over n we obtain the upper bound

$$\alpha \sum_{j=n}^{K-1} l(x(j), \mu_N(x(j)) \leq V_N(x(n)) - V_N(x(K)) \leq V_N(x(n)).$$

Hence, taking $K \to \infty$ gives us our assertion since the final inequality $V_N(x(n)) \leq V_\infty(x(n))$ follows by (3.2). □

Remark 3.4
In the formulation of Proposition 3.3 we have that α depends on the points $x(n)$ only, while in Proposition 3.2 it depends on all $x \in \mathbb{X}$. Hence, we expect α to be a less conservative approximation of the degree of suboptimality.
Yet, the degree of suboptimality α is valid only along the closed–loop trajectory, i.e. we cannot guarantee this estimate to hold in a neighborhood of the closed–loop trajectory. We like to mention that the degree of suboptimality is directly evaluated along a closed–loop solution. Therefore, we can compute the degree of suboptimality even along disturbed closed–loop solutions using identical methods. In this work, however, we focus on systems without disturbances like modelling errors, external forces, communication or measurement errors.

Remark 3.5
If Proposition 3.3 holds with $\alpha > 0$ for a given trajectory, we can conclude asymptotic stability of this trajectory if the stage cost $l(\cdot,\cdot)$ is positive definite and proper since by (3.8) *$V_N(\cdot)$ is a Lyapunov function for the closed–loop system, see also [92]. The positive definiteness of $l(\cdot,\cdot)$ can be replaced by a detectability condition, see [87].*

To distinguish between α values from Propositions 3.2 and 3.3, we introduce the following notation:

Definition 3.6 (Global/Local/Closed–loop Suboptimality Degree)
(1) The value $\alpha := \max\{\alpha \mid (3.6) \text{ holds for all } x \in \mathbb{X}\}$ is called *global suboptimality degree.*
(2) If $x(n) \in \mathbb{X}$ is fixed, then the maximal value of α satisfying (3.8) is called *local suboptimality degree.*
(3) The value $\alpha := \max\{\alpha \mid (3.8) \text{ holds for all } n \in \mathbb{N}_0\}$ is called *closed–loop suboptimality degree* for the considered trajectory.

According to Definition 4.29 we always have

$$\alpha_{\text{global}} \leq \alpha_{\text{closed–loop}} \leq \alpha_{\text{local}}.$$

Since all values in (3.8) are available at runtime, α can be computed online along the closed–loop trajectory and thus (3.8) yields a computationally feasible and numerically cheap way to estimate the degree of suboptimality of the trajectory. Moreover, using suitable controllability assumptions, one can show that $\alpha \to 1$ as $N \to \infty$, cf. [92, 102].

Remark 3.7
The knowledge of α can in principle be used to adapt the optimization horizon N online by increasing N if the computed α–value is too small, see also Algorithms 4.14, 4.21, 4.23 and 4.29 which are designed for this purpose.

For our programming, we consider the following algorithm to compute α, see also Section 6.2.2.1 for the actual implementation and Section 8.4.1 for numerical results.

Algorithm 3.8 (Computing a posteriori Suboptimality Bound)

Input:	$\alpha_{\min}$	—	Closed–loop suboptimality estimate
	$V_N(x(n))$	—	Value function in the previous step
	$V_N(x(n+1))$	—	Value function in the actual step
	$l(x(n), \mu_N(x(n)))$	—	Initial stage cost in the previous step

(1) If $V_N(x(n)) \leq V_N(x(n+1))$: Print warning "Solution may be unstable"

(2) Set $\alpha := \frac{V_N(x(n)) - V_N(x(n+1))}{l(x(n), \mu_N(x(n)))}$

(3) Set $\alpha_{\min} := \min\{\alpha_{\min}, \alpha\}$ and terminate

Output:	$\alpha_{\min}$	—	Closed–loop suboptimality estimate
	α	—	Local suboptimality estimate for the past step

In order to use Algorithm 3.8, we require $V_N(x(n+1))$ for the computation of α for the state $x(n)$.

Remark 3.9
At time n, the value of $V_N(x(n+1))$ can in principle be obtained by solving an additional optimal control problem, see also Section 6.2.2.1 for the actual implementation and Section 8.4.1 for numerical results. Proceeding this way, however, essentially doubles the computational effort and may thus not be feasible in real–time applications. Yet, our numerical results indicate that this effort pays off during the closed–loop runtime of the adaption algorithm, see Section 8.5. Additionally, one cannot expect that the value of $x(n+1)$ is known exactly in advance, e.g. due to disturbances as well as measurement or modelling errors.

If we wanted to use only those numerical information which is readily available at time n, then we would have to wait until time $n+1$ before the quality of the MPC feedback value $\mu_N(x(n))$ can be evaluated. In other words, (3.8) yields an *a posteriori* estimator, which is an obvious disadvantage if α is to be used for an online adaptation of N at time n. In the next section we present an alternative way in order to estimate α.

3.2 A priori Suboptimality Estimation

This section aims at reducing the amount of information necessary to give an estimate of the degree of suboptimality of the trajectory (3.3), (3.4) under consideration. In particular, we are interested in ignoring all future information, i.e. we try to avoid the use of $V_N(x(n+1))$ in our calculations. Of course, this yields a more conservative estimate in general.
The following estimates are similar to certain results in [102], where, however, they were defined and used globally for all $x \in \mathbb{X}$. In order to make those results computable without using a discretization of the state space $\mathbb{X}$ or analytical a priori information, we formulate and prove alternative versions of these results which can be used along trajectories.

Lemma 3.10
Consider $N \in \mathbb{N}$, a receding horizon feedback law $\mu_N(\cdot)$ and its associated closed–loop solution $x(\cdot)$ according to (2.16) with initial value $x(0) = x_0$. If

$$V_N(x(n+1)) - V_{N-1}(x(n+1)) \leq (1-\alpha) l(x(n), \mu_N(x(n))) \tag{3.10}$$

holds for some $\alpha \in [0,1]$ and all $n \in \mathbb{N}_0$, then $V_N(\cdot)$ satisfies (3.8) and

$$\alpha V_\infty(x(n)) \leq \alpha V_\infty^{\mu_N}(x(n)) \leq V_N(x(n)) \leq V_\infty(x(n)) \tag{3.11}$$

holds for all $n \in \mathbb{N}_0$.

Proof. Using the principle of optimality, i.e. Theorem 2.9 for $m = 1$, we obtain

$$\begin{aligned} V_N(x(n)) &= l(x(n), \mu_N(x(n))) + V_{N-1}(x(n+1)) \\ &\overset{(3.10)}{\geq} l(x(n), \mu_N(x(n))) + V_N(x(n+1)) - (1-\alpha) l(x(n), \mu_N(x(n))) \\ &= V_N(x(n+1)) + \alpha l(x(n), \mu_N(x(n))) \end{aligned}$$

Hence, (3.8) holds and Proposition 3.3 guarantees (3.11). □

In order to shorten notation, the following assumption contains the main ingredients for our results.

Assumption 3.11
For given $N \in \mathbb{N}$, $N \geq 2$, there exists a constant $\gamma > 0$ such that the inequalities

$$V_2(x_{u_N}(N-2, x(n))) \leq (\gamma+1)V_1(x_{u_N}(N-2, x(n))) \tag{3.12}$$
$$V_k(x_{u_N}(N-k, x(n))) \leq (\gamma+1)l(x_{u_N}(N-k, x(n)), \mu_k(x_{u_N}(N-k, x(n)))) \tag{3.13}$$

hold for all $k \in \{3, \ldots, N\}$ and all $n \in \mathbb{N}_0$ where the open–loop solution $x_{u_N}(j, x(n))$ is given by (2.10).

Remark 3.12
We would like to point out that $x(\cdot)$ in Assumption 3.11 is the closed–loop trajectory which is the outcome of the RHC algorithm, i.e. equations (2.15), (2.16). *In contrast to that, $x_{u_N}(\cdot, x(\cdot))$ represents the open–loop solutions coming from* (2.10), (2.14) *which are also known in every single step of this algorithm. Note that those two values are not identical in general, see Figure 3.1 for a schematic interpretation.*

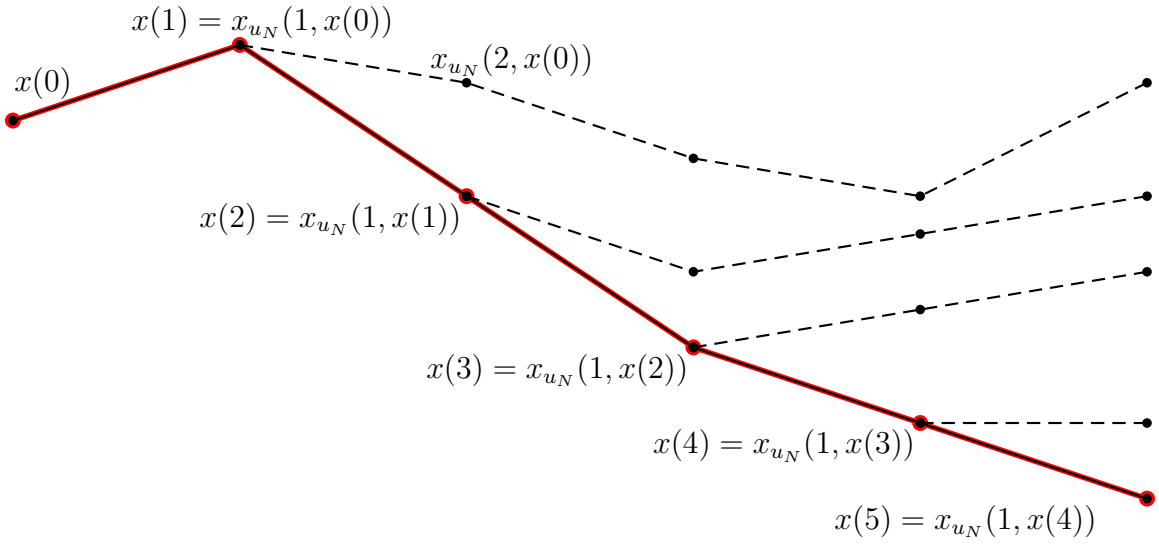


Figure 3.1: Difference between open–loop and closed–loop trajectory

Remark 3.13
Note that in order to obtain $V_1(x_{u_N}(N-2, x(n)))$ in (3.12), *one has to solve another optimal control problem, in this case a so called one–step problem, see e.g. [96]. Compared to the computation of $V_N(x(n+1))$ the computing time of a one–step problem is negligible.*

Proposition 3.14
Consider $N \geq 2$ and assume that Assumption 3.11 holds for this N. Then

$$\frac{(\gamma+1)^{N-2}}{(\gamma+1)^{N-2} + \gamma^{N-1}} V_N(x(n)) \leq V_{N-1}(x(n))$$

holds for all $n \in \mathbb{N}_0$.

Proof. In the following, we use the abbreviation $x_{u_N}(j) := x_{u_N}(j, x(n))$, $j = 0, \ldots, N$, since all our calculations using the open–loop trajectory defined by (2.14), (2.10) refer to

the fixed initial value $x(n)$.
Set $\tilde{n} := N - k$. First, we prove

$$V_{k-1}(f(x_{u_N}(\tilde{n}), \mu_k(x_{u_N}(\tilde{n})))) \leq \gamma l(x_{u_N}(\tilde{n}), \mu_k(x_{u_N}(\tilde{n}))) \tag{3.14}$$

for all $k \in \{3, \ldots, N\}$ and all $n \in \mathbb{N}$. Using the principle of optimality and Assumption 3.11, we obtain

$$\begin{aligned} V_{k-1}(f(x_{u_N}(\tilde{n}), \mu_k(x_{u_N}(\tilde{n})))) &= V_k(x_{u_N}(\tilde{n})) - l(x_{u_N}(\tilde{n}), \mu_k(x_{u_N}(\tilde{n}))) \\ &\overset{(3.13)}{\leq} (\gamma+1) l(x_{u_N}(\tilde{n}), \mu_k(x_{u_N}(\tilde{n}))) - l(x_{u_N}(\tilde{n}), \mu_k(x_{u_N}(\tilde{n}))) \\ &= \gamma l(x_{u_N}(\tilde{n}), \mu_k(x_{u_N}(\tilde{n}))) \end{aligned}$$

We show the main assertion by induction over $k = 2, \ldots, N$. For notational reason, we use the abbreviation

$$\eta_k = \frac{(\gamma+1)^{k-2}}{(\gamma+1)^{k-2} + \gamma^{k-1}}. \tag{3.15}$$

and prove

$$\eta_k V_k(x_{u_N}(\tilde{n})) \leq V_{k-1}(x_{u_N}(\tilde{n})). \tag{3.16}$$

Now we choose the initial value of the open–loop trajectory $x_{u_N}(0) = x(n)$ with n being arbitrary but fixed.
For $k = 2$ we obtain (3.16) via

$$V_2(x_{u_N}(N-2)) \overset{(3.12)}{\leq} (\gamma+1) V_1(x_{u_N}(N-2)) \overset{(3.15)}{=} \frac{1}{\eta_2} V_1(x_{u_N}(N-2)).$$

For the induction step $k \to k+1$, the following holds:

$$\begin{aligned} V_k(x_{u_N}(\tilde{n})) &= V_{k-1}(f(x_{u_N}(\tilde{n}), \mu_k(x_{u_N}(\tilde{n})))) + l(x_{u_N}(\tilde{n}), \mu_k(x_{u_N}(\tilde{n}))) \\ &\overset{(3.14)}{\geq} \left(1 + \frac{1-\eta_k}{\gamma+\eta_k}\right) V_{k-1}(f(x_{u_N}(\tilde{n}), \mu_k(x_{u_N}(\tilde{n})))) \\ &\quad + \left(1 - \gamma\frac{1-\eta_k}{\gamma+\eta_k}\right) l(x_{u_N}(\tilde{n}), \mu_k(x_{u_N}(\tilde{n}))) \\ &\overset{(3.16)}{\geq} \eta_k \left(1 + \frac{1-\eta_k}{\gamma+\eta_k}\right) V_k(f(x_{u_N}(\tilde{n}), \mu_k(x_{u_N}(\tilde{n})))) \\ &\quad + \left(1 - \gamma\frac{1-\eta_k}{\gamma+\eta_k}\right) l(x_{u_N}(\tilde{n}), \mu_k(x_{u_N}(\tilde{n}))) \\ &= \eta_k \frac{\gamma+1}{\gamma+\eta_k} \{V_k(f(x_{u_N}(\tilde{n}), \mu_k(x_{u_N}(\tilde{n})))) + l(x_{u_N}(\tilde{n}), \mu_k(x_{u_N}(\tilde{n})))\} \\ &= \eta_k \frac{\gamma+1}{\gamma+\eta_k} V_{k+1}(x_{u_N}(\tilde{n})) \end{aligned}$$

with

$$\eta_k \frac{\gamma+1}{\gamma+\eta_k} \overset{(3.15)}{=} \frac{(\gamma+1)^{k-2}}{(\gamma+1)^{k-2}+\gamma^{k-1}} \frac{\gamma+1}{\gamma + \frac{(\gamma+1)^{k-2}}{(\gamma+1)^{k-2}+\gamma^{k-1}}} = \frac{(\gamma+1)^{k-1}}{(\gamma+1)^{k-1}+\gamma^k} \overset{(3.15)}{=} \eta_{k+1}.$$

Hence, (3.16) holds true. If we choose $k = N$ then we get $\tilde{n} = 0$. Inserting this in (3.16) we can use $x_{u_N}(0) = x_{u_N}(0, x(n)) = x(n)$. Since n was chosen arbitrarily we obtain the assertion

$$\eta_N V_N(x(n)) = \eta_N V_N(x_{u_N}(0)) \leq V_{N-1}(x_{u_N}(0)) = V_{N-1}(x(n)) \qquad \forall n \in \mathbb{N}.$$

□

Using this technical construction, we are now prepared to prove our first main result:

Theorem 3.15
Consider $\gamma > 0$ and $N \in \mathbb{N}$, $N \geq 2$ such that $(\gamma+1)^{N-2} > \gamma^N$ holds. If Assumption 3.11 is fulfilled for these γ and N, then the estimate

$$\alpha V_\infty^{\mu_N}(x(n)) \leq V_N(x(n)) \leq V_\infty(x(n)) \quad \text{with} \quad \alpha = \frac{(\gamma+1)^{N-2} - \gamma^N}{(\gamma+1)^{N-2}} \tag{3.17}$$

holds for all $n \in \mathbb{N}$.
In particular, the inequality

$$\frac{V_\infty^{\mu_N}(x(n)) - V_\infty(x(n))}{V_\infty(x(n))} \leq \frac{\gamma^N}{(\gamma+1)^{N-2} - \gamma^N}$$

holds for the relative difference between $V_\infty^{\mu_N}(x(n))$ and $V_\infty(x(n))$.

Proof. From Proposition 3.14 we get

$$V_N(x(n)) - V_{N-1}(x(n)) \leq \left(\frac{(\gamma+1)^{N-2} + \gamma^{N-1}}{(\gamma+1)^{N-2}} - 1 \right) V_{N-1}(x(n)) = \frac{\gamma^{N-1}}{(\gamma+1)^{N-2}} V_{N-1}(x(n))$$

Considering $j = n - 1$ we obtain

$$\begin{aligned} V_N(x(j+1)) - V_{N-1}(x(j+1)) &\leq \frac{\gamma^{N-1}}{(\gamma+1)^{N-2}} V_{N-1}(x(j+1)) \\ = \frac{\gamma^{N-1} V_{N-1}(f(x(j), \mu_N(x(j))))}{(\gamma+1)^{N-2}} &= \frac{\gamma^{N-1} V_{N-1}(f(x_{u_N}(0, x(j)), \mu_N(x_{u_N}(0, x(j)))))}{(\gamma+1)^{N-2}} \end{aligned}$$

where we used the definition of the open–loop solution from (2.10) with $x_0 = x(j)$ similar to the proof of Proposition 3.14. Hence, we can use (3.14) with $k = N$ and get

$$\begin{aligned} V_N(x(j+1)) - V_{N-1}(x(j+1)) &\leq \frac{\gamma^N}{(\gamma+1)^{N-2}} l(x_{u_N}(0, x(j)), \mu_N(x_{u_N}(0, x(j)))) \\ &= \frac{\gamma^N}{(\gamma+1)^{N-2}} l(x(j), \mu_N(x(j))). \end{aligned}$$

Consequently, the assumptions of Lemma 3.10 are fulfilled with

$$\alpha = 1 - \frac{\gamma^N}{(\gamma+1)^{N-2}} = \frac{(\gamma+1)^{N-2} - \gamma^N}{(\gamma+1)^{N-2}} \tag{3.18}$$

and the assertion holds. □

Theorem 3.15 immediately leads to our second suboptimality estimation algorithm:

Algorithm 3.16 (Computing a priori Suboptimality Bound)

Input:	$\alpha_{\min}$	—	Closed–loop suboptimality estimate
	$V_N(x(n))$	—	Value function in the actual step
	N	—	Length of the horizon

(1) Compute $V_1(x_{u_N}(N-2, x(n)))$

(2) Set $\gamma := \frac{V_2(x_{u_N}(N-2,x(n)))}{V_1(x_{u_N}(N-2,x(n)))} - 1$

(3) For k from 3 to N do

(3a) Set $\tilde{\gamma} := \frac{V_k(x_{u_N}(N-k,x(n)))}{l(x_{u_N}(N-k,x(n)),\mu_k(x_{u_N}(N-k,x(n))))} - 1$

(3b) Set $\gamma := \max\{\gamma, \tilde{\gamma}\}$

(4) Set $\alpha := 1 - \frac{\gamma^N}{(\gamma+1)^{N-2}} = \frac{(\gamma+1)^{N-2}-\gamma^N}{(\gamma+1)^{N-2}}$

(5) If $\alpha < 0$: Print warning "Solution may be unstable"

(6) Set $\alpha_{\min} := \min\{\alpha_{\min}, \alpha\}$ and terminate

Output: $\alpha_{\min}$ — Closed–loop suboptimality estimate
α — Local suboptimality estimate for the actual step
γ — Characteristic of the problem

Hence, Algorithm 3.16 allows us to compute γ from the inequalities (3.12), (3.13) and the assumption in Theorem 3.15 at each time instant n and to evaluate α according to (3.17).

Remark 3.17
Note that the condition $(\gamma+1)^{N-2} \leq \gamma^N$ *stated in Theorem 3.15 is automatically checked in step (5). Analyzing the function*

$$f(\gamma) := (\gamma+1)^{N-2} - \gamma^N$$

on $\mathbb{R}^+ \setminus \{0\}$ *reveals the existence and uniqueness of a root of* $f(\cdot)$*. Moreover,* $f(\cdot)$ *is strictly monotonely decreasing for* γ*–values larger than this root, see also Figure 3.2. Hence, if* $\alpha < 0$ *holds true, then there exists no* $\widetilde{\gamma} > \gamma$ *satisfying* $(\widetilde{\gamma}+1)^{N-2} \leq \widetilde{\gamma}^N$ *and we are unable to apply Theorem 3.15.*

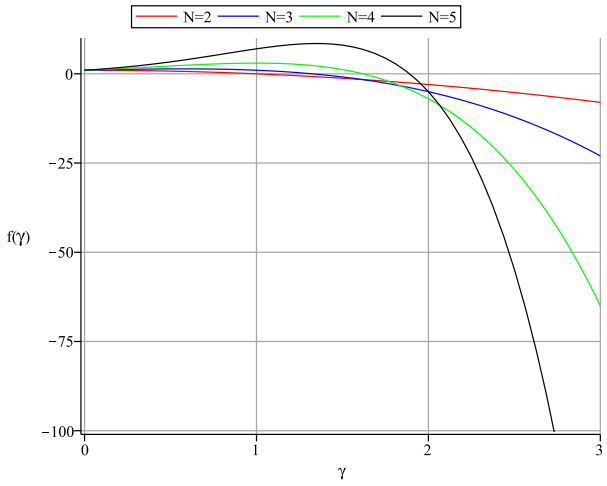


Figure 3.2: Development of the function $f(\cdot)$ defined to check applicability of Theorem 3.15

In contrast to computing α directly from (3.8), we obtain a criterion for the quality of $\mu_N(x(n))$ depending on the data available at time n (cf. Remark 3.26) and the solution of the optimal control problem of length $N = 1$ with initial value $x_{u_N}(N-2, x(n))$. Furthermore, if $l(\cdot,\cdot)$ is independent of u, the control value is not needed at all and thus γ can be computed only from the data available at time n. In either case, we obtain an *a priori* estimate which is available before the current step is actually carried out.

Remark 3.18
Intuitively, this result states that if the instantaneous running cost contains sufficient information about the optimal value function, then the resulting controller is suboptimal. Here, we can say that $V_N(\cdot)$ *and* $l(\cdot,\cdot)$ *contain "sufficient" information if we obtain* $\alpha > 0$ *from Proposition 3.3 by using* (3.18) *and there exists some* $\gamma > 0$ *such that the inequalities* (3.12) *and* (3.13) *hold.*

Now, our aim is to weaken the previous relation (3.18) between α and γ. To this end, we ease condition (3.12) which is required to establish the induction anker in proof of Proposition 3.14 only. This leads to the following relaxation of Assumption 3.11:

Assumption 3.19
For given N, $N_0 \in \mathbb{N}$, $N \geq N_0 \geq 2$, there exists a constant $\gamma > 0$ such that the inequalities

$$\frac{V_{N_0}(x_{u_N}(N-N_0, x(n)))}{\gamma+1} \leq \max_{j=2,\dots,N_0} l(x_{u_N}(N-j, x(n)), \mu_{j-1}(x_{u_N}(N-j, x(n)))) \quad (3.19)$$

$$\frac{V_k(x_{u_N}(N-k, x(n)))}{\gamma+1} \leq l(x_{u_N}(N-k, x(n)), \mu_k(x_{u_N}(N-k, x(n)))) \quad (3.20)$$

hold for all $k \in \{N_0+1, \dots, N\}$ and all $n \in \mathbb{N}_0$ where the open–loop solution $x_{u_N}(j, x(n))$ is given by (2.10).

Remark 3.20
Assumption 3.19 generalizes Assumption 3.11 as well as [102, Assumption 4.6] in which $N_0 = 2$ was used. In the numerical example shown in Section 8.4, we will see that a judicious choice of N_0 can considerably improve our suboptimality estimates. Since Assumptions 3.11 and 3.19 are fairly close, the corresponding Algorithms 3.16 and 3.23 are implemented in just one routine, see Section 6.2.2.2 for details.

Using these relaxed conditions, we can reformulate Proposition 3.14 in the following way:

Proposition 3.21
Consider $N \geq N_0 \geq 2$ and assume that Assummption 3.19 holds for these constants. Then the inequality

$$\frac{(\gamma+1)^{N-N_0}}{(\gamma+1)^{N-N_0} + \gamma^{N-N_0+1}} V_N(x(n)) \leq V_{N-1}(x(n)) \quad (3.21)$$

holds for all $n \in \mathbb{N}_0$.

Proof. Using the notation of the proof of Proposition 3.14, we again define $x_{u_N}(j) := x_{u_N}(j, x(n))$, $j \in \{0, \dots, N\}$ and

$$\eta_k = \frac{(\gamma+1)^{k-N_0}}{(\gamma+1)^{k-N_0} + \gamma^{k-N_0+1}} \quad (3.22)$$

to prove (3.16) for $k = N_0, \dots, N$.
According to the changes in our assumptions, we have to show is that (3.16) holds true for the induction anker only, i.e. $k = N_0$:

$$\begin{aligned} V_{N_0}(x_{u_N}(N-N_0)) &\overset{(3.19)}{\leq} (\gamma+1) \max_{j=2,\dots,N_0} l(x_{u_N}(N-j), \mu_{j-1}(x_{u_N}(N-j))) \\ &\leq (\gamma+1) \sum_{j=2}^{N_0} l(x_{u_N}(N-j), \mu_{j-1}(x_{u_N}(N-j))) \\ &\overset{(3.22)}{=} \frac{1}{\eta_{N_0}} V_{N_0-1}(x_{u_N}(N-N_0)). \end{aligned}$$

For the induction step, we have to prove (3.14) to hold for $k \in \{N_0+1, \dots, N\}$. This is exactly the index set of assumption (3.20). Hence, we can use the same induction step as in the proof of Proposition 3.14 showing the assertion. □

Using this proposition, we adapt Theorem 3.15:

Theorem 3.22
Consider $\gamma > 0$ and N, $N_0 \in \mathbb{N}$, $N \geq N_0$ such that $(\gamma+1)^{N-N_0} > \gamma^{N-N_0+2}$ holds. If Assumption 3.19 is fulfilled for these γ, N and N_0, then the estimate

$$\alpha V_\infty^{\mu_N}(x(n)) \leq V_N(x(n)) \leq V_\infty(x(n)) \qquad \text{with} \qquad \alpha = \frac{(\gamma+1)^{N-N_0} - \gamma^{N-N_0+2}}{(\gamma+1)^{N-N_0}} \tag{3.23}$$

holds for all $n \in \mathbb{N}$.
In particular, the inequality

$$\frac{V_\infty^{\mu_N}(x(n)) - V_\infty(x(n))}{V_\infty(x(n))} \leq \frac{\gamma^{N-N_0+2}}{(\gamma+1)^{N-N_0} - \gamma^{N-N_0+2}} \tag{3.24}$$

holds for the relative difference between $V_\infty^{\mu_N}(x(n))$ and $V_\infty(x(n))$.

Proof. Using the same steps as in the proof of Theorem 3.15 we get

$$V_N(x(n)) - V_{N-1}(x(n)) \leq \frac{\gamma^{N-N_0+1}}{(\gamma+1)^{N-N_0}} V_{N-1}(x(n))$$

from Proposition 3.21. Considering $j = n-1$, we obtain

$$V_N(x(j+1)) - V_{N-1}(x(j+1)) \leq \frac{\gamma^{N-N_0+1}}{(\gamma+1)^{N-N_0}} V_{N-1}(f(x_{u_N}(0, x(j)), \mu_N(x_{u_N}(0, x(j)))))$$

using our standard definition of the open–loop solution. Now, we can use (3.14) which holds true for $k \in \{N_0+1, \ldots, N\}$ according to the proof of Proposition 3.14 with $k = N$ and get

$$V_N(x(j+1)) - V_{N-1}(x(j+1)) \leq \frac{\gamma^{N-N_0+2}}{(\gamma+1)^{N-N_0}} l(x(j), \mu_N(x(j))).$$

Hence, the assumptions of Lemma 3.10 are fulfilled with

$$\alpha = 1 - \frac{\gamma^{N-N_0+2}}{(\gamma+1)^{N-N_0}} = \frac{(\gamma+1)^{N-N_0} - \gamma^{N-N_0+2}}{(\gamma+1)^{N-N_0}} \tag{3.25}$$

and the assertion follows. □

Similar to Theorem 3.15, we obtain a third algorithm to compute a suboptimality estimate:

Algorithm 3.23 (Computing a priori Suboptimality Bound)

Input:	$\alpha_{\min}$	—	Closed–loop suboptimality estimate
	$V_N(x(n))$	—	Value function in the actual step
	N	—	Length of the horizon
	N_0	—	Length of the comparison horizon

(1) Set $l_{\max} = 0$.

(2) For j from 2 to N_0 do

(2a) Compute $l(x_{u_N}(N-j,x(n)),\mu_{j-1}(x_{u_N}(N-j,x(n))))$

(2b) Set $l_{\max} = \max\{l_{\max}, l(x_{u_N}(N-j,x(n)),\mu_{j-1}(x_{u_N}(N-j,x(n))))\}$

(3) Set $\gamma := \frac{V_{N_0}(x_{u_N}(N-N_0,x(n)))}{l_{\max}} - 1$

(4) For k from N_0+1 to N do

(4a) Set $\tilde{\gamma} := \frac{V_k(x_{u_N}(N-k,x(n)))}{l(x_{u_N}(N-k,x(n)),\mu_k(x_{u_N}(N-k,x(n))))} - 1$

(4b) Set $\gamma := \max\{\gamma, \tilde{\gamma}\}$

(5) Set $\alpha := 1 - \frac{\gamma^{N-N_0+2}}{(\gamma+1)^{N-N_0}} = \frac{(\gamma+1)^{N-N_0} - \gamma^{N-N_0+2}}{(\gamma+1)^{N-N_0}}$

(6) If $\alpha < 0$: Print warning "Solution may be unstable"

(7) Set $\alpha_{\min} := \min\{\alpha_{\min}, \alpha\}$ and terminate

Output:	$\alpha_{\min}$	—	Closed–loop suboptimality estimate
	α	—	Local suboptimality estimate for the actual step
	γ	—	Characteristic of the problem

For the actual implementation and results of this algorithm, we refer to the Sections 6.2.2.2 and 8.4.2 respectively.
Using the relationship between α and the tupel (γ, N) from Theorem 3.22 for fixed N_0, we can see from Figure 3.3 which combinations of N and γ values actually guarantee stability and a certain degree of suboptimality. Moreover, one observes that in order to obtain stability of the closed–loop, i.e. $\alpha \geq 0$, we require

$$N \geq N_0 + \frac{2\ln(\gamma)}{\ln(\gamma+1) - \ln(\gamma)} =: h(\gamma, N_0) \tag{3.26}$$

to hold.

Remark 3.24
The relationship (3.26) *is analyzed in detail in Section 4.3.1 and allows us to derive a suitable prolongation strategy, see also Remark 3.7.*

Remark 3.25
Note that

$$\frac{\partial}{\partial\gamma} h(\gamma, N_0) = \frac{2\gamma\ln(\gamma+1) - 2\gamma\ln(\gamma) + 2\ln(\gamma+1)}{(\gamma+1)\gamma(-\ln(\gamma+1)+\ln(\gamma))^2}$$

is independent of N_0. *Hence, since for all* $\gamma \geq \overline{\gamma} \approx 0.01$ *we obtain* $h'(\gamma, N_0) > 1$ *for fixed* $N_0 \in [2, N]$, *we can use the positive definiteness and strict monotonicity of* $h'(\cdot, N_0)$ *to conclude that* $h(\cdot, N_0)$ *grows unboundedly and stronger than linear. Additionally, we obtain that* N *grows less than quadratic since for* $\overline{\gamma} \approx 5.7$ *we have* $h'(\gamma) < \gamma$ *for all* $\gamma \geq \overline{\gamma}$. *More precisely, we can show that* $h(\gamma, N_0)$ *approximates* $f(\gamma) := 2\gamma\ln(\gamma)$ *as* γ *tends to infinity since*

$$\begin{aligned}\lim_{\gamma\to\infty} \frac{h(\gamma, N_0)}{2\gamma\ln(\gamma)} &= \lim_{\gamma\to\infty} \frac{N_0}{2\gamma\ln(\gamma)} + \lim_{\gamma\to\infty} \frac{2\ln(\gamma)}{2\gamma\ln(\gamma)(\ln(\gamma+1)-\ln(\gamma))} \\ &= \lim_{\gamma\to\infty} \frac{\gamma^{-1}}{\ln(\gamma+1)-\ln(\gamma)} = \lim_{\gamma\to\infty} \frac{-\gamma^{-2}}{\frac{1}{\gamma+1} - \frac{1}{\gamma}} = \lim_{\gamma\to\infty} \frac{\gamma^2+\gamma}{\gamma^2} = 1.\end{aligned}$$

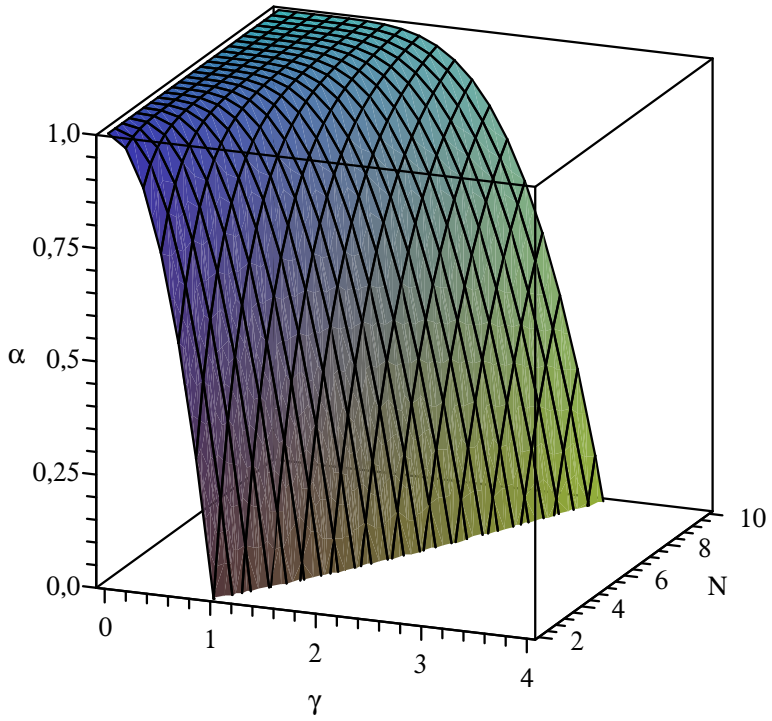


Figure 3.3: Relationship between α and (γ, N) for $N_0 = 2$

This, however, turns out to be twice the rate stated in [92] and up till now the origin of this multiplicative constant is unclear.

Remark 3.26
Assumption 3.19 involves both the state of the closed–loop trajectory $x(\cdot)$ from (3.3) at time n and the open–loop trajectory $x_{u_N}(\cdot, x(n))$ from (2.10) starting in $x(n)$. Both are available in the receding horizon control scheme at time n once the finite horizon optimization problem with initial value $x(n)$ is solved. From these, the optimal value functions on the left hand sides of the inequalities (3.19) and (3.20) can be derived using Bellman's optimality principle by summing up the running cost along the "tails" of the optimal trajectory $x_{u_N}(\cdot, x(n))$. The only values which are not immediately available are the controls $\mu_{j-1}(x_{u_N}(N-j, x(n)))$ in (3.19) which need to be determined by solving an additional optimal control problem with horizon $j \leq N_0 - 1$. Since typically N_0 is considerably smaller than N, this can be done with much less effort than computing $V_N(x(n+1))$. Furthermore, these control values are not needed at all if $l(\cdot, \cdot)$ is independent of u.

Remark 3.27
Another way of numerically computing suboptimality estimates is presented in [205] for linear finite dimensional system. The main difference to our approach is that the condition in [205] is verified by computing numerical approximations to the optimal value functions, which is feasible only for low dimensional linear systems but infeasible in our nonlinear setting on arbitrary metric spaces.

3.3 Practical Suboptimality

In general, one cannot expect that the conditions presented in Sections 3.1 and 3.2 hold in practice. This is due to the fact that in many cases the discrete–time system (3.3) is obtained from a discretization of a continuous–time system, e.g. sampling with zero order hold, cf. Section 1.2 for the theoretical and Section 5.1 for an implementational background. Hence, even if the continuous–time system is controllable to a fixed point x^*, it is likely that the corresponding sampled–data system is only practically stabilizable at x^*. In addition, numerical errors in the optimization algorithm may become predominant in a small neighborhood of x^*.
Here, we extend results from [102] which treat this problem globally by combining them with results from the previous Section 3.2 for the non–practical case to relax the necessary conditions.

If a system is only practically asymptotically stabilizable and one uses a positive definite stage cost $l(\cdot,\cdot)$ with respect to the desired equilibrium x^* in the first component such that $l(x,u)=0$ if and only if $x=x^*$, then it is impossible to calculate a control sequence which stears $l(\cdot,\cdot)$ to zero. Moreover, Assumptions 3.11 and 3.19 do not hold since $V_k(\cdot)$ grows unboundedly for $k\to\infty$ and Proposition 3.3 cannot be applied in general for practically stable systems. This is due to the possibility that $V_N(\cdot)$ may not be monotonely decreasing close to x^*. Hence, this result has to be adapted to be applicable in this situation.
One way to do this is to modify the stage cost $l(\cdot,\cdot)$ to be positive definite with respect to a forward invariant stabilizable neighborhood $\mathcal{N}$ of x^*. However, the computation of $\mathcal{N}$ and hence the design of $l(\cdot,\cdot)$ may be difficult if not impossible.
Still, even with the original $l(\cdot,\cdot)$ and in the presence of sampling and numerical errors, one can expect a typical receding horizon controller to drive the system towards a neighborhood of x^* and practically stabilize the closed–loop system, i.e., to drive the system towards a small neighborhood of x^*, see also [103]. Hence, an intuitive idea is to "virtually" cut off and shift the value function vertically as shown in Figures 3.4 and 3.5. This allows us to solve the original optimal control problem but interpret the resulting trajectory and control values in the context of a modified cost functional.
Here, we shift the stage cost $l(\cdot,\cdot)$ and hence $V(\cdot)$ using a constant ε which enables us to obtain estimates for the original problem similar to our previous results, cf. Proposition 3.3 and Theorems 3.15 and 3.22. More importantly, we want to choose the constant ε such that the relaxed Lyapunov inequality (3.8) holds outside the "zero region" of the shifted stage cost and value function. Hence, our estimates can be directly applied to points in this region which are visited by the trajectory.

Proposition 3.28
Consider a feedback law $\mu_N:\mathbb{X}\to U$ and its associated closed–loop trajectory $x(\cdot)$ according to (3.3). Assume there exists a function $V_N:\mathbb{X}\to\mathbb{R}_0^+$ satisfying the inequality

$$V_N(x(n)) \geq V_N(x(n+1)) + \min\left\{\alpha\left(l(x(n),\mu_N(x(n))) - \varepsilon\right), l(x(n),\mu_N(x(n))) - \varepsilon\right\} \tag{3.27}$$

for some $\alpha\in[0,1]$, some $\varepsilon>0$ and all $n\in\mathbb{N}_0$. Consider a discrete–time interval $I:=\{n_1,\ldots,n_2\}$. Let n_1, $n_2\in\mathbb{N}$, $n_1<n_2$, for which the inequality $l(x(n),\mu_N(x(n)))\geq\varepsilon$ holds for all $n\in I$ and set $\sigma:=V_N(x(n_2+1))$. Then, for the modified stage cost

$$\overline{l}(x(n),\mu_N(x(n))) := \max\left\{l(x(n),\mu_N(x(n))) - \varepsilon, 0\right\}, \tag{3.28}$$

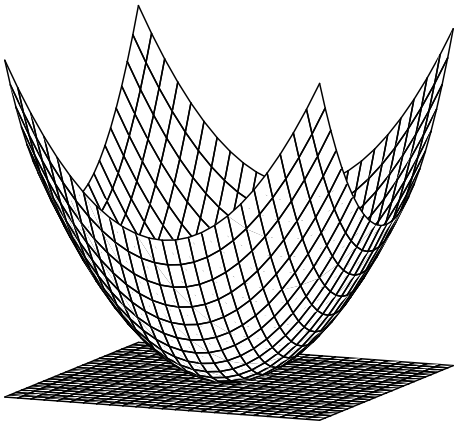


Figure 3.4: Example of a value function where the origin is asymptotically stable

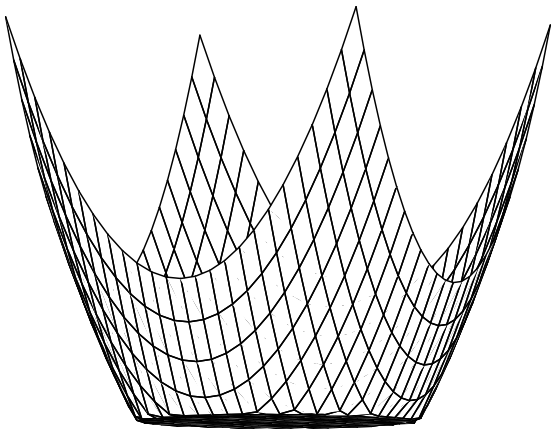


Figure 3.5: Example of a value function which is cut off and shifted vertically to zero

and the corresponding value function

$$\overline{V}_I^{\mu_N}(x(n)) := \sum_{j=n}^{n_2} \overline{l}(x(j), \mu_N(x(j))) \tag{3.29}$$

using the controller $\mu_N(\cdot)$, the estimate

$$\alpha \overline{V}_I^{\mu_N}(x(n)) \leq V_N(x(n)) - \sigma \leq V_\infty(x(n)) - \sigma \tag{3.30}$$

holds for all $n \in I$.

Proof. From the definition of $\overline{l}(\cdot,\cdot)$ and I we obtain

$$\begin{aligned} \alpha \overline{l}(x(n), \mu_N(x(n))) &= \max\{\alpha\,(l(x(n), \mu_N(x(n))) - \varepsilon)\,, 0\} \\ &\overset{(3.27)}{\leq} V_N(x(n)) - V_N(x(n+1)), \end{aligned}$$

for $n \in I$. Thus, summing over n gives us

$$\alpha \overline{V}_I^{\mu_N}(x(n)) = \alpha \sum_{j=n}^{n_2} \overline{l}(x(j), \mu_N(x(j))) \leq V_N(x(n)) - \sigma,$$

which implies the assertion since $V_N(x(n)) \leq V_\infty(x(n))$ follows by (3.2). □

Figure 3.6 illustrates a possible situation of Proposition 3.28.
If (3.27) is satisfied, then inequality (3.30) is true for all subintervals $I = \{n_1, \ldots, n_2\}$ of $\mathbb{N}_0$ on which $l(x(n), \mu_N(x(n))) \geq \varepsilon$ holds, implying that on I the corresponding trajectory behaves "almost" like an infinite horizon optimal one. In particular, $x(n)$ approaches x^* and thus the sequence of $l(\cdot,\cdot)$-values repeatedly (possibly infinitely often) enters and leaves the set $[0, \varepsilon]$. In Figure 3.6 the arrows at the bottom indicate the intervals on which the trajectory is approximately optimal.

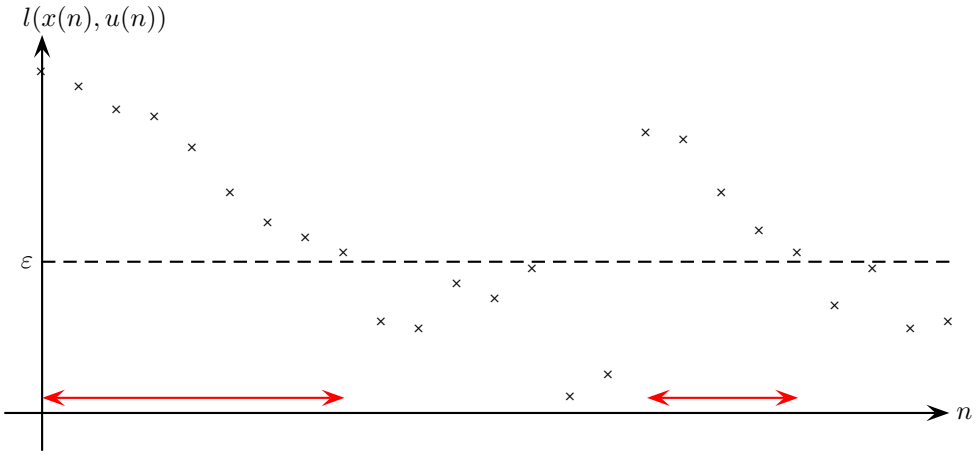


Figure 3.6: Illustration of Proposition 3.28

When leaving $[0, \varepsilon]$ the possible growth of $V_N(\cdot)$ is bounded by ε due to (3.27). Thus, once $l(x(n), \mu_N(x(n)))$ has entered $[0, \varepsilon]$ for the first time, the state of the system remains in a neighborhood of x^* defined by a sublevel set of $V_N(\cdot)$ whose precise shape, however, cannot be easily determined a priori, see also [102, Remark 5.2 and Example 5.10].
Similar to Definition 3.6, Proposition 3.28 induces the following declarations:

Definition 3.29 (Global/Local/Closed–loop Practical Suboptimality Degree)
(1) The value $\alpha := \max\{\alpha \mid (3.27) \text{ holds for all } x(n) \in \mathbb{X}\}$ is called *(semi–)global practical suboptimality degree.*
(2) If $x(n) \in \mathbb{X}$ is fixed, then the maximal value of α satisfying (3.27) is called *local practical suboptimality degree.*
(3) The value $\alpha := \max\{\alpha \mid (3.27) \text{ holds for all } n \in \mathbb{N}_0\}$ is called *closed–loop practical suboptimality degree.*

Implementing Proposition 3.28, we see that $V_N(x(n+1))$ is needed to compute the characteristic α. Hence, it can only be applied a posteriori or at a high additional computational cost, see Remark 3.9.
Within our implementation, see Section 6.2.2.3, we consider the following algorithm:

Algorithm 3.30 (Computing a posteriori practical Suboptimality Bound)

Input:	$\alpha_{\min}$	—	Closed–loop practical suboptimality estimate
	$V_N(x(n))$	—	Value function in the previous step
	$V_N(x(n+1))$	—	Value function in the actual step
	ε	—	Truncation constant

(1) If $l(x(n), \mu_N(x(n))) < \varepsilon$:
Print "Practical stability region reached", set $\alpha = 1$ and terminate

(2) Else

(2a) If $V_N(x(n)) \leq V_N(x(n+1))$: Print warning "Solution may be unstable"

(2b) Set $\alpha := \frac{V_N(x(n)) - V_N(x(n+1))}{l(x(n), \mu_N(x(n))) - \varepsilon}$

(2c) Set $\alpha_{\min} := \min\{\alpha_{\min}, \alpha\}$ and terminate

Output:	$\alpha_{\min}$	—	Closed–loop practical suboptimality estimate
	α	—	Local practical suboptimality estimate for the past step

For results of Algorithm 3.30 and a comparison to the outcome of Algorithm 3.8, we refer to Section 8.4.3.

Similar to Lemma 3.10, we can use the following observation:

Lemma 3.31
Consider $N \in \mathbb{N}$, a receding horizon feedback law $\mu_N(\cdot)$ and its associated closed–loop solution $x(\cdot)$ according to (3.3) with initial value $x(0) = x_0$. If

$$V_N(x(n+1)) - V_{N-1}(x(n+1)) \leq \max\{(1-\alpha)l(x(n), \mu_N(x(n))) + \alpha\varepsilon, \varepsilon\} \tag{3.31}$$

holds for some $\alpha \in [0,1]$, some $\varepsilon > 0$ and all $n \in [0, n_0] \subset \mathbb{N}_0$, then

$$\alpha \overline{V}_I^{\mu_N}(x(n)) \leq V_N(x(n)) - \sigma$$

holds for all $n \in [0, n_0]$ with $\overline{V}_I^{\mu_N}(\cdot)$ according to (3.29).

Proof. Using the principle of optimality we obtain

$$\begin{aligned} V_N(x(n)) &= l(x(n), \mu_N(x(n))) + V_{N-1}(x(n+1)) \\ &\overset{(3.31)}{\geq} l(x(n), \mu_N(x(n))) + V_N(x(n+1)) \\ &\qquad - \max\{(1-\alpha)l(x(n), \mu_N(x(n))) - \alpha\varepsilon, \varepsilon\} \\ &= \min\{\alpha\left(l(x(n), \mu_N(x(n))) - \varepsilon\right), l(x(n), \mu_N(x(n))) - \varepsilon\} + V_N(x(n+1)). \end{aligned}$$

Hence, (3.27) holds and Proposition 3.28 guarantees the assertion. □

Similar to the non–practical case, we can state sufficient conditions to establish our practical suboptimality estimate.

Assumption 3.32
For a given $N \in \mathbb{N}$, $N \geq 2$ there exist constants γ, $\varepsilon > 0$ such that the inequalities

$$\begin{aligned} V_2(x_{u_N}(N-2)) &\leq \max\{(\gamma+1)V_1(x_{u_N}(N-2)) + (1-\gamma)\varepsilon, \\ &\qquad V_1(x_{u_N}(N-2)) + \varepsilon\} \qquad (3.32) \\ V_k(x_{u_N}(N-k)) &\leq \max\{l(x_{u_N}(N-k), \mu_k(x_{u_N}(N-k))) + (k-1)\varepsilon, \\ &\qquad (\gamma+1)l(x_{u_N}(N-k), \mu_k(x_{u_N}(N-k))) + (k-\gamma-1)\varepsilon\} \qquad (3.33) \end{aligned}$$

hold for all $k \in \{3, \ldots, N\}$ and all $n \in [0, n_0] \subset \mathbb{N}_0$ using the abbreviation $x_{u_N}(j) := x_{u_N}(j, x(n))$ for the open–loop solution given by (2.10).

By choosing $\varepsilon = 0$ we obtain that this assumption is a direct relaxation of Assumption 3.11. From a geometrical point of view, this relaxation is a vertical shift of the running cost and simultaneously of the optimal value function, see (3.32) and (3.33) respectively. As a result, this allows us to avoid violations of the relaxed Lyapunov inequality (3.8) if the parameter ε is chosen appropriately. Hence, we persue a similar approach as in the previous section.

Proposition 3.33
Consider $N \in \mathbb{N}$ and assume that Assumption 3.32 holds for this N. Then

$$\begin{aligned} \min\left\{\frac{(\gamma+1)^{N-2}}{(\gamma+1)^{N-2} + \gamma^{N-1}}\left(V_N(x(n)) - N\varepsilon\right), V_N(x(n)) - N\varepsilon\right\} \\ \leq V_{N-1}(x(n)) - (N-1)\varepsilon \end{aligned}$$

holds for all $n \in [0, n_0] \subset \mathbb{N}_0$.

Proof. In order to prove the assertion, we show that given Assumption 3.32 for $V_k(\cdot)$, $k = 2, \dots, N$, Assumption 3.11 holds for a modified optimal value function $\widetilde{V}_k(\cdot)$ and Proposition 3.14 guarantees our assertion.
Again, we use the abbreviation $x_{u_N}(j) := x_{u_N}(j, x(n))$, $j = 0, \dots, N$, since all our calculations using the open–loop trajectory defined by (2.14), (2.10) refer to the fixed initial value $x(n)$ with $n \in [0, n_0] \subset \mathbb{N}_0$.
Now, we consider the stage cost

$$\tilde{l}(x, u) = l(x, u) - \varepsilon \tag{3.34}$$

which gives us the following dependency

$$\widetilde{V}_k(x) = V_k(x) - k\varepsilon \tag{3.35}$$

for arbitrary x and u. Using (3.33), we obtain

$$\begin{aligned}
\widetilde{V}_k(x_{u_N}(N-k)) &\overset{(3.33)}{\leq} \max\{(\gamma+1)l(x_{u_N}(N-k), \mu_k(x_{u_N}(N-k))) + (k-\gamma-1)\varepsilon, \\
&\qquad l(x_{u_N}(N-k), \mu_k(x_{u_N}(N-k))) + (k-1)\varepsilon\} - k\varepsilon \\
&= \max\{(\gamma+1)\left(l(x_{u_N}(N-k), \mu_k(x_{u_N}(N-k))) - \varepsilon\right), \\
&\qquad l(x_{u_N}(N-k), \mu_k(x_{u_N}(N-k))) - \varepsilon\} \\
&\overset{(3.34)}{=} \max\{(\gamma+1)\tilde{l}(x_{u_N}(N-k), \mu_k(x_{u_N}(N-k))), \\
&\qquad \tilde{l}(x_{u_N}(N-k), \mu_k(x_{u_N}(N-k)))\}
\end{aligned}$$

for $\widetilde{V}_k(\cdot)$ where $k \in [N_0, N]$ is arbitrary. Hence, we can see that Assumption (3.13) holds using either $\gamma = 0$ or γ from Assumption 3.32. Thus, the induction step in the proof of Proposition 3.14 can be utilized here as well.
The induction anker can be shown using (3.32)

$$\begin{aligned}
\widetilde{V}_2(x_{u_N}(N-2)) &\overset{(3.32)}{\leq} \max\{V_1(x_{u_N}(N-2)) + \varepsilon, (\gamma+1)V_1(x_{u_N}(N-2)) + (1-\gamma)\varepsilon\} - 2\varepsilon \\
&= \max\{V_1(x_{u_N}(N-2)) - \varepsilon, (\gamma+1)V_1(x_{u_N}(N-2)) - (\gamma+1)\varepsilon\} \\
&\overset{(3.35)}{=} \max\{\widetilde{V}_1(x_{u_N}(N-2)), (\gamma+1)\widetilde{V}_1(x_{u_N}(N-2))\}.
\end{aligned}$$

Hence, we proved that (3.12) holds for either $\gamma = 0$ or γ from Assumption 3.32 for $\widetilde{V}_k(\cdot)$ and $\tilde{l}(\cdot, \cdot)$. Therefore, according to Proposition 3.14, we can conclude

$$\min\{\eta_k \widetilde{V}_k(x_{u_N}(N-k)), \widetilde{V}_k(x_{u_N}(N-k))\} \overset{(3.16)}{\leq} \widetilde{V}_{k-1}(x_{u_N}(N-k)).$$

Last, inserting (3.35) and choosing $k = N$ proves the assertion. □

Theorem 3.34
Consider γ and $N \in \mathbb{N}$ such that $(\gamma+1)^{N-2} > \gamma^N$ holds. If Assumption 3.32 is fulfilled for these γ, N and some $\varepsilon > 0$, then the estimate

$$\alpha \overline{V}_I^{\mu_N}(x(n)) \leq V_\infty(x(n)) - \sigma \qquad \text{with} \qquad \alpha = \frac{(\gamma+1)^{N-2} - \gamma^N}{(\gamma+1)^{N-2}} \tag{3.36}$$

holds for all $n \in [0, n_0] \subset \mathbb{N}_0$ with $\overline{V}_I^{\mu_N}(\cdot)$ according to (3.29).

Proof. Using Proposition 3.33 we obtain

$$V_N(x(n)) - V_{N-1}(x(n)) - \varepsilon \leq \max\left\{\frac{\gamma^{N-1}}{(\gamma+1)^{N-2}}\left(V_{N-1}(x(n)) - (N-1)\varepsilon\right), 0\right\}. \quad (3.37)$$

Now we use a construction similar to the proof of Theorem 3.15. Again, the abbreviation $x_{u_N}(j) := x_{u_N}(j, x(n))$, $j = 0, \ldots, N$ denotes the open–loop trajectory defined by (2.14), (2.10).
First we obtain the inequality

$$V_{k-1}(f(x_{u_N}(\tilde{n}), \mu_k(x_{u_N}(\tilde{n})))) - (k-1)\varepsilon \leq \max\left\{\gamma\left(l(x_{u_N}(\tilde{n}), \mu_k(x_{u_N}(\tilde{n}))) - \varepsilon\right), 0\right\} \quad (3.38)$$

for $\tilde{n} := N - k$ and $k \in \{3, \ldots, N\}$. This is can be shown using Assumption 3.32 and the principle of optimality

$$\begin{aligned}
&V_{k-1}(f(x_{u_N}(\tilde{n}), \mu_k(x_{u_N}(\tilde{n})))) = V_k(x_{u_N}(\tilde{n})) - l(x_{u_N}(\tilde{n}), \mu_k(x_{u_N}(\tilde{n})))\\
\overset{(3.33)}{\leq}\ & \max\{(\gamma+1)l(x_{u_N}(\tilde{n}), \mu_k(x_{u_N}(\tilde{n}))) + (k-\gamma-1)\varepsilon,\\
&\qquad l(x_{u_N}(\tilde{n}), \mu_k(x_{u_N}(\tilde{n}))) + (k-1)\varepsilon\} - l(x_{u_N}(\tilde{n}), \mu_k(x_{u_N}(\tilde{n})))\\
=\ & \max\{\gamma(l(x_{u_N}(\tilde{n}), \mu_k(x_{u_N}(\tilde{n}))) - \varepsilon, 0\} + (k-1)\varepsilon\}
\end{aligned}$$

Coming back to our construction, we consider $j = n-1$ in (3.37) with $x_0 = x(j)$:

$$\begin{aligned}
&V_N(x(j+1)) - V_{N-1}(x(j+1)) - \varepsilon\\
&\leq \max\left\{\frac{\gamma^{N-1}}{(\gamma+1)^{N-2}}\left(V_{N-1}(x(j+1)) - (N-1)\varepsilon\right), 0\right\}\\
&= \max\left\{\frac{\gamma^{N-1}}{(\gamma+1)^{N-2}}\left(V_{N-1}(f(x_{u_N}(0), \mu_N(x_{u_N}(0)))) - (N-1)\varepsilon\right), 0\right\}
\end{aligned}$$

Inserting (3.38) with $k = N$ reveals

$$V_N(x(j+1)) - V_{N-1}(x(j+1)) - \varepsilon \leq \frac{\gamma^N}{(\gamma+1)^{N-2}} \max\left\{\left(l(x(j), \mu_k(x_{u_N}(j))) - \varepsilon\right), 0\right\}.$$

Now we can identify this inequality with (3.31) where

$$\alpha = 1 - \frac{\gamma^N}{(\gamma+1)^{N-2}} = \frac{(\gamma+1)^{N-2} - \gamma^N}{(\gamma+1)^{N-2}} \quad (3.39)$$

and hence Lemma 3.31 provides the assertion. □

According to Theorem 3.34, the following algorithm computes a practical suboptimality bound α:

Algorithm 3.35 (Computing a priori practical Suboptimality Bound)

Input:	$\alpha_{\min}$	—	Closed–loop practical suboptimality estimate
	$V_N(x(n))$	—	Value function in the actual step
	N	—	Length of the horizon
	ε	—	Truncation constant

(1) Set $\gamma = 0$

(2) Compute $V_1(x_{u_N}(N-2, x(n)))$

(3) If $V_2(x_{u_N}(N-2, x(n))) > V_1(x_{u_N}(N-2, x(n))) + \varepsilon$ or $V_1(x_{u_N}(N-2, x(n))) > \varepsilon$:
Set $\gamma := \frac{V_2(x_{u_N}(N-2,x(n)))-V_1(x_{u_N}(N-2,x(n)))-\varepsilon}{V_1(x_{u_N}(N-2,x(n)))-\varepsilon}$

(4) For k from 3 to N do

(4a) If $V_k(x_{u_N}(N-k, x(n))) - l(x_{u_N}(N-k, x(n)), \mu_k(x_{u_N}(N-k, x(n)))) > (k-1)\varepsilon$ or $l(x_{u_N}(N-k, x(n)), \mu_k(x_{u_N}(N-k, x(n)))) > \varepsilon$:

- Set $\tilde{\gamma} := \frac{V_k(x_{u_N}(N-k,x(n)))-l(x_{u_N}(N-k,x(n)),\mu_k(x_{u_N}(N-k,x(n))))-(k-1)\varepsilon}{l(x_{u_N}(N-k,x(n)),\mu_k(x_{u_N}(N-k,x(n))))-\varepsilon}$
- Set $\gamma := \max\{\gamma, \tilde{\gamma}\}$

(5) If $\gamma = 0$: Print "Practical stability region reached", set $\alpha = 1$ and terminate

(6) Set $\alpha := \frac{(\gamma+1)^{N-2}-\gamma^N}{(\gamma+1)^{N-2}}$

(7) If $\alpha < 0$: Print warning "Solution may be unstable"

(8) Set $\alpha_{\min} := \min\{\alpha_{\min}, \alpha\}$ and terminate

Output:	$\alpha_{\min}$	—	Closed–loop practical suboptimality estimate
	α	—	Local practical suboptimality estimate for the actual step
	γ	—	Characteristic of the problem

Remark 3.36
In Step (3) γ can only be computed if the maximum occurs in the first argument of the right hand side of (3.32)*. Similarly, in Step (4) γ is set only if the maximum is attained in the second argument of* (3.33)*. This allows us to determine whether the practical region has been reached by checking for the parameter γ to be zero. In case of Algorithm 3.30, we are able to use the characterization of the region directly, see Figure 3.6. Here, however, the condition relates the optimal value function of subproblems and running costs to the cutting parameter ε.*

Since in our numerical examples results using Assumption 3.19 instead of Assumption 3.11 are significantly better, cf. Section 8.4.2, we want to adapt this more general case to cover practical stability as well. Note that one cannot use the same argumentation to relax the previous result due to the necessary positive definiteness of the modified stage cost $\tilde{l}(\cdot,\cdot)$ in the proof of Proposition 3.21.

Assumption 3.37
For a given N, $N_0 \in \mathbb{N}$, $N \geq N_0 \geq 2$ there exist constants γ, $\varepsilon > 0$ such that the inequalities

$$\begin{aligned} V_{N_0}(x_{u_N}(N-N_0)) \;\leq\; & \max_{j=2,\dots,N_0} \Big\{ l(x_{u_N}(N-j), \mu_{j-1}(x_{u_N}(N-j))) + \varepsilon, \\ & (\gamma+1) l(x_{u_N}(N-j), \mu_{j-1}(x_{u_N}(N-j))) + (\gamma + 1 - N_0\gamma)\varepsilon \Big\} \qquad (3.40) \\ V_k(x_{u_N}(N-k)) \;\leq\; & \max\{ l(x_{u_N}(N-k), \mu_k(x_{u_N}(N-k))) + (k-1)\varepsilon, \\ & (\gamma+1) l(x_{u_N}(N-k), \mu_k(x_{u_N}(N-k))) + (k-\gamma-1)\varepsilon \} \qquad (3.41) \end{aligned}$$

hold for all $k \in \{N_0+1,\dots,N\}$ and all $n \in [0,n_0] \subset \mathbb{N}_0$ where the open–loop solution $x_{u_N}(j) := x_{u_N}(j,x(n))$ is given by (2.10).

Again, one can see that Assumption 3.32 is a special case of Assumption 3.32 by setting $N_0 = 2$.

Proposition 3.38
Consider $N, N_0 \in \mathbb{N}$, $N \geq N_0 \geq 2$, and assume that Assumption 3.37 holds for these constants. Then

$$\begin{aligned}\min\left\{\frac{(\gamma+1)^{N-N_0}}{(\gamma+1)^{N-N_0}+\gamma^{N-N_0+1}}\left(V_N(x(n))-N\varepsilon\right), V_N(x(n))-N\varepsilon\right\}\\ \leq V_{N-1}(x(n))-(N-1)\varepsilon\end{aligned}$$

holds for all $n \in [0,n_0] \subset \mathbb{N}_0$.

Proof. Note that $\tilde{l}(\cdot,\cdot)$ and $\widetilde{V}_k(\cdot)$ may become negative. Hence, in the proof of Proposition 3.14, the construction showing the induction anker to hold cannot be applied here. However, the induction step itself holds for $k = N_0+1,\dots,N$ since we do not have to assume those terms to be nonnegative.
According to the changes in our assumptions, we only need to prove the induction anker. Using (3.35) and (3.40) we obtain

$$\begin{aligned}\widetilde{V}_{N_0}(x_{u_N}(N-N_0)) &\leq \max_{j=2,\dots,N_0}\Big\{l(x_{u_N}(N-j),\mu_{j-1}(x_{u_N}(N-j)))+\varepsilon,\\ &\qquad (\gamma+1)l(x_{u_N}(N-j),\mu_{j-1}(x_{u_N}(N-j)))+(\gamma+1-N_0\gamma)\varepsilon\Big\}-N_0\varepsilon\\ &\leq \max\Bigg\{\sum_{j=2}^{N_0}\left[(\gamma+1)l(x_{u_N}(N-j),\mu_{j-1}(x_{u_N}(N-j)))\right]+(\gamma+1-N_0\gamma)\varepsilon,\\ &\qquad \sum_{j=2}^{N_0}\left[l(x_{u_N}(N-j),\mu_{j-1}(x_{u_N}(N-j)))\right]+\varepsilon\Bigg\}-N_0\varepsilon\end{aligned}$$

which can be reformulated given the definition of the optimal value function as

$$\begin{aligned}\widetilde{V}_{N_0}(x_{u_N}(N-N_0)) &\leq \max\Big\{(\gamma+1)V_{N_0-1}(x_{u_N}(N-N_0))+(\gamma+1-N_0\gamma)\varepsilon,\\ &\qquad V_{N_0-1}(x_{u_N}(N-N_0))+\varepsilon\Big\}-N_0\varepsilon\\ &= \max\Big\{(\gamma+1)\widetilde{V}_{N_0-1}(x_{u_N}(N-N_0))+(\gamma+1-N_0\gamma)\varepsilon+(N_0-1)(\gamma+1)\varepsilon,\\ &\qquad \widetilde{V}_{N_0-1}(x_{u_N}(N-N_0))+\varepsilon+(N_0-1)\varepsilon\Big\}-N_0\varepsilon\\ &\overset{(3.35)}{=} \max\Big\{(\gamma+1)\widetilde{V}_{N_0-1}(x_{u_N}(N-N_0)),\widetilde{V}_{N_0-1}(x_{u_N}(N-N_0))\Big\}\end{aligned}$$

Now, we can conclude our assertion analogously to the proof of Proposition 3.33. □

Theorem 3.39
Consider $\gamma > 0$ and N, $N_0 \in \mathbb{N}$, $N \geq N_0$ such that $(\gamma+1)^{N-N_0} > \gamma^{N-N_0+2}$ holds. If Assumption 3.37 is fulfilled for these γ, N, N_0 and some $\varepsilon > 0$, then the estimate

$$\alpha\overline{V}_I^{\mu_N}(x(n)) \leq V_\infty(x(n))-\sigma \qquad \text{with} \qquad \alpha = \frac{(\gamma+1)^{N-N_0}-\gamma^{N-N_0+2}}{(\gamma+1)^{N-N_0}} \tag{3.42}$$

holds for all $n \in [0,n_0] \subset \mathbb{N}_0$ with $\overline{V}_I^{\mu_N}(\cdot)$ according to (3.29).

Proof. Using Proposition 3.38 we obtain

$$V_N(x(n)) - V_{N-1}(x(n)) - \varepsilon \leq \max\left\{\frac{\gamma^{N-N_0+1}}{(\gamma+1)^{N-N_0}}\left(V_{N-1}(x(n)) - (N-1)\varepsilon\right), 0\right\}. \quad (3.43)$$

Now, we use the same construction as in the proof of Theorem 3.15. Similarly, we use the abbreviation $x_{u_N}(j) := x_{u_N}(j, x(n))$, $j = 0, \ldots, N$, since again all our calculations using the open–loop trajektory defined by (2.14), (2.10) refer to the fixed initial value $x(n)$ with $n \in [0, n_0] \subset \mathbb{N}_0$.
We consider $j = n - 1$ in (3.43)

$$\begin{aligned}
&V_N(x(j+1)) - V_{N-1}(x(j+1)) - \varepsilon \\
&\leq \max\left\{\frac{\gamma^{N-N_0+2}}{(\gamma+1)^{N-N_0}}\left(V_{N-1}(x(j+1)) - (N-1)\varepsilon\right), 0\right\} \\
&= \max\left\{\frac{\gamma^{N-N_0+2}}{(\gamma+1)^{N-N_0}}\left(V_{N-1}(f(x_{u_N}(0), \mu_N(x_{u_N}(0)))) - (N-1)\varepsilon\right), 0\right\}
\end{aligned}$$

with $x_0 = x(j)$. Inserting (3.38) with $k = N$ gives us

$$\begin{aligned}
&V_N(x(j+1)) - V_{N-1}(x(j+1)) - \varepsilon \\
&\leq \max\left\{\frac{\gamma^{N-N_0+2}}{(\gamma+1)^{N-N_0}} \max\left\{\left(l(x_{u_N}(0), \mu_k(x_{u_N}(0))) - \varepsilon\right), 0\right\}, 0\right\} \\
&= \frac{\gamma^{N-N_0+2}}{(\gamma+1)^{N-N_0}} \max\left\{\left(l(x(j), \mu_k(x_{u_N}(j))) - \varepsilon\right), 0\right\}.
\end{aligned}$$

Hence, we obtain our assertion using Lemma 3.31 with

$$\alpha = 1 - \frac{\gamma^{N-N_0+2}}{(\gamma+1)^{N-N_0}} = \frac{(\gamma+1)^{N-N_0} - \gamma^{N-N_0+2}}{(\gamma+1)^{N-N_0}}. \quad (3.44)$$

□

To compute the practical suboptimality estimate given by Theorem 3.39, we use the following algorithm, see also Section 6.2.2.4 for the implementation and Section 8.4.4 for numerical results.

Algorithm 3.40 (Computing a priori practical Suboptimality Bound)

Input:	$\alpha_{\min}$	—	Closed–loop practical suboptimality estimate
	$V_N(x(n))$	—	Value function in the actual step
	N	—	Length of the horizon
	N_0	—	Length of the comparison horizon
	ε	—	Truncation constant

(1) Set $l_{\max} = 0$, $\gamma = 0$

(2) For j from 2 to N_0 do

(2a) Compute $l(x_{u_N}(N-j, x(n)), \mu_{j-1}(x_{u_N}(N-j, x(n))))$

(2b) Set $l_{\max} = \max\{l_{\max}, l(x_{u_N}(N-j, x(n)), \mu_{j-1}(x_{u_N}(N-j, x(n))))\}$

(3) If $V_{N_0}(x_{u_N}(N-N_0)) > l_{\max} + \varepsilon$ and $l_{\max} > (N_0 - 1)\varepsilon$:

Set $\gamma := \frac{V_{N_0}(x_{u_N}(N-N_0))-l_{\max}-\varepsilon}{l_{\max}-(N_0-1)\varepsilon}$

(4) For k from N_0+1 to N do

(4a) If $V_k(x_{u_N}(N-k,x(n))) - l(x_{u_N}(N-k,x(n)),\mu_k(x_{u_N}(N-k,x(n)))) > (k-1)\varepsilon$ and $l(x_{u_N}(N-k,x(n)),\mu_k(x_{u_N}(N-k,x(n)))) > \varepsilon$:

- Set $\tilde{\gamma} := \frac{V_k(x_{u_N}(N-k,x(n)))-l(x_{u_N}(N-k,x(n)),\mu_k(x_{u_N}(N-k,x(n))))-(k-1)\varepsilon}{l(x_{u_N}(N-k,x(n)),\mu_k(x_{u_N}(N-k,x(n))))-\varepsilon}$
- Set $\gamma := \max\{\gamma,\tilde{\gamma}\}$

(5) If $\gamma = 0$: Print "Practical stability region reached", set $\alpha = 1$ and terminate

(6) Set $\alpha := \frac{(\gamma+1)^{N-N_0}-\gamma^{N-N_0}+2}{(\gamma+1)^{N-N_0}}$

(7) If $\alpha < 0$: Print warning "Solution may be unstable"

(8) Set $\alpha_{\min} := \min\{\alpha_{\min},\alpha\}$ and terminate

Output: $\alpha_{\min}$ — Closed–loop practical suboptimality estimate
α — Local practical suboptimality estimate for the actual step
γ — Characteristic of the problem

Assumption 3.19 represents sufficient conditions separately for each N_0. This allows us to choose the best resulting suboptimality estimate. Algorithmically, we can call Algorithm 3.40 for various N_0, see also Section 6.2.2.4. The computational costs, however, rise dramatically if the parameter N_0 is increased since a larger number of long optimal control problems have to be solved.

Remark 3.41
In some references, an inequality of the form

$$V_k(x) \leq \Phi(x)$$

for some function $\Phi : \mathbb{X} \to \mathbb{R}_0^+$, all $k \in \mathbb{N}$ and all $x \in \mathbb{X}$ is imposed in order to conclude stability or practical stability of the RHC closed loop, cf. e.g. Assumption 4 in [87]. Note that Assumption 3.19 fits into this framework while Assumption 3.37 is more general.

3.4 Other Stability and Suboptimality Results

In the past 20 years different possibilities to guarantee stability of the closed–loop system have been proposed which can be divided in three classes.
The first class is characterized by an additional *terminal point constraint* connecting every initial value to the desired equilibrium point by a feasible trajectory over the open–loop horizon, see e.g. [37, 126, 158, 159, 162]. Additionally, we obtain a first estimate for suboptimality of the receding horizon controller.
A relaxation of this approach is the so called *quasi–infinite horizon method*. Here, the terminal constraint regards a closed connected set. To compensate this, the cost functional is modified by adding a Mayer term $F(\cdot)$, see also Remark 2.14, which in principle overestimates the remaining infinite time cost corresponding to the terminal state, see e.g. [38, 39, 68, 117, 164]. In this setting, the concept of *inverse optimality* as described

in [22, 150, 160, 173] can be utilized to qualify a resulting controller to be optimal in some sense.
Last, a suboptimality estimation approach using observability and controllability conditions has been proposed in [87, 92, 100, 102] which turns out to be checkable using a small linear program.
Here, we first describe the two endpoint–based stability approaches as well as the term of inverse optimality shortly. Thereafter, we state the controllability–based result and conclude by discussing conceptual advantages and disadvantages as well as correctness of these methods compared to our proposed stability results of Propositions 3.3, 3.33 and Theorems 3.22, 3.39.

3.4.1 Terminal Point Constraint

In 1988, a first stability result for nonlinear receding horizon control was established in [126] using the idea of a moving terminal set $\mathbb{X}_f$. For linear system, this has been shown earlier in [215, 216]. For simplicity of exposition, we consider the origin to be the desired equilibrium. The key idea is to add a *terminal point constraint*

$$x_{u_N}(N, x) = 0 \tag{3.45}$$

to the problems RHC_N and SDOCP_N respectively. Basically, if a trajectory is feasible for the underlying open–loop problem SDOCP_N, then the equilibrium $x^\star = 0$ is reached within the considered time horizon.
To show stability we assume the following:

Assumption 3.42 (1) The constraint set $\mathbb{X}$ is closed.

(2) The dynamic of the system $f : \mathbb{R}^n \times \mathbb{U} \to \mathbb{R}^n$ is continuous and $f(0, 0) = 0$.

(3) The stage cost $l : \mathbb{R}^n \times \mathbb{U} \to \mathbb{R}_0^+$ is lower semi–continuous and $l(0, 0) = 0$.

(4) There exists a class $\mathcal{K}_\infty$–function L_1 such that

$$l(x, u) \geq L_1(\|(x, u)\|)$$

holds for all $(x, u) \in \mathbb{R}^n \times \mathbb{U}$.

Theorem 3.43 (Stability using a Terminal Point Constraint)
Consider the problems RHC_N and $SDOCP_N$ with additional terminal point constraint (3.45) *respectively. Suppose Assumption 3.42 to hold. If x_0 is chosen such that problem $SDOCP_N$ exhibits a feasible solution, then $V_\infty(x_0)$, $V_\infty^{\mu_N}(x_0)$ and $V_N(x_0)$ are finite and satisfy*

$$V_\infty(x_0) \leq V_\infty^{\mu_N}(x_0) \leq V_N(x_0). \tag{3.46}$$

Within this context, another remarkable property of the problem SDOCP_N can be observed — that is, the value of the finite horizon cost functional converges to the value of the infinite horizon cost functional. To show this we require the following:

Assumption 3.44 (1) The origin is contained in $\mathbb{X} \times \mathbb{U}$.

(2) There exists a class $\mathcal{K}_\infty$–function L_2 such that

$$l(x,u) \leq L_2(\|(x,u)\|)$$

holds for all $(x,u) \in \mathbb{R}^n \times \mathbb{U}$.

Theorem 3.45 (Convergence of the Costfunctional using a Terminal Point Constraint) *Consider Assumptions 3.42 and 3.44 to be satisfied and $\delta > 0$ to be a given real constant. Moreover, we assume x_0 to be chosen such that problem SDOCP$_N$ together with* (3.45) *possesses a feasible solution. Then, there exists a horizon length $\overline{N} = N(\delta, x_0)$ such that for all $N \geq \overline{N}$ the problem SDOCP$_N$ together with* (3.45) *exhibits a feasible solution and the estimate*

$$V_N(x_0) \leq V_\infty(x_0) + \delta \tag{3.47}$$

holds. Moreover, there exists a radius $r > 0$ and a horizon length $\overline{N} = N(\delta)$ such that (3.47) *holds for all $N \geq \overline{N}$ and all $x_0 \in B_r(0)$.*

For proofs of Theorems 3.43 and 3.45 we refer to [126].

Remark 3.46
In 1982 — even earlier than [126] — the method of using the value function of a finite horizon optimal control problem as a Lyapunov function to establish stability of continuous–time systems employing a terminal equality contraint has been presented in [37]. Unfortunately, it stayed unnoticed and unextended until 1989 in [158, 159].

3.4.2 Regional Terminal Constraint and Terminal Cost

In 1998 a second idea was proposed in [39] relaxing the terminal point constraint (3.45) to a regional terminal constraint

$$x_{u_N}(N,x) \in \Omega \tag{3.48}$$

respectively for problems RHC$_N$ and SDOCP$_N$. Additionally, the cost functional in both problems is modified by adding a Mayer term $F(\cdot)$ defined on this terminal set, that is

$$J_N(x,u_N) = \sum_{i=0}^{N-1} l\left(x_{u_N}(i,x), u_N(x_{u_N}(0,x),i)\right) + F(x_{u_N}(N,x)). \tag{3.49}$$

This approach is also termed *quasi–infinite horizon NMPC*. In [39], this approach was formulated for continuous–time systems. Here, we give a discrete–time equivalent, see also [160].
More precisely, the following condition are assumed to apply:

Assumption 3.47 (1) $\Omega \subset \mathbb{X}$ is closed and contains the origin.

(2) For a given local feedback $u_f(\cdot)$ we have $u_f(x) \in \mathbb{U}$ for all $x \in \Omega$.

(3) The set Ω is forward invariant for $f(\cdot,\cdot)$ under $u_f(\cdot)$, i.e. $f(x,u_f(x)) \in \Omega$ for all $x \in \Omega$.

(4) The Mayer term $F(\cdot)$ is a local Lyapunov function, i.e.

$$F(f(x, u_f(x))) - F(x) + l(x, u_f(x)) \leq 0.$$

holds for each $x \in \Omega$.

Then stability of the resulting closed–loop system can be shown.

Theorem 3.48 (Stability using Terminal Costs)
Consider the problem RHC_N with modified cost functional (3.49) *and additional constraint* (3.48). *Suppose that Assumption 3.47 holds and the stated problem exhibits a feasible solution for a set X_N of initial values x_0. Then the resulting closed–loop solution* (2.16) *is asymptotically stable for all initial values $x_0 \in X_N$.*

For corresponding assumptions and a proof in the continuous–time case, we refer to [38,39]. For the discrete–time setting, a proof can be found in [160].

3.4.3 Inverse Optimality

In the previous Sections 3.4.1 and 3.4.2, we dealed with the stability aspect of the closed–loop solution (2.16) only. Now, we aim at finding a feedback control $u(x(n))$ which stabilizes the system

$$x(n+1) = f\left(x(n), u(x(n))\right), \qquad x(0) = x_0$$

while minimizing the cost

$$J_\infty(x_0, u) = \sum_{i=0}^{\infty} l\left(x(i), u(x(i))\right). \tag{3.50}$$

This issue has been considered earlier, see e.g. [22].
Within our analysis of the receding horizon controller in Sections 3.1 – 3.3, we obtained a parameter α characterizing the degree of suboptimality of the closed–loop solution regarding the infinite horizon optimal solution. In the *inverse control problem*, see e.g. [22, 150, 160, 173], a Lyapunov function is given. Then, the aim is to determine whether a control law $u(\cdot)$ is optimal for a cost of the form (3.50). This leads to the so called L_gV–control for control–affine systems studied, among others, in [116, 131].
In order to apply this idea to the receding horizon controller, we need the following assumptions:

Assumption 3.49
Suppose $F(\cdot)$ to be modeled implying the existence of a closed set $\mathbb{X}_N \subset \mathbb{R}^n$ and a control law u_N such that the following conditions hold:

(1) For all $x_0 \in \mathbb{X}_N$ there exists a feasible control $u_N(x_0) \in \mathbb{U}$.

(2) If the initial value x_0 is contained in $\mathbb{X}_N$, then the subsequent state along the trajectory stays in this set, i.e. $f(x_0, \mu_N(x_0)) \in \mathbb{X}_N$.

(3) For all $x_0 \in \mathbb{X}_N$ the Mayer term $F(\cdot)$ is decreasing in one step at least by the size of the stage cost, that is

$$F(x_{u_N}(N, u_N(x(1)))) - F(x_{u_N}(N, u_N(x_0))) \leq -l\left(x_0, u_N(x_0)\right).$$

In particular, condition (3) reveals $F(\cdot)$ to be decreasing along the closed–loop trajectory. Note that $F(\cdot)$ is not necessarily a local control Lyapunov function as in Section 3.4.2 to exhibit such a characteristic, see also [117]. This property can be used to establish the following result shown e.g. in [150, 173].

Theorem 3.50 (Inverse Optimality)
Consider a problem RHC_N with cost functional (3.49) *such that Assumption 3.49 holds. Then there exists a set of initial values $\mathbb{X}_0 \subset \mathbb{R}^n$ and a function $\mathcal{L} : \mathbb{R}^n \times \mathbb{U} \to \mathbb{R}_0^+$ with*

$$\mathcal{L}(x, u_N(x)) \geq l(x, u_N(x)) \qquad \forall x \in \mathbb{X}_0 \tag{3.51}$$

such that for all $x_0 \in \mathbb{X}_0$ the resulting closed–loop controller $\mu_N(\cdot)$ minimizes the modified cost functional

$$J_\infty(x_0, \mu_N) = \sum_{i=0}^{\infty} \mathcal{L}\left(x_{\mu_N}(i, x_0), \mu_N(x(i))\right). \tag{3.52}$$

3.4.4 Controllability–based Suboptimality Estimates

In [92], a controllability assumption based on the running cost $l(\cdot, \cdot)$ instead of the trajectory is imposed to obtain an estimate for the trade–off between the infinite horizon optimal control (2.8) and the receding horizon control law (2.15):

Assumption 3.51
There exists a function $W : \mathbb{X} \to \mathbb{R}_0^+$ and constants $\vartheta, C > 0$ and $0 \leq \sigma < 1$ such that for all $x \in \mathbb{X}$ we have:

(1) $l(x, u) \geq \vartheta W(x)$ holds for all $u \in \mathbb{U}$.

(2) There exists a control sequence $u^\star \in \mathcal{U}$ such that

$$l(x(n), u^\star(n)) \leq C\sigma^n W(x) \tag{3.53}$$

holds along the solution $x(n)$ defined by (2.10) with $u(\cdot) := u^\star(\cdot)$ emanating from the initial value $x(0) = x_0$.

Using this assumption, we retrieve a connection to our results from Theorem 3.15, see [92] for a proof:

Theorem 3.52
Suppose Assumption 3.51 to hold. Then Assumption 3.11 holds with

$$\gamma := \frac{C}{\vartheta(1-\sigma)} - 1 \tag{3.54}$$

and the suboptimality degree α can be computed for all $x \in \mathbb{X}$ according to (3.17).

Using a different controllability assumption, one can define a linear program to compute the degree of suboptimality. In [99, 100, 102], the following setting was analyzed:

Assumption 3.53
Given a function $\beta \in \mathcal{KL}_0$, for each initial value $x_0 \in \mathbb{X}$ there exists a control function $u_N(x_0, \cdot) \in \mathcal{U}$ satisfying

$$l(x_{u_N}(x_0, n), u_N(x_0, n)) \leq \beta(l^\star(x_0), n) \tag{3.55}$$

for all $n \in \mathbb{N}_0$.

Here, a continuous function $\beta : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is of class $\mathcal{KL}_0$ if for each $r > 0$ we have $\lim_{t\to\infty} \beta(r,t) = 0$ and for each $t \geq 0$ we either have $\beta(\cdot,t) \in \mathcal{K}_\infty$ or $\beta(\cdot,t) \equiv 0$. Using Assumption 3.53, for any $r \geq 0$ and any $N \geq 1$ one defines the values

$$B_N(r) := \sum_{n=0}^{N-1} \beta(r,n) \tag{3.56}$$

as well as

$$\begin{aligned} \lambda_n &:= l(x_{u_N}(n), u_N(x_{u_N}(0),n)) \\ \nu &:= V_N(x_{u_N}(0)) \end{aligned}$$

with $u_N(\cdot,\cdot)$ defined by (2.14). This allows to show the following, see [92] for a proof:

Proposition 3.54
Suppose Assumption 3.53 holds and consider a horizon $N \geq 1$ within problem RHC_N using a m–step feedback (2.18) with $m \in \{1,\dots,N-1\}$. Assume $u_N(x_0,\cdot) \in \mathcal{U}$ to be a minimizing control of problem $SDOCP_N$ with initial value $x_0 \in \mathbb{X}$. If $\lambda_n > 0$, $n = 0,\dots,N-1$, holds, then we have

$$\sum_{n=k}^{N-1} \lambda_n \leq B_{N-k}(\lambda_k), \quad k = 0,\dots,N-2 \tag{3.57}$$

and if furthermore $\nu > 0$ we obtain

$$\nu \leq \sum_{n=0}^{j-1} \lambda_{n+m} + B_{N-j}(\lambda_{j+m}), \quad j = 0,\dots,N-m-1. \tag{3.58}$$

Proposition 3.54 can be seen as necessary condition for suboptimality of the m–step MPC feedback law $\mu_{N,m}$. In order to obtain sufficient conditions as shown in [92], we require the following:

Theorem 3.55
Consider $\beta \in \mathcal{KL}_0$, a horizon $N \geq 1$ within problem RHC_N using a m–step feedback (2.18) with $m \in \{1,\dots,N-1\}$ and assume that all sequences $\lambda_n > 0$, $n = 0,\dots,N-1$ and values $\nu > 0$ fulfilling (3.57), (3.58) satisfy the inequality

$$\sum_{n=0}^{N-1} \lambda_n - \nu \geq \alpha \sum_{n=0}^{m-1} \lambda_n \tag{3.59}$$

for some $\alpha \in (0,1]$. Then, for each optimal control problem $SDOCP_N$ satisfying Assumption 3.53, the inequality

$$\alpha V_\infty(x) \leq \alpha V_\infty^{\mu_{N,m}}(x) \leq V_N(x)$$

holds for all $x \in \mathbb{X}$.

Combining Proposition 3.54 and Theorem 3.55 allow us to formulate the linear program

$$\begin{aligned} \text{Minimize } \alpha := \inf_{\lambda_0,\ldots,\lambda_{N-1},\nu} & \frac{\sum_{n=0}^{N-1}\lambda_n - \nu}{\sum_{n=0}^{m-1}\lambda_n} \\ \text{ST. } \nu \leq & \sum_{n=0}^{j-1}\lambda_{n+m} + B_{N-j}(\lambda_{j+m}), \quad j = 0,\ldots,N-m-1 \\ & \sum_{n=k}^{N-1}\lambda_n \leq B_{N-k}(\lambda_k), \quad k = 0,\ldots,N-2 \end{aligned}$$

to compute the suboptimality estimate α.

3.4.5 Subsumption of the presented Approaches

Considering Theorem 3.43, the stated Assumptions 3.42 are motivated by establishing stability only, in particular there is no direct connection between the physical plant and the introduced terminal point condition (3.45). This results in a modification of the problem which does not represent the original one in general, hence the resulting controls and the closed–loop trajectories of these two problems most likely deviate. Additionally, the necessary computing effort is high since the required horizon length N must be chosen large such that existence of a feasible solution of the optimization problem can be guaranteed.
The approach used in Theorem 3.48 relaxes this issue by enlarging the terminal constraint region. Hence, feasible trajectories are obtained for significantly shorter horizon lengths N. Moreover, the additive terminal cost can be justified as an (over-)approximation of the anticipated costs on the infinite horizon. Still, the strictly monotone decrease in $F(\cdot)$, see Assumption 3.47(5), needs to be established. Moreover, the resulting closed–loop trajectory again deviates from the original one since by adding the additional constraint and modifying the cost functional the optimization problem exhibits a different solution in general.
The setting considered within Proposition 3.3 — and related statements thereafter — represents the majority of industrially implemented receding horizon controllers, see e.g. [16, 54]. In order to establish these results, no further assumptions changing the optimal control problem have to be made. Therefore, the resulting controls can be regarded as correctly computed. Moreover, no additional (analytical or numerical) effort is necessary to obtain a suitable endcost term and no terminal set is required. According to these arguments, our results can be regarded as superior compared to these settings.
The estimates from Section 3.4.4, however, are based on the setvalued relaxed Lyapunov inequality (3.6). They correspond to a RHC_N problem formulation which is identical to our setting from Sections 3.1 – 3.3 and hence have to be regarded as correctly computed as well. Yet, it appears to be difficult to obtain the required parameters and functions of Assumptions 3.51 or 3.53 for general nonlinear problems. Hence, this methodology is, at least until now, restricted to a smaller class of problems compared to our approach. Moreover, we expect our results to be less conservative since the estimates are not required to hold for the whole state space $\mathbb{X}$ but only those points which are visited by the closed–loop trajectory (2.16).

As mentioned before, the primary assertion of Theorem 3.43 is the stability property of the closed–loop. Still, it also offers an estimate for suboptimality via $V_\infty^{\mu_N}(x_0) \leq V_N(x_0)$

but the optimal control problems corresponding to $V_N(\cdot)$ and $V_\infty(\cdot)$ differ due to the introduction of the terminal point constraint in the finite horizon problem. As a result, we can not establish the inequality $V_N(x) \leq V_\infty(x)$ to obtain an upper bound in terms of $V_\infty(\cdot)$. Such a bound can be set up using Theorem 3.45 but until now there exists no common formula to compute the required horizon length N. Hence, the method proposed in Section 3.4.1 is suitable to obtain some bound on the cost of the closed–loop, but it cannot be employed to develop an quantitative suboptimality property.
Using the inverse optimality concept stated in Theorem 3.50 allows us to show optimality of the closed–loop regarding the cost functional $\mathcal{L}(\cdot,\cdot)$. Additionally, the robustness property of the infinite horizon controller carries over to receding horizon controller. However, the function $\mathcal{L}(\cdot,\cdot)$ is in general unknown or computationally costy since it requires the solution of two optimization problems, see e.g. [160,173]. Moreover, since the closed–loop controller is optimal for a different cost functional, we can e.g. utilize [92, Theorem 6.4] to obtain an upper bound on $V_\infty^{\mu_N}(\cdot)$. However, this result reveals no quantitative estimate with respect to $V_\infty(\cdot)$ which makes it unapplicable for our adaption strategies in the following Chapter 4.
The methods proposed in Sections 3.1, 3.2 and 3.3 allow for a computation of the introduced suboptimality degree α relating the costs of the closed–loop to the costs of the infinite horizon control. Since the corresponding algorithms can be applied along the closed–loop trajectory without significant additional effort, they are well suited to develop algorithms to locally guarantee a lower bound on this parameter. As a result, we additionally obtain stability of the closed–loop. Moreover, the basic Proposition 3.3 gives rise to further theoretical insight of the receding horizon control problem, that is, among others, to compute stability and performance bounds [90] or to show improved performance of the closed–loop considering longer control horizons [99].

Chapter 4

Adaptive Receding Horizon Control

Using the suboptimality estimates from the previous Chapter 3, our aim is to develop an automatic adaptation strategy in order to reduce the computing time necessary to solve the receding horizon control problem.
The main idea is to accelerate the calculation of the open–loop control u_N given by (2.14) in every step of the receding horizon controller setup. At the same time, however, we want to guarantee a certain degree of suboptimality $\overline{\alpha} \in [0, 1)$ of the solution (3.3), (3.4) compared to the infinite horizon solution (1.1), (2.8) with $u(n) = \mu(x(n))$. This aim renders our desired algorithm to be close to step size control methods of ODE solvers like Runge–Kutta–Fehlberg, Dormand–Prince or Radau. Yet, the parameter of choice is different: From the literature we know that longer optimization horizons N (apart from numerical difficulties) lead to improved closed–loop solutions. The parameter N also has a major impact on the computing time of each open–loop control law u_N. Therefore, designing strategies to adapt the horizon length N according to the problem and the state of the system is a natural choice.
In order to develop such an adaptation strategy, we utilize the a priori and a posteriori estimates given in the previous Chapter 3. These estimates allow us to quantify the degree of suboptimality which is the comparison criterion we intend to use to develop shortening and prolongation strategies for the optimization horizon N.
So far, we have seen that the existence of $\alpha \geq \overline{\alpha}$ is a sufficient condition for this purpose if the horizon length N is not changed. Yet, the resulting lower bound $\overline{\alpha}$ does not hold true for the whole trajectory if we vary N along the closed–loop solution. Therefore, we extend the result of Proposition 3.3 and 3.28 to our actual setting in Section 4.1. In the following Section 4.2, we develop a basic adaptation algorithm from the a posteriori result of Propositions 3.3. This scheme is extended in Section 4.3 using the a priori suboptimality estimate of Theorem 3.22. In the final Section 4.4, we consider the practical suboptimality estimates of Proposition 3.28 and Theorem 3.39 to cover the case of practically asymptotically stabilizable systems.

4.1 Suboptimality Estimates for varying Horizon

In the previous Chapter 3 we assumed the optimization horizon N to be fixed for all iterates of the receding horizon controller and developed a local suboptimality estimate α for this controller. Now, our aim is to vary the optimization horizon of the receding horizon controller along the closed–loop solution 2.16. Note that Algorithms 3.8, 3.23, 3.30 and 3.40 can still be applied for every point along the closed–loop if the optimization

horizon is known. The stability proofs of Propositions 3.3 and 3.28, however, do not hold since the terms in the telescope sum argument do not cancel out each other in general. Here, we generalize the results for fixed optimization horizon N from Propositions 3.3 and 3.28. To this end, we assume that if there exists a horizon length parameter N such that $\alpha(N) \geq \overline{\alpha}$ holds, then the controller shows a bounded guaranteed performance if the horizon length is increased, i.e.

Assumption 4.1
Given an initial value $x \in \mathbb{X}$ and a horizon length $N < \infty$ such that $\mu_N(\cdot)$ guarantees local suboptimality degree $\alpha(N) \geq \overline{\alpha}$, $\overline{\alpha} \in (0,1)$, we assume that for $\widetilde{N} \geq N$, $\widetilde{N} < \infty$, there exist constants $C_l, C_\alpha > 0$ such that the inequalities

$$l(x, \mu_N(x)) \leq C_l l(x, \mu_{\widetilde{N}}(x)) \tag{4.1}$$

$$\alpha(N) \leq \frac{1}{C_\alpha} \alpha(\widetilde{N}) \tag{4.2}$$

hold where $\alpha(\widetilde{N})$ is the local suboptimality degree of the controller $\mu_{\widetilde{N}}(\cdot)$ corresponding to the horizon length $\widetilde{N}$.

Remark 4.2
In order to fit the context of varying optimization horizons, we intuitively extend our notation from Chapters 2 and 3 by adding the used optimization horizon as an argument, i.e. $\alpha(N)$ denotes the variable α from Chapter 3 with horizon N. Moreover, since the resulting closed–loop control now depends on a sequence $(N_i)_{i \in \mathbb{N}}$ we denote such a control law by $\mu_{(N_i)}$.

The aim of Assumption 4.1 is to allow for non–monotone developments of the suboptimality degree $\alpha(\cdot)$ if the horizon length is increased. Still, we want to guarantee that if a certain suboptimality degree $\overline{\alpha} \in (0,1)$ holds for a horizon length N, then the estimate does not drop below zero if the horizon length is increased.

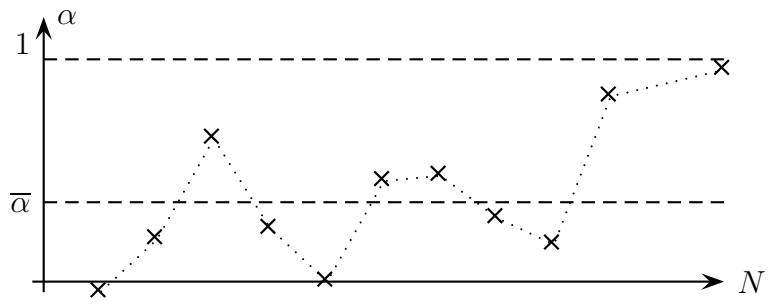


Figure 4.1: Possible development of the suboptimality degree $\alpha(\cdot)$ depending on the horizon length N

Remark 4.3
Since $l(x, \mu_{\widetilde{N}}(x))$ may tend to zero if $\widetilde{N}$ is increased, we obtain that C_l is in general unbounded. The special case $l(x, \mu_{\widetilde{N}}(x)) = 0$, however, states that the equilibrium of our problem has been reached and can be neglected in this context.

Given Assumption 4.1, we obtain stability and a performance estimate of the closed–loop in the context of changing horizon lengths similar to Proposition 3.3.

Theorem 4.4 (Stability of Adaptive RHC)
Consider $\overline{\alpha} \in [0,1)$ *and a sequence* $(N_i)_{i\in\mathbb{N}_0}$, $N_i \in \mathbb{N}$, *where* $N^\star = \max\{N_i \mid i \in \mathbb{N}\}$, *such that the receding horizon feedback law* $\mu_{(N_i)}$ *defining the closed–loop solution* (2.16) *guarantees*

$$V_{N_i}(x(i)) \geq V_{N_i}(x(i+1)) + \overline{\alpha} l(x(i), \mu_{N_i}(x(i))) \tag{4.3}$$

for all $i \in \mathbb{N}_0$. *If additionally Assumption 4.1 is satisfied for all pairs of initial values and horizons* $(x(i), N_i)$, $i \in \mathbb{N}_0$, *then we obtain*

$$\alpha_C V_\infty(x(n)) \leq \alpha_C V_\infty^{\mu_{(N_i)}}(x(n)) \leq V_{N^\star}(x(n)) \leq V_\infty(x(n)) \tag{4.4}$$

to hold for all $n \in \mathbb{N}_0$ *where* $\alpha_C := \min\limits_{j\in\mathbb{N}_{\geq n}} \frac{C_\alpha^{(j)}}{C_l^{(j)}}\overline{\alpha}$.

Proof. Given a pair $(x(i), N_i)$, Assumption 4.1 guarantees $\alpha(N_i) \leq \frac{1}{C_\alpha^{(i)}}\alpha(\widetilde{N})$ for $\widetilde{N} \geq N_i$. Now we choose $\widetilde{N} = N^\star$ within this local suboptimality estimation. Hence, we obtain

$$\overline{\alpha} \overset{(4.3)}{\leq} \alpha(N_i) \leq \frac{\alpha(N^\star)}{C_\alpha^{(i)}}.$$

using the relaxed Lyapunov inequality (4.3). Multiplying by the stage cost $l(x(i), \mu_{N_i}(x(i)))$, we can conclude

$$\begin{aligned}
\overline{\alpha} l(x(i), \mu_{N_i}(x(i))) &\leq \frac{\alpha(N^\star)}{C_\alpha^{(i)}} l(x(i), \mu_{N_i}(x(i))) \\
&= \frac{V_{N^\star}(x(i)) - V_{N^\star}(x(i+1))}{C_\alpha^{(i)} l(x(i), \mu_{N^\star}(x(i)))} l(x(i), \mu_{N_i}(x(i))) \\
&\leq \frac{C_l^{(i)}}{C_\alpha^{(i)}} \left(V_{N^\star}(x(i)) - V_{N^\star}(x(i+1))\right)
\end{aligned}$$

using (4.3) and (4.1). Summing the running costs along the closed–loop trajectory reveals

$$\alpha_C \sum_{j=i}^{K} l(x(j), \mu_{N_j}(x(j))) \leq V_{N^\star}(x(i)) - V_{N^\star}(x(K+1))$$

where we defined $\alpha_C := \min\limits_{n\in[i,\dots,K]} \frac{C_\alpha^{(n)}}{C_l^{(n)}}\overline{\alpha}$. Since $V_{N^\star}(x(K+1)) \geq 0$ holds, we can neglect it in the inequality. Taking K to infinity reveals

$$\alpha_C V_\infty^{\mu_{(N_i)}}(x(i)) = \alpha_C \lim_{K\to\infty} \sum_{j=i}^{K} l(x(j), \mu_{N_j}(x(j))) \leq V_{N^\star}(x(i)).$$

Since the first and the last inequality of (4.4) hold by the principle of optimality, the assertion follows. □

Remark 4.5
Comparing Proposition 3.3 and Theorem 4.4, we see that (4.3) *is only a sufficient condition for guaranteeing the suboptimality degree* $\overline{\alpha}$ *if the horizon length is fixed. Here, the bound* α_C *may become very small depending on* C_α *and* C_l *from Assumption 4.1 but in our numerical experiments no such case occured, see also Section 8.5.4 for numerical results.*

Similar to Proposition 3.28, we can extend Theorem 4.4 to the practical case:

Theorem 4.6 (Practical Stability of Adaptive RHC)
Suppose $\overline{\alpha} \in [0,1)$ *and* $\varepsilon > 0$ *are fixed. Consider a sequence* $(N_i)_{i\in\mathbb{N}_0}$, $N_i \in \mathbb{N}$, *where* $N^\star = \max\{N_i \mid i \in \mathbb{N}\}$, *such that the receding horizon feedback law* $\mu_{(N_i)}$ *defining the closed–loop solution* (2.16) *guarantees*

$$V_{N_i}(x(i)) \geq V_{N_i}(x(i+1)) + \min\left\{\overline{\alpha}\left(l(x(i),\mu_{N_i}(x(i))) - \varepsilon\right), l(x(i),\mu_{N_i}(x(i))) - \varepsilon\right\} \tag{4.5}$$

for all $i \in \mathbb{N}_0$. *Consider a discrete–time interval* $I := \{n_1,\dots,n_2\}$. *Let* n_1, $n_2 \in \mathbb{N}$, $n_1 < n_2$, *for which the inequality* $l(x(i),\mu_{N_i}(x(i))) \geq \varepsilon$ *holds for all* $i \in I$ *and set* $\sigma := V_{N^\star}(x(n_2+1))$. *If additionally Assumption 4.1 is satisfied for all pairs of initial values and horizons* $(x(i),N_i)$, $i \in I$, *then we obtain*

$$\alpha_C \overline{V}_I^{\mu_{(N_i)}}(x(n)) \leq V_{N^\star}(x(n)) - \sigma \leq V_\infty(x(n)) - \sigma \tag{4.6}$$

to hold for all $n \in I$ *where* $\alpha_C := \min\limits_{j\in I_{\geq n}} \frac{C_\alpha^{(j)}}{C_l^{(j)}}\overline{\alpha}$ *and* $\overline{l}(x(i),\mu_{N_i}(x(i)))$, $\overline{V}_I^{\mu_{(N_i)}}(x(n))$ *are defined in* (3.28) *and* (3.29) *respectively and* $I_{\geq n} := \{i \in I \mid i \geq n\}$.

Proof. The proof combines Proposition 3.3 and Theorem 4.4. From the definition of $\overline{l}(\cdot,\cdot)$ and I we obtain

$$\overline{\alpha}\overline{l}(x(i),\mu_{N_i}(x(i))) = \max\left\{\overline{\alpha}\left(l(x(i),\mu_{N_i}(x(i))) - \varepsilon\right), 0\right\}$$

for $i \in I$. Similar to the proof of Theorem 4.4, we obtain $\overline{\alpha} \leq \alpha(N_i) \leq \frac{\alpha(N^\star)}{C_\alpha^{(i)}}$ from Assumption 4.1. Hence, we have

$$\begin{aligned}\overline{\alpha}\overline{l}(x(i),\mu_{N_i}(x(i))) &\leq \max\left\{\frac{\alpha(N^\star)}{C_\alpha}\left(l(x(i),\mu_{N_i}(x(i))) - \varepsilon\right), 0\right\} \\ &\leq \frac{C_l^{(i)}}{C_\alpha^{(i)}}\left(V_{N^\star}(x(i)) - V_{N^\star}(x(i+1))\right)\end{aligned}$$

using (4.5) and (4.1). Thus, summing over i and defining $\alpha_C := \min\limits_{j\in I_{\geq n}} \frac{C_\alpha^{(j)}}{C_l^{(j)}}\overline{\alpha}$ gives us

$$\alpha_C \overline{V}_I^{\mu_{(N_i)}}(x(n)) = \alpha_C \sum_{j=n}^{n_2} \overline{l}(x(j),\mu_{N_j}(x(j))) \leq V_{N^\star}(x(n)) - \sigma,$$

which implies the assertion since $V_{N^\star}(x(n)) \leq V_\infty(x(n))$ follows by (3.2). □

Now, we are ready to design adaptive algorithms based on our local suboptimality estimates from the previous Chapter 3.

4.2 Basic Adaptive RHC

A basic version of an adaptive receding horizon controller (ARHC) can be implemented using Proposition 3.3. Note that this proposition represents an *a posteriori* estimate. However, the necessary data can be obtained by solving two consecutive optimal control problems using a forward prediction of the state trajectory.
A basic assumption we require is the existence of a finite horizon length parameter N guaranteeing stability with suboptimality degree greater than $\overline{\alpha}$, i.e.

Assumption 4.7
Given $\overline{\alpha} \in [0,1)$, for all $x_0 \in \mathbb{X}$ there exists a finite horizon length $\overline{N} = N(x_0) \in \mathbb{N}$ such that the relaxed Lyapunov inequality (3.6) holds with $\alpha(N) \geq \overline{\alpha}$ for all horizon lengths $N \geq \overline{N}$.

Regarding Theorem 4.4, we first fix the lowest tolerable suboptimality degree $\overline{\alpha} \in [0,1)$, then we compute a suitable horizon length parameter N such that $\alpha \geq \overline{\alpha}$ holds for α from (3.8) and last implement the resulting control value u_N. More formally, we use the following algorithm:

Algorithm 4.8 (A posteriori Adaptive Receding Horizon Control (ARHC))

Input: $\overline{\alpha}$ — Fixed suboptimality bound
N — Length of the horizon
$x(n)$ — Initial value of the plant

(1) Compute $V_N(x(n))$, $V_N(x(n+1))$

(2) Compute $\alpha(N)$ by calling Algorithm 3.8

(3) If $\alpha(N) > \overline{\alpha}$:

- Shortening Strategy

(4) Else:

- Prolongation Strategy

(5) Goto (1)

Output: $\alpha(N)$ — Local suboptimality estimate
$x(\cdot)$ — Closed–loop trajectory

In order to shorten the actual horizon length, we present two quite similar strategies which are based on the following conclusion.

Lemma 4.9 (Shortening Strategy)
Consider an optimal control problem (2.10), (2.14) *with initial value* $x_0 = x(n)$, $N \in \mathbb{N}$ *and* $\overline{\alpha} \in [0,1)$ *to be fixed. Suppose there exists an integer* $\overline{i} \in \mathbb{N}_0$, $0 \leq \overline{i} < N$ *such that*

$$V_{N-i}(x_{u_N}(i+1,x_0)) + \overline{\alpha} l(x_{u_N}(i,x_0), \mu_{N-i}(x_{u_N}(i,x_0))) \leq V_{N-i}(x_{u_N}(i,x_0)) \tag{4.7}$$

holds true for all $0 \leq i \leq \overline{i}$ *where* $x_{u_N}(i,x_0)$ *is defined by* (2.10). *Then the first* $(\overline{i}+1)$ *elements of* (2.14) *can be implemented with local suboptimality degree* $\overline{\alpha}$.

Proof. Consider the optimal control problem with fixed horizon length parameter N emanating from the given initial value. Using $i \geq 0$, $i \leq \overline{i}$, the open–loop trajectory (2.10) and (4.7) we conclude that (3.9) holds true for any optimal control problem with horizon length $N-i$ and initial value $x_{u_N}(i,x_0)$. Hence, the assertion follows. □

The idea of Lemma 4.9 is not only to allow for a shortening of the current optimization horizon and to guarantee Assumption (3.8) to hold. It also reveals an computationally cheap algorithm which implements $i \in \{0,\dots\overline{i}\}$ elements of the open–loop control sequence $u_N(\cdot,x_0)$ and checks (3.8) to hold iteratively for the corresponding horizon length $N-i$ along the resulting solution.

Reformulating Lemma 4.9 we obtain the following two slightly different algorithms:

Algorithm 4.10 (A posteriori Simple Shortening Strategy)
Closed-loop strategy:

Input:	$\overline{\alpha}$	—	Fixed suboptimality bound
	N	—	Length of the horizon
	$u_N(x(n), \cdot)$	—	Open–loop control
	$V_N(x(n))$	—	Value function
	$V_N(x(n+1))$	—	Predicted value function

(1) Implement $u_N(x(n), 0)$, obtain $x(n+1)$ and set $n := n+1$

(2) Set $u_N(x(n), k) := u_N(x(n-1), k+1)$ for all $k \in \{0, \ldots, N-2\}$

(3) If $N \geq 2$:

(3a) Set $N := N-1$

(3b) Compute $V_N(x(n+1))$

(3c) Compute $\alpha(N)$ by calling Algorithm 3.8

(3d) If $\alpha(N) < \overline{\alpha}$:

- Set $N := N+1$
- Set $u_N(x(n), N-1)$ to default value according to (2.9)

Else

- Set $u_N(x(n), k) := u_{N+1}(x(n), k)$ for all $k \in \{0, \ldots, N-1\}$

Output:	$\alpha(N)$	—	Local suboptimality estimate
	N	—	New length of the horizon
	$x(n)$	—	New initial value
	$u_N(x(n), \cdot)$	—	Shifted open–loop control on new horizon

Algorithm 4.11 (A posteriori Simple Shortening Strategy 2)
Mixed closed–loop open-loop strategy:

Input:	$\overline{\alpha}$	—	Fixed suboptimality bound
	N	—	Length of the horizon
	$\alpha(N)$	—	Local suboptimality estimate
	$u_N(x(n), \cdot)$	—	Open–loop control
	$V_N(x(n))$	—	Value function
	$V_N(x(n+1))$	—	Predicted value function

(1) While $\alpha(N) > \overline{\alpha}$ do

- Compute N and $\alpha(N)$ by calling Algorithm 4.10

Output:	$\alpha(N)$	—	Local suboptimality estimate
	N	—	New length of the horizon
	$x(n)$	—	New initial value
	$u_N(x(n), \cdot)$	—	Open–loop control on new horizon

Note that in both algorithms we suppose $x_{u_N}(x(n), j)$ to be the outcome of (2.10) with $x_0 = x(n)$. The second strategy opens up the closed–loop for $(\overline{i}+1)$ steps while using the

first strategy the general problem structure remains the same. Hence, the first strategy is in general more robust with respect to perturbations. The second strategy, on the other hand, allows for an increased computing time to solve e.g. the successive optimal control problem if $\overline{i} \geq 1$.

Remark 4.12
If we consider an m–step feedback as in Definition 2.22, these two strategies can be adapted quite easily by shifting the horizon m times in Step (1) of Algorithm 4.10 and adapting the new horizon length to $N - m$.

Note that we have to solve one additional optimal control problem in every step of Algorithm 4.10. This can not be avoided since the initial point $x(n+1)$ is already shifted in time and hence is not identical to the basis of the predicted value function in general.

Remark 4.13
In order to speed up the algorithm, we can reuse the endpiece of the control of the first optimal control problem as initial guess for the second optimal control problem, see also Section 8.3.3 for impact of the initial guess. Additionally, we can use synergy effects in two consecutive receding horizon control steps since in the adaptation algorithm the second optimal control problem of the old step is (apart from deviations of the initial value of the state) identical to the first problem in the actual RHC step.

The prolongation of the optimization horizon is a quite difficult task. One strategy for this is straight forward but computationally expensive. Yet, it can be applied to any optimal control problem satisfying Assumption 4.7.

Algorithm 4.14 (A posteriori Simple Prolongation Strategy)
Suboptimality guaranteeing approach:

Input:	$\overline{\alpha}$	—	Fixed suboptimality bound
	$\alpha(N)$	—	Local suboptimality estimate
	N	—	Length of the horizon

(1) Set $i := 0$

(2) While $\alpha(N) < \overline{\alpha}$ do

(2a) Set $i := i + 1$

(2b) Compute $V_{N+i}(x(n))$ and $V_{N+i}(x(n+1))$

(2c) Compute $\alpha(N)$ by calling Algorithm 3.8

(3) Set $N := N + i$

(4) Implement $u_N(x(n), 0)$

(5) Obtain $x(n+1)$ and set $n := n + 1$

Output:	$\alpha(N)$	—	Local suboptimality estimate
	N	—	New length of the horizon
	$x(n)$	—	New initial value

Lemma 4.15 (Prolongation Strategy)
Consider an optimal control problem (2.10), (2.14) *with initial value $x_0 = x(n)$ and $N \in$*

$\mathbb{N}$. *Moreover,* $\overline{\alpha} \in [0,1)$ *is supposed to be fixed and Assumption 4.7 to be satisfied. Then, Algorithm 4.14 terminates in finite time and computes a horizon length* N *such that the first element of the open–loop control* (2.14) *can be implemented with local suboptimality degree* $\overline{\alpha}$.

Proof. According to Assumption 4.7, there exists a finite horizon length $\overline{N}$ such that $u_N(x(n),0)$, $N \geq \overline{N}$, can be implemented with local suboptimality degree $\overline{\alpha}$. Hence, the while–loop in Algorithm 4.14 will terminate at latest after $i = \overline{N} - N$ steps satisfying the stopping criterion and the assertion follows. □

4.3 Modifications of the ARHC Algorithm

This section aims at reducing the computing time necessary to obtain a horizon length which guarantees a certain degree of suboptimality $\overline{\alpha}$. Similar to Section 3.2, we want to ignore all future information, i.e. to avoid the use of $V_N(x(n+1))$ in our calculations. To this end, we use the *a priori* estimate which we derived in Section 3.2 using Assumption 3.19 and Theorem 3.22. While this estimate is slightly more conservative than the estimate from Proposition 3.3 it is also computationally less demanding if the value N_0 is small. Since no knowledge of $V_N(x(n+1))$ is required, we can reformulate Algorithm 4.8 in the following way:

Algorithm 4.16 (A priori Adaptive Receding Horizon Control)

Input:	$\overline{\alpha}$	—	Fixed suboptimality bound
	N	—	Length of the horizon
	N_0	—	Fixed internal length of subproblems
	$x(n)$	—	Initial value of the plant

(1) Compute $V_N(x(n))$

(2) Compute $\alpha(N)$ by calling Algorithm 3.23

(3) If $\alpha(N) > \overline{\alpha}$:

- Shortening Strategy

(4) Else:

- Prolongation Strategy

(5) Goto (1)

Output:	$\alpha(N)$	—	Local suboptimality estimate
	$x(\cdot)$	—	Closed–loop trajectory

Remark 4.17
Here, we only use the more general result of Theorem 3.22 based on Assumption 3.19 and do not discuss the special case $N_0 = 2$ *of Theorem 3.15 and Assumption 3.11.*

Similar to the changes made in Algorithm 4.8, we can modify Lemma 4.9 to match to the conditions of Assumption 3.19.

Lemma 4.18 (Shortening Strategy)
Consider an optimal control problem (2.10), (2.14) *with initial value* $x_0 = x(n)$ *and* $N, N_0 \in \mathbb{N}$, $N \geq N_0 \geq 2$. *Moreover* $\overline{\alpha} \in [0,1)$ *is supposed to be fixed inducing some* $\overline{\gamma}(\cdot)$ *via* (3.23). *If there exists an integer* $\overline{i} \in \mathbb{N}_0$, $0 < \overline{i} < N - N_0 - 1$ *such that there exist* $\gamma_i < \overline{\gamma}(N-i)$ *satisfying*

$$V_{N_0}(x_{u_N}(N-N_0,x_0)) \leq (\gamma_i+1) \max_{j=2,\ldots,N_0} l(x_{u_N}(N-j,x_0), \mu_{j-1}(x_{u_N}(N-j,x_0))) \quad (4.8)$$

$$V_{k_i}(x_{u_N}(N-k_i,x_0)) \leq (\gamma_i+1) l(x_{u_N}(N-k_i,x_0), \mu_{k_i}(x_{u_N}(N-k_i,x_0))) \quad (4.9)$$

for all $k_i \in \{N_0+1,\ldots,N-i\}$ *and all* $0 \leq i \leq \overline{i}$ *where* $x_{u_N}(i,x_0)$ *is given by* (2.10) *with* $x_0 = x(n)$, *then the first* $(\overline{i}+1)$ *elements of the open–loop control* $u_N(x_0,\cdot)$ *defined by* (2.14) *can be implemented with local suboptimality degree* $\overline{\alpha}$.

Proof. Since (4.8), (4.9) hold for $i = 0$, Theorem 3.22 guarantees that the local suboptimality degree is at least as large as $\overline{\alpha}$. If $\overline{i} > 0$ holds, we can make use of the open–loop trajectory (2.10) and arbitrarily choose $i \in \{0,\ldots,\overline{i}\}$ to compute $x_{u_N}(i,x_0)$. By (4.8), (4.9), we obtain our assertion for the optimal control problem with initial value $x_{u_N}(i,x_0)$ and horizon length $N-i$ via Theorem 3.22. Hence, since i was arbitrary, this holds for all $i \leq \overline{i}$ and concludes the proof. □

Now we use Lemma 4.18 to adapt our shortening strategies:

Algorithm 4.19 (A priori Simple Shortening Strategy)
Closed-loop strategy:

Input:	$\overline{\alpha}$	—	Fixed suboptimality bound
	N	—	Length of the horizon
	$u_N(x(n),\cdot)$	—	Open–loop control
	$V_N(x(n))$	—	Value function

(1) Implement $u_N(x(n),0)$, obtain $x(n+1)$ and set $n := n+1$

(2) Set $u_N(x(n),k) := u_N(x(n-1),k+1)$ for all $k \in \{0,\ldots,N-2\}$

(3) If $N \geq N_0$:

(3a) Set $N := N-1$

(3b) Compute $\alpha(N)$ by calling Algorithm 3.23

(3c) If $\alpha(N) < \overline{\alpha}$:

- Set $N := N+1$
- Set $u_N(x(n),N-1)$ to default value according to (2.9)

Else

- Set $u_N(x(n),k) := u_{N+1}(x(n),k)$ for all $k \in \{0,\ldots,N-1\}$

Output:	$\alpha(N)$	—	Local suboptimality estimate
	N	—	New length of the horizon
	$x(n)$	—	New initial value
	$u_N(x(n),\cdot)$	—	Open–loop control on new horizon

Again, we can implement Lemma 4.18 in a mixed closed–loop open–loop fashion as shown in Algorithm 4.11.

Algorithm 4.20 (A priori Simple Shortening Strategy 2)
Mixed closed–loop open-loop strategy:

Input:	$\overline{\alpha}$	—	Fixed suboptimality bound
	$\alpha(N)$	—	Local suboptimality estimate
	N	—	Length of the horizon
	$u_N(x(n), \cdot)$	—	Open–loop control
	$V_N(x(n))$	—	Value function

(1) While $\alpha(N) > \overline{\alpha}$ and $N \geq N_0$ do

- Compute N and $\alpha(N)$ by calling Algorithm 4.19

Output:	$\alpha(N)$	—	Local suboptimality estimate
	N	—	New length of the horizon
	$x(n)$	—	New initial value
	$u_N(x(n), \cdot)$	—	Open–loop control on new horizon

To prolongate the horizon, we can use an approach which is similar to Algorithm 4.14 but based on the estimates from Theorem 3.22 and Assumption 4.7.

Algorithm 4.21 (A priori Simple Prolongation Strategy)
Suboptimality guaranteeing approach:

Input:	$\overline{\alpha}$	—	Fixed suboptimality bound
	N	—	Length of the horizon
	$\alpha(N)$	—	Local suboptimality estimate

(1) Set $i := 0$

(2) While $\alpha(N) < \overline{\alpha}$

(2a) Set $i := i + 1$

(2b) Compute $V_{N+i}(x(n))$

(2c) Compute $\alpha(N)$ by calling Algorithm 3.23

(3) Set $N := N + i$

(4) Implement $u_N(x(n), 0)$ given by (2.14)

(5) Obtain $x(n+1)$ and set $n := n + 1$

Output:	$\alpha(N)$	—	Local suboptimality estimate
	N	—	New length of the horizon
	$x(n)$	—	New initial value

Note that finite termination and the guaranteed local suboptimality degree $\overline{\alpha}$ can be concluded by Lemma 4.15 directly.

4.3.1 Fixed Point Iteration based Strategy

Since the shortening strategies based on both the *a posteriori* and the *a priori* estimates can be implemented without any additional computational effort, we do not have to refine them. In contrast to that, the prolongation strategies, cf. Algorithms 4.14 and 4.21, may

result in solving a finite but unknown number of optimal control problems. To reduce the number of optimal control problems to be solved to guarantee a local suboptimality degree of at least $\overline{\alpha}$, we analyze equation (3.23) more closely.
If we consider $\overline{\alpha} \in [0,1)$, we obtain a lower bound for N from (3.23) by

$$N \geq N_0 + \frac{2\ln(\gamma(N)) - \ln(1-\overline{\alpha})}{\ln(\gamma(N)+1) - \ln(\gamma(N))}. \tag{4.10}$$

Since $x(n)$, N_0 and $\overline{\alpha}$ are fixed, the right hand side of (4.10) is in fact a function of N. Moreover, we want the horizon length $N \in \mathbb{N}$ to guarantee local suboptimality degree $\overline{\alpha}$ but also to be as small as possible due to the computing time necessary to solve the corresponding optimal control problem. Hence, we seek a horizon length N satisfying

$$N = \Phi(N) := \left\lceil N_0 + \frac{2\ln(\gamma(N)) - \ln(1-\overline{\alpha})}{\ln(\gamma(N)+1) - \ln(\gamma(N))} \right\rceil, \tag{4.11}$$

i.e. a fixed point of the function $\Phi(\cdot)$. During our numerical experiments we experienced that the mapping $\Phi(\cdot)$ overestimates the required adaptation of the horizon length in most cases. In particular, if the actual horizon length N needs to be enlarged, then $\Phi(N)$ is typically larger than the required minimal horizon length to guarantee suboptimality degree $\overline{\alpha}$. Similar, if N can be shortened, then $\Phi(N)$ is too short in general. Yet, we experienced that if the mapping is applied iteratively, then the distance between two iterates is shrinking in many cases, i.e.

$$|\Phi(\Phi(N)) - \Phi(N)| \leq \theta |\Phi(N) - N| \qquad \theta \in [0,1)\ \forall N \geq N_0. \tag{4.12}$$

In the following Theorem 4.22 we show that we can use the mapping $\Phi(\cdot)$ to iteratively compute a sequence $(N_i)_{i\in\mathbb{N}_0}$ which converges to such a fixed point if it satisfies (4.12):

Theorem 4.22. *Consider $N, N_0 \in \mathbb{N}$, $N \geq N_0 \geq 2$, and $\overline{\alpha} \in [0,1)$ to be fixed and $\gamma(N)$ to minimally satisfy Assumption 3.19. If for a given $n \in \mathbb{N}_0$ there exists a constant $\theta \in [0,1)$ such that the function $\Phi(\cdot)$ defined in (4.11) satisfies (4.12) and $\Phi^k(N) \geq N_0$ for all $k \in \mathbb{N}$, then there exists a solution $N^\star \in \mathbb{N}$ with $N^\star = \Phi(N^\star)$ and $\Phi^k(N) \to \mathbb{N}^\star$, $k \to \infty$. If additionally Assumption 4.1 holds and $N^\star$ is used as horizon length for the actual step of the RHC problem, then the resulting solution exhibits local suboptimality degree $\alpha(N^\star) \geq \overline{\alpha}$.*

Proof. Since $\gamma(N)$ satisfies all requirements of Theorem 3.22, we can estimate α via (3.23). In order to guarantee a certain degree of suboptimality $\overline{\alpha}$ we have to show

$$\overline{\alpha} \leq \alpha(N) = \frac{(\gamma+1)^{N-N_0} - \gamma^{N-N_0+2}}{(\gamma+1)^{N-N_0}}.$$

This can be solved for N giving

$$N \geq \left\lceil N_0 + \frac{2\ln(\gamma(N)) - \ln(1-\overline{\alpha})}{\ln(\gamma(N)+1) - \ln(\gamma(N))} \right\rceil =: \Phi(N).$$

Due to (4.12) we have

$$|\Phi^k(N) - \Phi^{k-1}(N)| \leq \theta^{k-1} |\Phi(N) - N|. \tag{4.13}$$

Since $\theta \in [0,1)$ the right hand side of (4.13) tends to zero. Hence, there exists an index $\overline{k} \in \mathbb{N}$ such that $\theta^{\overline{k}-1}|\Phi(N) - N| < 1$. Defining the sequence of optimization horizons via $(N^{(i)})_{i\in\mathbb{N}_0} := (\Phi^i(N))_{i\in\mathbb{N}_0}$ we obtain $N^{(j)} = N^{(k)} \geq N_0$ for all $j,k \geq \overline{k}$. Hence, the sequence $(N^{(i)})_{i\in\mathbb{N}_0}$ is converging and $N^\star = \Phi(N^\star)$ holds for $N^\star = N^{(\overline{k})}$.
Choosing $N = N^\star$, the local suboptimality degree satisfies $\alpha(N) \geq \overline{\alpha}$ by construction of $\Phi(\cdot)$. Hence, a new initial value can be obtained by implementing this controller in a receding horizon fashion. Since this procedure can be applied along the resulting trajectory, i.e. for all $n \in \mathbb{N}$, asymptotic stability of the closed–loop follows by Assumption 4.1 and Theorem 4.4. □

If the assumptions of Theorem 4.22 hold true then an implementation of the fixed point iteration provides a reasonable prolongation strategy. Note that, in general, we cannot a priori check whether $\Phi(\cdot)$ satisfies (4.12). Moreover, an algorithm derived from Theorem 4.22 may show bad performance as shown in Figure 4.2, i.e. the resulting horizon length N may be chosen too large. To this end — and to avoid overshoots — we bound the change in the horizon length from above and below.

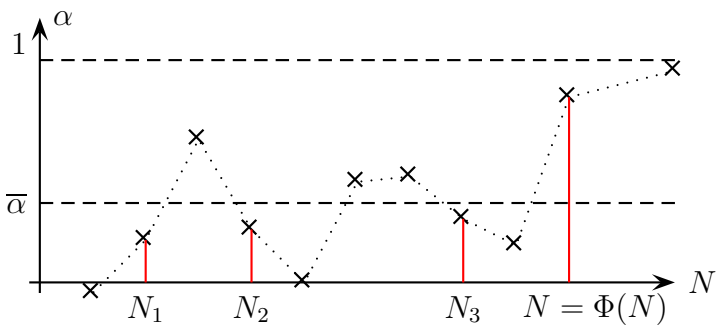


Figure 4.2: Possible outcome of the iteration defined in Theorem 4.22

Due to the integer property of the horizon length, this maximal change is implemented dynamically using an additional variable $\sigma \in \mathbb{N}$, $\sigma > 1$. From our numerical experience, $\sigma = 5$ seems to be a suitable choice, yet, this variable should be chosen depending on the considered problem, see also Section 8.5.

Algorithm 4.23 (A priori Fixed Point Prolongation Strategy)
Suboptimality guaranteeing approach:

Input:	$\overline{\alpha}$	—	Fixed suboptimality bound
	$\gamma(N)$	—	Characteristic of the problem
	$\alpha(N)$	—	Local suboptimality estimate
	N	—	Length of the horizon
	σ	—	Maximal allowable change in horizon length

(1) Set $\delta_{\text{actual}} := \infty$

(2) While $\alpha(N) < \overline{\alpha}$ do

(2a) Compute $\Phi(N)$ according to (4.11)

(2b) Set $\delta_{\text{previous}} := \delta_{\text{actual}}$ and $N_{\text{previous}} := N$

(2c) Set $N := \max\{\min\{N+\sigma, \Phi(N)\}, N-\sigma, N_0\}$ and $\delta_{\text{actual}} := N - N_{\text{previous}}$

(2d) If $\delta_{\text{actual}} > \delta_{\text{previous}}$: Print warning "Iteration may diverge"

(2e) Compute $V_N(x(n))$

(2f) Compute $\gamma(N)$ and $\alpha(N)$ by calling Algorithm 3.23

(3) Implement $u_N(x(n), 0)$ given by (2.14)

(4) Obtain $x(n+1)$ and set $n := n+1$

Output:	$\alpha(N)$	—	Local suboptimality estimate
	N	—	New length of the horizon
	$x(n)$	—	New initial value

Remark 4.24
Within our implementation, the choice of σ is fixed. Yet, numerical simulations indicate that the "best" choice of σ depends on the occuring horizon lengths N_i, i.e. larger horizons allow for larger choices of σ.

Remark 4.25
Theorem 4.22 can also be utilized to shorten the horizon. However, the computation of $N^\star$ requires nonnegligible effort. Hence, this strategy should only be considered when $\alpha(N) < \overline{\alpha}$. Yet, the result of step (2c) of Algorithm 4.23 may be a suitable choice for the optimization horizon in the subsequent optimal control problem.

4.3.2 Monotone Iteration

As indicated by the discussion of Theorem 4.22, no check whether the function $\Phi(\cdot)$ satisfies (4.12) could be derived yet. The aim of this section is to provide a reasonable alternative to the prolongation strategy of Algorithm 4.23. Similar to the previous Section 4.3.1, we want to develop an iteration procedure which guarantees local suboptimality estimate $\alpha(\cdot) \geq \overline{\alpha}$ for a given suboptimality bound $\overline{\alpha}$ after a finite number of iteration steps. Here, we design this procedure by guaranteeing $\alpha(\cdot)$ to be monotonely increasing during the iteration process.
More formally, our aim is to develop an iteration operator $\Psi(\cdot)$ which generates a sequence of optimization horizons $(N^{(i)})$ via

$$N^{(i+1)} := \Psi(N^{(i)})$$

such that the suboptimality estimate

$$\alpha(N^{(i)}) := \frac{(\gamma(N^{(i)})+1)^{N^{(i)}-N_0} - \gamma(N^{(i)})^{N^{(i)}-N_0+2}}{(\gamma(N^{(i)})+1)^{N^{(i)}-N_0}} = 1 - \gamma(N^{(i)})^2 \left(\frac{\gamma(N^{(i)})}{\gamma(N^{(i)})+1}\right)^{N^{(i)}-N_0} \tag{4.14}$$

is monotonely increasing along this sequence where $\gamma(N^{(i)})$ satisfies Assumption 3.19. Moreover, we want to compute a horizon length $N^\star$ such that $\alpha(N^\star) \geq \overline{\alpha}$ holds.
Since we require the given lower bound $\overline{\alpha}$ to be satisfied by an element of this sequence, we want to generate an increase of at least $\overline{\alpha} - \alpha(N^{(i)})$ in $\alpha(\cdot)$ to obtain $N^\star$ during the iteration process.

Lemma 4.26 (Monotonicity of the Suboptimality Estimate)
Suppose $N, N_0 \in \mathbb{N}$, $N \geq N_0 \geq 2$, $\overline{\alpha} \in [0,1)$ and $0 \leq \delta < 1 - \alpha(N)$ are given and Assumption 3.19 holds. Suppose there exists a constant $\vartheta > 0$ such that $\gamma(\tilde{N}) \leq \vartheta\gamma(N)$ holds for

$$\tilde{N} \geq \left\lceil N_0 + \left(\frac{\ln\left(\left(\frac{\gamma(N)}{\gamma(N)+1}\right)^{N-N_0} - \frac{\delta}{\gamma(N)^2}\right) - 2\ln(\vartheta)}{\ln(\vartheta\gamma(N)) - \ln(\vartheta\gamma(N)+1)} \right) \right\rceil, \tag{4.15}$$

then $\alpha(\tilde{N})$ *as defined in* (4.14) *satisfies*

$$\alpha(\tilde{N}) \geq \alpha(N) + \delta \tag{4.16}$$

Proof. In order to show $\alpha(\tilde{N}) \geq \alpha(N) + \delta$ we use (4.14) on both sides of (4.16) which gives us

$$1 - \gamma(\tilde{N})^2 \left(\frac{\gamma(\tilde{N})}{\gamma(\tilde{N})+1}\right)^{\tilde{N}-N_0} \geq 1 - \gamma(N)^2 \left(\frac{\gamma(N)}{\gamma(N)+1}\right)^{N-N_0} + \delta$$

and is equivalent to

$$\gamma(\tilde{N})^2 \left(\frac{\gamma(\tilde{N})}{\gamma(\tilde{N})+1}\right)^{\tilde{N}-N_0} \leq \gamma(N)^2 \left(\frac{\gamma(N)}{\gamma(N)+1}\right)^{N-N_0} - \delta.$$

If we have $\gamma(\tilde{N}) \leq \vartheta\gamma(N)$, then we can overestimate the left hand side and obtain

$$\gamma(\tilde{N})^2 \left(\frac{\gamma(\tilde{N})}{\gamma(\tilde{N})+1}\right)^{\tilde{N}-N_0} \leq \vartheta^2\gamma(N)^2 \left(\frac{\vartheta\gamma(N)}{\vartheta\gamma(N)+1}\right)^{\tilde{N}-N_0}.$$

Hence, it suffices to show

$$\vartheta^2\gamma(N)^2 \left(\frac{\vartheta\gamma(N)}{\vartheta\gamma(N)+1}\right)^{\tilde{N}-N_0} \leq \gamma(N)^2 \left(\frac{\gamma(N)}{\gamma(N)+1}\right)^{N-N_0} - \delta$$

to guarantee (4.16). This inequality is equivalent to

$$\left(\tilde{N} - N_0\right)\left[\ln\left(\frac{\vartheta\gamma(N)}{\vartheta\gamma(N)+1}\right)\right] \leq \ln\left(\left(\frac{\gamma(N)}{\gamma(N)+1}\right)^{N-N_0} - \frac{\delta}{\gamma(N)^2}\right) - 2\ln(\vartheta)$$

since $\vartheta > 0$. By negative definiteness of $\ln\left(\frac{\vartheta\gamma(N)}{\vartheta\gamma(N)+1}\right)$, the latter is equivalent to

$$\tilde{N} \geq N_0 + \left(\frac{\ln\left(\left(\frac{\gamma(N)}{\gamma(N)+1}\right)^{N-N_0} - \frac{\delta}{\gamma(N)^2}\right) - 2\ln(\vartheta)}{\ln\left(\vartheta\gamma(N)\right) - \ln\left(\vartheta\gamma(N)+1\right)}\right).$$

Since the last inequality holds by assumption (4.15), the assertion follows. □

As a result of Lemma 4.26, we obtain the iteration operator

$$\Psi(N) := \left\lceil N_0 + \left(\frac{\ln\left(\left(\frac{\gamma(N)}{\gamma(N)+1}\right)^{N-N_0} - \frac{\delta}{\gamma(N)^2}\right) - 2\ln(\vartheta)}{\ln\left(\vartheta\gamma(N)\right) - \ln\left(\vartheta\gamma(N)+1\right)}\right)\right\rceil \tag{4.17}$$

Remark 4.27
In the special case $\delta = 0$*, we can simplify the expression* (4.17) *to*

$$\Psi(N) := \left\lceil N_0 + \frac{(N-N_0)\left(\ln\left(\gamma(N)+1\right) - \ln\left(\gamma(N)\right)\right) + 2\ln\left(\vartheta\right)}{\ln\left(\vartheta\gamma(N)+1\right) - \ln\left(\vartheta\gamma(N)\right)}\right\rceil. \tag{4.18}$$

Since $\frac{\ln(x+1)-\ln(x)}{\ln(\vartheta(x+1))-\ln(\vartheta(x))} > 1$ *holds, we obtain* $\Psi(N) > N$ *if we assume* $\vartheta \geq 1$*. Moreover, increasing* δ *results in an enlarged value* $\Psi(\cdot)$ *and hence we obtain* $\Psi(N) > N$ *for* (4.17) *as well if* $\vartheta \geq 1$ *holds.*

Remark 4.28
In Lemma 4.26, we only assume $\vartheta > 0$. Hence, we can see from (4.17) that a decrease in the horizon length is possible during the iteration. Within our implementation, however, we exclude this possibility by allowing for $\vartheta \geq 1$ only.
Obtaining a suitable approximation of ϑ is probably the most tricky part of this prolongation method. One possibility to overcome this issue is to solve the problem $SDOCP_N$ for all initial values $x \in \mathbb{X}$ and all $N \geq N_0$ and compute the corresponding values $\gamma(\cdot)$. Such a method, however, is computationally very demanding. For our implementation as shown in Algorithm 4.29 below, we iteratively update the value of ϑ by setting $\vartheta := \max\left\{\vartheta, \frac{\gamma(N^{(i+1)})}{\gamma(N^{(i)})}\right\}$ separately for each step in the receding horizon control procedure, cf. Steps (2f) and (1) in Algorithm 4.29 respectively. This method is not only computationally cheap and gives us a lower bound for ϑ, it also moderates a possible overshoot of the iteration operator $\Psi(\cdot)$ defined in (4.17).
The used lower approximation, however, also leads to a conservativeness of the algorithm since the fundamental idea of this strategy is based on a priori knowledge of this value. Yet, from our numerical experience this approximation works well and the values of ϑ stay fairly small, i.e. upon termination of one call of Algorithm 4.29 we usually have $\vartheta \in [2, 3]$. Still, this small change may obstruct the prolongation.

Considering Assumptions 3.19 and 4.7, we construct the following algorithm from Lemma 4.26. Similar to Algorithm 4.23 we add a security parameter $\sigma \in \mathbb{N}$, $\sigma > 1$, to avoid overshoots which may be caused by the conservativeness of the suboptimality estimates, see also Figure 4.2.

Algorithm 4.29 (A priori Monotone Prolongation Strategy)
Suboptimality guaranteeing approach:

Input:	$\overline{\alpha}$	—	Fixed suboptimality bound
	N	—	Length of the horizon
	$\alpha(N)$	—	Local suboptimality estimate
	$\gamma(N)$	—	Characteristic of the problem
	σ	—	Maximal allowable increase in horizon length

(1) Set $\vartheta := 1$

(2) While $\alpha(N) < \overline{\alpha}$ do

(2a) If $\alpha(N) \geq \overline{\alpha} - \delta$ holds true
Set $\delta := \overline{\alpha} - \alpha(N)$

(2b) Compute $\Psi(N)$ according to (4.17)

(2c) Set $N_{\text{previous}} := N$ and $N := \min\{N + \sigma, \Psi(N)\}$

(2d) Compute $V_N(x(n))$

(2e) Compute $\gamma(N)$ and $\alpha(N)$ by calling Algorithm 3.23

(2f) Set $\vartheta := \max\left\{\vartheta, \frac{\gamma(N)}{\gamma(N_{\text{previous}})}\right\}$

(3) Implement $u_N(x(n), 0)$ given by (2.14)

(4) Obtain $x(n+1)$ and set $n := n + 1$

Output:	$\alpha(N)$	—	Local Suboptimality Estimate
	N	—	New length of the horizon
	$x(n)$	—	New initial value

Last, we show that Algorithm 4.29 terminates in finite time revealing a horizon length N which guarantees the local suboptimality bound $\overline{\alpha}$.

Theorem 4.30 (Monotone Iteration)
Suppose Assumptions 3.19 and 4.7 hold. Then Algorithm 4.29 computes a horizon length N which guarantees local suboptimality degree $\alpha(N) \geq \overline{\alpha}$.

Proof. Due to the stopping criterion of the while–loop, we always have $\delta > 0$. Hence, by Remark 4.27, we can conclude that the horizon length N is increasing in every step of the while–loop due to $\vartheta \geq 1$. Since Assumption 4.7 guarantees the existence of a finite horizon length $\overline{N} \in \mathbb{N}$, $\overline{N} < \infty$, such that $\alpha(N) \geq \overline{\alpha}$ holds for all $N \geq \overline{N}$, Algorithm 4.29 terminates in finite time. Last, $\alpha(N) \geq \overline{\alpha}$ is guaranteed by the stopping criterion of the while–loop. □

Note that we do not assume $\gamma(\cdot)$ in (4.10) to be computed in a specific way but only to satisfy Assumption 3.19. Hence, our iteration procedure also fulfills our aim of deriving a horizon length N such that $\alpha(N) \geq \overline{\alpha}$ holds if it is based on any characteristic $\tilde{\gamma}(\cdot) \geq \gamma(\cdot)$. In particular, this allows us to use our a priori practical estimates from Theorems 3.34 and 3.39.

4.3.3 Integrating the a posteriori Estimate

The prolongation strategies presented in Algorithms 4.23 and 4.29 are based on the characteristic $\gamma(\cdot)$. Hence, they are not readily applicable if we want to base them on the *a posteriori* estimate of Proposition 3.3 since the value of $\gamma(\cdot)$ is unknown from the utilized Lyapunov inequality (3.8).
Yet, given a fixed horizon length N, we know the connecting formula (3.23) relating the known suboptimality estimate $\alpha(N)$ and the desired characteristic $\gamma(N)$, that is

$$\alpha(N) = \frac{(\gamma(N)+1)^{N-N_0} - \gamma(N)^{N-N_0+2}}{(\gamma(N)+1)^{N-N_0}} = 1 - \frac{\gamma(N)^{N-N_0+2}}{(\gamma(N)+1)^{N-N_0}}. \tag{4.19}$$

Now, our first aim is to show that this relation is a bijective mapping:

Lemma 4.31 (Bijectivity)
Given $N, N_0 \in \mathbb{N}$, $N, \mathbb{N}_0 \geq 2$, the mapping $\Gamma : [0,\infty) \to (-\infty, 1]$ defined by

$$\Gamma(x) := 1 - \frac{x^{N-N_0+2}}{(x+1)^{N-N_0}} \tag{4.20}$$

is bijective.

Proof. In order to show bijectivity, we use continuity of $\Gamma(\cdot)$ on $[0,\infty)$ and $\Gamma(0) = 1$ and $\lim_{x\to\infty} \Gamma(x) = -\infty$ to show surjectivity. In order to obtain injectivity, we show that $\Gamma(\cdot)$ is strictly monotone on $[0,\infty)$, i.e. for all $x \in [0,\infty)$ and $\varepsilon > 0$ we have $\Gamma(x) > \Gamma(x+\varepsilon)$. Using (4.20) in this last inequality, we have to show

$$1 - \frac{x^{N-N_0+2}}{(x+1)^{N-N_0}} > 1 - \frac{(x+\varepsilon)^{N-N_0+2}}{(x+\varepsilon+1)^{N-N_0}}$$

to hold true. This is equivalent to

$$\frac{(x+\varepsilon+1)^{N-N_0}}{(x+1)^{N-N_0}} < \frac{(x+\varepsilon)^{N-N_0+2}}{x^{N-N_0+2}}$$

and reformulating this expression gives us

$$\left(1+\frac{\varepsilon}{x+1}\right)^{N-N_0} < \left(1+\frac{\varepsilon}{x}\right)^{N-N_0+2}$$

which holds true for $x > 0$ since $N - N_0 \geq 0$. Last, considering $x = 0$, we see that $\Gamma(\varepsilon) < 1$ for all $\varepsilon > 0$ and hence the assertion follows. □

As a conclusion from Lemma 4.31, we can invert the mapping $\Gamma(\cdot)$. Back in our suboptimality estimate context, Lemma 4.31 shows that if a computed suboptimality degree $\alpha(N)$ is available, then there exists a unique characteristic $\gamma(N)$. Since the function $\Gamma(\cdot)$ is twice continuously differentiable on $[0,\infty)$, the nonlinear equation (4.19) can be solved using Newton's method, see e.g. [51]. The resulting iteration is given by

$$\gamma^{(k+1)}(N) := \gamma^{(k)}(N) + \frac{1-\alpha-\frac{\gamma^{(k)}(N)^{N-N_0+2}}{(\gamma^{(k)}(N)+1)^{N-N_0}}}{\left(\frac{\gamma^{(k)}(N)}{\gamma^{(k)}(N)+1}\right)^{N-N_0+1}(N-N_0+2+2\gamma^{(k)}(N))} \tag{4.21}$$

Remark 4.32
Here, we do not follow the approach to evaluate $\gamma(N)$ via Algorithm 3.23 since the additional computational effort of using Newton's method is low compared to solving an optimal control problem required to check Assumption 3.19. Moreover, computing $\gamma(N)$ using Algorithm 3.23 gives us a more conservative estimate which renders this approach impractical.

Remark 4.33
Note that in the context of the a posteriori estimate, we are free to choose $N_0 \in \{2,\dots,N\}$. However, it is not clear which value should be chosen. On the one hand, we might want to approximate the smallest possible characteristic $\gamma(N) > 0$ by setting $N_0 := 2$ within the Newton iteration (4.21)*. On the other hand, formulas* (4.11) *and* (4.17) *which are used to compute the new horizon length logarithmically depend on $\gamma(N)$. Here, we face the problem that the logarithmic part of nominator of* (4.17) *which depends on $\gamma(N)$ also depends on $-\delta/\gamma(N)^2 < 0$ which generates a vertical asymptote for $N := \Psi(\gamma(N))$. Hence, small values of $\gamma(N)$ might lead to drastically overestimate the minimal required horizon length which guarantees the local suboptimality bound $\overline{\alpha}$. Within our implementation we always consider $N_0 = 2$ and handle the overshoot by setting σ appropriately. Note that using formula* (4.11) *we do not face this problem since the logarithmic parts depend on $\gamma(N)$ only.*

Upon initialization of the Newton method (4.21), we can define $\gamma^{(0)}(N) := 1$ or, if we use it during the run of the receding horizon control algorithm, we can also set $\gamma^{(0)}(N)$ of the current iteration to the computed approximation of $\gamma(N)$ for the last step of the receding horizon controller.

Remark 4.34
Since we do not expect the characteristic $\gamma(N)$ to vary massively along the closed–loop, the reuse of information from previous steps of the receding horizon control algorithm is convenient. For numerical results concerning the values of $\gamma(\cdot)$, we refer to Section 8.4.

In order to use the a posteriori estimate of Proposition 3.3 within Algorithms 4.23 and 4.29, we have to additionally compute $V_N(x(n+1))$, estimate $\alpha(N)$ according to Algorithm 3.8 and approximate $\gamma(N)$ via the Newton iteration (4.21). This gives us the following:

Algorithm 4.35 (A posteriori Fixed Point Prolongation Strategy)
Suboptimality guaranteeing approach:

Input:	$\overline{\alpha}$	—	Fixed suboptimality bound
	$\gamma(N)$	—	Characteristic of the problem
	$\alpha(N)$	—	Local suboptimality estimate
	N	—	Length of the horizon
	σ	—	Maximal allowable change in horizon length

(1) Set $\delta_{\text{actual}} := \infty$

(2) While $\alpha(N) < \overline{\alpha}$ do

(2a) Compute $\gamma(N)$ via the Newton iteration defined in (4.21)

(2b) Compute $\Phi(N)$ according to (4.11)

(2c) Set $\delta_{\text{previous}} := \delta_{\text{actual}}$ and $N_{\text{previous}} := N$

(2d) Set $N := \max\{\min\{N+\sigma, \Phi(N)\}, N-\sigma, N_0\}$ and $\delta_{\text{actual}} := N - N_{\text{previous}}$

(2e) If $\delta_{\text{actual}} > \delta_{\text{previous}}$: Print warning "Iteration may diverge"

(2f) Compute $V_N(x(n))$ and $V_N(x(n+1))$

(2g) Compute $\alpha(N)$ by calling Algorithm 3.8

(3) Implement $u_N(x(n), 0)$ given by (2.14)

(4) Obtain $x(n+1)$ and set $n := n+1$

Output:	$\alpha(N)$	—	Local suboptimality estimate
	N	—	New length of the horizon
	$x(n)$	—	New initial value

Algorithm 4.36 (A posteriori Monotone Prolongation Strategy)
Suboptimality guaranteeing approach:

Input:	$\overline{\alpha}$	—	Fixed suboptimality bound
	N	—	Length of the horizon
	$\alpha(N)$	—	Local suboptimality estimate
	$\gamma(N)$	—	Characteristic of the problem
	σ	—	Maximal allowable increase in horizon length

(1) Set $\vartheta := 1$

(2) While $\alpha(N) < \overline{\alpha}$ do

(2a) Compute $\gamma(N)$ via the Newton iteration defined in (4.21)

(2b) If $\alpha(N) \geq \overline{\alpha} - \delta$ holds true

Set $\delta := \overline{\alpha} - \alpha(N)$

(2c) Compute $\Psi(N)$ according to (4.17)

(2d) Set $N_{\text{previous}} := N$ and $N := \min\{N+\sigma, \Psi(N)\}$

(2e) Compute $V_N(x(n))$ and $V_N(x(n+1))$

(2f) Compute $\alpha(N)$ by calling Algorithm 3.8

(2g) Set $\vartheta := \max\left\{\vartheta, \frac{\gamma(N)}{\gamma(N_{\text{previous}})}\right\}$

(3) Implement $u_N(x(n), 0)$ given by (2.14)

(4) Obtain $x(n+1)$ and set $n := n+1$

Output: $\alpha(N)$ — Local Suboptimality Estimate
N — New length of the horizon
$x(n)$ — New initial value

For a comparison of the presented methods we refer to Section 8.5. Yet, we may expect these methods to show poor performance in the context of sampled–data systems, see also the discussion at the beginning of Section 3.3. To cover practical stability we next incorporate our practical suboptimality estimates from Proposition 3.28 and Theorem 3.39 into our adaptation algorithms.

4.4 Extension towards Practical Suboptimality

In this section, we extend Algorithms 4.8 and 4.16 to the practical stability case described in Section 3.3.
Comparing Propositions 3.3 and 3.28, we see that Algorithm 4.8 can be adapted by replacing the computation of $\alpha(N)$ via Algorithm 3.8 by a call of Algorithm 3.30. In particular, this requires us to define a tolerance level ε in (3.27) for the deviation of the stage cost characterizing the violation area of the relaxed Lyapunov inequality.

Algorithm 4.37 (Adaptive practical horizon length control)
Input: $\overline{\alpha}$ — Fixed suboptimality bound
ε — Tolerance level
N — Length of the horizon
$x(n)$ — Initial value of the plant

(1) Compute $V_N(x(n))$, $V_N(x(n+1))$

(2) Compute $\alpha(N)$ by calling Algorithm 3.30

(3) If $\alpha(N) > \overline{\alpha}$ or $l(x(n), \mu_N(x(n))) < \varepsilon$:

- Shortening Strategy

(4) Else:

- Prolongation Strategy

(5) Goto (1)

Output: $\alpha(N)$ — Local suboptimality estimate
$x(\cdot)$ — Closed–loop trajectory

Moreover, we can still use the same ideas for shortening and prolongation strategies as in Section 4.2. From the discussion of Proposition 3.28 we know that on subintervals I the trajectory behaves "almost" like an infinite horizon optimal one. This allows us to obtain extensions of Lemma 4.9 and 4.15 for the practical case:

Lemma 4.38
Consider an optimal control problem (2.10), (2.14) *with initial value* $x_0 = x(n)$ *and* $N \in \mathbb{N}$. *Moreover,* $\overline{\alpha} \in [0,1)$ *is fixed. Consider* $\varepsilon > 0$ *to be chosen such that there exists an index* $\overline{i} \in \mathbb{N}_0$, $0 \leq \overline{i} < N$ *such that*

$$\begin{aligned} \min \{\overline{\alpha}\,(l(x_{u_N}(i,x_0), \mu_{N-i}(x_{u_N}(i,x_0))) - \varepsilon)\,, l(x_{u_N}(i,x_0), \mu_{N-i}(x_{u_N}(i,x_0))) - \varepsilon\} \leq \\ \leq V_{N-i}(x_{u_N}(i,x_0)) - V_{N-i}(x_{u_N}(i+1,x_0)) \end{aligned} \tag{4.22}$$

holds true for all $0 \leq i \leq \overline{i}$ *where* $x_{u_N}(i,x_0)$ *is defined by* (2.10). *Then the first* $(\overline{i}+1)$ *elements of* (2.14) *can be implemented with local practical suboptimality degree* $\overline{\alpha}$.

Lemma 4.39
Consider an optimal control problem (2.10), (2.14) *with initial value* $x_0 = x(n)$ *and* $N \in \mathbb{N}$. *Moreover,* $\overline{\alpha} \in [0,1)$ *and* $\varepsilon > 0$ *are supposed to be fixed and Assumption 4.7 to be satisfied. Then, Algorithm 4.14 with Step (2c) replaced by*

(2c') Compute $\alpha(N)$ *by calling Algorithm 3.30*

terminates in finite time and computes a horizon length N *such that the first element of the open–loop control* (2.14) *can be implemented with local practical suboptimality degree* $\overline{\alpha}$.

Proof of Lemma 4.38 and 4.39: If $l(\cdot,\cdot) > \varepsilon$, the result follows directly from Lemma 4.9. In the other case, there is nothing to show since $\alpha(N-i)$ can be chosen independently of $V_{N-i}(\cdot)$ and $l(\cdot,\cdot)$. □

Hence, the modification of Algorithms 4.10, 4.11 and 4.14 yield suitable shortening and prolongation strategies for the practical case if we replace the calls of Algorithm 3.8 by calls of Algorithm 3.30.

The extension of the modified ARHC algorithm 4.16 which is based on the a priori suboptimality estimate from Theorem 3.22 to the practical case can be obtained similarly. Here, we substitute Algorithm 3.23 by Algorithm 3.40.

Algorithm 4.40 (Adaptive practical horizon length control)

Input:	$\overline{\alpha}$	—	Fixed suboptimality bound
	ε	—	Tolerance level
	N	—	Length of the horizon
	N_0	—	Fixed internal length of subproblems
	$x(n)$	—	Initial value of the plant

(1) Compute $V_N(x(n))$

(2) Compute $\alpha(N)$ by calling Algorithm 3.40

(3) If $\alpha(N) > \overline{\alpha}$ or $l(x(n), \mu_N(x(n))) < \varepsilon$:

- Shortening Strategy

(4) Else:

- Prolongation Strategy

(5) Goto (1)

Output: $\alpha(N)$ — Local suboptimality estimate
$x(\cdot)$ — Closed–loop trajectory

In order to shorten or prolongate the optimization horizon, we use the following results:

Lemma 4.41
Consider an optimal control problem (2.10), (2.14) *with initial value* $x_0 = x(n)$ *and* $N, N_0 \in \mathbb{N}$, $N \geq N_0 \geq 2$. *Moreover,* $\overline{\alpha} \in [0,1)$ *is supposed to be fixed inducing* $\overline{\gamma}(\cdot)$ *via* (3.23). *If there exists an* $\overline{i} \in \mathbb{N}_0$, $0 < \overline{i} < N - N_0 - 1$ *such that there exist* $\gamma_i < \overline{\gamma}(N-i)$ *satisfying*

$$V_{N_0}(x_{u_N}(N-N_0,x_0)) \leq \max_{j=2,\dots,N_0} \Big\{ l(x_{u_N}(N-j,x_0),\mu_{j-1}(x_{u_N}(N-j,x_0))) + \varepsilon, \\ (\gamma_i+1) l(x_{u_N}(N-j,x_0),\mu_{j-1}(x_{u_N}(N-j,x_0))) + (\gamma_i + 1 - N_0\gamma)\varepsilon \Big\} \tag{4.23}$$

$$V_{k_i}(x_{u_N}(N-k_i,x_0)) \leq \max\{ l(x_{u_N}(N-k_i,x_0),\mu_{k_i}(x_{u_N}(N-k_i,x_0))) + (k_i-1)\varepsilon, \\ (\gamma_i+1) l(x_{u_N}(N-k_i,x_0),\mu_k(x_{u_N}(N-k_i,x_0))) + (k_i - \gamma_i - 1)\varepsilon \} \tag{4.24}$$

for all $k_i \in \{N_0+1,\dots,N-i\}$ *and all* $0 \leq i \leq \overline{i}$ *where* $x_{u_N}(i,x_0)$ *is given by* (2.10) *with* $x_0 = x(n)$, *then the first* $(\overline{i}+1)$ *elements of* (2.14) *can be implemented with local practical suboptimality degree* $\overline{\alpha}$.

Lemma 4.42
Consider an optimal control problem (2.10), (2.14) *with initial value* $x_0 = x(n)$ *and* $N \in \mathbb{N}$. *Moreover,* $\overline{\alpha} \in [0,1)$ *and* $\varepsilon > 0$ *are supposed to be fixed and Assumption 4.7 to be satisfied. Then, Algorithm 4.21 with Step (2c) replaced by*

(2c') Compute $\alpha(N)$ *by calling Algorithm 3.40*

terminates in finite time and computes a horizon length N *such that the first element of the open–loop control* (2.14) *can be implemented with local practical suboptimality degree* $\overline{\alpha}$.

Proof of Lemma 4.41 and 4.42. Similar to the proof of Lemma 4.38 and 4.42, we distinguish the cases $l(\cdot,\cdot) > \varepsilon$ and $l(\cdot,\cdot) \leq \varepsilon$. For the latter one, γ_i can be chosen freely, hence the corresponding suboptimality estimate $\alpha(N)$ is equal to one. In the first case, the result follows directly from Lemma 4.18 which concludes the proof. □

Hence, the algorithmical extension to the practical case using the a priori estimate from Theorem 3.22 reduces to replacing the calls of Algorithm 3.23 by calls of Algorithm 3.40 in Algorithms 4.19, 4.20 and 4.21.
The more sophisticated prolongation strategy presented in Theorem 4.22 relies on inequality (3.26). In particular, the horizon length is obtained by utilizing inequalities (3.19) and (3.20) for $\gamma(\cdot)$ within the iteration procedure. Since in Assumption 3.37 these inequalities are extended to the practical case leaving the relation (3.26) untouched, we can replace Step (2f) in Algorithm 4.23 by

(2f') Compute $\gamma(N)$ and $\alpha(N)$ by calling Algorithm 3.40

Then the following holds:

Theorem 4.43. *Consider $N, N_0 \in \mathbb{N}$, $N \geq N_0 \geq 2$, and $\overline{\alpha} \in [0,1)$ to be fixed and $\gamma(N)$ to minimally satisfy Assumption 3.37. If for a given $n \in \mathbb{N}_0$ there exists a constant $\theta \in [0,1)$ such that the function $\Phi(\cdot)$ defined in (4.11) satisfies (4.12) and $\Phi^k(N) \geq N_0$ for all $k \in \mathbb{N}$, then there exists a solution $N^\star \in \mathbb{N}$ with $N^\star = \Phi(N^\star)$ and $\Phi^k(N) \to \mathbb{N}^\star$, $k \to \infty$. If additionally Assumption 4.1 holds and $N^\star$ is used as horizon length for the actual step of the RHC problem, then the resulting solution exhibits local practical suboptimality degree $\alpha(N^\star) \geq \overline{\alpha}$.*

Proof. Similar to the proof of Theorem 4.22, we obtain $\Phi(\cdot)$ from the definition of $\alpha(\cdot)$ in Theorem 3.22 and the requirement

$$\overline{\alpha} \leq \alpha(N) = \frac{(\gamma+1)^{N-N_0} - \gamma^{N-N_0+2}}{(\gamma+1)^{N-N_0}}.$$

Moreover, we obtain existence of the fixed point $N^\star = \Phi(N^\star)$ by arguments of the proof of Theorem 4.22. Choosing $N = N^\star$, the local practical suboptimality degree satisfies $\alpha(N) \geq \overline{\alpha}$ by construction of $\Phi(\cdot)$. Applying this procedure along the resulting closed–loop, practical asymptotic stability follows by Assumption 4.1 and Theorem 4.6. □

Considering the strategies of Theorem 4.30 in the practical case, there is nothing to show since this strategy is based on an arbitrary estimate $\gamma(\cdot)$ satisfying Assumption 3.37. Hence, we may use Algorithm 3.40 instead of Algorithm 3.23 within the implementation of Theorem 4.30 in Algorithm 4.29.

Concluding, we have obtained algorithms which reasonably compute a horizon length N such that the suboptimality bound $\overline{\alpha}$ holds for the given initial value $x(n)$. Moreover, checkable sufficient conditions have been derived which are computable online. Still, minimality of the resulting horizon length N cannot be guaranteed. Within this thesis, however, we will not perform any further investigation of this issue.

Chapter 5

Numerical Algorithms

In the previous Chapters 2 – 4, we stated the receding horizon controller, analyzed its stability and suboptimality properties and derived adaptation methods for this type of controller. In all parts, we assumed that the minimizing open–loop control (2.14) is at hand. The aim of this chapter is to link the theoretical part of the receding horizon controller from Chapters 2 – 4 to our implementation and the obtained results presented in Chapters 6 and 8 respectively.
To this end, we now define the mathematical terms which we require in the implementation of the receding horizon controller and outline several algorithms. We first state a discretization technique in Section 5.1 which is not only suitable to convert a possibly continuous–time optimal control problem into a optimization problem in standard form, but also fits the needs of a fast receding horizon controller. The fundamental idea of the discretization technique, however, is closely related to the optimization methods which we use in our implementation. To show these connections, we present required basics of nonlinear optimization theory in Section 5.2. In both sections, we establish links to theory and practice of the receding horizon controller to illustrate interactions of both parts.

5.1 Discretization Technique for RHC

In Section 2.4, we stated the discrete–time receding horizon control problem RHC_N to be of Lagrangian form which suited our analytical purposes in Chapters 3 and 4. Since within the literature also Bolza type cost functionals are considered, cf. Section 3.4 and references therein, our implementation also covers this aspect. In particular, we consider the generalized problem

$$
\begin{aligned}
\text{Find } \mu_N(x(k)) &:= u_{[0]} \\
\text{ST. } u_{[0,N-1]} &= \operatorname*{argmin}_{u_N \in \mathcal{U}_N} J_N(x(k), u_N) \\
J_N(x(k), u_N) &= \sum_{i=0}^{N-1} l\left(x_{u_N}(i, x(k)), u_N(x(k), i)\right) \\
&\quad + F(x_{u_N}(N, x(k))) \\
x_{u_N}(i+1, x(k)) &= f\left(x_{u_N}(i, x(k)), u_N(x(k), i)\right) \quad \forall i \in \mathcal{I}_u \\
x_{u_N}(0, x(k)) &= x(k) \\
x_{u_N}(i, x(k)) &\in \mathbb{X} \qquad \forall i \in \mathcal{I}_x \\
u_N(x_{u_N}(0, x(k)), i) &\in \mathbb{U} \qquad \forall i \in \mathcal{I}_u
\end{aligned}
\tag{$\mathrm{RHC}_{\mathrm{N}}^{\mathrm{B}}$}
$$

Note that all our examples stated in Chapter 7 are continuous–time models and therefore do not meet the discrete–time form of problem $\mathrm{RHC}_{\mathrm{N}}^{\mathrm{B}}$. The aim of this section is to state and motivate the discretization technique used within our implementation *PCC2*[1] of a receding horizon controller. Moreover, we show how to transform every problem in the sequence $\mathrm{RHC}_{\mathrm{N}}^{\mathrm{B}}$ to meet the standard form of nonlinear optimization

$$\boxed{\begin{array}{rll} \text{Minimize} & F(x) & \text{over all } x \in \mathbb{R}^n \\ \text{ST.} & G_i(x) = 0, & i \in \mathcal{E} \\ & H_i(x) \geq 0, & i \in \mathcal{I} \end{array}} \qquad \text{(NLP)}$$

which is required by the minimization procedures used within our implementation. The theoretical issues of nonlinear optimization are treated in the following Section 5.2 whereas realizations are addressed in Sections 6.2.4 and 6.3 considering discretization and optimization respectively.

Within the receding horizon control setting, we make use of the predefined sampling grid given by the sampled–data system, see Definition 1.23.

Definition 5.1 (Sampling Grid)
Consider a problem RHC_N. Given a sampling instant k in discrete–time and corresponding instant t_k in continuous–time, we denote the *sampling grid* of all sampling instances of the problem SDOCP_N with initial time t_k by

$$\mathbb{T}_k := \{t_k^0, t_k^1, \ldots, t_k^N\} = \{t_k, t_k + T, \ldots, t_k + NT\} \tag{5.1}$$

with $t_k^0 := t_k$ where t_k is identified with the k–th closed–loop sampling instant.

Similar to Section 2.4, we define the sets $\mathcal{I} = \{0, \ldots, N\}$ and $\mathcal{I}_u = \{0, \ldots, N-1\}$ to be the superindex sets of the open–loop sampling instants t_k^i from Definition 5.1. Thus, we can define

$$x_{u_N}(i+1, x(k)) := \Phi\left(x_{u_N}(i, x(k)), u_N(x_{u_N}(x(k), i)\right) \tag{5.2}$$

for all $i \in \mathcal{I}_u$ where $\Phi : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is an operator representing a solution method for the underlying differential equation (1.3).
The *approximation issue*, that is the mismatch between the exact discrete–time system and the numerical approximation, and its effect on stability of the resulting closed–loop are treated in Section 1.3.3. This allows us to apply the computed control to the considered plant. In particular, if we can guarantee *stability* and *suboptimality* as described in Chapter 3, then the plant inherits these properties if the requirements on the approximation are satisfied. For details on the solvers which we use to implement the operator $\Phi(\cdot, \cdot)$ in (5.2), we refer to Section 6.4 and references therein.
For the rest of this chapter, we assume the differential equation to be solved exactly.

[1] Webpage: `http://http://www.nonlinearmpc.com`

5.1.1 Full Discretization

Now our aim is to obtain an optimization problem in standard form NLP. The dynamic (5.2) itself does not meet the standard form for constraints. However, it can be rewritten as

$$x_{u_N}(i+1, x(k)) - \Phi\left(x_{u_N}(i, x(k)), u_N(x_{u_N}(x(k), i))\right) = 0 \qquad \forall i \in \mathcal{I}_u \tag{5.3}$$

$$x_{u_N}(0, x(k)) - x(k) = 0 \tag{5.4}$$

and we obtain a set of equality constraints. Moreover, equation (5.2) allows us to evaluate the constraints

$$\begin{aligned} x_{u_N}(i, x(k)) &\in \mathbb{X} \qquad \forall i \in \mathcal{I}_x \\ u_N(x(k), i) &\in \mathbb{U} \qquad \forall i \in \mathcal{I}_u. \end{aligned}$$

Here, we assume $\mathbb{X}$ and $\mathbb{U}$ to be characterized by a set of functions $g_i : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, $i \in \mathcal{E}_g = \{0, \ldots, r_g\}$, and $h_i : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, $i \in \mathcal{I}_h = \{0, \ldots, r_h\}$ such that

$$\begin{aligned} g_i(x, u) &= 0 \quad \forall i \in \mathcal{E}_g \\ h_i(x, u) &\geq 0 \quad \forall i \in \mathcal{I}_h \end{aligned}$$

holds for all feasible points. Evaluating these function at each sampling point, we obtain the sets of equality and inequality constraints $\overline{G}(\cdot)$ and $\overline{H}(\cdot)$,

$$\overline{G}_{i\cdot(N+1)+j}(x) := g_i\left(x_{u_N}(j, x(k)), u_N(x(k), j)\right) \quad \forall i \in \mathcal{E}_g, j \in \mathcal{I}_x \tag{5.5}$$

$$\overline{H}_{i\cdot(N+1)+j}(x) := h_i\left(x_{u_N}(j, x(k)), u_N(x(k), j)\right) \quad \forall i \in \mathcal{I}_h, j \in \mathcal{I}_x \tag{5.6}$$

with $u_N(x(k), N) := 0 \in \mathbb{R}^m$ and optimization variable

$$x := \left(x_{u_N}(0, x(k))^\top, \ldots, x_{u_N}(N, x(k))^\top, u_N(x(k), 0)^\top, \ldots, u_N(x(k), N-1)^\top\right)^\top \tag{5.7}$$

The constraint functions $\overline{G}(\cdot)$ and $\overline{H}(\cdot)$ are already given in standard form. Now, we can combine the equality constraints defined in (5.3), (5.4) and (5.5) to obtain

$$G(x) = \begin{pmatrix} \left[g_i\left(x_{u_N}(j, x(k)), u_N(x(k), j)\right)\right]_{i \in \mathcal{E}_g, j \in \mathcal{I}_x} \\ \left[x_{u_N}(i+1, x(k)) - \Phi\left(x_{u_N}(i, x(k)), u_N(x_{u_N}(x(k), i))\right)\right]_{i \in \mathcal{I}_u} \\ x_{u_N}(0, x(k)) - x(k) \end{pmatrix}$$

$$H(x) = \begin{pmatrix} \left[h_i\left(x_{u_N}(j, x(k)), u_N(x(k), j)\right)\right]_{i \in \mathcal{I}_h, j \in \mathcal{I}_x} \end{pmatrix}$$

and the corresponding sets $\mathcal{E}$ and $\mathcal{I}$, i.e.

$$\mathcal{E} = \{0, \ldots, (r_g + n) \cdot (N+1)\} \qquad \text{and} \qquad \mathcal{I} = \{0, \ldots, r_h \cdot (N+1)\}.$$

Using the optimization variable (5.7) allows us to rewrite the cost function

$$F(x) := J_N(x(k), u_N). \tag{5.8}$$

Hence, we can combine the cost function (5.8) and the constraints (5.3), (5.4), (5.5), (5.6) to obtain a discretized optimal control problem in standard form NLP.

Definition 5.2 (Fully discretized Optimal Control Problem)
Consider a single optimal control problem of the sequence $\mathrm{RHC}_{\mathrm{N}}^{\mathrm{B}}$. The corresponding nonlinear optimization problem in standard form NLP given by

$$\boxed{\begin{array}{lll} \text{Minimize} & F(x) & \text{over all } x \in \mathbb{R}^{n\cdot(N+1)+m\cdot N} \\ \text{ST.} & G_i(x) = 0, & i \in \mathcal{E} \\ & H_i(x) \geq 0, & i \in \mathcal{I} \end{array}}$$

where

$$\begin{aligned} x &= \left(x_{u_N}(0,x(k))^\top, \ldots, x_{u_N}(N,x(k))^\top, u_N(x(k),0)^\top, \ldots, u_N(x(k),N-1)^\top\right)^\top \\ F(x) &= J_N(x(k), u_N) \\ G(x) &= \begin{pmatrix} \left[g_i\left(x_{u_N}(j,x(k)), u_N(x(k),j)\right)\right]_{i\in\mathcal{E}_g, j\in\mathcal{I}_x} \\ \left[x_{u_N}(i+1,x(k)) - \Phi\left(x_{u_N}(i,x(k)), u_N(x_{u_N}(x(k),i))\right)\right]_{i\in\mathcal{I}_u} \\ x_{u_N}(0,x(k)) - x(k) \end{pmatrix} \\ H(x) &= \left(\left[h_i\left(x_{u_N}(j,x(k)), u_N(x(k),j)\right)\right]_{i\in\mathcal{I}_h, j\in\mathcal{I}_x} \right) \\ \mathcal{E} &= \{0, \ldots, (r_g+n)\cdot(N+1)\} \\ \mathcal{I} &= \{0, \ldots, r_h\cdot(N+1)\} \end{aligned}$$

is called *fully discretized optimal control problem.*

Note that there are $(n\cdot(N+1)+m\cdot N)$ optimization variables and $((r_g+r_h+n)\cdot(N+1))$ constraint functions which renders the optimization problem to be quite large. Hence, every step of our receding horizon problem is difficult to solve in real–time, that is starting the optimization at time instant t_k, the result must be computed latest at time instant of implementation t_{k+j} for a fixed $j \geq 1$.

5.1.2 Recursive Discretization

As we will see in the following Section 5.2, the main computational cost within every step of the receding horizon control problem is caused by the calculation of the Jacobian of the constraints. Here, the combined vectors $G(\cdot)$ and $H(\cdot)$ have to be differentiated with respect to the optimization variable. Since the constraints $g_i(\cdot,\cdot)$ and $h_i(\cdot,\cdot)$ are given by the plant model, we cannot influence their quantity. Hence, we focus on analyzing the optimization variable and the dynamic of the system to reduce the arising computational effort.

Remark 5.3
Not all constraints within the combined vectors $G(\cdot)$ and $H(\cdot)$ have to be considered for the optimization since in most cases only a few of them are "active", cf. Definitions 5.14 and 5.22. This allows us to circumvent recomputing all derivatives in every step, see Section 5.2.3. In particular, the implemented optimization routines use the so called BFGS update formula to reduce the computational effort even further, see Section 5.2.4.4 for details.

Here, we exploit the fact that — due to implementation reasons of the sampled–data setting, cf. Definition 1.23 — equations (5.3), (5.4) are always satisfied. Hence, the set of equality constraints can be reduced to

$$G(x) = \left(\left[g_i\left(x_{u_N}(j,x(k)), u_N(x(k),j)\right)\right]_{i\in\mathcal{E}_g, j\in\mathcal{I}_x} \right) \tag{5.9}$$

and $\mathcal{E} = \{0, \ldots, r_g\cdot(N+1)\}$. Moreover, in every iterate of the receding horizon control scheme, the state trajectory (5.2) only depends on the given initial value $x(k)$ and the control sequence $u_N(x(k),\cdot) \in \mathbb{U}^{\mathcal{I}_u}$ only. In particular, the values $x_{u_N}(\cdot,x(k))$ can only be

changed by modifying the control sequence $u_N(x(k),\cdot)$. This allows us to outsource the dynamic of the system from the optimization and solve the underlying continuous–time control system (1.3) in a parallel manner. As a result, the optimization variable is reduced to

$$x := \left(u_N(x(k),0)^\top, \ldots, u_N(x(k),N-1)^\top\right)^\top \tag{5.10}$$

Hence, we obtain the following smaller optimization problem:

Definition 5.4 (Recursively discretized Optimal Control Problem)
Consider a single optimal control problem of the sequence $\mathrm{RHC}_{\mathrm{N}}^{\mathrm{B}}$. The corresponding nonlinear optimization problem in standard form NLP given by

$$\begin{array}{rll} \text{Minimize} & F(x) & \text{over all } x \in \mathbb{R}^{m\cdot N} \\ \text{ST.} & G_i(x) = 0, & i \in \mathcal{E} \\ & H_i(x) \geq 0, & i \in \mathcal{I} \end{array}$$

where

$$\begin{aligned} x &= \left(u_N(x(k),0)^\top, \ldots, u_N(x(k),N-1)^\top\right)^\top \\ F(x) &= J_N(x(k),u_N) \\ G(x) &= \left(\left[g_i\left(x_{u_N}(j,x(k)),u_N(x(k),j)\right)\right]_{i\in\mathcal{E}_g, j\in\mathcal{I}_x} \right) \\ H(x) &= \left(\left[h_i\left(x_{u_N}(j,x(k)),u_N(x(k),j)\right)\right]_{i\in\mathcal{I}_h, j\in\mathcal{I}_x} \right) \\ \mathcal{E} &= \{0,\ldots,r_g\cdot(N+1)\} \\ \mathcal{I} &= \{0,\ldots,r_h\cdot(N+1)\} \end{aligned}$$

is called *recursively discretized optimal control problem.*

As a result, the dimension of the optimization variable is reduced from $(n\cdot(N+1)+m\cdot N)$ to $(m\cdot N)$ and the number of constraints from $((r_g+r_h+n)\cdot(N+1))$ to $((r_g+r_h)\cdot(N+1)$. Hence, we expect this problem to be solved faster than the fully discretized optimal control problem from Definition 5.2.

5.1.3 Multiple Shooting Method for RHC

Another aspect of a discretization of a receding horizon control problem is the availability of a target information, i.e. the equilibrium $x^\star$ or the tracking signal $x_{\mathrm{ref}}(\cdot)$ which shall be stabilized is known in advance. This information can be utilized to speed up the optimization routine and even shorten the optimization horizon by adding multiple shooting nodes to the problem, see Section 8.3.4 for numerical results.
Here, a multiple shooting node is a state value at a fixed sampling instant which can be set arbitrarily and acts as an additional control value.

Definition 5.5 (Multiple Shooting Node)
Consider the optimal control problem of the sequence $\mathrm{RHC}_{\mathrm{N}}^{\mathrm{B}}$ corresponding to the initial sampling instant k. We call the i-th element of the state vector $x_{u_N}(j,x(k))$ at a fixed sampling instant j a *multiple shooting node* if this part of the state vector is added as an optimization variable. The *vector of multiple shooting nodes* is denoted by $s_x := (x_1,\ldots,x_s)$ where x_i corresponds to the i-th multiple shooting node. Moreover, the

functions $\varsigma : \{1, \dots, s\} \to \{1, \dots, N\}$ and $\iota : \{1, \dots, s\} \to \{1, \dots, n\}$ which identify the indices of the multiple shooting nodes and the vector of shooting nodes via

$$x_{u_N}(\varsigma(i), x(k))_{\iota(i)} = x_i \tag{5.11}$$

are refered to as *shooting index functions*.

Remark 5.6
In our implementation, the notation for positions in the time grid t_i and the number of the shooting horizon index $\sigma(\cdot)$ differ by one. This is due to an efficient memory allocation and the fact that the first position in the time grid corresponds to the initial value which is fixed by definition of the receding horizon control problem. Hence, no shooting node can be set for this time instant and the shooting horizon index zero corresponds to t_1 in the time grid.
Moreover, it does not make sense to define the last state vector to be a shooting node since no further control will be applied to this vector. Hence, setting a shooting node to this time instant would only increase the required computing times and is therefore not allowed in our implementation.

Using multiple shooting nodes we may improve the initial guess of the optimization vector, i.e. by presetting this value to the target value $x^\star$ or $x_{\text{ref}}(j)$, see Figure 5.1 for a schematic representation and Section 8.3.4 for numerical results.

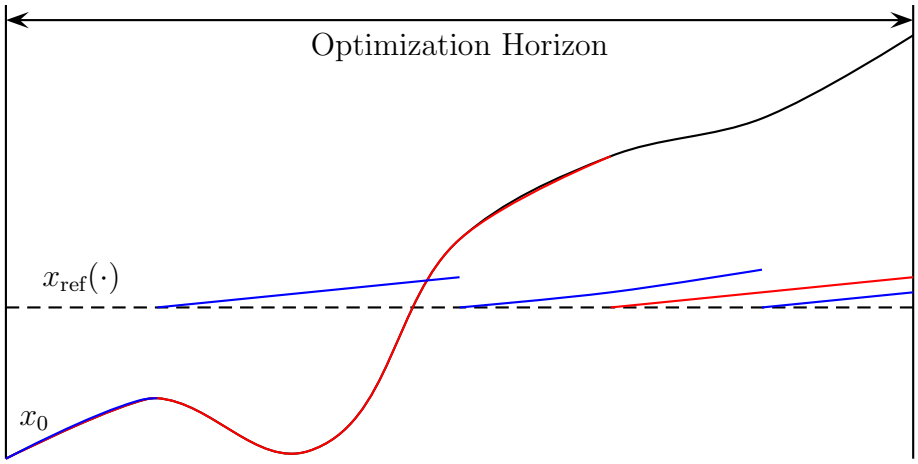


Figure 5.1: Resulting trajectories for initial guess u using no multiple shooting nodes (black), one shooting node (red) and three shooting nodes (blue)

Clearly, the initial guess reveals a trajectory which is closer to the optimum. This, in turn, eases the optimization.
Yet, if we introduce such a multiple shooting node into the recursively discretized optimal control problem from Definition 5.4, we cannot guarantee equations (5.3) to hold automatically. Hence, we have to add those equations which are affected by adding multiple shooting nodes, that is

$$\Big[x_i - \Phi\left(x_{u_N}(\varsigma(i) - 1, x(k)), u_N(x_{u_N}(x(k), \varsigma(i) - 1))\right)_{\iota(i)} = 0\Big]_{i \in \{1,\dots,s\}} \tag{5.12}$$

to the equality constraint function $G(\cdot)$. Moreover, we have to extend our optimization variable (5.10) by the vector of multiple shooting nodes s_x to enable the optimization

routine to deal with the additional constraints (5.12). As a result, we obtain the following optimization problem:

Definition 5.7 (Recursively discretized Optimal Control Problem with Shooting Nodes) Consider a single optimal control problem of the sequence $\mathrm{RHC}_{\mathrm{N}}^{\mathrm{B}}$. The corresponding nonlinear optimization problem in standard form NLP given by

$$\boxed{\begin{array}{lll} \text{Minimize} & F(x) & \text{over all } x \in \mathbb{R}^{m\cdot N+s} \\ \text{ST.} & G_i(x) = 0, & i \in \mathcal{E} \\ & H_i(x) \geq 0, & i \in \mathcal{I} \end{array}}$$

where

$$\begin{aligned} x &= \left(u_N(x(k),0)^\top, \ldots, u_N(x(k),N-1)^\top, s_x^\top\right)^\top \\ F(x) &= J_N(x(k), u_N) \\ G(x) &= \begin{pmatrix} \left[g_i\left(x_{u_N}(j,x(k)), u_N(x(k),j)\right)\right]_{i\in\mathcal{E}_g, j\in\mathcal{I}_x} \\ \left[x_i - \Phi\left(x_{u_N}(\varsigma(i)-1, x(k)), u_N(x_{u_N}(x(k),\varsigma(i)-1))\right)_{\iota(i)}\right]_{i\in\{1,\ldots,s\}} \end{pmatrix} \\ H(x) &= \left(\left[h_i\left(x_{u_N}(j,x(k)), u_N(x(k),j)\right)\right]_{i\in\mathcal{I}_h, j\in\mathcal{I}_x} \right) \\ \mathcal{E} &= \{0,\ldots,r_g\cdot(N+1)+s\} \\ \mathcal{I} &= \{0,\ldots,r_h\cdot(N+1)\} \end{aligned}$$

is called *recursively discretized optimal control problem with multiple shooting nodes.*

The possible improvement of the initial guess of the optimization vector induces additional s optimization variables and constraints compared to the optimization problem of Definition 5.4. Thus, a decrease in the computing time cannot be guaranteed but the numerical problem may become easier.

Remark 5.8
Note that the technique of using multiple shooting nodes within a recursively discretized optimal control problem represents a mixture of both full and recursive discretization. In particular, the full discretization can be obtained if the state values for all sampling instants and all state dimensions are considered as multiple shooting nodes.

Remark 5.9
As mentioned before, the use of multiple shooting nodes may allow us to shorten the optimization horizon N, see also Section 8.3.4, which may compensate for the additional control variables and constraints. Still, a balance between improving the initial guess of the control and the number of multiple shooting points needs to be found.

Remark 5.10
Within the receding horizon control setting, we usually consider the positions of the multiple shooting nodes to be fixed in a single optimal control problem of the sequence RHC_N^B. Yet, the number of nodes as well as the index function $\varsigma(\cdot)$ and $\iota(\cdot)$ may vary, see Sections 6.1.1.2 and 6.2.4 for the function definition and the discretization respectively.
Background for such an adaptation may be that information about the previous optimal control can be used to improve the first initial guess of the actual optimization problem, see Remark 5.9 and Section 8.3.3 which may be unnecessary for consecutive problems. Hence, the number of shooting points should be reconsidered in every step of the receding horizon control problem.

5.2 Optimization Technique for RHC

In Section 2.4 we saw that in every single step of the RHC algorithm a finite optimal control problem has to be solved to obtain the minimizing control sequence. Using one of the disretization techniques stated in the previous Section 5.1, we obtain a nonlinear optimization problem. Within this section, we treat the general problem of minimizing a nonlinear objective function $F(\cdot)$ subject to nonlinear equality and inequality constraints $G(\cdot)$ and $H(\cdot)$, that is we consider the problem

$$\boxed{\begin{array}{rll} \text{Minimize} & F(x) & \text{over all } x \in \mathbb{R}^n \\ \text{ST.} & G_i(x) = 0, & i \in \mathcal{E} \\ & H_i(x) \geq 0, & i \in \mathcal{I} \end{array}} \qquad \text{(NLP)}$$

where x is an n–dimensional parameter vector. Similar to Section 5.1 we denote the index sets of $G(\cdot)$ and $H(\cdot)$ by $\mathcal{E}$ and $\mathcal{I}$ respectively.

One of the most effective methods for solving nonlinearly constrained optimization problems is the so called *sequential quadratic programming* approach (SQP) which iteratively solves the problem by generating quadratic subproblems.
This approach is mainly used in two different frameworks, the trust–region or the line search variant, and is appropriate for small and large problems. In particular, SQP methods show their strength if one wants to solve problems with significant nonlinearities in the constraints.

Remark 5.11
Another well known approach to solve the problem NLP are so called interior point methods. Although our implementation contains a wrapper of the interior point method `IpOpt`[2]*, we do not present any details here since this part of the implementation has not been tested extensively yet and first results showed poor performance in terms of the computing time. For these reasons, we focus on the implemented SQP methods* `NLPQLP`[3] *and* `e04wdc`[4]*, see also Section 6.2 for details on the integration of these methods in our receding horizon controller.*

In this chapter we show how SQP methods work in general whereas details on the used implementations and their differences are described in Section 6.3.
There are two types of SQP methods. In the IQP approach, a general inequality–constrained quadratic program is solved at each iteration, with the twin goals of computing a step and generating an estimate of the optimal active set. EQP methods, however, decouple these computations. They first compute an estimate of the optimal active set and then solve an equality–constrained quadratic program to find the step.

The idea behind the SQP approach is to solve the problem NLP iteratively by

(1) approximating the problem at the current iterate $x^{(k)}$ using a quadratic programming subproblem and then

(2) use the minimizer of this subproblem to define a new iterate $x^{(k+1)}$.

[2]Webpage: `https://projects.coin-or.org/Ipopt`
[3]Webpage: `http://www.math.uni-bayreuth.de/~kschittkowski/nlpqlp.htm`
[4]Webpage: `http://www.nag.co.uk`

The first challenge here is to design a quadratic subproblem

$$\boxed{\begin{aligned} \text{Minimize} \quad & f_1(x^{(k)}) + f_2(x^{(k)})^\top d^{(k)} + \frac{1}{2} d^{(k)\top} f_3(x^{(k)}) d^{(k)} \qquad \text{over all } d^{(k)} \in \mathbb{R}^n \\ \text{ST.} \quad & f_4(x^{(k)})^\top d^{(k)} + f_5(x^{(k)}) \geq 0 \end{aligned}} \quad \text{(QP)}$$

which offers a good search direction $d^{(k)} \in \mathbb{R}^n$ for the nonlinear optimization problem. In particular, we have to suitably define the functions $f_1 : \mathbb{R}^n \to \mathbb{R}$, $f_2 : \mathbb{R}^n \to \mathbb{R}^n$ and $f_3 : \mathbb{R}^n \to \mathbb{R}^{n \times n}$ characterizing the cost criterion and the constraint functions $f_4 : \mathbb{R}^n \to \mathbb{R}^{p \times n}$ and $f_5 : \mathbb{R}^n \to \mathbb{R}^p$ where p is the number of equality and inequality constraints.

5.2.1 Analytical Background

We derive the subproblem QP using the first order necessary conditions for NLP, the so called Karush–Kuhn–Tucker (KKT) conditions. To this end, we require some basic definitions from optimization theory. In particular, we only worry about points satisfying all constraints, the so called feasible points.

Definition 5.12 (Feasible Points)
We call a point $x \in \mathbb{R}^n$ *feasible* if $G_i(x) = 0$ holds for all $i \in \mathcal{E}$ and $H_i(x) \geq 0$ for all $i \in \mathcal{I}$. The set $\mathcal{X} = \{x \in \mathbb{R}^n \mid x \text{ is feasible}\}$ is called *feasible set*.

These feasible points form a subset in $\mathbb{R}^n$ and we are interested in minimizers of the function $F(\cdot)$ over this set.

Definition 5.13 (Minimizer)
Consider a feasible point $x^\star \in \mathbb{R}^n$. Then we call $x^\star$

(1) *local minimizer* if there exists a neighborhood $\mathcal{N}(x^\star) \subset \mathbb{R}^n$ such that $F(x^\star) \leq F(x)$ holds for all $x \in \mathcal{X} \cap \mathcal{N}$.

(2) *strong local minimizer* if there exists a neighborhood $\mathcal{N}(x^\star) \subset \mathbb{R}^n$ such that $F(x^\star) < F(x)$ holds for all $x \in \mathcal{X} \cap \mathcal{N}$, $x \neq x^\star$.

(3) *global minimizer* if $F(x^\star) \leq F(x)$ holds for all $x \in \mathcal{X}$.

(4) *strong global minimizer* if $F(x^\star) < F(x)$ holds for all $x \in \mathcal{X}$, $x \neq x^\star$.

Moreover, we need to distinguish between inequality constraints which are satisfied with equality, i.e. which are *tight* or *active*, and those allowing for further relaxation.

Definition 5.14 (Active Set)
The *active set* $\mathcal{A}(x)$ at any feasible point x consists of the equality constraint indices from $\mathcal{E}$ together with the indices of the inequality constraints $i \in \mathcal{I}$ where $H_i(x) = 0$ holds, that is

$$\mathcal{A}(x) := \mathcal{E} \cup \{i \in \mathcal{I} \mid H_i(x) = 0\}. \tag{5.13}$$

Remark 5.15
For simplicity of exposition, we assume that indices within the active set $\mathcal{A}(x)$ represent exactly one constraint. For example, this can be acchieved by sorting the indices: If p_1 and p_2 are the number of constraints of the sets $\mathcal{E}(x)$ and $\mathcal{I}(x)$ respectively, then the first p_1 indices within the set $\mathcal{A}(x)$ can be identified with the p_1 indices of set $\mathcal{E}$. Thereafter, the indices $p_1 + i$ with $i \in \{j \in \mathcal{I}(x) \mid H_j(x) = 0\}$ are contained in the set $\mathcal{A}(x)$. Note that this is also the numbering of constraints within the used SQP implementations, see also Section 6.3.

Before stating the KKT conditions we introduce a *constraint qualification*. Conditions like this are often used in the design of algorithms to guarantee that the linearized approximation to the feasible set captures the essential shape of $\mathcal{X}$.

Definition 5.16 (Constraint Qualifications)
Consider a feasible point x and the active set $\mathcal{A}(x)$. If $F(\cdot)$, $G(\cdot)$ and $H(\cdot)$ are continuously differentiable and

(1) the elements of the gradient set

$$[\nabla_x G_i(x)]_{i \in \mathcal{A}(x)} \tag{5.14}$$

are linearly independent and there exists a $v \in \mathbb{R}^p \setminus \{0\}$ such that

$$\nabla_x G_i(x) v = 0 \tag{5.15}$$
$$\nabla_x H_i(x) v > 0 \tag{5.16}$$

holds for all $i \in \mathcal{A}(x)$, then we call x *regular*.

(2) the elements of the gradient set

$$[\nabla_x G_i(x)]_{i \in \mathcal{A}(x)} \cup [\nabla_x H_i(x)]_{i \in \mathcal{A}(x)} \tag{5.17}$$

are linearly independent we call x *normal* or *strongly regular*.

Condition (1) of Definition 5.16 is also called *Mangasarian–Fromowitz condition* (MFCQ), see [151], while (2) is referred to as *linear independence constraint qualification* (LICQ), cf. [175]. There also exist other constraint qualifications such as the *Slater condition*, see, e.g., [77, 196]. The proposed conditions, however, can be more easily checked analytically and numerically. In the following we assume the stated LICQ condition to hold.
Moreover, we introduce the *Lagrangian* of the problem NLP which represents the energy of the system regarding the present constraints.

Definition 5.17 (Lagrangian)
Consider $x \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}^p$. The function $L : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}$,

$$L(x, \lambda) := F(x) - \lambda^\top \begin{pmatrix} [G_i(x)]_{i \in \mathcal{E}(x)} \\ [H_i(x)]_{i \in \mathcal{I}(x)} \end{pmatrix}. \tag{5.18}$$

is called *Lagrangian* of the nonlinear optimization problem NLP. The components of λ are called the *Lagrange multipliers*.

Now we are ready to state the first order necessary conditions for a local minimum of the problem NLP shown, e.g., in [64, 80, 175]. These conditions are also the basis of the algorithm which we present to solve the problem NLP.

Theorem 5.18 (KKT — 1st Order Necessary Conditions)
Consider $x^\star \in \mathbb{R}^n$ to be a local minimizer of NLP. If $x^\star$ is regular, then there exists a Lagrange multiplier vector $\lambda^\star$ with components $\lambda_i^\star$, $i \in \mathcal{E} \cup \mathcal{I}$, such that the conditions

$$\nabla_x L(x^\star, \lambda^\star) = 0, \tag{5.19}$$
$$G_i(x^\star) = 0, \qquad \forall i \in \mathcal{E}, \tag{5.20}$$
$$H_i(x^\star) \geq 0, \qquad \forall i \in \mathcal{I}, \tag{5.21}$$
$$\lambda_i^\star \geq 0, \qquad \forall i \in \mathcal{I}, \tag{5.22}$$
$$\lambda_i^\star H_i(x^\star) = 0, \qquad \forall i \in \mathcal{I}. \tag{5.23}$$

are satisfied at $(x^\star, \lambda^\star)$. Moreover, the multiplier vector $\lambda^\star$ is unique if $x^\star$ is normal.

Geometrically speaking, if for $x^\star$ the set of active contraints $\mathcal{A}(x^\star)$ is the empty set, then the standard results for unconstrained problems can be applied. If, however, there exists at least one active constraint, then for any point satisfying the equality and inequality constraints any small deviation such that the point is still feasible causes a raise in the cost function, i.e. the gradient of the Lagrangian is a negative linear combination of the outward normal vectors of the active constraints.
Yet, the KKT conditions are not sufficient to check whether a candidate $x^\star$ is a local minimizer.

Definition 5.19 (Critical Points)
Points $x^\star$ satisfying the KKT conditions for a Lagrange multiplier λ are called *critical* or *KKT point*.

In order to see if a critical point actually is a local minimizer, sufficient conditions have to be checked. Within numerical optimization second order conditions have become standard but require the function to be at least twice differentiable. Here, we concentrate on this standard case which is shown, e.g., in [64, 80, 175]. First order sufficient conditions, see e.g. [114, 156], are not considered in the following.

Theorem 5.20 (2nd Order Conditions)
Consider a feasible, normal point $x^\star \in \mathbb{R}^n$. Moreover, $F(\cdot)$, $G(\cdot)$ and $H(\cdot)$ are twice continuously differentiable in a neighborhood $\mathcal{N}(x^\star)$. We define the cone

$$C(x^\star) := \left\{ v \in \mathbb{R}^n \;\middle|\; \begin{array}{l} \nabla_x H_i(x^\star) v \leq 0,\ i \in \mathcal{A}(x^\star),\ \text{if } \lambda_i = 0 \\ \nabla_x H_i(x^\star) v = 0,\ i \in \mathcal{A}(x^\star),\ \text{if } \lambda_i > 0 \\ \nabla_x G_i(x^\star) v = 0,\ i \in \mathcal{A}(x^\star) \end{array} \right\}. \tag{5.24}$$

(1) Consider a local minimizer $x^\star$ of problem NLP. Then there exists a unique Lagrange multiplier vector $\lambda \in \mathbb{R}^p$ satisfying

$$\lambda_i \geq 0, \quad \forall i \in \mathcal{A}(x^\star) \cap \mathcal{I} \qquad \text{and} \qquad \lambda_i = 0, \quad \forall i \notin \mathcal{A}(x^\star) \tag{5.25}$$
$$\nabla_x L(x^\star, \lambda) = 0, \tag{5.26}$$
$$v^T \nabla_{xx}^2 L(x^\star, \lambda) v \geq 0, \quad \forall v \in C(x^\star) \backslash \{0\}. \tag{5.27}$$

(2) Consider a Lagrange multiplier vector $\lambda \in \mathbb{R}^p$ such that the conditions

$$\lambda_i \geq 0, \quad \forall i \in \mathcal{A}(x^\star) \cap \mathcal{I} \qquad \text{and} \qquad \lambda_i = 0, \quad \forall i \notin \mathcal{A}(x^\star) \tag{5.28}$$
$$\nabla_x L(x^\star, \lambda) = 0, \tag{5.29}$$
$$v^T \nabla_x^2 L(x^\star, \lambda) v > 0, \quad \forall v \in C(x^\star) \backslash \{0\} \tag{5.30}$$

hold. Then there exist constants $c, \epsilon \in \mathbb{R}$, $c > 0$ *and* $\epsilon > 0$ *such that the inequality*

$$F(x) \geq F(x^\star) + c\|x - x^\star\|^2 \tag{5.31}$$

holds for $x \in C(x^\star)$ *where* $\|x - x^\star\| \leq \epsilon$.

(3) Consider a Lagrange multiplier vector $\lambda \in \mathbb{R}^p$ *such that* (5.28) — (5.30) *and the strict complementarity condition hold, i.e.*

$$\lambda_i > 0 \quad \forall i \in \mathcal{A}(x^\star) \cap \mathcal{I}. \tag{5.32}$$

Then the assertion of (2) holds true and the representation of $C(x^\star)$ *reduces to*

$$C(x^\star) = Ker\left(\{\nabla_x G_i(x^\star) \mid i \in \mathcal{A}(x^\star)\} \cup \{\nabla_x H_i(x^\star) \mid i \in \mathcal{A}(x^\star)\}\right).$$

Remark 5.21
The second order sufficient conditions may be used to analyze the sensitivity of optimal solutions regarding disturbances. Yet, for the RHC setting this requires a large amount of data which has to be saved and would result in an improvement of the already closed–loop nature of the RHC feedback only if the time instant of computation and implementation are not identical and if measurement errors occured in the meantime.

Geometrically, we obtain a quadratically increasing lower bound in the Lagrangian and the cost function from Theorem 5.20(2) for all neighbouring points of a candidate $x^\star$ within the cone $C(x^\star)$ which is sufficient for a local minimizer. Theorem 5.20(3) additionally states that a relaxation of any active inequality constraint leads to a further decrease of the cost function. The reduction is characterized by the Lagrange multiplier vector λ which can be interpreted as the shadow price of the corresponding constraints. Since the Lagrange multipliers of all inactive constraints are zero, it also motivates local search algorithms which consider active constraints only.

Definition 5.22 (Active Constraints)
Consider $x^\star$ to be feasible and $\mathcal{A}(x^\star)$ to be the active set. Then we call

$$A(x^\star) := \begin{pmatrix} (G_i)_{i \in \mathcal{A}(x^\star)} \\ (H_i)_{i \in \mathcal{A}(x^\star)} \end{pmatrix} \tag{5.33}$$

the *set* or *vector of active constraints* and $n_A = \sharp\mathcal{A}(x^\star)$ the *number* or *dimension of active constraints*. Moreover, we denote the corresponding Lagrange multiplier vector by λ^A.

The active set allows us to simplify the feasible cone.

Lemma 5.23
If $x^\star$ *is a normal optimal solution, then the cone* $C(x^\star)$ *is given by*

$$C(x^\star) = Ker(\nabla_x A(x^\star)). \tag{5.34}$$

Proof. The assertion follows directly from the normality property and the definition of $C(\cdot)$ in (5.24). □

Due to the constraint of the positive definiteness of the Hessian of the Lagrangian on the cone $C(x^\star)$ and due to the usually high dimension of the nonlinear optimization problem, the sufficient conditions stated in Theorem 5.20 cannot be checked without further ado. Here, we show how these conditions can be verified numerically. To this end, we need some technical results:

Definition 5.24 (Kernel of the Jacobian)
Consider a matrix $H \in \mathbb{R}^{n \times n-n_A}$ such that its columns are an orthogonal basis of $\mathrm{Ker}\,(\nabla_x A(x^\star))$, i.e.

$$\nabla_x A(x^\star) H = 0. \tag{5.35}$$

Then we call H the *kernel of the Jacobian* of the active constraints.

Lemma 5.25
If $x^\star$ is normal then we have

$$\mathit{rank}(\nabla_x A(x^\star)) = n_A. \tag{5.36}$$

Proof. The assertion is a direct conclusion of the normality of $x^\star$. □

Lemma 5.26
If H is the kernel of the Jacobian of the active constraints and $x^\star$ is normal, then H has full column rank, that is

$$\mathit{rank}(H) = n - n_A. \tag{5.37}$$

Proof. Since $x^\star$ is normal we have $\mathrm{rank}(\nabla_x A(x^\star)) = n_A$ and the assertion follows by the definition of H. □

Lemma 5.27
If H is the kernel of the Jacobian of the active constraints and $v \in \mathit{Ker}(\nabla_x A(x^\star))$ where $x^\star$ is normal, then there exists a vector $w \in \mathbb{R}^{n-n_A}$ such that

$$v = Hw. \tag{5.38}$$

Proof. Follows directly from Definition 5.24 and Lemma 5.26. □

Definition 5.28 (Projected Hessian)
Consider H to be the kernel of the Jacobian of the active constraints, $x^\star$ to be normal and let $\nabla^2_{xx} L(x^\star, \lambda^\star)$ denote the Hessian of the Lagrangian. Then we define the *projected Hessian* or *reduced Hessian* of the Lagrangian as

$$\nabla^2_{xx} L^p(x^\star, \lambda^\star) := H^\top \nabla^2_{xx} L(x^\star, \lambda^\star) H. \tag{5.39}$$

Now we can reformulate the sufficient condition (5.30) of Theorem 5.20.

Theorem 5.29 (2nd Order Sufficient Conditions)
Consider $\nabla^2_{xx} L^p(x^\star, \lambda^\star)$ to be the projected Hessian of the Lagrangian and $x^\star$ to be normal. Then condition (5.30) *is equivalent to*

$$w^\top \nabla^2_{xx} L^p(x^\star, \lambda^\star) w > 0 \tag{5.40}$$

to hold for all $w \in \mathbb{R}^{n-n_A} \setminus \{0\}$.

Proof. The equivalence is clear from Lemma 5.23 and the definition of the projected Hessian. □

Using Theorem 5.29, the verification of (5.30) is reduced to a check on positivity of the smallest eigenvalue of $\nabla^2_{xx} L(x^\star, \lambda^\star)$ provided H is known. To compute the missing matrix H, we need another technical lemma.

Lemma 5.30
If $x^\star$ is normal then there exists a RQ decomposition where $R \in \mathbb{R}^{n_A \times n}$ is triangular and has full row rank and $Q \in \mathbb{R}^{n \times n}$ is orthogonal, that is

$$\nabla_x A(x^\star) = RQ. \tag{5.41}$$

Proof. Clear since $\operatorname{rank}(\nabla_x A(x^\star)) = n_A$. □

Lemma 5.31
Consider $x^\star$ to be normal and R, Q as in Lemma 5.30. Then R and Q can be partitioned such that

$$R = [R_1 \mid 0], \quad Q = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix}. \tag{5.42}$$

Moreover, we can define

$$H := Q_2^\top \tag{5.43}$$

as the kernel of the Jacobian of the active constraints.

Proof. The existence of the partition matrices is clear by dimension arguments and Lemma 5.30. Moreover, we have

$$\nabla_x A(x^\star) H = RQQ_2^\top = R \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} Q_2^\top = [R_1 \mid 0] \begin{bmatrix} 0 \\ \mathrm{Id} \end{bmatrix} = 0$$

which shows the assertion. □

Note that the matrices Q_1 and R_1 are uniquely defined which is not true for Q_2. However, due to similarity of the projected Hessian to various bases of the kernel of the Jacobian of the constraints $\ker(\nabla_x A(x^\star))$ the eigenvalues of $\nabla_{xx} L^p(x^\star, \lambda)$ stay unchanged in (5.39).

5.2.2 Basic SQP Algorithm for Equality Constrained Problems

In order to derive an algorithm to solve the problem NLP, we first consider the simpler equality constrained problem

$$\boxed{\begin{array}{rll} \text{Minimize} & F(x) & \text{over all } x \in \mathbb{R}^n \\ \text{ST.} & A_i(x) = 0, & i \in \mathcal{A}(x) \end{array}} \tag{ECNLP}$$

of NLP where $\mathcal{A}(\cdot)$ and $A(\cdot)$ denote the active set and vector of active constraints according to Definitions 5.14 and 5.22 respectively. Moreover, we assume $F(\cdot)$, $G(\cdot)$ and $H(\cdot)$ to be twice continuously differentiable.
The KKT conditions for this problem are

$$M(x, \lambda^A) := \begin{pmatrix} \nabla_x L(x, \lambda^A) \\ A(x) \end{pmatrix} = 0. \tag{5.44}$$

This is a standard problem for nonlinear equation system solver. If we apply Newton's method to this problem, see e.g. [51], then the resulting iteration is given by

$$\begin{pmatrix} x^{(k+1)} \\ \lambda^{A^{(k+1)}} \end{pmatrix} = \begin{pmatrix} x^{(k)} \\ \lambda^{A^{(k)}} \end{pmatrix} - \left(\nabla_{x,\lambda^A} M(x^{(k)}, \lambda^{A^{(k)}})\right)^{-1} \begin{pmatrix} \nabla_x L(x^{(k)}, \lambda^{A^{(k)}}) \\ A(x^{(k)}) \end{pmatrix}. \tag{5.45}$$

Hence, if sufficient conditions for locally quadratic convergence of the resulting sequence are fulfilled, we are able to numerically compute a solution of the problem ECNLP, see [51, 213] for a convergence analysis. Since we assumed $F(\cdot)$, $G(\cdot)$ and $H(\cdot)$ to be twice continuously differentiable, we only have to check whether the Jacobian of (5.44) is invertible.

Theorem 5.32 (Regularity)
Consider $x^\star$ to be normal and a minimizer of $F(\cdot)$. Moreover, assume $x^\star$ to satisfy the sufficient conditions from Theorem 5.20. Then the matrix

$$A_0 := \begin{bmatrix} \nabla^2_{xx} L(x^\star, \lambda^\star) & \nabla_x A(x^\star)^\top \\ \nabla_x A(x^\star) & 0 \end{bmatrix} \tag{5.46}$$

is regular.

Proof. Consider $(v, w) \in \mathbb{R}^{n+n_A}$ such that $A_0(v^\top, w^\top)^\top = 0$. Then, we obtain

$$(v^\top, 0) A_0 \begin{pmatrix} v \\ w \end{pmatrix} = (v^\top \nabla^2_{xx} L(x^\star, \lambda^\star), v^\top \nabla_x A(x^\star)^\top) \begin{pmatrix} v \\ w \end{pmatrix} = v^\top \nabla^2_{xx} L(x^\star, \lambda^\star) v = 0$$

since $v^\top \nabla_x A(x^\star)^\top = 0$. Due to the positive definiteness of $\nabla^2_{xx} L(x^\star, \lambda^\star)$ on the kernel of the Jacobian of the active constraints in Theorem 5.20, see Theorem 5.29, we have $v = 0$. Hence, we see that $A_0(v^\top, w^\top)^\top = 0$ and $\nabla_x A(x^\star)^\top w = 0$ are equivalent and obtain $w = 0$ since $\nabla_x A(x^\star)$ has rank n_A due to the normality of $x^\star$. Concluding, we have $A_0(v^\top, w^\top)^\top = 0$ if and only if $(v, w) = (0, 0)$ and the assertion follows. □

Instead of Newton's iteration we can derive an iterative method for solving the problem NLP differently. To this end, we replace the cost function in problem ECNLP by its Lagrangian and approximate it by an equality constraint quadratic program (ECQP). In particular, we use a quadratic approximation of the cost function $F(\cdot)$ and linearize the active constraints $A(\cdot)$.

$$\begin{array}{rl} \text{Minimize} & F(x^{(k)}) + \nabla_x F(x^{(k)})^\top d^{(k)} + \dfrac{1}{2} d^{(k)\top} \nabla^2_{xx} L(x^{(k)}, \lambda^{A^{(k)}}) d^{(k)} \quad \text{over all } d^{(k)} \in \mathbb{R}^n \\ \text{ST.} & A_i(x^{(k)}) + \nabla_x A_i(x^{(k)})^\top d^{(k)} = 0, \qquad i \in \mathcal{A}(x^{(k)}) \end{array} \tag{ECQP}$$

Note that we can replace $\nabla_x F(x^{(k)})^\top d^{(k)}$ by $\nabla_x L^p(x^{(k)}, \lambda^{A^{(k)}})^\top d^{(k)}$ since the linearization of the equality constraints makes these two choices equivalent.
Now we can utilize the necessary and sufficient conditions from Theorems 5.18 and 5.20 respectively to obtain a basic SQP algorithm.

Lemma 5.33
If $\lambda^{(k)}$ denotes the Lagrange multipliers corresponding to the optimal solution $d^{(k)}$ of ECQP, then we have

$$\nabla^2_{xx} L(x^{(k)}, \lambda^{(k)}) d^{(k)} + \nabla_x F(x^{(k)}) = -\left(\lambda^{A^{(k)}}\right)^\top \nabla_x A(x^{(k)}). \tag{5.47}$$

Proof. Direct conclusion of Theorem 5.18. □

Lemma 5.34
Consider $x^\star$ to be a normal optimal solution of problem ECNLP. Moreover, assume the actual iterate $x^{(k)}$ to be close to the optimum $x^\star$ and $\lambda^{A^{(k)}}$, $d^{(k)}$ as in Lemma 5.33. Then

$$x^{(k)} \approx x^\star - d^{(k)} \tag{5.48}$$

$$\lambda^{A^{(k)}} \approx \lambda^\star \tag{5.49}$$

can be used as approximations of $(x^\star, \lambda^\star)$.

Proof. The assertion follows directly from the construction of ECQP as a local Taylor expansion in a neighborhood of the optimal solution. □

Lemma 5.35
Given an iterate $x^{(k)}$, then the corresponding Lagrange multiplier $\lambda^{A^{(k)}}$ and the search direction $d^{(k)}$ fulfill the linear equation system

$$\begin{bmatrix} \nabla^2_{xx} L(x^{(k)}, \lambda^{A^{(k)}}) & \left(\nabla_x A(x^{(k)})\right)^\top \\ \nabla_x A(x^{(k)}) & 0 \end{bmatrix} \begin{pmatrix} d^{(k)} \\ \lambda^{A^{(k)}} \end{pmatrix} + \begin{pmatrix} \nabla_x F(x^{(k)}) \\ A(x^{(k)}) \end{pmatrix} = 0. \tag{5.50}$$

Proof. The resulting linear equation system is a combination of the linearized constraints of problem ECQP and (5.47). □

Lemma 5.36
Consider $x^\star$ to be normal and a minimizer of $F(\cdot)$. Moreover, assume $x^\star$ to satisfy the sufficient conditions from Theorem 5.20. Then there exists a neighborhood $\mathcal{N}$ of $x^\star$ such that there exists a unique solution of (5.50).

Proof. Given $x^\star$ to be a normal minimizer of $F(\cdot)$, the matrix A_0 is invertible as shown in Theorem 5.32 and the assertion follows. □

The resulting iterate $(d^{(k)}, \lambda^{A^{(k)}})$ can therefore be calculated either as solution of problem ECQP or as iterate generated by Newton's method applied to the optimality conditions of problem ECNLP. The sequence of solution iterates of the problem ECQP defines the socalled SQP framework. While Newton's method is useful for the convergence analysis, practical algorithms and extensions to the inequality constrained case NLP can be derived using SQP. In particular, SQP methods avoid computing the Hessian of the cost function and utilize update techniques instead, see Section 5.2.4.4, which makes them computationally advantageous.
The simplest form of an SQP algorithm, the so called EQP algorithm, is motivated by Lemma 5.35.

Algorithm 5.37 (Basic local EQP Algorithm)
Input: $(x^{(0)}, \lambda^{(0)})$ — Pair of initial values

(1) Set $k := 0$

(2) While convergence test is not satisfied

(2a) Compute $F(x^{(k)})$, $\nabla_x F(x^{(k)})$, $\nabla^2_{xx} L(x^{(k)}, \lambda^{(k)})$, $\nabla_x A(x^{(k)})$ and $A(x^{(k)})$

(2b) Solve ECQP via (5.50) to obtain $d^{(k)}$ and $\lambda^{A^{(k)}}$

(2c) Set $x^{(k+1)} := x^{(k)} + d^{(k)}$

(2d) Set $k := k + 1$

Output: $(x^{(k)}, \lambda^{(k)})$ — Locally optimal pair

The quadratic problems ECQP are by now well known and there exists a wide variety of algorithms to solve these problems. Among them are direct approaches such as Gaussian elimination as shown e.g. in [51, 213] or symmetric indefinite factorization, cf. [34, 84], the Schur complement method and the null–space method as stated e.g. [79] and [10, 82] respectively as well as iterative methods, i.e. preconditioned conjugate gradient or projected conjugate gradient algorithms, see e.g. [14, 86, 201].
Since the optimization problem resulting from the discretization of optimal control problem within the receding horizon control problem possibly contains inequality constraints as well, our next aim is to incorporate this aspect in the SQP framework.

5.2.3 Extension to Inequality Constrained Problems

If we consider inequality constraints in Algorithm 5.37, this leads to the so called IQP algorithms. These methods differ only in treating *quadratic subproblems* instead of the entire problem which is why we generalize the problem ECQP to QP

$$\begin{array}{lll} \text{Minimize} & F(x^{(k)}) + \nabla_x F(x^{(k)})^\top d^{(k)} + \frac{1}{2} d^{(k)\top} \nabla^2_{xx} L(x^{(k)}, \lambda^{(k)}) d^{(k)} & \text{over all } d^{(k)} \in \mathbb{R}^n \\ \text{ST.} & G_i(x^{(k)}) + \nabla_x G_i(x^{(k)})^\top d^{(k)} = 0, \quad i \in \mathcal{E} & \text{(QP)} \\ & H_i(x^{(k)}) + \nabla_x H_i(x^{(k)})^\top d^{(k)} \geq 0, \quad i \in \mathcal{I} & \end{array}$$

The SQP algorithm can then be rewritten by exchanging the call for solving the ECQP problem by computing the solution of the problem QP:

Algorithm 5.38 (Basic Local IQP Algorithm)
Input: $(x^{(0)}, \lambda^{(0)})$ — Pair of initial values

(1) Set $k := 0$

(2) While convergence test is not satisfied

(2a) Compute $F(x^{(k)})$, $\nabla_x F(x^{(k)})$, $\nabla^2_{xx} L(x^{(k)}, \lambda^{(k)})$, $\nabla_x G(x^{(k)})$, $G(x^{(k)})$, $\nabla_x H(x^{(k)})$ and $H(x^{(k)})$

(2b) Solve QP to obtain $d^{(k)}$ and $\lambda^{(k)}$

(2c) Set $x^{(k+1)} := x^{(k)} + d^{(k)}$

(2d) Set $k := k + 1$

Output: $(x^{(k)}, \lambda^{(k)})$ — Locally optimal pair

Note that until now we have not specified how the problem QP can be solved. Similar to the equality constrained case, there exists a wide variety of algorithms designed for this purpose. These algorithm can be assigned to two classes, *interior point* and *active set methods*.
All such algorithms are based on the KKT conditions which are sufficient if $\nabla^2_{xx} L(x^{(k)}, \lambda^{(k)})$ is positive semidefinite. Hence, we face two sources of difficulties of Theorem 5.20: If the Hessian is not positive definite then there may exist more than one solution which in particular does not have to be a minimum. Secondly, there may exist degenerate Lagrange

multipliers $\lambda_i^\star = 0$ where $i \in \mathcal{A}(x^\star)$ or the active constraint gradients may be linearly dependent. In this case, a factorization may not be possible or the algorithm is unable to decide whether the constraint is active and repeatedly switches this assignment.

Remark 5.39
Since we focus on SQP algorithms in our implementation, see Remark 5.11, we do not present any analytical background for interior point methods here. For further details on this class of solvers, we refer to [175, 227].

Note that, given the described setup, if the SQP method can identify and does not change the optimal active set, then it acts like a Newton method for problem ECQP and converges (locally) quadratically. Sufficient conditions for this behaviour are proved in [195]:

Theorem 5.40 (Robinson)
Consider $x^\star$ to be a KKT point and $\lambda^\star$ to be a corresponding Lagrange multiplier. Moreover, we suppose the conditions of Theorem 5.20(3) to hold and $(x^{(k)}, \lambda^{(k)})$ to be sufficiently close to $(x^\star, \lambda^\star)$. Then there exists a local solution of problem QP whose active set $\mathcal{A}(x^{(k)})$ is the same as the active set $\mathcal{A}(x^\star)$ of problem NLP.

Remark 5.41
For the real–time applicability of the receding horizon control problem this corresponds to finding a good initial guess of the control to speed up the optimization process, see also Section 8.3.3 for numerical results.

For reasons of simplicity, we only consider the convex case of problem QP here, for the nonconvex case we refer to [85]. Moreover, we consider the case given by Algorithm 5.38, in particular this means that the derivatives of the problem NLP are not recomputed.
The main challenge considering QP problems is the proper identification of the set of active constraints. If these were known in advance, we could apply Algorithm 5.37 for the equality constrained problem.
The fundamental idea used in active–set methods is similar to the simplex method, see e.g. [172]. First, we consider a so called *working set* $\mathcal{W}^{(k)}$ which contains some of the inequality and all equality constraints. All constraints within the working set are then treated as equality constraints and the resulting quadratic problem is solved. Here, we require that the gradients of the constraints contained in the working set are linearly independent, even if the full set of active constraints at that point has linearly dependent gradients.
Solving the described problem may lead to two cases: Either the actual iterate $x^{(k)}$ is a local minimizer, or we compute a step $d^{(k)}$. We first consider the second case and have to face the problem that we neglected the inactive inequality constraints. Hence, the new iterate $x^{(k+1)} := x^{(k)} + d^{(k)}$ may not be feasible and the step length may have to be shortened. To this end, we introduce a step length parameter $\alpha^{(k)} \in [0, 1]$. Note that for the constraints contained in the working set $\mathcal{W}^{(k)}$, this problem does not occur along the search direction since the equality constraints are linear in $d^{(k)}$.
Our next task is to derive a suitable step length $\alpha^{(k)}$ such that

$$x^{(k+1)} := x^{(k)} + \alpha^{(k)} d^{(k)} \tag{5.51}$$

is feasible and $\alpha^{(k)}$ is as large as possible. To this end, we consider all constraints which are not contained in the working set $\mathcal{W}^{(k)}$. If $\nabla_x H_i(x^{(k)}) d^{(k)} \geq 0$ holds for $i \notin \mathcal{W}^{(k)}$, then we have

$$H_i(x^{(k)}) + \nabla_x H_i(x^{(k)})^\top \cdot (x^{(k)} + \alpha^{(k)} d^{(k)}) \geq H_i(x^{(k)}) + \nabla_x H_i(x^{(k)})^\top x^{(k)} \geq 0$$

by the feasibility assumption of $x^{(k)}$, and hence $\alpha^{(k)} \geq 0$ can be chosen freely. In the case $\nabla_x H_i(x^{(k)})^\top d^{(k)} < 0$, we obtain $H_i(x^{(k)}) + \nabla_x H_i(x^{(k)})^\top \cdot (x^{(k)} + \alpha^{(k)} d^{(k)}) \geq 0$ only if

$$\alpha^{(k)} \leq \frac{-H_i(x^{(k)}) - \nabla_x H_i(x^{(k)})^\top x^{(k)}}{\nabla_x H_i(x^{(k)})^\top d^{(k)}}$$

holds true. In order to maximize the decrease in the cost function we define

$$\alpha^{(k)} := \min \left(1, \min_{i \notin \mathcal{W}^{(k)},\, \nabla_x H_i(x^{(k)})^\top d^{(k)} < 0} \frac{-H_i(x^{(k)}) - \nabla_x H_i(x^{(k)})^\top x^{(k)}}{\nabla_x H_i(x^{(k)})^\top d^{(k)}} \right) \tag{5.52}$$

Note that $\alpha^{(k)} = 0$ is possible since there might exist an active constraint which is not an element of the working set $\mathcal{W}^{(k)}$ and exhibits $\nabla_x H_i(x^{(k)})^\top d^{(k)} < 0$. Moreover, we call the constraints $H_i(x^{(k)})$ for which the minimum is achieved *blocking constraints* and denote the *set of blocking constraints* by

$$\mathcal{C}^{(k)} := \left\{ j \in \mathcal{W}^{(k)} \;\middle|\; \alpha^{(k)} = \frac{-H_j(x^{(k)}) - \nabla_x H_j(x^{(k)})^\top x^{(k)}}{\nabla_x H_j(x^{(k)})^\top d^{(k)}} \right\}. \tag{5.53}$$

If $\alpha^{(k)}$ can be chosen to 1, we can continue to the next iterate. If $\alpha^{(k)} < 1$, however, we know that at least one constraint is active for $x^{(k+1)}$ which is not contained in $\mathcal{W}^{(k)}$. Hence, we construct a new working set $\mathcal{W}^{(k+1)}$ by adding one of the blocking constraints. This can be done in an iterative manner until we reach a point $x^\star$ which minimizes the quadratic objective function over its current working set $\mathcal{W}^\star$. Note that for such a point $x^\star$, we have $d^\star = 0$ and the optimality conditions are satisfied by construction of the working set. Hence, we obtain the Lagrange multiplier $\lambda_i^\star$ for all $i \in \mathcal{W}^\star$ such that

$$\sum_{i \in \mathcal{W}^\star} \nabla_x H_i(x^\star)^\top \lambda_i^\star = -\nabla_{xx}^2 L(x^\star, \lambda^\star) x^\star - \nabla_x F(x^\star) \tag{5.54}$$

holds according to Lemma 5.33. This multiplier can be extended to a complete one by defining the multipliers corresponding to those constraints not contained in $\mathcal{W}^\star$ to be zero. In particular, we obtain a search direction if $d^{(k)}$ is nonzero and the value of the cost function is descending if the second–order sufficient conditions hold, i.e.

Theorem 5.42 (Decrease along Search Direction)
If $d^{(k)}$ is nonzero and conditions (5.28)–(5.30) *hold, then $F(\cdot)$ is strictly decreasing along the direction $d^{(k)}$.*

Proof. From conditions (5.28)–(5.30) and Lemma 5.36 we know that $d^{(k)}$ is the unique solution of the actual problem ECQP. Since $d^{(k)} = 0$ is feasible by definition but must exhibit a larger cost function value, we compare these two solutions and obtain

$$\frac{1}{2} d^{(k)\top} \nabla_{xx}^2 L(x^{(k)}, \lambda^{(k)}) d^{(k)} + \nabla_x F(x^{(k)}) d^{(k)} < 0.$$

Due to convexity the quadratic term is positive and hence $\nabla_x F(x^{(k)}) d^{(k)} < 0$ holds. Last, testing the search direction shows the assertion since

$$F(x^{(k)} + \alpha^{(k)} d^{(k)}) = F(x^{(k)}) + \alpha^{(k)} \nabla_x F(x^{(k)}) d^{(k)} + \frac{{\alpha^{(k)}}^2}{2} d^{(k)\top} \nabla_{xx}^2 L(x^{(k)}, \lambda^{(k)}) d^{(k)} < F(x^{(k)})$$

holds if $\alpha^{(k)}$ is chosen sufficiently small. □

Next, we analyze the multipliers corresponding to the inequality constraints contained in the working set. If these multipliers show nonnegative sign, then all KKT conditions are satisfied. If moreover $\nabla^2_{xx} L(x^{(k)}, \lambda^{(k)})$ is positive semidefinite, we know from Theorem 5.20 and our convexity assumption that $x^\star$ is actually a global minimizer.
The more interesting case comes up if one or more of the multipliers $\lambda^\star$ are negative. Hence, a further decrease in the cost function can be obtained if these constraints are removed from the working set. Note that until now we have only used knowledge from the equality constrained context shown in Section 5.2.2. Here, we drop one of the active constraints and we have to show that the solution at the next iterate is feasible.

Theorem 5.43 (Feasibility)
Consider $x^{(k)}$ to be normal for the equality constrained problem with working set $\mathcal{W}^{(k)}$ and assume $\lambda_j^{(k)} < 0$ for some $j \in \mathcal{W}^{(k)}$. If $d^{(k)}$ is the solution of the equality constrained problem with working set $\mathcal{W}^{(k)} \setminus \{j\}$, then $d^{(k)}$ is a feasible direction for the j-th constraint, i.e. $\nabla_x H_j(x^{(k)})^\top d^{(k)} \geq 0$ holds. If $d^{(k)}$ additionally satisfies the second–order sufficient conditions, then $\nabla_x H_j(x^{(k)})^\top d^{(k)} > 0$ is true and $F(\cdot)$ is decreasing along the search direction $d^{(k)}$.

Proof. Since $d^{(k)}$ solves problem ECQP, the linearity property of the considered (linearized) constraints guarantee that there exists multiplier $\tilde{\lambda}_i^{(k)}$ for all $i \in \mathcal{W}^{(k)}$, $i \neq j$ such that

$$\sum_{i \in \mathcal{W}^{(k)}, i \neq j} \nabla_x H_i(x^{(k)})^\top \tilde{\lambda}_i^{(k)} = -\nabla^2_{xx} L(x^{(k)}, \lambda^{(k)}) \cdot (x^{(k)} + d^{(k)}) - \nabla_x F(x^{(k)})$$

holds. Since $x^{(k)}$ is normal, we can use (5.54) to obtain

$$\sum_{i \in \mathcal{W}^{(k)}, i \neq j} \nabla_x H_i(x^{(k)})^\top \cdot (\tilde{\lambda}_i^{(k)} - \lambda_i^{(k)}) - \nabla_x H_i(x^{(k)})^\top \lambda_j^{(k)} - = -\nabla^2_{xx} L(x^{(k)}, \lambda^{(k)}) d^{(k)}. \quad (5.55)$$

Taking the inner products with $d^{(k)}$ we have

$$-d^{(k)\top} \nabla_x H_i(x^{(k)})^\top \lambda_j = d^{(k)\top} \nabla^2_{xx} L(x^{(k)}, \lambda^{(k)}) d^{(k)}$$

and the assertion follows.
If the second–order sufficient conditions hold, then $\nabla^2_{xx} L(x^{(k)}, \lambda^{(k)})$ is positive definite. Moreover, the last inequality reveals a strikt descrease if $d^{(k)} \neq 0$. If this was the case, then we would obtain $\lambda_j = 0$ from (5.55) which contradicts our choice of j and completes the proof. □

Remark 5.44
In practice, not just any index is chosen but the one corresponding to the most negative Lagrange multiplier λ_j which is motivated by sensitivity aspects.

Now we combine all these steps to the *active–set algorithm for convex QP*:

Algorithm 5.45 (Active–Set Algorithm for Convex QP)
Input: $x^{(0)}$ — Feasible optimization vector

(1) Set $\mathcal{W}^{(0)} \subset \mathcal{A}(x^{(0)})$

(2) While not terminated

(2a) Solve ECQP with constraint set $\mathcal{W}^{(k)}$

(2b) If $d^{(k)} = 0$

- Compute Lagrange multipliers $\lambda^\star$ satisfying (5.54) with $\mathcal{W}^\star = \mathcal{W}^{(k)}$
- If $\lambda_j^\star \geq 0$ for all $i \in \mathcal{W}^{(k)} \cap \mathcal{I}$: Set $x^\star := x^{(k)}$ and terminate
  Else: Set $j := \underset{j \in \mathcal{W}^{(k)} \cap \mathcal{I}}{\operatorname{argmin}} \lambda_j^\star$, $x^{(k+1)} := x^{(k)}$, $\mathcal{W}^{(k+1)} := \mathcal{W}^{(k)} \setminus \{j\}$

Else

- Compute $\alpha^{(k)}$ from (5.52) and $\mathcal{C}^{(k)}$ from (5.53)
- Set $x^{(k+1)} := x^{(k)} + \alpha^{(k)} d^{(k)}$
- If $\mathcal{C}^{(k)} \neq \emptyset$: Set $\mathcal{W}^{(k+1)} := \mathcal{W}^{(k)} \cup \{j\}$, $j \in \mathcal{C}^{(k)}$
  Else: Set $\mathcal{W}^{(k+1)} := \mathcal{W}^{(k)}$

Output: $x^\star$ — Optimal solution

This algorithm allows us to maintain the linear independence property of constraints which are contained in $\mathcal{W}^{(k)}$. In particular, if the gradients of the active constraints of the initial value are linearly dependent, then we can consider a subset of linear independent constraints. During the iteration we may encounter blocking constraints. The normals of these constraints, however, cannot be represented by a linear combination of the normals of the constraints contained in the working set $\mathcal{W}^{(k)}$, i.e. linear independence is preserved if one constraint is added. Last, the deletion of a constraint from the working set $\mathcal{W}^{(k)}$ does not lead to linear dependency of the remaining constraint normals.

Remark 5.46
The details of Algorithm 5.45 show the importance of a good initial guess for the optimizer and hence also for our receding horizon control problem. For a bad guess, repeated updates on the working set are required which causes the step length to be shortened. In turn, this leads to an increase of the number of SQP steps.

Remark 5.47
The receding horizon control problem offers us additional structure which allows us to speed up the optimization process. In particular, considering the n–th problem in the sequence of optimal control problems, the result of the $(n-1)$–th problem reveals a good initial guess for the n–th problem. Hence, reinitialization of one the routines `NLPQLP` *or* `e04wdc` *with default values should not be considered, see also Section 8.3.3.*
Additionally, we can supply knowlegde of the target to the optimizer by utilizing multiple shooting nodes, see Section 5.1.3, which may also help to identify the correct working set $\mathcal{W}^\star$ faster. In this case, we hope to at least cancel out the additional effort of computing a larger gradient of the cost function and a larger Jacobian of the constraints if the number of required SQP steps is reduced, see also Section 8.3.4 for numerical results.

Theorem 5.48 (Finite Termination)
If the problem QP is strictly convex then Algorithm 5.45 terminates in a finite number of steps.

Proof. If we have $d^{(k)} = 0$, then the current point $x^{(k)}$ is the unique global minimizer of the cost function $F(\cdot)$ for the working set $\mathcal{W}^{(k)}$, see discussion after Theorem 5.42. If $x^{(k)}$ is no solution of problem NLP, then there exists at least one negative Lagrange multiplier. Hence, Theorems 5.42 and 5.43 show existence of a feasible step into a search direction

such that the cost function decreases once the corresponding constraint is dropped. Since the step length satisfies $\alpha^{(k)} > 0$, we obtain a strict decrease and hence the previous working set $\mathcal{W}^{(k)}$ is never regarded again since $x^{(k)}$ already was the global minimizer for this set.
Moreover, the algorithm visits a point such that $d^{(k)} = 0$ at least every n–th iteration. To show this, we first consider an iterate j such that $d^{(j)} \neq 0$. Hence, we either have $\alpha^{(j)} = 1$ and reach the minimizer of the cost function on the current working set $\mathcal{W}^{(j)}$ such that $d^{(j+1)} = 0$ holds, or a constraint is added to the working set $\mathcal{W}^{(j)}$. The second case may occur repeatedly but after at most n added constraints the working set contains n linearly independent vectors. Hence, the only possible solution is the zero solution.
Concluding, we have that the algorithm at least periodically finds the global minimum of the cost function on the current working set and thereafter never vists this set again. Since the number of working sets is finite, the algorithm locates a minimizer for a current working set such that the optimality conditions from Theorem 5.20 for problem NLP are satisfied and terminates in finite time. □

The algorithms `NLPQLP` and `e04wdc` which we use in our implementation of the receding horizon controller are two different types of such active–set methods, i.e. a *line search* method and a *trust–region* variant. The general procedures of these two methods are discussed in Sections 5.2.5 and 5.2.6 respectively. In the following Section 5.2.4, we describe four implementation aspects which are by now standard for most solvers including `NLPQLP` and `e04wdc`. Details of the algorithms `NLPQLP` and `e04wdc` are discussed in Sections 6.3.2 and 6.3.3 which also deal with their integration in our package *PCC2*.

5.2.4 Implementation Issues

Until now we focused on the basic ideas of SQP algorithms and how the iteratively generated subproblems are solved. For these subproblems, however, we can in general not guarantee that conditions like feasibility, LICQ or convexity hold. Here, we introduce techniques to work these issues. Later, we use the obtained knowledge in the two main categories of practically implemented SQP methods, i.e. line search and trust–region methods, see Sections 5.2.5 and 5.2.6.

5.2.4.1 Merit Function

As we have seen in Section 5.2.3 for the active–set methods, accepting the new iterate $x^{(k+1)} := x^{(k)} + d^{(k)}$ as proposed in step (2c) of Algorithm 5.38 may cause the consecutive problem to be infeasible. A *merit function* incorporates the violation aspect within a new objective which shall be decreasing along the iterates of the optimization process. To this end, it uses a measure of the constraint violation to decide whether a trial step should be accepted. Here, the main implementations of line search and trust–region differ: For the latter one the merit function determines if the step is accepted or rejected and if the trust–region radius needs to be adapted. In contrast to that, the merit function controls the steplength itself within the line search setting.

Remark 5.49
For the receding horizon controller, this may result in different points in the control sequence space $\mathbb{U}^N$ to be visited by the optimization routine. Hence, if stopping criteria such as computing time termination or maximal iteration numbers are considered — which we use within our implementation — then results for the methods `NLPQLP` and `e04wdc` most

likely deviate. Note that no method is preferable in general since the steps of the optimizer depend on the initial guess of the control.
Moreover, the optimization methods `NLPQLP` *and* `e04wdc` *may end up in different local optima since they only converge locally. But even if the same optimum is identified within every step of the receding horizon control algorithm described in Section 2.4, the closed–loop solutions are most likely not identical since the allowed optimization tolerance leads to errors in the control. Still, if the system is stabilized by the controller, the difference between the solutions stays bounded due to the feedback nature of the control.*

In the literature, several merit functions are considered, e.g. the non–differentiable function

$$\widetilde{L}(x,\mu) := F(x) + \mu\|A(x)\|_1 \tag{5.56}$$

is used in [112, 184] while

$$\widetilde{L}(x,\mu) := F(x) + \frac{\mu\|A(x)\|_2^2}{2}. \tag{5.57}$$

is chosen in [200, 201]. Typically only active constraints are considered. This leads to the introduction of a vector of slack variables $s \geq 0$ such that inequality constraints are converted to equality constraints via

$$\tilde{H}(x,s) := H(x) - s = 0$$

and are hence also contained in the active set. Here, we are going to consider the merit function (5.56). In particular, we have to show that the search direction $d^{(k)}$ of Algorithm 5.37 is also a direction of decrease for the merit function.

Theorem 5.50
Consider $d^{(k)}$ and $\lambda^{(k)}$ calculated according to Algorithm 5.37. Then the directional derivative of $\widetilde{L}(x^{(k)},\mu^{(k)})$ in direction $d^{(k)}$ is given by

$$D\left(\widetilde{L}(x^{(k)},\mu^{(k)}); d^{(k)}\right) = \nabla_x F(x^{(k)})^\top d^{(k)} - \mu^{(k)}\|A(x^{(k)})\|_1 \tag{5.58}$$

and

$$D\left(\widetilde{L}(x^{(k)},\mu^{(k)}); d^{(k)}\right) \leq -d^{(k)\top}\nabla^2_{xx}L(x^{(k)},\lambda^{(k)})d^{(k)} - (\mu^{(k)} - \|\lambda^{(k+1)}\|_\infty)\|A(x^{(k)})\|_1 \tag{5.59}$$

holds.

Proof. Using the Taylor expansion we obtain

$$\begin{aligned}\widetilde{L}(x^{(k)} + \alpha d^{(k)},\mu^{(k)}) - \widetilde{L}(x^{(k)},\mu^{(k)}) \leq{}& \alpha\nabla_x F(x^{(k)})^\top d^{(k)} + \gamma\alpha^2\|d^{(k)}\|^2 \\ &+ \mu^{(k)}\|A(x^{(k)}) + \alpha\nabla_x A(x^{(k)})d^{(k)}\|_1 - \mu^{(k)}\|A(x^{(k)})\|_1.\end{aligned}$$

Since by (5.50) we have $\nabla_x A(x^{(k)})d^{(k)} = -\nabla_x F(x^{(k)})$ the two inequalities

$$\begin{aligned}\widetilde{L}(x^{(k)} + \alpha d^{(k)},\mu^{(k)}) - \widetilde{L}(x^{(k)},\mu^{(k)}) &\leq \alpha\left[\nabla_x F(x^{(k)})^\top d^{(k)} - \mu^{(k)}\|A(x^{(k)})\|_1\right] + \gamma\alpha^2\|d^{(k)}\|^2 \\ \widetilde{L}(x^{(k)} + \alpha d^{(k)},\mu^{(k)}) - \widetilde{L}(x^{(k)},\mu^{(k)}) &\geq \alpha\left[\nabla_x F(x^{(k)})^\top d^{(k)} - \mu^{(k)}\|A(x^{(k)})\|_1\right] - \gamma\alpha^2\|d^{(k)}\|^2\end{aligned}$$

hold. Hence, we can take $\alpha \to 0$ and the assertion (5.58) follows. Hence, we obtain

$$D(\widetilde{L}(x^{(k)}, \mu^{(k)}); d^{(k)}) = \\ = -d^{(k)\top} \nabla^2_{xx} L(x^{(k)}, \lambda^{(k)}) d^{(k)} + d^{(k)\top} \nabla_x A(x^{(k)})\top \lambda^{(k+1)} - \mu^{(k)} \|A(x^{(k)})\|_1.$$

where $\lambda^{(k)}$ is the Lagrangian multiplier vector corresponding to the iterate $x^{(k)}$. Now we can replace

$$d^{(k)\top} \nabla_x A(x^{(k)})^\top \lambda^{(k+1)} = -\nabla_x F(x^{(k)}) \lambda^{(k+1)} \leq \|\nabla_x F(x^{(k)})\|_1 \|\lambda^{(k+1)}\|_\infty$$

in the last equality which shows (5.59) and concludes the proof. □

As we can see from (5.59), this decrease condition holds if the parameter $\mu^{(k)}$ is chosen sufficiently large. For the choice of $\mu^{(k)}$, we define a piecewise quadratic model of $\widetilde{L}(\cdot, \cdot)$ by

$$Q_{\mu^{(k)}}(d^{(k)}) := F(x^{(k)}) + \nabla_x F(x^{(k)})^\top d^{(k)} + \frac{\sigma}{2} d^{(k)\top} \nabla^2_{xx} L(x^{(k)}, \lambda^{(k)}) d^{(k)} + \mu^{(k)} M(d^{(k)})$$

where $M(d^{(k)}) = \|A(x^{(k)}) + \nabla_x A(x^{(k)}) d^{(k)}\|_1$ and

$$\sigma = \begin{cases} 1 & \text{if } d^{(k)\top} \nabla^2_{xx} L(x^{(k)}, \lambda^{(k)}) d^{(k)} > 0 \\ 0 & \text{else} \end{cases}$$

This choice of σ allows us to consider non positive definite matrices $\nabla^2_{xx} L(x^{(k)}, \lambda^{(k)})$ as well. The parameter $\mu^{(k)}$ is chosen sufficiently large such that

$$Q_{\mu^{(k)}}(0) - Q_{\mu^{(k)}}(d^{(k)}) \geq \rho \mu^{(k)} \left[M(0) - M(d^{(k)}) \right] \tag{5.60}$$

holds for some $\rho \in (0, 1)$. From the definition of the model of $\widetilde{L}(\cdot, \cdot)$ and the linearized constraints of the problem NLP with slack variables vector s we obtain that

$$\mu^{(k)} \geq \frac{\nabla_x F(x^{(k)})^\top d^{(k)} + \frac{\sigma}{2} d^{(k)\top} \nabla^2_{xx} L(x^{(k)}, \lambda^{(k)}) d^{(k)}}{(1 - \rho) \|A(x^{(k)})\|_1} \tag{5.61}$$

is sufficient for (5.60) to hold.

5.2.4.2 Maratos Effect

Imposing a merit function may also obstruct the optimization algorithm. This phenomenon is called the *Maratos effect*, see [152, 186]. If no measures are taken, it slows down the considered method by rejecting steps which else make good progress towards a solution and by preventing superlinear convergence. Here, we mention two strategies for avoiding the Maratos effect:

(1) a second–order correction of the search direction

(2) a nonmonotone strategy which allows for an increase in the merit function

These two approach differ significantly: The second–order correction aims at improving the merit function and hence the satisfaction of the constraints of the original problem. The nonmonotone strategy tries to enhance feasibility and optimality at the same time by allowing for temporary increases in the merit function.

The second–order correction term $\hat{d}^{(k)}$ of the search direction can be obtained as the minimum–norm solution of the equation

$$\nabla_x A(x^{(k)})\hat{d}^{(k)} + A(x^{(k)} + d^{(k)}) = 0,$$

that is

$$\hat{d}^{(k)} = -\nabla_x A(x^{(k)})^\top \left(\nabla_x A(x^{(k)})\nabla_x A(x^{(k)})^\top\right)^{-1} A(x^{(k)} + d^{(k)}). \tag{5.62}$$

A correction algorithm can then be implemented straight forward:

Algorithm 5.51 (Second–order Correction Algorithm)

Input:	$x^{(k)}$	—	Current optimization vector
	$d^{(k)}$	—	Search direction
	$\eta \in (0, 0.5)$	—	Decrease parameter
	$\tau_1, \tau_2 \in (0, 1)$, $\tau_1 < \tau_2$	—	Search parameter
	$\alpha^{(k)} = 1$	—	Step length parameter

While not terminated

If $\widetilde{L}\left(x^{(k)} + \alpha^{(k)} d^{(k)}, \mu^{(k)}\right) \leq \widetilde{L}\left(x^{(k)}, \mu^{(k)}\right) + \eta\alpha^{(k)} D\left(\widetilde{L}\left(x^{(k)}, \mu^{(k)}\right); d^{(k)}\right)$:

- Set $x^{(k+1)} := x^{(k)} + \alpha^{(k)} d^{(k)}$ and terminate.

Else if $\alpha^{(k)} = 1$:

- Compute $\hat{d}^{(k)}$ according to (5.62)
- If $\widetilde{L}\left(x^{(k)} + d^{(k)} + \hat{d}^{(k)}, \mu^{(k)}\right) \leq \widetilde{L}\left(x^{(k)}, \mu^{(k)}\right) + \eta\alpha^{(k)} D\left(\widetilde{L}\left(x^{(k)}, \mu^{(k)}\right); d^{(k)}\right)$:

  Set $x^{(k+1)} := x^{(k)} + d^{(k)} + \hat{d}^{(k)}$ and terminate.

  Else: Choose new $\alpha^{(k)} \in [\tau_1\alpha^{(k)}, \tau_2\alpha^{(k)}]$

Else: Choose new $\alpha^{(k)} \in [\tau_1\alpha^{(k)}, \tau_2\alpha^{(k)}]$

Output: $x^{(k+1)}$ — Enhanced solution

Within this algorithm, the second–order correction term is neglected if no reduction in the merit function is achieved. In particular, no intermediate solutions along $d^{(k)} + \hat{d}^{(k)}$ are searched since there is no guarantee for a decrease in the merit function along this direction.
In practice, this strategy turns out to be very effective and the additional computing costs are compensated by the improved efficiency and robustness.

In many cases, the decrease along the merit function is unstable, i.e. due to round–off errors, calculation errors in the gradient approximations or other error sources. To compensate these effects, a nonmonotone strategy can be used. In contrast to the second–order correction, these algorithms allow for an increase of the merit function for a certain amount of steps, typically 5 or 8. For reasons of simplicity, we allow for one increase only before a sufficient decrease must be attained. The hope is for the subsequent steps to more than compensate the temporary increase in the merit function.
Similar to the second–order correction algorithm, the so called *watchdog* algorithm is straight forward:

Algorithm 5.52 (Nonmonotone Strategy – Watchdog Algorithm)

Input: $x^{(k)}$ — Current optimization vector
$d^{(k)}$ — Search direction
$\eta \in (0, 0.5)$ — Decrease parameter
$\alpha^{(k)}$ — Step length parameter

While not terminated

If $\widetilde{L}\left(x^{(k)} + d^{(k)}, \mu^{(k)}\right) \leq \widetilde{L}\left(x^{(k)}, \mu^{(k)}\right) + \eta D\left(\widetilde{L}\left(x^{(k)}, \mu^{(k)}\right); d^{(k)}\right)$:

- Set $x^{(k+1)} := x^{(k)} + \alpha^{(k)} d^{(k)}$ and terminate

Else:

- Compute search direction $d^{(k+1)}$ from $x^{(k+1)}$
- Find $\alpha^{(k+1)}$ such that

$$\widetilde{L}\left(x^{(k+1)} + \alpha^{(k+1)} d^{(k+1)}, \mu^{(k)}\right) \leq \widetilde{L}\left(x^{(k+1)}, \mu^{(k)}\right) + \eta \alpha^{(k+1)} D\left(\widetilde{L}\left(x^{(k+1)}, \mu^{(k)}\right); d^{(k+1)}\right)$$

  and set $x^{(k+2)} := x^{(k+1)} + \alpha^{(k+1)} d^{(k+1)}$
- If $\widetilde{L}\left(x^{(k+1)}, \mu^{(k)}\right) \leq \widetilde{L}\left(x^{(k)}, \mu^{(k)}\right)$
  or $\widetilde{L}\left(x^{(k+2)}, \mu^{(k)}\right) \leq \widetilde{L}\left(x^{(k)}, \mu^{(k)}\right) + \eta D\left(\widetilde{L}\left(x^{(k)}, \mu^{(k)}\right); d^{(k)}\right)$:
  Set $k := k + 1$ and terminate
  Else if $\widetilde{L}\left(x^{(k+2)}, \mu^{(k)}\right) > \widetilde{L}\left(x^{(k)}, \mu^{(k)}\right)$:
  Find $\alpha^{(k)}$ such that

$$\widetilde{L}\left(x^{(k)} + \alpha^{(k)} d^{(k)}, \mu^{(k)}\right) \leq \widetilde{L}\left(x^{(k)}, \mu^{(k)}\right) + \eta \alpha^{(k)} D\left(\widetilde{L}\left(x^{(k)}, \mu^{(k)}\right); d^{(k)}\right)$$

  holds, set $x^{(k+3)} := x^{(k)} + \alpha^{(k)} d^{(k)}$, $k := k + 2$ and terminate
  Else
  Find $\alpha^{(k+2)}$ such that $\widetilde{L}\left(x^{(k+2)} + \alpha^{(k+2)} d^{(k+2)}, \mu^{(k)}\right) \leq \widetilde{L}\left(x^{(k+2)}, \mu^{(k)}\right) +$
  $\eta \alpha^{(k+2)} D\left(\widetilde{L}\left(x^{(k+2)}, \mu^{(k)}\right); d^{(k+2)}\right)$ holds,
  set $x^{(k+3)} := x^{(k+2)} + \alpha^{(k+2)} d^{(k+2)}$, $k := k + 2$ and terminate

Output: $x^{(k+1)}$ — Enhanced solution

Note that both algorithms compute the step length $\alpha^{(k)}$ without solving the one–dimensional optimization problem along the search direction as in *penalty methods* and hence their computing costs are significantly lower. For further details on the mentioned penalty methods we refer to [19, 64, 83].

5.2.4.3 Inconsistent Linearization

Since we are using a first order Taylor approximation of the Jacobian of the constraints, deviations from the manifold defined by those constraints may occur and need to be treated, e.g. by projecting the resulting solution such that the constraints are satisfied. One possibility to circumvent this difficulty is to state the problem NLP as a so called l_1–penalty problem

$$\begin{aligned} \text{Minimize} \quad & F(x) + \mu^{(k)} \sum_{i \in \mathcal{E}} (v_i + w_i) + \mu^{(k)} \sum_{i \in \mathcal{I}} t_i \qquad \text{over all } (x, v, w, t) \in \mathbb{R}^{n+2r_g+r_h} \\ \text{ST.} \quad & G_i(x) = v_i - w_i, \qquad i \in \mathcal{E} \\ & H_i(x) \geq -t_i, \qquad i \in \mathcal{I} \\ & v, w, t \geq 0 \end{aligned}$$

Note that there always exists a solution to this problem which allows us to simultaneously keep track of the feasibility and the optimality issue, see also [64] for details. However, as shown in [175], the solution corresponds to the solution of our basic problem only if certain additional conditions are fulfilled:

Theorem 5.53
Suppose that $x^\star$ is a solution of the l_1–penalty problem for all $\mu \geq \overline{\mu} > 0$. If $x^\star$ is feasible for the problem NLP then it satisfies the KKT conditions of Theorem 5.18 for the problem NLP.

5.2.4.4 Hessian Quasi–Newton Approximation

Within the proposed SQP algorithms the calculation of the Hessian of the cost function causes large computing costs. The fundamental idea to reduce this cost is to replace the computation of $\nabla^2_{xx} L(x^{(k)}, \lambda^{(k)})$ by an approximation $B^{(k)}$ and was first proposed in [48]. To this end, we define

$$Q^{(k)}(d^{(k)}) := F(x^{(k)}) + \nabla_x L(x^{(k)}, \lambda^{(k)})^\top d^{(k)} + \frac{1}{2} d^{(k)\top} B^{(k)} d^{(k)}$$

and match the gradients of $Q(\cdot)$ at step $k+1$ to the last two iterates $x^{(k)}$ and $x^{(k-1)}$, i.e.

$$\nabla_d Q^{(k)}(-\alpha^{(k-1)} d^{(k-1)}) = \nabla_x L(x^{(k)}, \lambda^{(k)}) - B^{(k)} \alpha^{(k-1)} d^{(k-1)} = \nabla_x L(x^{(k-1)}, \lambda^{(k-1)}).$$

Setting $s^{(k-1)} := x^{(k)} - x^{(k-1)}$ and $y^{(k-1)} := \nabla_x L(x^{(k)}, \lambda^{(k)}) - \nabla_x L(x^{(k-1)}, \lambda^{(k-1)})$ this can be rearranged to the so called *secant equation*

$$B^{(k)} s^{(k-1)} = y^{(k-1)}. \tag{5.63}$$

Note that this gives us a positive definite matrix $B^{(k)}$ if and only if the *curvature condition*

$$s^{(k-1)\top} y^{(k-1)} > 0 \tag{5.64}$$

holds. Again, we start with the convex case of the optimization problem and try to render $B^{(k)}$ to be positive definite and symmetric. Hence, we obtain $n(n+1)/2$ degrees of freedom within this matrix while the curvature condition (5.64) states only n conditions. The same holds true if we consider the matrix $H^{(k)} := {B^{(k)}}^{-1}$ and hence we are looking for the matrix H which is closest to the matrix $H^{(k-1)}$. This reveals the optimization problem

$$\begin{aligned} \text{Minimize} \quad & \left\| H - H^{(k-1)} \right\| \qquad \text{over all } H \in \mathbb{R}^{n \times n} \\ \text{ST.} \quad & H = H^\top \\ & H y^{(k-1)} = s^{(k-1)} \end{aligned}$$

If we consider B and $B^{(k-1)}$ in this problem, we obtain the *DFP (Davidon–Fletcher–Powell) updating formula*, see [48, 66]. Here, we present the *BFGS (Broydon–Fletcher–Goldfarb–Shanno) formula* [33, 63, 81, 206] since it shows better performance.
Within the stated minimization problem we choose the weighted Frobenius norm $\|A\|_W \equiv \|W^{1/2}AW^{1/2}\|_F$ where $\|\cdot\|_F$ is defined as $\|A\|_F^2 = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}^2$. This allows for an easy and scale–invariant optimization. The weighting matrix W is set via

$$W := \left[\int_0^1 \nabla_{dx}^2 L(x^{(k-1)} + \tau\alpha^{(k-1)}d^{(k-1)}, \lambda^{(k-1)})d\tau\right]^{-1}, \tag{5.65}$$

that is the *average Hessian*. Hence, the weighted Frobenius norm is non–dimensional and the unique solution of the optimization problem

$$H^{(k)} := \left(\mathrm{Id} - \frac{s^{(k-1)}y^{(k-1)\top}}{y^{(k-1)\top}s^{(k-1)}}\right) H^{(k-1)} \left(\mathrm{Id} - \frac{y^{(k-1)}s^{(k-1)\top}}{y^{(k-1)\top}s^{(k-1)}}\right) + \frac{s^{(k-1)}s^{(k-1)\top}}{y^{(k-1)\top}s^{(k-1)}} \tag{5.66}$$

is independent of the units of the problem. Note that any norm and any matrix W satisfying $Wy^{(k-1)} = s^{(k-1)}$ can be chosen instead of the stated ones. The resulting BFGS formula (5.66), however, has shown to be computationally efficient and effectively self–correcting with respect to bad estimates of the Hessian. To start the iteration, the initial guess $H^{(0)}$ is usually taken as a multiple of the identity matrix. Moreover, a scaling can be used before the first update of the approximation is done, i.e. by defining

$$H^{(1)} := \frac{y^{(0)\top}s^{(0)}}{y^{(0)\top}y^{(0)}}\mathrm{Id} \tag{5.67}$$

which aims at an eigenvalue approximation of $\nabla_{xx}^2 F(x^{(0)})$.
The BFGS update can then be summarized as follows:

Algorithm 5.54 (BFGS Update)

Input:	$x^{(k)}$	—	Current optimization vector
	$x^{(k-1)}$	—	Preceding optimization vector
	$\nabla_x F(x^{(k)})$	—	Current gradient of the cost function
	$\nabla_x F(x^{(k-1)})$	—	Preceding gradientof the cost function
	$H^{(k-1)}$	—	Preceding inverse of the Hessian

If $k = 0$:

- Set $H^{(0)} := \mathrm{Id}$

Else:

- Set $s^{(k-1)} := x^{(k)} - x^{(k-1)}$ and $y^{(k-1)} := \nabla_x L(x^{(k)}, \lambda^{(k)}) - \nabla_x L(x^{(k-1)}, \lambda^{(k-1)})$
- If $k = 1$:

  Compute $H^{(1)}$ according to (5.67)
- Compute $H^{(k)}$ according to (5.66)

Output:	$H^{(k)}$	—	Updated inverse of the Hessian

Remark 5.55
There also exists a version of the BFGS formula which uses the approximation of the Hessian $B^{(k)}$ instead of $H^{(k)}$ via

$$B^{(k)} = B^{(k-1)} - \frac{B^{(k-1)} d^{(k-1)} d^{(k-1)\top} B^{(k-1)}}{d^{(k-1)\top} B^{(k-1)} d^{(k-1)}} + \frac{y^{(k-1)} y^{(k-1)\top}}{y^{(k-1)\top} d^{(k-1)}} \tag{5.68}$$

If this formula is used within Algorithm 5.54, then the computation of the search direction via $B^{(k)} d^{(k)} = -\nabla_x F(x^{(k)}, \lambda^{(k)})$ can be accelerated by using updates on the Cholesky factors of $B^{(k)}$.

If we replace the correct Hessian by the BFGS approximation in our SQP algorithms 5.37 and 5.38, we can still guarantee convergence. For a proof we refer to [35, 182].

Theorem 5.56 (Convergence)
Consider $H^{(k)}$ to be positive definite and symmetric for all $k \in \mathbb{N}_0$. Moreover, we consider $x^{(0)} \in \mathbb{R}^n$ such that the level set

$$\mathcal{L} := \{x \in \mathbb{R}^n \mid F(x) \leq F(x^{(0)})\}$$

is convex and there exists positive constants c_1, c_2 such that

$$c_1 \|x\|^2 \leq x^\top \nabla^2_{xx} L(\hat{x}, \hat{\lambda}) x \leq c_2 \|x\|^2$$

holds for all $x \in \mathbb{R}^n$ and $\hat{x} \in \mathcal{L}$ with corresponding Lagrange multiplier $\hat{\lambda}$. If the Hessian in Algorithms 5.37 and 5.38 is replaced by the BFGS update according to Algorithm 5.54 then the resulting sequences $(x^{(k)})_{k \in \mathbb{N}_0}$ converge to a minimizer $x^\star$ of $F(\cdot)$.
If additionally the Hessian is Lipschitz–continuous in a neighborhood of this minimizer $x^\star$, then the convergence rate is superlinear.

In order to guarantee

$$H^{(k)^{-1}} = B^{(k)} = B^{(k-1)} - \frac{B^{(k-1)} s^{(k-1)} s^{(k-1)\top} B^{(k-1)}}{s^{(k-1)\top} B^{(k-1)} s^{(k-1)}} + \frac{y^{(k-1)} y^{(k-1)\top}}{s^{(k-1)\top} y^{(k-1)}}$$

to stay positive definite during the iteration we use an interpolation of the two latest matrices $B^{(k-1)}$ and $B^{(k)}$. To this end, we define

$$r^{(k-1)} = \theta^{(k-1)} y^{(k-1)} + (1 - \theta^{(k-1)}) B^{(k-1)} s^{(k-1)}$$

with safeguarded scalar

$$\theta^{(k-1)} = \begin{cases} 1, & \text{if } \frac{s^{(k-1)\top} y^{(k-1)}}{s^{(k-1)\top} B^{(k-1)} s^{(k-1)}} \geq 0.2 \\ \frac{0.8 s^{(k-1)\top} B^{(k-1)} s^{(k-1)}}{s^{(k-1)\top} B^{(k-1)} s^{(k-1)} - s^{(k-1)\top} y^{(k-1)}}, & \text{else} \end{cases} \tag{5.69}$$

Within the BFGS formula (5.66) this results in replacing the gradient difference $y^{(k-1)}$ by the interpolant $r^{(k-1)}$, i.e.

$$B^{(k)} = B^{(k-1)} - \frac{B^{(k-1)} s^{(k-1)} s^{(k-1)\top} B^{(k-1)}}{s^{(k-1)\top} B^{(k-1)} s^{(k-1)}} + \frac{r^{(k-1)} r^{(k-1)\top}}{s^{(k-1)\top} r^{(k-1)}}. \tag{5.70}$$

Moreover, we obtain $B^{(k)}$ and hence $H^{(k)}$ to be positive definite since $s^{(k-1)\top} y^{(k-1)} = 0.2 s^{(k-1)\top} B^{(k-1)} s^{(k-1)} > 0$ holds due to the definition of $\theta^{(k-1)}$ in (5.69), see also [184].

Still, if the Hessian of the Lagrangian is not positive definite, the damped BFGS update ultimately fails. If no update can be found which is positive definite, then the update should be skipped. To obtain a skipping criterion, the curvature condition (5.64) is considered and whenever $S^{(k-1)\top}y^{(k-1)} < \sigma$ with $\sigma = \alpha^{(k-1)}(1-\nu)d^{(k-1)\top}B^{(k-1)}d^{(k-1)}$ holds for a preassigned constant $\nu \in (0,1)$, then the curvature is considered to be not sufficiently positive and the update is skipped. Not updating the approximation of the Hessian, however, leads to poor performance since $B^{(k)}$ cannot capture important curvature information.

Remark 5.57
Another possiblity to deal with an indefinite Hessian is the so called SR1 method, see e.g. [44] for its use in line search and [128] for a trust–region implementation. Since both `NLPQLP` *and* `e04wdc` *use the BFGS update, we do not present any details on the SR1 method here.*

Remark 5.58
The computational effort can be reduced even further by applying reduced Hessian Quasi–Newton approximations. These methods only consider the orthogonal subspace of the range space of $A^{(k)}$ since $d^{(k)}$ is influenced by $\nabla^2_{xx}L(x^{(k)}, \lambda^{(k)})$ on this space only, see e.g. [21] for further details.

5.2.5 Line Search SQP

The implementation issues shown in Section 5.2.4 indicate that there exists a large number of possible implementations of SQP algorithms. Here, we first consider the so called *line search SQP algorithm* which, in a basic version, contains the following characteristic steps:

Algorithm 5.59 (Line Search SQP Algorithm)
Input: $(x^{(0)}, \lambda^{(0)})$ — Initial pair
$\eta \in (0, 0.5)$ — Decrease parameter

Set $k = 0$

While convergence test is not satisfied

(1) Evaluate $F(x^{(k)})$ and $A(x^{(k)})$
(2) Compute Hessian
(3) Evalutate $\nabla_x F(x^{(k)})$ and $\nabla_x A(x^{(k)})$
(4) Compute $d^{(k)}$ by solving the problem QP with corresponding multiplier $\hat{\lambda}^{(k)}$
(5) Set $d_\lambda := \hat{\lambda} - \lambda^{(k)}$
(6) Choose $\mu^{(k)}$ such that (5.61) holds for $\sigma = 1$
(7) Find $\alpha^{(k)}$ satisfying

$$\widetilde{L}\left(x^{(k)} + \alpha^{(k)}d^{(k)}, \mu^{(k)}\right) \leq \widetilde{L}\left(x^{(k)}, \mu^{(k)}\right) + \eta\alpha^{(k)}D\left(\widetilde{L}\left(x^{(k)}, \mu^{(k)}\right); d^{(k)}\right)$$

(8) Set $x^{(k+1)} := x^{(k)} + \alpha^{(k)}d^{(k)}$, $\lambda^{(k+1)} := \lambda^{(k)} + \alpha^{(k)}d_\lambda$

Output: $x^\star$ — Optimal solution

A theoretical overview for line search SQP can be found e.g. in [64,175] whereas ideas of the algorithm and convergence results are, among others, described in [111,183].
Within this basic version, we can utilize the algorithms of Section 5.2.4 to speed up the optimization process. In particular, we can implement Step (2) as an update formula, see Section 5.2.4.4. Steps (7) and (8) can be replaced by either the Watchdog Algorithm 5.52 or the Second–order Correction Algorithm 5.51. Moreover, if the linearization of the constraints in Step (3) causes an inconsistency we can switch to a l_1–penalty problem formulation, see Section 5.2.4.3.

Remark 5.60
Due to the Quasi–Newton nature of the SQP methods, the gradients are not computed completely in every iteration of the while loop, see also Section 6.3. Hence, the computational effort is reduced further. Note that both implementations `NLPQLP` *and* `e04wdc` *offer this possibility.*

5.2.6 Trust–Region SQP

In contrast to line search algorithms, the class of trust–region methods does not require the Hessian or its approximation to be positive definite. In particular, they show good progress even in presence of Hessian and Jacobian singularities. Trust–region variants are studied, e.g., in [36,225], see also [28,85] for literature surveys and [234] for a convergence analysis.
The trust–region problem considers the standard QP problem including an additional regional constraint $\Delta^{(k)}$:

$$\begin{array}{lll} \text{Minimize} & F(x^{(k)}) + \nabla_x F(x^{(k)})^\top d^{(k)} + \dfrac{1}{2} d^{(k)\top} \nabla^2_{xx} L(x^{(k)}, \lambda^{(k)}) d^{(k)} & \text{over all } d^{(k)} \in \mathbb{R}^n \\ \text{ST.} & G_i(x^{(k)}) + \nabla_x G_i(x^{(k)})^\top d^{(k)} = 0, \qquad i \in \mathcal{E} & \\ & H_i(x^{(k)}) + \nabla_x H_i(x^{(k)})^\top d^{(k)} \geq 0, \qquad i \in \mathcal{I} & \\ & \|d^{(k)}\| \leq \Delta^{(k)} & \end{array}$$

This additional constraint may cause the problem to be infeasible even if there exists a solution satisfying all other constraints. Yet, we have to keep in mind that we are only dealing with linearizations of the constraints and these can only be trusted in a certain neighborhood of the linearization point. Hence, there is no reason to satisfy the linearized constraints exactly except the trusted region permits it. Instead, our aim has to be the improvement of the feasibility and at the same time the improvement of the solution of the problem NLP. Here, we give a short glance at relaxation and l_1–penalty methods. Since none of our used SQP solver uses filter methods, we skip this topic here and refer to [65] and references therein.
The relaxation approach considers the problem ECQP and modifies the constraints by a relaxation vector $r^{(k)}$, that is

$$A_i(x^{(k)}) + \nabla_x A_i(x^{(k)})^\top d^{(k)} = r_i^{(k)},$$

and by adding the trust–region constraint. The vector $r^{(k)}$ is chosen to allow for a consistent solution even for a reduced radius of the trusted region. A radius $r^{(k)}$ guaranteeing these properties can be obtained by setting

$$r^{(k)} := \nabla_x A(x^{(k)}) v^{(k)} + A(x^{(k)}) \tag{5.71}$$

where $v^{(k)}$ is the minimizing solution of

$$\begin{aligned} \text{Minimize} \quad & \|\nabla_x A(x^{(k)})v^{(k)} + A(x^{(k)})\| \qquad \forall v^{(k)} \in \mathbb{R}^n \\ \text{ST.} \quad & \|v^{(k)}\|_2 \leq 0.8\Delta^{(k)} \end{aligned} \tag{5.72}$$

The safeguard factor 0.8 guarantees existence of a consistent solution of the relaxed problem. Hence, the resulting problem exhibits consistent constraints and can be solved by any of the algorithms mentioned in Section 5.2.2. The merit function is chosen according to the Euclidean setting of the additional trust–region constrained,

$$\widetilde{L}(x^{(k)}, \mu^{(k)}) := F(x^{(k)}) + \mu^{(k)} \|\nabla_x A(x) d^{(k)} + A(x^{(k)})\|_2$$

Furthermore, the ratio

$$\rho^{(k)} := \frac{\widetilde{L}(x^{(k)}, \mu^{(k)} - \widetilde{L}(x^{(k)} + d^{(k)}, \mu^{(k)}}{Q_{\mu^{(k)}}(0) - Q_{\mu^{(k)}}} \tag{5.73}$$

concerning the difference in the merit function (characterizing the decrease) and the augmented quadratic model problem $Q_{\mu^{(k)}}(d^{(k)})$ as defined in Section 5.2.4.1 considering the Euklidean norm (characterizing the feasibility) is used to decide whether a step is accepted.
Different to the relaxation approach, the l_1–penalty methods consider and modify the problem QP. As shown in Section 5.2.4.3, two additional terms are added to the cost functional such that the optimization similarly keeps track of the feasibility aspect. Moreover, a trust–region constraint is taken into account. Hence, the l_1–penalty problem is written as follows:

$$\begin{aligned} \text{Minimize} \quad & F(x^{(k)}) + \nabla_x F(x^{(k)})^\top d^{(k)} + \frac{1}{2} d^{(k)\top} \nabla^2_{xx} L(x^{(k)}, \lambda^{(k)}) d^{(k)} \\ & + \mu \sum_{i \in \mathcal{E}} (v_i + w_i) + \mu \sum_{i \in \mathcal{I}} t_i \qquad \text{over all } (d^{(k)}, v, w, t) \in \mathbb{R}^{n+2r_g+r_h} \\ \text{ST.} \quad & G_i(x^{(k)}) + \nabla_x G_i(x^{(k)})^\top d^{(k)} = v_i - w_i, \qquad i \in \mathcal{E} \\ & H_i(x^{(k)}) + \nabla_x H_i(x^{(k)})^\top d^{(k)} \geq -t_i, \qquad i \in \mathcal{I} \\ & v, w, t \geq 0 \\ & \|d^{(k)}\| \leq \Delta^{(k)} \end{aligned}$$

Similar to the relaxation approach we use the ratio $\rho^{(k)}$ from (5.73) to determine whether a step is accepted or the trusted region needs to be adapted. Here, however, the l_1 function

$$\widetilde{L}(x^{(k)}, \mu^{(k)}) := F(x^{(k)}) + \mu \sum_{i \in \mathcal{E}} |G_i(x^{(k)})| + \mu \sum_{i \in \mathcal{I}} \max\{0, -H_i(x^{(k)})\}$$

is used as merit function within this setting.
Summing up, a trust–region algorithm takes the following general form:

Algorithm 5.61 (Trust–Region SQP Algorithm)

Input:	$(x^{(0)}, \lambda^{(0)})$	—	Initial pair
	$\Delta^{(0)}$	—	Initial trust region radius
	$\varepsilon > 0$	—	Optimality parameter
	$\eta \in (0, 1)$	—	Acceptance parameter
	$\gamma \in (0, 1)$	—	Contraction parameter

Set $k = 0$

While convergence test is not satisfied

(1) Evaluate $A(x^{(k)})$ and $\nabla_x A(x^{(k)})$

(2) Solve problem (5.72) for $v^{(k)}$

(3) Compute $r^{(k)}$ from (5.71)

(4) Compute Hessian

(5) Compute $d^{(k)}$ by solving the problem QP with additional constraint $\|d^{(k)}\| \leq \Delta^{(k)}$ with corresponding multiplier $\hat{\lambda}^{(k)}$

(6) Evalutate $F(x^{(k)})$ and $\nabla_x F(x^{(k)})$

(7) If $\|\nabla_x F(x^{(k)}) - \nabla_x A(x^{(k)})^\top \hat{\lambda}^{(k)}\| < \varepsilon$ and $\|A(x^{(k)})\|_\infty < \varepsilon$:

Set $x^\star := x^{(k)}$ and terminate

(8) Choose $\mu^{(k)}$ such that (5.61) holds

(9) Compute $\rho^{(k)}$ according to (5.73)

(10) If $\rho^{(k)} > \eta$

Set $x^{(k+1)} := x^{(k)} + d^{(k)}$ and $\Delta^{(k+1)}$ such that $\Delta^{(k+1)} \geq \Delta^{(k)}$ holds

Else

Set $x^{(k+1)} := x^{(k)}$ and $\Delta^{(k+1)}$ such that $\Delta^{(k+1)} \leq \gamma \|d^{(k)}\|$ holds

Output: $x^\star$ — Optimal solution

Remark 5.62
Accoring to numerical comparisons, trust–region SQP variant and line search methods are equally efficient in terms of the number of function and gradient evaluations, see e.g. [56].

Remark 5.63
For our receding horizon controller setting, the line search variant of Algorithm 5.59, i.e. `NLPQLP`*, is in most cases superior concerning computing times. The reason for this is simple but can be easily overlooked: While in Algorithm 5.59 the computation of $F(x^{(k)})$ and $A(x^{(k)})$ as well as $\nabla_x F(x^{(k)})$ and $\nabla_x A(x^{(k)})$ are performed in Steps (1) and (3), in Algorithm 5.61 we compute $A(x^{(k)})$ and $\nabla_x A(x^{(k)})$ in Step (1) and $F(x^{(k)})$ and $\nabla_x F(x^{(k)})$ in Step (6). In standard nonlinear optimization, this is irrelevant. However, the structure of the receding horizon control problem offers synergy effects in evaluating the underlying dynamic of the system in the first case since variations in the control vector can be used in the difference quotient to compute both $\nabla_x F(x^{(k)})$ and $\nabla_x A(x^{(k)})$.*

5.2.7 Classical Convergence Results

Here, we are not going to state the whole convergence theory which has been derived for SQP methods over the last 25 years. For a rather general convergence result we refer to the survey [45]. Instead, we present a selection of classical convergence result. To this end, we now state a condition under which a standard SQP method in any case identifies a KKT point of the nonlinear program.

Assumption 5.64
Consider an SQP algorithm to satisfy the following assumptions:

(1) The SQP method computes the search direction by solving the problem QP.

(2) Within the cost function of problem QP the Hessian $\nabla^2_{xx} L(x^{(k)}, \lambda^{(k)})$ is replaced by some symmetric, positive definite and bounded approximation $B^{(k)}$.

(3) The iterate $x^{(k+1)} := x^{(k)} + \alpha^{(k)} d^{(k)}$ is defined according to Algorithm 5.51.

(4) The problem QP is feasible in each step and exhibits a bounded solution $d^{(k)}$.

(5) The penalty parameter $\mu^{(k)}$ is fixed for all k and sufficiently large.

(6) $F(\cdot)$, $G(\cdot)$ and $H(\cdot)$ are continuously differentiable.

(7) The multipliers $\lambda^{(k)}$ are bounded.

Then the following convergence result proved in [185] applies:

Theorem 5.65 (Global Convergence)
Consider Assumptions 5.64 to hold and the sequences $\left(x^{(k)}\right)_{k\in\mathbb{N}_0}$ *and* $\left(x^{(k)} + d^{(k)}\right)_{k\in\mathbb{N}_0}$ *to remain in a closed, bounded and convex subset of* $\mathbb{R}^n$*. Then all limits points of the sequence* $\left(x^{(k)}\right)_{k\in\mathbb{N}_0}$ *are KKT points of the nonlinear problem NLP.*

Next, instead of establishing global results, we aim at obtaining local convergence of the SQP methods. To this end, we consider the following situation for the standard SQP Algorithm 5.37, i.e. without inequality constraints:

Assumption 5.66
Consider $x^\star$ to be a local solution of problem NLP such that the following conditions hold:

(1) $F(\cdot)$ and $G(\cdot)$ are twice differentiable in a neighborhood of $x^\star$ and their second derivative is Lipschitz continuous.

(2) At $x^\star$ the LICQ holds, see Definition 5.16.

(3) The second–order sufficient conditions hold at $(x^\star, \lambda^\star)$, see Theorem 5.20.

Theorem 5.67 (Quadratic Convergence using Newton Methods)
If Assumptions 5.66 hold and $(x^{(0)}, \lambda^{(0)})$ *is sufficiently close to* $(x^\star, \lambda^\star)$*, then the sequence of pairs* $(x^{(k)}, \lambda^{(k)})$ *given by Algorithm 5.37 converges quadratically to* $(x^\star, \lambda^\star)$*.*

Proof. Since given the stated assumption Algorithm 5.37 is identical to the Newton iteration the assertion follows directly. □

Remark 5.68
This convergence result also holds true for the inequality constrained problem if the working set $\mathcal{W}^\star$ *of the optimal value* $x^\star$ *has been reached (and remains unchanged) and* $H(\cdot)$ *is twice differentiable.*

The following result from [29] allows us to substitute the Hessian of the Lagrangian by an approximation as shown in Section 5.2.4.4 and obtain superlinear convergence.

Theorem 5.69 (Superlinear Convergence using Quasi–Newton Methods)
Suppose Assumptions 5.66 to hold and that the sequence $(x^{(k)})_{k\in\mathbb{N}_0}$ generated by Algorithm 5.37 converges using quasi–Newton approximated Hessians $B^{(k)}$. Then we obtain superlinear convergence of $(x^{(k)})_{k\in\mathbb{N}_0}$ to $x^\star$ if and only if

$$\lim_{k\to\infty} \frac{\left\| P^{(k)} \left(B^{(k)} - \nabla^2_{xx} L(x^\star, \lambda^\star) \right) (x^{(k+1)} - x^{(k)}) \right\|}{\|x^{(k+1)} - x^{(k)}\|} = 0 \tag{5.74}$$

holds for the Hessian approximation $B^{(k)}$ where

$$P^{(k)} := \left[Id - \nabla_x A(x^{(k)})^\top \left[\nabla_x A(x^{(k)}) \nabla_x A(x^{(k)})^\top \right]^{-1} \nabla_x A(x^{(k)}) \right]. \tag{5.75}$$

Hence, we can substitute the quasi–Newton approximation of the Hessian of the cost function $F(\cdot)$ by the BFGS update formula. Due to the results from Section 5.2.4.4 we still obtain a good approximation and convergence of the iterates computed by our scheme. This follows immediately by the positive definiteness of the iterates $B^{(k)}$ according to the damped BFGS update. More formally, we obtain the following result directly from Theorem 5.69 and the properties of the BFGS formula shown in Section 5.2.4.4:

Theorem 5.70 (Superlinear Convergence using BFGS)
Suppose Assumptions 5.66 to hold and $\nabla^2_{xx} L(x^\star, \lambda^\star)$ and $B^{(0)}$ are symmetric and positive definite. If additionally the norm differences $\|x^\star - x^{(0)}\|$ and $\|\nabla^2_{xx} L(x^\star, \lambda^\star) - B^{(0)}\|$ are sufficiently small and (5.74), (5.75) *hold for the BFGS Hessian approximations* (5.70), *then the sequence $(x^{(k)})_{k\in\mathbb{N}_0}$ converges superlinearly to $x^\star$.*

Chapter 6

Numerical Implementation

In order to make the description of the receding horizon control implementation *PCC2*[1] (**P**redictive **C**omputed **C**ontrol 2) intuitively understandable we will consider our standard example, the inverted pendulum, and explain the software package in detail, that is we relate

- *analytical background* shown in Chapters 1 – 4,
- *numerical algorithms* described in Chapters 3 – 5

and show reasons for the chosen way of implementing these methods.
Here, we focus on the implementation of the *PCC2* software, the conceptual background for the chosen class definitions and explain their functionality. Using the examples from Chapter 7, a detailed comparison of the selectable methods is presented in Chapter 8.

In the following Section 6.1 we motivate the structure of our implementation of a receding horizon controller in the *PCC2* package which is divided in three major parts. This corresponds to the topics considered within the subsequent Sections 6.2, 6.3 and 6.4. The setup of the receding horizon controller itself is discussed in Section 6.2. Since the controller automatically discretizes continuous–time problems, this issue is reviewed within this section as well. Moreover, we show three different implementations connecting the optimization routine shown in Section 6.3 and the differential equation solvers as presented in Section 6.4 to this discretization.
Last, within Section 6.5, we give an overview on the *PCC2* package in a tutorial way. To this end, we illustrate the general setup by implementing the inverted pendulum example from Section 7.2 and a receding horizon controller for this example.

6.1 Programming Scheme of PCC2

The numerical implementation of the receding horizon control algorithm is organized hierarchically. The package is separated in three major parts: the *receding horizon controller* itself, *minimization algorithms* and *methods to solve the underlying dynamic of the system*, see Figure 6.1.

[1] Webpage: `http://www.nonlinearmpc.com`

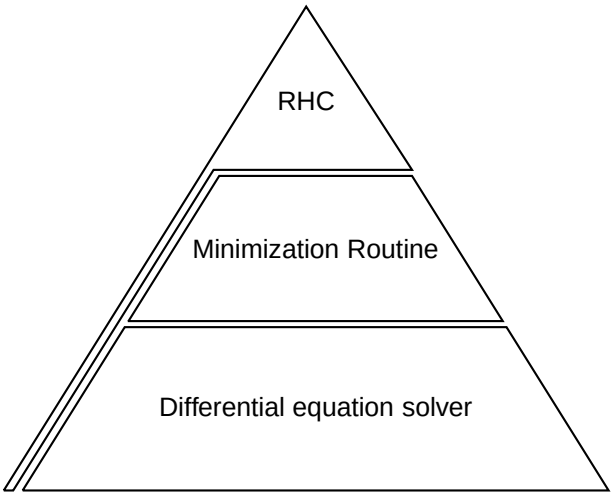


Figure 6.1: Hierarchy of the implemented control algorithm

To this end, several C++ classes and libraries have been implemented. Within this programming structure, the receding horizon controller implementation is designed independently of the minimizer and the differential equation solver. This allows for modular and exchangable subcomponents rendering the program flexible and extendable.
The definition of a control problem is handled outside this hierarchy. To this end, a *prototype model* has been introduced which forces the user to implement necessary functions like the dynamic of the system or the cost functional in a fixed but intuitive way, see e.g. Listings A.1 and A.2. Hence, controller and model are independent classes which allows for reuse of models in different controller setups and vice versa.

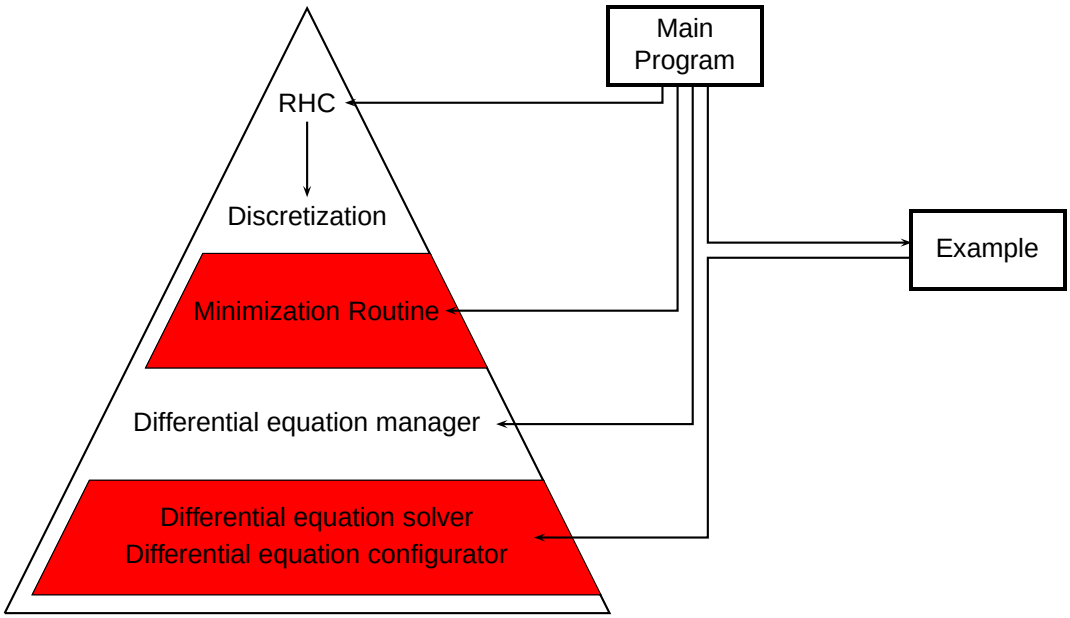


Figure 6.2: Structure of an implemented program

Note that here, motivated by model inherent properties such as the time set, dimensionality or stiffness, the model defines the type of *differential equation solver* to be used and the *configuration of this solver*. Hence, a suitable choice of solver and parameters needs to be obtained only once and can be reused for other controller settings. Still, the class structure allows the main program to change the configuration of the solver if necessary.

An appropriate choice of the *minimization routine* and its parameters, however, is depending on the size of the discretization, i.e. the dimension of the model and its restrictions, but also the horizon length chosen in the controller design. Hence, it seems reasonable to define this component within the main program.
The *discretization* of the optimal control problem is done automatically by the receding horizon controller once the sampling instants are fixed. Hence, no further work has to be done by the user. The connecting link between the discretized problem, the minimization routine and the differential equation solver is established by a so called *differential equation manager*. Since our three implementations differ in terms of speed, accuracy, practicability and reliability, the choice is left to the user.

6.1.1 Class Model

According to the hierarchy of the program as shown in Figure 6.2, an example has to be defined before setting up the receding horizon control problem. To this end, an example class has to be derived from the class `Model` and the user has to provide functions with a strict function header which are given by virtual functions within the class `Model`. For ease of notation and to make our software intuitively understandable, we choose the standard control systems nomenclature in our implementation as well.

6.1.1.1 Constructor / Destructor

The constructor method of an example does not only allocate the internal variables of the class itself, but it also specifies and constructs the `OdeSolve` and `OdeConfig` objects — that is the differential equation solver object and its configuration object — to be used for this particular example. This is convenient since one main program may be used for several examples without having to worry about a suitable choice of the ODE solver routine. In particular, the size and stiffness properties of every single example are different and the ODE solver should be selected carefully. For further details on the correlation between size, stiffness and computing times of the implemented routines we refer to Sections 8.1.1 and 8.1.2.
The `OdeConfig` object enables the user to specify all internal variables of the differential equation solver object `OdeSolve`, see also Section 6.4.2. Most importantly, *absolute and relative tolerance* can (and should) be set here. For their impact on computing times see Section 8.1.3.
Last, parameters of the control system can be set within the constructor. This enables us to defined parametrized example classes, e.g. to analyze sensitivity or robustness aspects. For an illustrative example and the usage of the constructor, see Section 6.5.1.
Within the destructor, the `OdeSolve` and `OdeConfig` objects as well as all internally allocated variables have to be deleted.

6.1.1.2 Defining the Control Problem

In order to use an example to setup a receding horizon control using the package *PCC2*, it has to meet the standard form defined by the class `Model`. In particular, the cost functional, the dynamic of the system, the box constraints as well as the constraint function need to be defined:

Function	Description
`dglfunction`	Dynamic of the system

Function	Description
`objectiveFunction`	Integral part of the cost functional
`pointcostFunction`	Discrete part of the cost functional
`getObjectiveWeight`	Weighting between both parts of the cost functional
`restrictionFunction`	Constraint functions
`getControlBounds`	Box constraints of the control u
`getModelBounds`	Box constraints of the state x
`getDefaultState`	Default initial value
`getDefaultControl`	Default value of the control
`getMaxShootingDataLength`	Maximal number of multiple shooting points
`getShootingDataLength`	Number of used multiple shooting points
`getShootingDataInfo`	Specifying multiple shooting point positions
`eventBeforeMPC`	Update possibility before next RHC iterate

Table 6.1: Required functions within a class `Model` object

The *cost functional* can exhibit both integral and discrete parts

$$J_N(x, u_N) = \sum_{i=0}^{N-1} \int_{t_i}^{t_{i+1}} L\left(x_{u_N}(\tau, x), u_N(x), \tau\right) d\tau + \sum_{i=0}^{N-1} l\left(x_{u_N}(t_i, x), u_N(x), t_i\right) + F(x_{u_N}(t_N, x))$$

which have to be implemented in the corresponding functions `objectiveFunction` and `pointcostFunction`. The discrete part of the cost functional implementation can be used to simulate a Mayer–term, but it is also possible to introduce a weighting of some or all points within the time grid `t`. The first part implements an integral form, i.e. a Lagrangian cost functional, if the underlying dynamic of the system is given by a differential equation. In the discrete–time case, i.e. if the system is given by a difference equation, this term is a sum of equally weighted costs caused by the state of the system at the grid points. These two parts of the cost functional can be used at the same time. To this end, a *weighting* between both functions has to be defined within the method `getObjectiveWeight`, see also Figure 6.3.

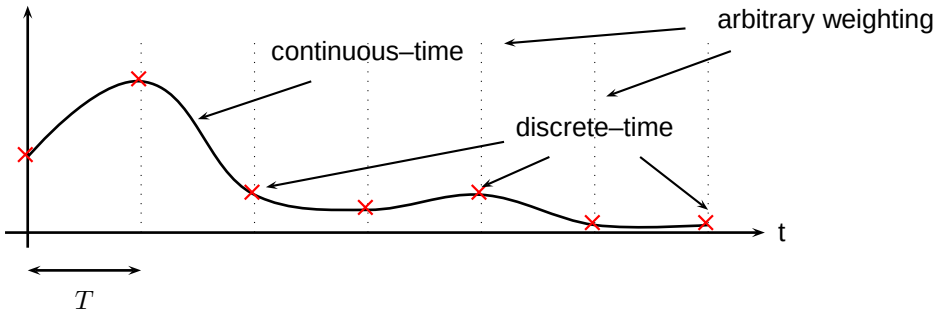


Figure 6.3: Weighting of the cost functionals

The non–constant *restrictions* are defined using `restrictionFunction` where restrictions at every grid point can be implemented. These restrictions have to meet the standard

form in nonlinear optimization $f(t, x, u) \geq 0$, see also Definition 2.2, and may consist of pure state constraints as well as mixed state and control constraints and can also be time–dependent.

Remark 6.1
Note that without adapting the time grid itself violations of these restrictions cannot be recognized and avoided in between two gridpoints t_i and t_{i+1}.

The methods `getControlBounds` and `getModelBounds` are implemented to supply the *box constraints* for state and control variables by defining two vectors in each method.
In order to initialize the optimal control problem in the main program, the user is required to set *initial conditions* for the state. Again, this has to be done in the main program, either manually or by utilizing the *default state* implementation `getDefaultState`.

Remark 6.2
There exists no method within the class `Model` *to set initial conditions automatically. Representing a model, that is a mathematical description of the behaviour of an application, it is independent of the actual state. Hence, measurement data concerning the current state of the system must be provided to the controller by the external program, i.e. the main program.*

Similarly, a default vector of the *control* can be provided to the controller. To this end, the method `getDefaultControl` of the class `Model` may be used. Last, *multiple shooting nodes* can be specified using the three methods `GetMaxShootingDataLength`, `GetShootingDataLength` and `getShootingDataInfo`. The first two get–routines are necessary to distinguish between the number of actually used shooting nodes and the maximal number of shooting nodes. The second one is required by the receding horizon control algorithm to allocate the maximal necessary memory. The last routine reveals for which state of the system and time instant (as multiples of the sampling time) a shooting node is set.
Setting the values of each multiple shooting node may be repeated before restarting the optimization within the RHC algorithm. For this purpose, the method `eventBeforeMPC` can be used and has to be modified to fit this task. Note that `eventBeforeMPC` also allows for other changes within the model, e.g. switching the right hand side of the differential equation or the cost functional.

6.1.2 Setup of the Libraries

As mentioned before, the software package is structured in several C++ libraries. This allows us to update parts of the software without having to recompile the whole package. Each package is equipped with a version checking mechanism, i.e. compilation and installation of parts of the package can be carried out only if all requirements of the update are satisfied. Currently, the following libraries are used:

Name	Content
libbtipopt	Wraparound for the minimizer IpOpt[1]
libbtmbutils	Collection of useful tools like automated storing of data, time measurements and exception debugging manager
libminprog	Parenting library for all minimization routines
libmpc2	Main library containing all routines necessary within the receding horizon control algorithm, but also all differential equation manager classes and the discretization class

Name	Content
libodesol2	Wraparound of all implemented ODE solvers[2]
libsqpf	Wraparound for the minimizer NLPQLP[3]
libsqpnagc	Wraparound for the minimizer e04wdc from the NAGC library[4]

Table 6.2: Libraries contained within the *PCC2* package

Table 6.3 displays the classes of the different libraries.

Library	Class
libbtmbutils	BitField
	Code
	DataItem
	DataQueue
	DebugMaster
	DebugMasterFile
	DebugMasterFileTextStream
	DebugClient
	OptimalDiff
	RTClock
	SaveData
	Uuid
libmpc2	CacheOdeManager
	ControlClipboard
	ControlSequence
	Cycle
	Discretization
	IOController
	IOdeManager
	IOInterface
	Model
	MPC
	SimpleOdeManager
	SuboptimalityMPC
	SyncOdeManager

Library	Class
libbtipopt	BtIpopt
	NLPProblem
libminprog	GaussNewton
	MinProg
libodesol2	DoPri
	DoPri5
	DoPri853
	DoPriConfig
	Euler
	OdeConfig
	OdeFunction
	OdeSolEx
	OdeSolveFirst
	OdeSolve
	Radau
	Radau5
	Radau5913
	Radauconfig
	Radau5913Config
	RecurseSequence
libsqpf	SqpFortran
libsqpnagc	SqpNagC

Table 6.3: Classes within the *PCC2* libraries

Within these libraries several external sources are used. For further details of the corresponding codes, we refer to stated webpages.

Remark 6.3
Moreover, various exception classes are included in each of these libraries which can be used to either automatically handle different error situations or to provide a full error report to the user.

[1]Webpage: `https://projects.coin-or.org/Ipopt`
[2]Webpage: `http://www.unige.ch/~hairer/software.html`
[3]Webpage: `http://www.math.uni-bayreuth.de/~kschittkowski/nlpqlp.htm`
[4]Webpage: `http://www.nag.co.uk`

The following sections deal with the implementation and interaction of the major classes within the package. The class `BtIpOpt` has not been optimized and tested extensively yet. Hence, no description of its implementation is given here.

6.2 Receding Horizon Controller

The receding horizon controller is contained in the library **libmpc2** and implements Figure 6.2. It allows for a user defined choice of algorithms, i.e. minimization routine and differential equation solver, and configuration of the corresponding objects. The implementation itself handles the interaction required to solve the problem RHC_N quietly and allows the routine to be used as a black box in a larger setting. In this section, we explain the "ingredients" of this controller and the connection of its classes.

6.2.1 Class MPC

The class `MPC` offers the possibility to construct and solve the sequence of optimal control problems with sampling and zero order hold

$$
\begin{aligned}
\text{Find } \mu_N(x(t_k)) &:= u_{[0]} \\
\text{ST. } u_{[0,N-1]} &= \operatorname*{argmin}_{u_N \in \mathcal{U}_N} J_N(x(t_k), u_N) \\
J_N(x(t_k), u_N) &= \sum_{i=0}^{N-1} \int_{t_i^k}^{t_{i+1}^k} L\left(x_{u_N}(\tau, x(t_k)), u_N(x(t_k), \tau)\right) d\tau \\
&\quad + \sum_{i=0}^{N-1} l\left(x_{u_N}(t_i^k, x(t_k)), u_N(x(t_k), t_i^k)\right) + F(x_{u_N}(t_N^k, x(t_k))) \\
\dot{x}_{u_N}(t) &= f(x_{u_N}(t, x(t_k)), u_N(x(t_k), t)) \qquad \forall t \in [t_0^k, t_N^k] \\
x_{u_N}(0, x(t_k)) &= x(t_k) \\
x_{u_N}(t, x(t_k)) &\in \mathbb{X} \qquad \forall t \in [t_0^k, t_N^k] \\
u_N(x(t_k), t) &= c_i \in \mathbb{U} \qquad \forall t \in [t_i^k, t_{i+1}^k)
\end{aligned}
$$

To this end, the user has to implement the example in a class `Model` object, to specify a choice of algorithms from all lower levels of the hierarchy and to setup a main program, see Section 6.1.1 for further details and Section 6.5 for an illustrative example.
Note that within the class `MPC`, this problem is still in its original form, i.e. if the user supplies a continuous–time model with piecewise constant control on some given time grid $(t_i^0)_{i=1,\ldots,N}$, there is no discretization done within this class. At this point we focus on the mathematically and numerically more challenging part of continuous–time systems but we like to mention that it is also possible to implement discrete–time systems.
To generate and solve the receding horizon control problem, several procedures are required for

- Initializing a class `Discretization` object
- Solving the discretized version of the optimal control problem using a supplied optimization routine and a differential equation solver

- Shifting the problem forward in time

Moreover the class contains procedures

- to set parameters within the optimal control problem itself, e.g. the dynamic of the system or the cost functional, and
- to adapt the number of switching points within the control function, i.e. to change the length of the horizon.

6.2.1.1 Constructor

Within the constructor method the receding horizon problem is defined. This requires the user to create a class `Model` object

```
33    Model * object_model = new InvertedPendulum();
```

Listing 6.1: Constructing call of a class `Model` object

which in turn implicitly defines a suitable *differential equation solver* and its *configuration*, cf. Listing 6.37. Moreover, the calls

```
34    IOdeManager * object_odemanager = new SimpleOdeManager();
```

Listing 6.2: Constructing call of a class `IOdeManager` object

```
35    btmb::MinProg::MinProg * object_minimizer = new SqpFortran();
```

Listing 6.3: Constructing call of a class `MinProg` object

```
38    btmb::MPC2::MPC * mpc_problem = new MPC ( INF );
39    mpc_problem->reset ( object_odemanager, object_minimizer,
         object_model, HORIZON );
```

Listing 6.4: Constructing call of a class `MPC` object

are used to construct objects from lower level classes of the hierarchy, i.e.

- a model of the example `object_model`, see Section 6.1.1,
- a minimization routine `object_minimizer`, see Section 6.3.1, and
- a differential equation manager `object_odemanager`, see Section 6.2.5,

have to be handed over to the class `MPC` object. Additionally, the size of the problem, i.e. the length of the horizon `HORIZON` must be known by the controller.
Note that all other problem specifications, that is

- the dynamic,
- the cost functional,
- the restriction functions and
- shooting nodes and values

as well as the dimension of the state vector and the control vector are given by the class `Model` object `object_model`.

Remark 6.4
Multiple shooting nodes and values are treated as optimization variables within this setting. Internally, these variables are combined to one optimization variable of the discretized problem, see Section 6.2.4 for further details and Section 8.3.4 for the computational impact of multiple shooting nodes.

Finally, some value `INF` can be supplied which internally sets the value of $\pm\infty$ for the optimization routine. If this is not stated explicitly, the default value $\pm 10^{19}$ will be used. Coming back to the mathematical setting, the constructor method generates an optimal control problem of the form

$$
\begin{aligned}
\text{Find } \hat{u}_N &= \underset{u_N \in \mathcal{U}_N}{\operatorname{argmin}} J_N(x_0, u_N) \\
\text{ST. } J_N(x(t), u_N) &= \sum_{i=0}^{N-1} \int_{t_i}^{t_{i+1}} L\left(x_{u_N}(\tau, x(t)), u_N(x(t), \tau)\right) d\tau \\
&\quad + \sum_{i=0}^{N-1} l\left(x_{u_N}(t_i, x(t)), u_N(x(t), t_i)\right) + F(x_{u_N}(t_N, x(t))) \\
\dot{x}_{u_N}(t) &= f(x_{u_N}(t, x(t)), u_N(x(t), t)) \qquad \forall t \in [t_0, t_N] \\
x_{u_N}(0, x(t)) &= x(t) \\
x_{u_N}(t, x(t)) &\in \mathbb{X} \qquad \forall t \in [t_0, t_N] \\
u_N(x(t), t) &= c_i \in \mathbb{U} \qquad \forall t \in [t_i, t_{i+1})
\end{aligned}
$$

where the functions can be identified as follows:

	Function	Program
Control function	$\hat{u}_N$	`u` `sdatavalues`
Stage cost	$l\left(x_{u_N}, u_N\right)$	`pointcostFunction`
Terminal cost	$F(x_{u_N})$	`getObjectiveWeight`
Running cost	$L\left(x_{u_N}, u_N\right)$	`objectiveFunction`
Dynamic	$f(x_{u_N}, u_N)$	`dglFunction`

Restrictions	$\mathbb{X}, \mathbb{U}$	`restrictionFunction`
		`getControlBounds`
		`getModelBounds`

Table 6.4: Notation used for implementation of a RHC problem

From Table 6.4 one can see that the state trajectory $x_{u_N}(\cdot, \cdot)$ is not used during the construction of the class `MPC` object. The class `MPC` object only depends on the initial state and the switching structure. As we will see in Section 6.2.1.2, this allows for using this object as a black box in a user–defined control setup.

Moreover, one can see that the problem defined in a class `MPC` object is not yet discretized. Hence, at this point we consider the control function $\hat{u}_N$ to be piecewise constant on some time grid $\mathbb{T}$. Additionally, this time grid is not fixed since the memory allocation only depends on the number of switching points, i.e. `HORIZON`-1, and does not specify the values of these points.

6.2.1.2 Initialization

As seen in the construction of a class `MPC` object, the time grid is not yet fixed, i.e. the switching structure of the piecewise constant control function is unknown by now. Calling

```
53    mpc_problem->allocateMemory ( t, u );
54    mpc_problem->initCalc ( t, u );
```

Listing 6.5: Initializing a class `MPC` object

allocates and initializes a predefined class `MPC` object with a time grid `t` supplied by the user. Moreover, an initial guess of the control sequence `u` has to be supplied. Note that using this procedure, these two arrays can be changed at any time during the RHC iteration process.
The standard setup in the literature for the time discretization is to use an equidistant grid, see e.g. [16, 95, 134].

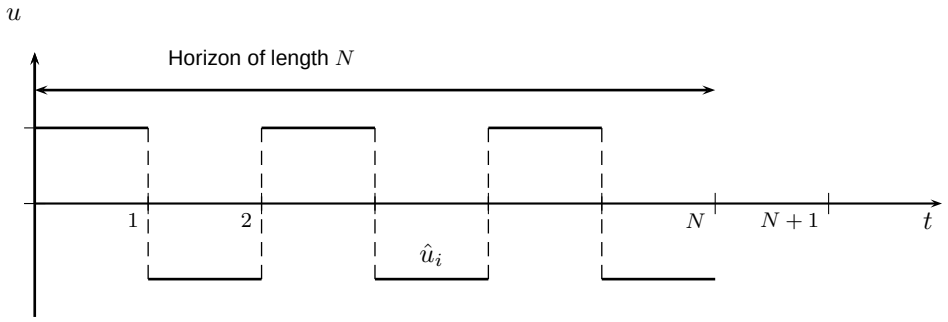


Figure 6.4: Setting up the time grid and initial guess of the control

However, since in some applications this is not a desireable feature, the time grid can be defined as an arbitrary strict ascending sequence

$$\mathtt{t} = [t_0, t_1, \ldots, t_N], \quad t_0 < t_1 < \ldots < t_N,$$

The switching structure of the receding horizon control problem can be modified after every step of the RHC algorithm. Moreover, one does not have to care about numerical errors by choosing this structure since the underlying differential equations are solved using an adaptive step sizing algorithm which is independent of the chosen step length and not some kind of Runge–Kutta scheme such that the choice of $\Delta_i = t_{i+1} - t_i$ may become critical. This offers the possibility to test several switching structures before implementing one control value.
We like to mention that sampling points within the overlapping time intervals in two consecutive receding horizon control steps do not have to coincide at all. However, from a numerical and, in particular, from an optimization procedure point of view, consecutive time grids with coinciding sampling points are a very desireable feature. In this case, the outcome of the previous optimal control problem — in general — represents a good initial guess for the control in the following optimal control problem. Hence, this turns out to be a possible implementation for supplying a new initial guess after shifting the time grid, see also Section 6.2.1.5.
Of course any initial guess can be used by changing `u`. Finding such an initial guess for the control, however, is a hard task and massively influences the computing time necessary to solve the discretized optimal control problem, cf. Section 8.3.3. Moreover, if the supplied

guess is not close to the optimal solution, the optimization routine may converge to some local optimum since we use local minimization methods to solve this problem only.
However, one may use external information, if available, to improve or ease the search for an optimal solution. This can be done for any control values within the discretized optimal control problem. Here, it is particularly useful to subsequently reset the values of the multiple–shooting nodes which are also controls in the discretized problem and included in `u`. A more sophisticated analysis of this topic can be found in Sections 8.3.3 and 8.3.4.

Remark 6.5
Since equidistant time grids with fixed sampling period T are standard in model predictive control we consider this setup exclusively. In particular, we use the term horizon length $H = N \cdot T$ instead of the switching structure of the control. However, all routines can also be applied for an arbitrary discretization.

6.2.1.3 Resizing the Horizon

Since the possiblity to adapt the length of the optimization horizon is required by the adaptation algorithms presented in Chapter 4, i.e. to shorten or prolongate the horizon, a method to deal with this matter is implemented within the class `MPC`. This procedure can be used either manually in the `main` program or in an automated manner within an adaptive receding horizon control algorithm by calling

```
70    mpc_problem->resizeHorizon ( 17, H_NEW );
```

Listing 6.6: Resetting the length of the optimization horizon

where the first variable is used as new quantity of grid points. In the case of an prolongation of the horizon, additional grid points are added equidistantly at the end with grid diameter `H_NEW`. Internally, this leads to a destruction of the discretized problem which is due to memory requirements and a new object of this type with correct values is constructed.

Remark 6.6
Note that memory allocation at this point is not strictly dynamic. A class `MPC` object is initialized with a maximal size of the horizon length which cannot be extended further. This is due to the optimization of the code towards speed which requires a certain setup of the used memory.
To extend the horizon even further, the class `MPC` object must be destructed and a new one with extended maximal horizon length must be created.

The equidistant setting of grid points in the prolongation case can be adapted by setting the array `t`. Alternatively, a different array can be supplied to a class `MPC` object `mpc_problem` using the initialization routine described in Section 6.2.1.2.

Remark 6.7
Concerning the impact of the horizon length on the computing time necessary to solve the discretized optimal control problem, this feature is particularly useful if one can somehow calculate the minimal stabilizing horizon length, especially if this lower bound varies during the iteration process. For an analysis of this issue we refer to Section 8.3.2 for numerical results concerning the horizon length and to Section 8.5 which deals with our adaptation strategies.

6.2.1.4 Starting the Calculation

Since it is likely that the initial value of the actual optimal control problem has to be redefined, e.g. in the case when an external observer is used for state estimation as shown in Figures 6.5, 6.6, this value can be set to an externally computed one before starting the optimization routine.

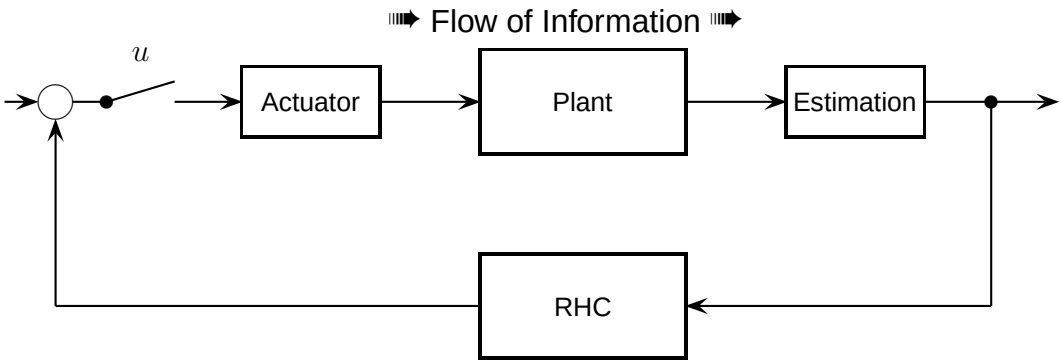


Figure 6.5: Schematic representation of the usage of a receding horizon controller

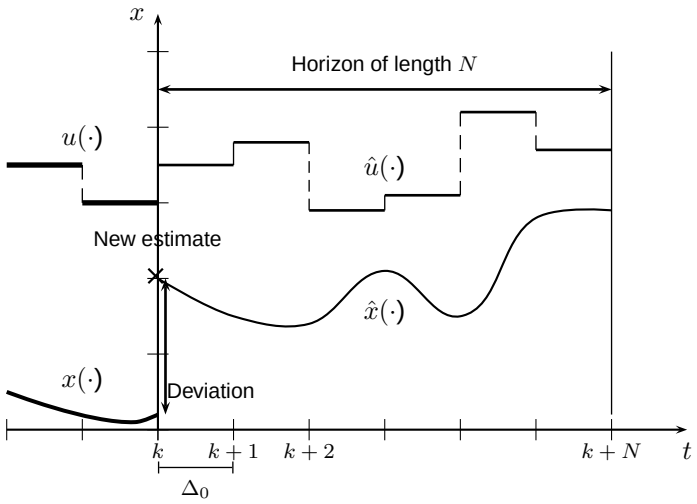


Figure 6.6: Evaluation of the optimal control problem and setting initial value

In order to solve the optimal control problem the procedure

```
77          mpc_problem->calc ( x );
```

Listing 6.7: Solving the underlying optimal control problem of the RHC problem

has to be called where external resets of the state vector `x` can be handed over directly to the class `MPC` object. On exit, this call causes the results, i.e. the control sequence $\hat{u}_N$, to be stored in the array `u`. Additionally, as long as the class `MPC` object is not destructed, the values of the internally predicted trajectory as well as the cost functional values can be extracted if necessary.

6.2.1.5 Shift of the Horizon

The time shift of the optimization horizon, that is the receding horizon step of the RHC method, see Step 3 in Section 2.4, is carried out via

```
86  mpc_problem->shiftHorizon ( x, next_u, H_NEW, mstep );
```

Listing 6.8: Executing the shift within the RHC scheme

Here, the user can choose between three different variants of this shift. The call

```
mpc_problem->setConfigShiftHorizon( H_NEW, method,
   reltolerance, abstolerance, tolerancevector );
```

Listing 6.9: Configurating the shift within the RHC scheme

can be used to change the configuration of the class `MPC` object where the different variables have the following:

Variable	Description
`H_NEW`	Step length of the added sampling intervals at the end of the actual optimization horizon
`method`	Decision variable $= 0$: Shift of the control sequence by `mstep` sampling intervals and copying the last `mstep` entries for the new sampling intervals (default) $= 1$: Shift of the control sequence by `mstep` sampling intervals and optimization of the last `mstep` entries for the new sampling intervals $= 2$: Successive (for `i` from 1 to `mstep - 1`) shift of the control sequence by one sampling interval and re–optimization of the last `HORIZON - mstep + i` entries for the new influencable sampling intervals
`reltolerance`	Value or vector of relative tolerances of the differential equation solver used for implementing the shift
`abstolerance`	Value or vector of relative tolerances of the differential equation solver used for implementing the shift
`tolerancevector`	Decision variable allowing to supply tolerance values (`tolerancevector`= 0, default) or vectors (`tolerancevector`= 1)

Table 6.5: Configuration parameters of RHC scheme shift

Remark 6.8
Supplying the parameter `reltolerance`, `abstolerance` *and* `tolerancevector` *allows the*

user to distinguish between accuracy of the differential equation solver used for optimization and the one used for simulation. Note that, if these values are not identical, the outcome of the simulated implementation of the control is in general not identical to the predicted trajectory within the optimization.

The parameter `H_NEW` is used to add a total of `mstep` new sampling instants at the end of the optimization horizon with sampling distance `H_NEW`, i.e. the subsequent control problem possesses the time grid

$$\begin{aligned} t_i &:= t_{i+\texttt{mstep}}, \quad i \in \{0, \ldots, N - \texttt{mstep}\}, \\ t_{N-\texttt{mstep}+i} &:= t_{N-\texttt{mstep}} + i \cdot \texttt{H\_NEW}, \quad i \in \{1, \ldots, \texttt{mstep}\}. \end{aligned}$$

In order to change this equidistant setting, a redefinition of the variable `t` in the main program can be used.
Calling the method `shiftHorizon` causes the internally predicted state value at time instant t_{mstep} to be stored in `x`. In case of external disturbances, model uncertainties or if an observer is used to estimate the state vector, this prediction should be reset by a corrected vector before starting the next optimization, see Section 6.2.1.4. Additionally, the control vectors $\hat{u}_{[0,\texttt{mstep}]}$ which are to be applied at time instants $t_0, \ldots, t_{\text{mstep}}$ are copied to `next_u`.
Here, we like to point out that shifting the horizon does not only affect the internal time grid. Within this implementation, the control sequence is shifted as well. Depending on the parameter `optimize`, different control vectors are the result of such a shift:

- If `method` is set to zero (default), the last control vector of the unshifted problem is copied such that in the shifted problem the last `mstep` control vectors are identical. This method is computationally efficient and still offers a good initial guess for the subsequent optimization problem, at least if the trajectory at the end of the horizon is close to the optimum.

- For the second choice, i.e. `method`= 1, the initial guess of the option `method`= 0 is used and an additional optimization problem considering the last `mstep` control vectors is generated and solved. Even for `mstep` being small, the additional computational effort is not negligible. Yet, significant reduction of the computing times for solving the subsequent optimization problem can be experienced, see Section 8.3.3.

- The third implementation is computationally demanding since a whole series of large optimization problems have to be solved. Yet, it has shown to be of good use if large control horizons `mstep` are considered. In this case, the option `method`= 2 allows us to subsequentually keep track of a good initial guess for the following optimization problem.

6.2.1.6 Destructor

A class `MPC` object can be destroyed by using the destructor

```
93    delete mpc_problem;
```

Listing 6.10: Destructing call of a class `MPC` object

which deletes the internally allocated minimization problem but leaves all supplied data untouched. Hence, the last open–loop control sequence `u` as well as the initial state vector `x` can still be accessed and used for other purposes.

6.2.2 Class SuboptimalityMPC

The class `SuboptimalityMPC` is derived from the class `MPC` and contains the methods described in Chapter 3 to obtain stability and suboptimality results. In particular, all methods from class `MPC` are inherited such that a RHC controller utilizing the additional estimation methods can be implemented identically. Hence, only the class name within the construction/destruction of the object needs to be replaced:

```
btmb::MPC2::SuboptimalityMPC * mpc_problem = new
   SuboptimalityMPC ( INF );
delete mpc_problem;
```

Listing 6.11: Constructing/Destructing call of a class `SuboptimalityMPC` object

Moreover, internal variables are set to default values. The meaning and usage of these variables are explained within the description of the estimation procedures.

6.2.2.1 A posteriori Suboptimality Estimate

Calling the method

```
mpc_problem->aposterioriEstimate ( x, alpha );
```

Listing 6.12: Calculating α according to Proposition 3.3

leads to an evaluation of Proposition 3.3 using Algorithm 3.8 to compute the suboptimality degree $\alpha =$ `alpha`.
To this end, the values of $V(x(n))$ and $V(x(n+1))$ as well as $l(x(n), \mu_N(x(n)))$ must be known. In this implementation, these values will **not** be computed by the procedure within every step. Instead, the previous optimal value function $V(x(n-1))$ and the corresponding first stage cost $l(x(n-1), \mu_N(x(n-1)))$ are stored within a class `SuboptimalityMPC` object `mpc_problem` and $V(x(n))$ is computed by the procedure by a *postoptimal evaluation of the cost function*, i.e. no additional optimization is required. To this end, the variable `x` can be used to incorporate measurement or prediction errors. Hence, this setting is purely *a posteriori*. Internally calculating the optimal finite horizon cost $V(x(n))$ gives us, as a byproduct, the stage cost $l(x(n), \mu_N(x(n)))$ and both are used to replace the stored values of the last step, see also Section 6.2.3.3 below for further details.

Remark 6.9
This setup can be extended to analyze the suboptimality of a so called m–step feedback, see e.g. [99, 100], by storing a sum of the stage costs. This is implemented within the method `mstepAposterioriEstimate`*, see Listing 6.13.*

```
mpc_problem->mstepAposterioriEstimate ( x, alpha, m );
```

Listing 6.13: Calculating α similar to Proposition 3.3 for an m–step feedback

The internal boolean variable `single` can be used to obtain *local* or *closed–loop* suboptimality results. If α is to be computed for every single point of the trajectory, the variable `single` is set to the default value true. If `single` is set to false by calling

```
mpc_problem->setSingle ( false );
```

Listing 6.14: Setting the internal variable `single`

the minimal α along the points under consideration is the result of this method. Note that the variable `alpha` must not be changed within the calling program.

Remark 6.10
The underlying problem has to meet the form which is the basis of Proposition 3.3. Hence, the user must check the implemented cost functional `objectiveFunction` ***and*** *the additive discrete–time cost functional* `pointcostFunction` *defined within the class* `Model` *object used by a class* `SuboptimalityMPC` *object.*

6.2.2.2 A priori Suboptimality Estimate

In contrast to Section 6.2.2.1 using Algorithm 3.8, we now compute an estimate $\alpha =$ `alpha` *a priori* based on Algorithm 3.16. In particular, it is not necessary for this procedure to store any past data of the class `SuboptimalityMPC` problem since all values can be computed based on the actual initial value `x` and the calculated optimal control sequence `u` which is available within a class `SuboptimalityMPC` object `mpc_problem`.
In order to obtain $V_k(x_{u_N}(N-k, x(n)))$ and $l(x_{u_N}(N-k, x(n)), \mu_k(x_{u_N}(N-k, x(n))))$ for all $k \in \{N_0, \ldots, N\}$, it suffices to solve the dynamic of the problem using the precomputed optimal control sequence `u`. However, to retrieve the value of $l(x_{u_N}(N-j, x(n)), \mu_{j-1}(x_{u_N}(N-j, x(n))))$ for all $j \in \{2, \ldots, N_0\}$ we have to solve a whole series of optimal control problems. This requires the user to call the routine

```
mpc_problem->aprioriEstimate ( x, alpha, gamma, N0 );
```

Listing 6.15: Calculating α according to Theorem 3.22

where `x` is a possibly updated estimate of the current state. Internally a series of new, smaller optimal control problems and their discretizations are generated. To solve the resulting minimization problems, another instance of the supplied optimization routine `object_minimizer` and of the differential equation manager `object_odemanager` need to be generated.
Again, the internal boolean variable `single` can be used to obtain the local suboptimality estimates α by setting it to true and the closed–loop suboptimality estimate if `single` is set to false.
Last, the variable `N0` which specifies the shortened optimal tail to be considered in the analysis needs to be set by the user.

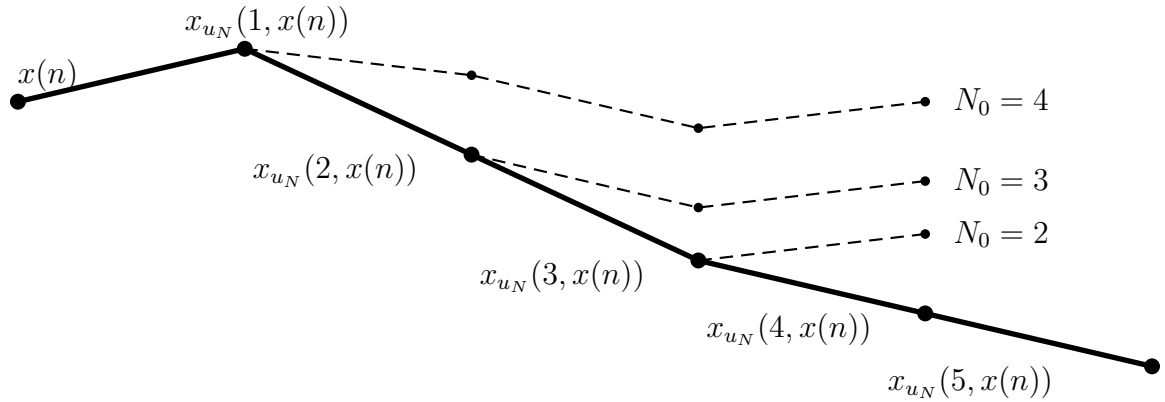


Figure 6.7: Schematic representation of solution "tails" used within the a priori suboptimality estimation of Theorem 3.22 for $N = 5$

Remark 6.11
For large values of `NO` $= N_0 \in \{2, \ldots, N-1\}$ *the additional optimal control problem causes this procedure to be not negligible considering the computing time. Hence, in the context of time–critical applications, only small values of* `NO` *should be used, see also Section 8.4 for numerical tests of this routine and the impact of the parameter* `NO` *and Section 8.5 for results in the adaptive receding horizon control setting.*

6.2.2.3 A posteriori Practical Suboptimality Estimate

Similar to the routine for the non–practical case, the call

```
mpc_problem->aposterioriPracticalEstimate ( x, alpha );
```

Listing 6.16: Calculating α according to Proposition 3.28

computes the suboptimality degree $\alpha =$ `alpha` according to Proposition 3.28, i.e. Algorithm 3.30. Here, we use the same setup as in Section 6.2.2.1 for storing past data, supplying the actual state vector `x` and influencing the output to be a *local* or *closed–loop* estimate via the internal boolean variable `single`.
Different to the non–practical case, the internal variable `epsilon` is utilized for cutting the stage costs. This variable is set to `epsilon` $= \varepsilon = 10^{-6}$ and can be reset using

```
mpc_problem->setEpsilon ( 1E-10 );
```

Listing 6.17: Setting the internal variable `epsilon`

In general, its magnitude should be of the size of the optimality tolerance of the underlying class `MinProg` object `object_minimizer`. However, this rule of thumb only holds if the tolerances of the class `OdeSolve` object used by the class `IOdeManager` object `object_odemanager` are lower than the optimality tolerance of the minimizer.
The underlying problem has to meet the form used within Proposition 3.28. To this end, the implemented cost functional `objectiveFunction` **and** the additive discrete–time cost functional `pointcostFunction` defined within a class `Model` object `object_model` used by a `SuboptimalityMPC` object must be reviewed.

Remark 6.12
Similar to the non–practical case, an extension to m–step feedbacks is implemented within the method `mstepAposterioriPracticalEstimate`*, cf. Listing 6.13.*

6.2.2.4 A priori Practical Suboptimality Estimate

The a priori practical estimate uses the same extension as the a posteriori estimate from the previous Section 6.2.2.3. Hence, the call

```
mpc_problem->aprioriPracticalEstimate ( x, alpha, gamma, N0 )
   ;
```

Listing 6.18: Calculating α according to Proposition 3.39

from Listing 6.15 uses the internal variable `epsilon` to cut the stage cost. Here, Algorithm 3.40 is implemented to compute the estimate $\alpha =$ `alpha`. Similar to Section 6.2.2.2, the values of $l(x_{u_N}(N-j, x(n)), \mu_{j-1}(x_{u_N}(N-j, x(n))))$ have to be obtained via a series of new smaller optimal control problems. These problems are generated, discretized and solved by instances of the objects created within the class `SuboptimalityMPC` object.

Similar to Section 6.2.2.2, the variable `N0` specifies the shortened optimal tail and the boolean `single` is utilized to distinguish between *local* and *closed–loop* suboptimality results.

6.2.3 Class AdaptiveMPC

The class `AdaptiveMPC` implements the optimization horizon adaptation strategies described in Chapter 4. Since the corresponding Algorithms 4.8, 4.10, 4.14, 4.16, 4.19, 4.21, 4.23, 4.29, 4.37 and 4.40 are based on the suboptimality estimates from Chapter 3 it is derived from the class `SuboptimalityMPC` implementing these estimates.
In particular, all methods from the class `MPC` are inherited such that a receding horizon controller utilizing the additional estimation methods can be implemented identically. The methods `calc` and `shiftHorizon`, however, are replaced to suit the adaptivity algorithms from Chapter 4. Similar to the constructor of a class `SuboptimalityMPC` object, all headers are identical such that only the class name within the construction/destruction of the object needs to be replaced:

```
btmb::MPC2::AdaptiveMPC * mpc_problem = new AdaptiveMPC ( INF
    );
delete mpc_problem;
```

Listing 6.19: Constructing/Destructing call of a class `AdaptiveMPC` object

6.2.3.1 Starting the Calculation

In order to obtain an open–loop optimal control satisfying a predefined suboptimality bound $\overline{\alpha}$, by calling

```
mpc_problem->setConfigAdaptiveMPC ( alphabound, estimate,
   implementation, N0, sigma );
```

Listing 6.20: Calculation call for a class `AdaptiveMPC` object

the used suboptimality bound `alphabound`$= \overline{\alpha}$ is set and the user chooses the suboptimalty estimation method and the implementation of the adaptation algorithm by defining the parameter `estimate` and `implementation`. Optionally, the configuration parameter `N0` $= N_0$ of the a priori estimation method and the maximal steps `sigma` $= \sigma$ can be set.

Variable	Description
`alphabound`	Lower suboptimality bound $\overline{\alpha}$ to be satisfied
`estimate`	Suboptimality estimation method 0: No estimation, standard `MPC` `calc` and `shiftHorizon` methods are applied 1: A posteriori estimate according to Proposition 3.3 2: A posteriori practical estimate according to Proposition 3.28 3: A priori estimate according to Theorem 3.22 4: A priori practical estimate according to Theorem 3.39

Variable	Description
`implementation`	Adaptation strategy 0: No adaptation applied 1: Simple shortening strategies 4.10, 4.19 and prolongation strategies 4.14, 4.21 depending on the choice of the suboptimality estimation method 2: Simple shortening strategies 4.10, 4.19 and fixed point strategy 4.23 for prolongation 3: Simple shortening strategies 4.10, 4.19 and monotone strategy 4.29 for prolongation
`N0`	Auxilliary parameter N_0 required by suboptimality estimates according to Theorems 3.22 and 3.39
`sigma`	Auxilliary parameter σ required by Algorithms 4.23 and 4.29

Table 6.6: Configuration parameters of a class `AdaptiveMPC` object

Once a class `AdaptiveMPC` object is configured, the method

```
mpc_problem->calc( x );
```

Listing 6.21: Calculation call for a class `AdaptiveMPC` object

can be called to start the optimization and adaptation process. Due to the general structure of the receding horizon controller implemented in the *PCC2* package, this call is not identical to the algorithms described in Chapter 4 since the computation and implementation of control sequences are separated. For the implementation, the method `shiftHorizon` can be applied, see also Section 6.2.3.4.

6.2.3.2 Implemented Strategies

The different strategies of Algorithms 4.8, 4.16, 4.37 and 4.40 using the simple shortening and prolongation strategies of Algorithms 4.10, 4.14, 4.19 and 4.21 are implemented in the private method `calcSimple`, `simpleShortening` and `simpleProlongation` respectively. Here, the configuration parameter `implementation` is used to choose a particular method.

Remark 6.13
Algorithms 4.11 and 4.20 which represent a mixed closed–open–loop strategy, are not implemented separately. To obtain results for these algorithms, a repeated call of `calc` *and* `shiftHorizon` *can be used due to the principle of optimality. In this case, the endpiece of the previous optimal control problem is identical to the actual (shifted) optimal control problem. Hence, if the original solution was optimal, calling* `calc` *does not change the solution and a call of* `shiftHorizon` *implements the actual first component of the control vector.*

The more sophisticated prolongation strategies of Algorithms 4.23 and 4.29 are implemented in the private methods `calcSimpleFixpoint` and `calcSimpleMonotone`. Note that these implementations are designed to allow for shortening the horizon only by the

shortening strategies 4.10 and 4.19. Again, the parameter `implementation` is used to start the chosen methods.

Remark 6.14
The adaptation algorithms of Chapter 4 in methods `calcSimple`*,* `calcSimpleFixpoint` *and* `calcSimpleMonotone` *are separated from the required calls to the suboptimality estimates of Chapter 3 since they are designed independently. The latter ones are implemented in method* `calcAlpha` *described in the following Section 6.2.3.3.*

6.2.3.3 Using Suboptimality Estimates

The parameter `estimate` defines the estimation method to be used by a call of the private method `calcAlpha`, i.e. a posteriori, a priori or the corresponding practical variants, see also Propositions 3.3 and 3.28 and Theorems 3.22 and 3.39.
In order to use the a priori methods from class `SuboptimalityMPC`, no additional computations to call these methods are required since these methods rely on the actual state vector `x` only, see Sections 6.2.2.2 and 6.2.2.4. The required calls are implemented in the private method `aprioriCalculatedEstimate`.
The a posteriori methods on the other hand have to be based on a future state vector which holds true at the first sampling instant in the actual time grid. The private method `predictedAposterioriEstimate` computes this value and solves the required additional shifted optimal control problem. Moreover, the call to the a posteriori estimation method `aposterioriEstimate` or `aposterioriPracticalEstimate` is executed in this method and the computed suboptimality estimate `alpha` $= \alpha$ is set.

6.2.3.4 Shift of the Horizon

Within our implementation of an adaptive receding horizon controller, we replace the shift strategy `shiftHorizon` described in Section 6.2.1.5 if an a posteriori estimate is used, i.e. if `estimate` $\in \{1, 2\}$. Here, we use the already acquired information of solving the predicted shifted optimal control problem, see also Section 6.2.3.3. This allows us — if no errors and state estimation errors occur — to reduce the additional effort for the adaptation strategy to zero since the predicted shifted optimal control problem is identical to the now actual one and no reoptimization is necessary.
In any other case, calling

```
mpc_problem->shiftHorizon( x, H_NEW );
```

Listing 6.22: Shifting call for a class `AdaptiveMPC` object

calls the routine previously described in Section 6.2.1.5. Note that, in either case, the calls are identical.

6.2.4 Class Discretization

The class `Discretization` can be used to create a class `Discretization` object which transforms a possibly continuous–time optimal control problem into a discrete–time one. Although this class can also be used outside a class `MPC` object, the underlying problem still has to meet the specifications mentioned in Section 6.2.1.
Since we focus on solving a receding horizon control problem, we restate the discretization in this context, i.e. we transform the optimal control problem from Section 6.2.1 to a discrete problem of the following form:

$$
\begin{aligned}
\text{Find } u_{[0,N-1]} &= \operatorname*{argmin}_{u_N \in \mathcal{U}_N} J_N(x_0, u_N) \\
\text{ST. } J_N(x_0, u_N) &= \sum_{i=0}^{N-1} l_i\left(x_{u_N}(i, x_0), u_N(x_0, i)\right) \\
&\quad + F(x_{u_N}(N, x_0)) \\
x_{u_N}(i+1, x_0) &= f\left(x_{u_N}(i, x_0), u_N(x_0, i)\right) \quad \forall i \in \{0, \ldots, N-1\} \\
x_{u_N}(0, x_0) &= x_0 \\
x_{u_N}(i, x_0) &\in \mathbb{X} \qquad \forall i \in \{0, \ldots, N\} \\
u_N(x_0, i) &\in \mathbb{U} \qquad \forall i \in \{0, \ldots, N-1\}
\end{aligned}
$$

Here, the functions $l_i(\cdot,\cdot)$ correspond to the integrated Lagrangian term $L(\cdot,\cdot)$ plus the summation term $l(\cdot,\cdot)$ which is to be evaluated at the sampling instants, i.e.

$$
l_i(x, u) := \int_{t_i}^{t_{i+1}} L(x(\tau), u(\tau)) d\tau + l(x(t_i), u(t_i)). \tag{6.1}
$$

The classes `MPC` and `Discretization` are handled separately since one class `MPC` object may use discretizations of several optimal control problems or several discretizations of one optimal control problem.
Within the class `Discretization` we use the recursive discretization technique with multiple shooting nodes, see Sections 5.1.2 and 5.1.3. Hence, we obtain a discretized optimal control problem stated in Definition 5.7 which is given in standard form for nonlinear optimization

$$
\begin{aligned}
\text{Minimize} \quad & F(x) && \forall x \in \mathbb{R}^n \\
\text{ST.} \quad & G_i(x) = 0, && i \in \mathcal{E} \\
& H_i(x) \geq 0, && i \in \mathcal{I}
\end{aligned}
$$

with optimization variable

$$
x := \left(u_{[0]}^\top, \ldots, u_{[N-1]}^\top, s_x^\top\right)^\top
$$

Remark 6.15
Even if the underlying dynamic within a class `MPC` object is already given in discrete–time, the discretization step is still necessary for memory allocation purposes.

6.2.4.1 Constructor / Destructor

Similar to a class `MPC` object, a class `Discretization` object is set up by calling

```
btmb::MPC2::Discretization * problem = new btmb::MPC2::
    Discretization ( object_odemanager, object_minimizer,
    object_model, length_horizon, infinity );
```

Listing 6.23: Constructing call of a class `Discretization` object

Due to the separation of the discretization from the receding horizon control problem itself, the memory allocation necessary to handle variables and constraints is done within this object. Background for this is the consideration of a possibly continuous–time optimal control problem in a class `MPC` object and a discrete–time transformation within a class `Discretization` object. The resulting optimization problem is of the general form

$$\begin{array}{rll} \text{Minimize} & F(x) & \forall x \in \mathbb{R}^n \\ \text{ST.} & G_i(x) = 0, & i \in \mathcal{E} \\ & H_i(x) \geq 0, & i \in \mathcal{I} \end{array}$$

Moreover, no computation of any kind is performed during this construction. In particular, it does not make sense to calculate any trajectory within a class `Discretization` object itself since it cannot compute an optimal control sequence. This is the job of the supplied class `MinProg` object. Hence, a class `Discretization` object makes use of an class `SqpFortran`, `SqpNagC` or `BtIpOpt` object from one of the subclasses of class `MinProg`, see Section 6.3.1, to solve this task. To this end, it manages the necessary memory and provides procedures for the class `MinProg` object to compute the cost functional, its gradient and the restriction function as well as the Jacobian of the restriction function.

6.2.4.2 Initialization

The initialization method of the class `Discretization` object is implemented identically to a class `MPC` object and supplied for external use of this class only.

6.2.4.3 Calculation

After creating and initializing a class `Discretization` object, the underlying minimization method from class `MinProg` can be called to solve the minimization problem. Depending on the choice of the minimizer the procedure

```
problem->calc ( x );
```

Listing 6.24: Calculation call for a class `Discretization` object

executes this calculation. Here, a class `Discretization` object supplies the mentioned private methods for computing the *cost functional*, its *gradient* and the *restriction function* as well as the *Jacobian of the restriction function* at the discretization instances to the class `MinProg` object. This defines the abstract minimization problem stated in Section 6.2.4.1.
For solving the abstact optimization problem, this results in the following:
A class `Discretization` object defines the abstract optimization problem by setting function pointers for the cost function, restrictions and the dynamic of the control problem. Then, the method `calc` triggers the minimization routine to start the optimization. In turn, the corresponding method of a class `MinProg` object calls a class `IOdeManager` object to evalutate all required functions. Then, the class `IOdeManager` object passes the function pointers and required evaluation data over to the differential equation solver which is a class `OdeSolve` object.
Once all evaluations are done, the minimization routine computes a new set of optimization variables. Since the implemented minimization methods are iterative solvers, the predescribed process is repeated within every step until the minimization is terminated.

Remark 6.16
The advantage of this setting is not only the modularity of the considered components of the algorithm, but also the separation of the allocated memory. In particular, once all required objects are initialized, no further allocation or deletion of memory is triggered — except for user modifications. This allows for an effective allocation of the memory regarding the subproblems and increases the performance of the computation.

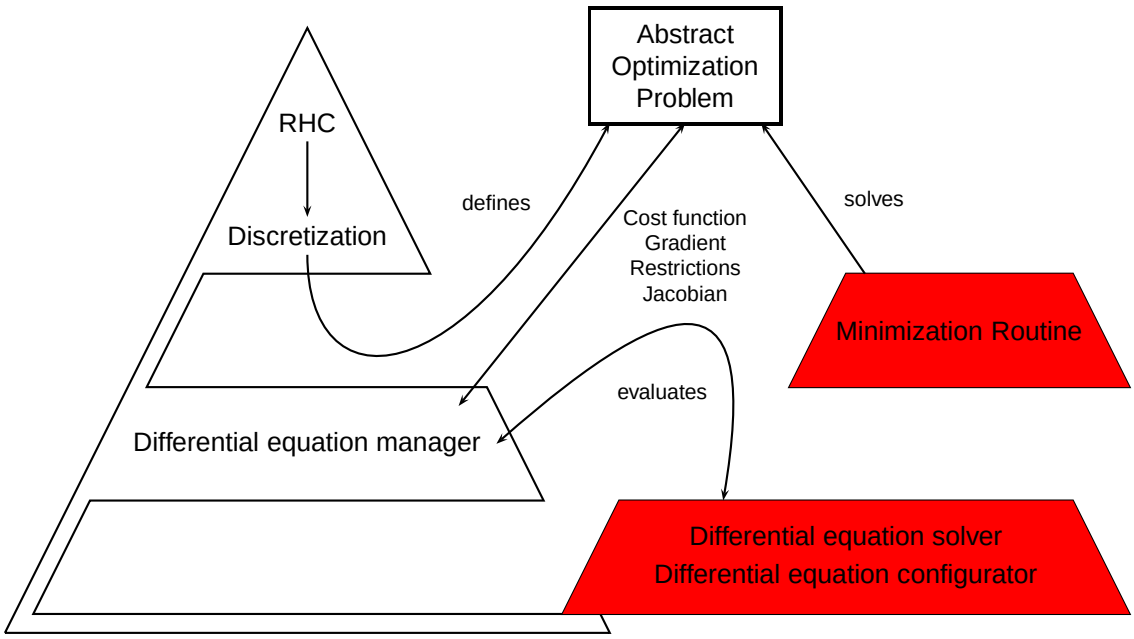


Figure 6.8: Definition of the abstract optimization problem

Here, the vector `x` represents the initial value for the state trajectory to be controlled. This allows us to implement the receding horizon controller independently from the actual state and time of the plant. This is necessary since we cannot expect the optimal control to be computed instantaneously, i.e. we have to use some kind of external prediction to a future state when the computed control will be applied.

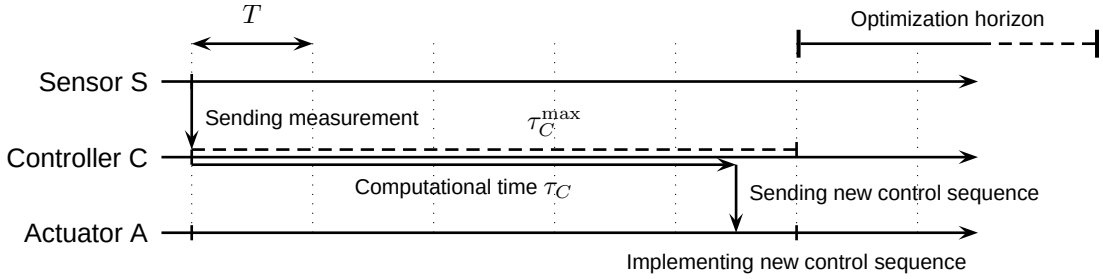


Figure 6.9: Timescale of the closed–loop system

Remark 6.17
The timescale in Figure 6.9 indicates that the calculating procedure `calc` *can/should be initialized with a prediction of the state vector* `x` *at the implementation time instant. In order to be applicable, this time instant should correspond to one of the sampling instants after the new control sequence has arrived at the actuator.*

Remark 6.18
Since the receding horizon control algorithm relies on an input initial state, the usage of estimation procedures, sensor data or other input methods is possible and allows the software to be used as a black box within a control setting.

6.2.4.4 Other functions

The functions to compute the cost functional, its gradient and the restriction function as well as the Jacobian of the restriction function are for internal use of minimization object `object_minimizer` of class `MinProg` only. Calling these functions results in solving a series of differential equation problems using a class `OdeSolve` object. The sequencing of this series of calls is organized by class IOdeManager objects which we treat in the following Section 6.2.5. For further details of the interaction of the classes, see Section 6.2.4.3.

6.2.5 Class IOdeManager

The purpose of the class `IOdeManager` is to cover three implementations for calculating the values of the cost function, the constraints and their derivatives required by a class `MinProg` object. The derivatives need to be computed with respect to every component of the optimization variable (5.7). This allows for a simple definition and approximation of the derivative of some function $f(\cdot)$ with respect to all components of x via the difference quotient

$$D_v f(x) = \lim_{t \to 0} \frac{f(x + t \cdot v) - f(x)}{t} \approx \frac{f(x + t \cdot v) - f(x)}{t} \tag{6.2}$$

where $v \in \mathbb{R}^{Nm+s}$, $\|v\| = 1$ and $t \in \mathbb{R}$.

Remark 6.19
In the receding horizon control context, we think of $f(\cdot)$ being either the cost functional or one of the restriction functions. Moreover, the steplength t should be chosen depending on the used computer, i.e. its floating point ability. Note that given an optimal control problem, the computation of the difference quotient (6.2) *may be sensitive if different discretizations are used to compute $f(x + t \cdot v)$ and $f(x)$. To circumvent this problem, we additionally implemented a synchronized computation, see Section 6.2.5.3 below.*

Within the context of optimizing a discretized optimal control problem, the function values to be computed are depending on the state of the system and the control. Since the state of the system is not contained in the optimization variable (5.7), neither the cost function and the restrictions nor their derivatives can be evaluated.
Now, we use the additional information on the initial state of the system x_0. This information is provided by the user to the class `MPC` object by calling the `calc` method. By this method, the information is passed over to the class `Discretization` object defining the abstract optimization problem and hence the class `MinProg` object. Within the class `IOdeManager` object called by the minimization routine of class `MinProg`, this allows us to call a differential equation solver of class `OdeSolve` to solve the dynamic of the system using this initial value x_0 and the optimization variable x.
Hence, a class `IOdemanager` object provides a link between the classes `MinProg` and `OdeSolve` to solve an optimal control problem.
Using the predescribed methodology, we can evaluate all required functions $f(\cdot)$ and, using the difference quotient (6.2), all derivatives.
For the implementation of such a *differential equation manager* derived from the class `IOdeManager`, we have to keep the following in mind:

(1) Memory requirements: A naive implementation is to compute and store all necessary values, i.e. the trajectories for nominal and varied parameter sets. Such a strategy,

however, requires a rather large storage capacity even for small systems. Considering large differential equation systems like a discretization of a PDE, this may exceed hardware limitations.

(2) Speed: Every evaluation of a function $f(\cdot)$ possibly triggers a call of the differential equation solver. Hence, if the number of function evaluations is not reduced on implementation, many calculations have to be performed which makes this part computationally expensive.

(3) Correctness: The Lagrangian part of the cost function has to be evaluated along the trajectory. To this end, an identical discretization has to be used for the evaluation of this integral and the dynamic of the problem, see also [104] for a detailed analysis. The same holds for the derivative of a function $f(\cdot)$, that is for the two required evaluations of $f(\cdot)$ the same discretization grid should be considered.

Remark 6.20
The Lagrangian part of the cost function is appended to the dynamic of the system within the class `Discretization` *object. Since this new dynamic is handed over to the class* `IOdeManager`*, the first part of the correctness requirement is fulfilled for all deduced classes of class* `IOdeManager`*. Appending the cost function is convenient since in the minimization routines of class* `MinProg`*, every evaluation of the cost function at the current iterate is accompanied by an evaluation of the restriction functions.*
Moreover, the class `Discretization` *object combines the optimization variable according to* (5.7)*.*

Within the receding horizon control setting, the Jacobian of the constraints has the following general structure:

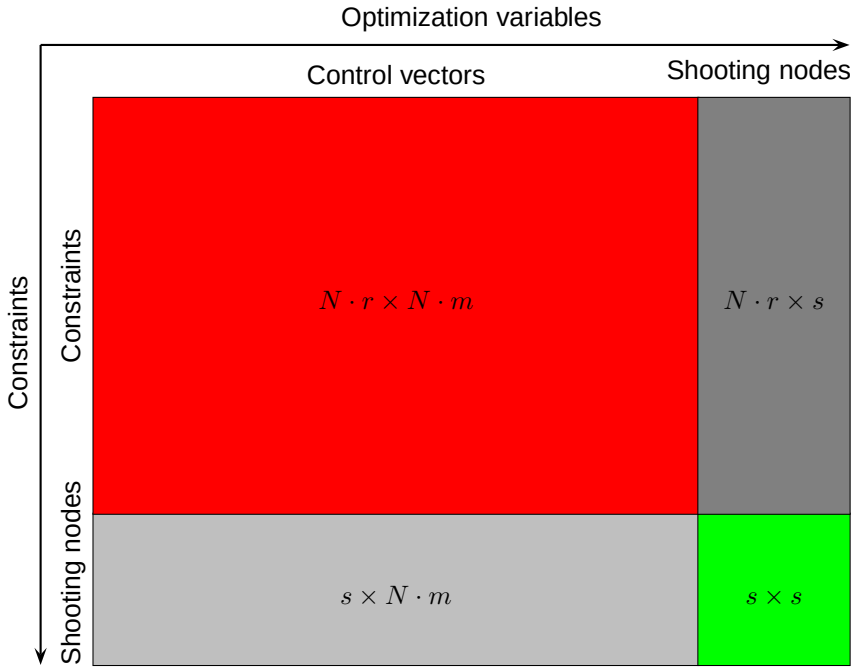


Figure 6.10: Structure of the Jacobian of the constraints

The number of function evaluations is growing quadratically in the horizon length N. Moreover, a recomputation of these values has to be performed within a large number

of the iteration steps of the minimization routine. Hence, an efficient implementation is required for the receding horizon controller. For computing time comparisons of the different `IOdeManager` subclasses see Section 8.2.

Remark 6.21
Note that the dimension of the constraints r may also depend on the optimization procedure since not all methods treat box constraints as nonlinear constraints.

6.2.5.1 Class SimpleOdeManager

The class `SimpleOdeManager` is a straight forward implementation of a interface described in the previous Section 6.2.5 which calculates all function values by solving the differential equation on the whole optimization horizon for the given optimization variable. To utilize an object of this type within the receding horizon controller, it can be constructed/destructed by calling

```
IOdeManager * object_odemanager = new SimpleOdeManager ;
delete object_odemanager;
```

Listing 6.25: Constructing/destructing call of a class `SimpleOdeManager` object

For memory purposes, this method is effective since no intermediate results are stored. However, it is not computationally efficient since many identical calculation steps are repeated, see also Table 6.7. Moreover, the correctness aspect concerning the discretization used to compute the derivatives is not satisfied.

Method	Description
`comdgl`	Attaches the Lagrangian part of the cost function to the differential equation system
`calcObjectiveFunction`	Calculates the value of the cost function for a fixed set of optimization variables considering the appended differential equation of the dynamic, the summation term and the terminal cost term, see Section 6.2.4
`calcObjectiveFunctionGrad`	Computes the gradient of the cost function with respect to all optimization variables
`calcRestrFunction`	Calculates the values of all restrictions for all sampling instances and a fixed set of optimization variables
`calcRestrFunctionGrad`	Computes the Jacobian update

Table 6.7: Calculation methods within class `SimpleOdeManager`

Remark 6.22
Since all derivatives are computed within the Jacobian of the constraints, the QP iteration of the optimization routine is actually a Newton method with complete BFGS–update.

6.2.5.2 Class CacheOdeManager

To utilize the advantages of the quasi–Newton approach based on a reduced Hessian approximation in the QP iterations, a class `CacheOdeManager` object calculates only the required function values. An object of this class is constructed/destructed via

```
IOdeManager * object_odemanager = new CacheOdeManager ;
delete object_odemanager;
```

Listing 6.26: Constructing/destructing call of a class `CacheOdeManager` object

In particular, it considers two aspects:

(1) The array `needc` of the optimization method is used to identify the columns in the Jacobian of the constraints which have to be recomputed.

(2) Repeated function evaluations for identical optimization variables are avoided.

The implementation of the second aspect is a simple but very effective reordering of calculations. Considering the control vectors and the multiple shooting nodes separately, we note that changing the j-th value in this part of the optimization vector does not affect the state trajectory values $x(i)$ for $i \in [0, \ldots, j]$. Therefore, we first compute the nominal trajectory and store it. Then, the gradient of the cost functional and the required columns of the Jacobian of the constraints are calculated starting from the end of the vectors of control vectors. This is repeated for the vector of shooting nodes. In Figure 6.11, we indicated the proposed method by thick black arrows.
Using the illustration of Figure 6.10, the algorithm computes the Jacobian update by solving the dynamic of the control system in the top to bottom order of the arrows within Figure 6.11. Here, the aspect (1) is integrated, i.e. only the required updates within the Jacobian are actually computed. Within Figure 6.11, these are indicated in blue. Since we have stored the nominal trajectory, one solution of the differential equation is sufficient to compute one column of the Jacobian which yields the implementation to be computationally efficient.

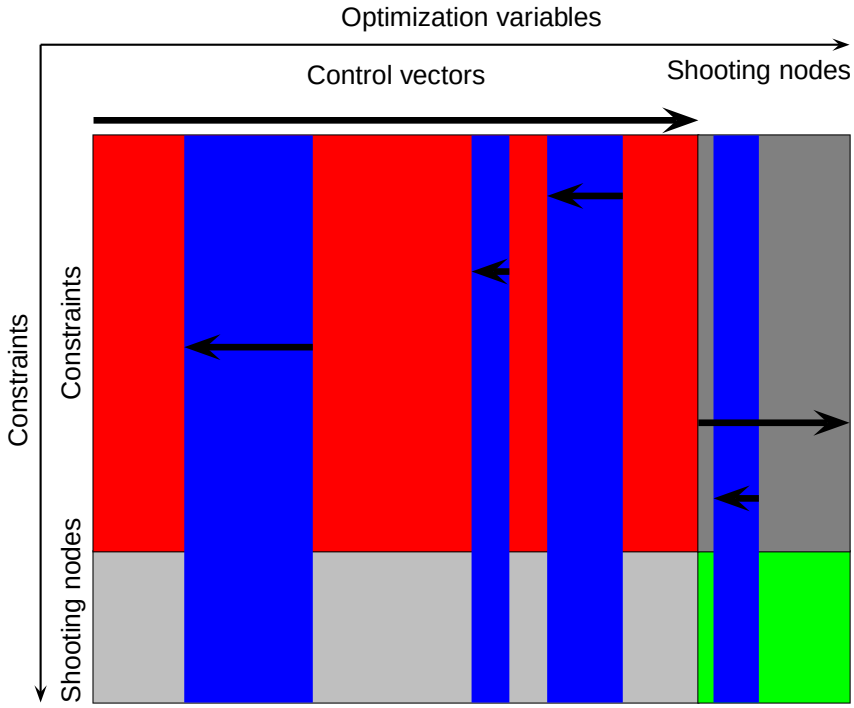


Figure 6.11: Sequence of calculations within a class `CacheOdeManager` object

The methodology of the `CacheOdeManager` is split up in several methods:

Method	Description
`comdgl`	Attaches the Lagrangian part of the cost function to the differential equation system
`compareCache`	Compares actual set of time set, initial values and control values to previous calls of the class `OdeSolve` object
`calcCache`	Calculates the value of the cost function whenever a combination of time set, initial values and control values has not been handled yet
`calcCacheDerived`	Calculates the value of the cost function whenever a combination of time set, initial values and optimization variables has not been handled yet
`calcObjectiveFunction`	Utilizes the `compareCache` and `calcCache` methods to compute the value of the cost function
`calcObjectiveFunctionGrad`	Utilizes the `compareCache`, `calcCache` and `calcCacheDerived` methods to compute the gradient of the cost function with respect to all optimization variables
`calcRestrFunction`	Utilizes the `compareCache` and `calcCache` methods to compute the value of the restriction function at one sampling instant
`calcRestrFunctionGrad`	Computes the Jacobian update for columns indicated by the array `needc` of the optimization routine using the methods `compareCache`, `calcCache` and `calcCacheDerived`

Table 6.8: Calculation methods within class `CacheOdeManager`

Remark 6.23
Class `CacheOdeManager` objects are not as robust as class `SimpleOdeManager` objects regarding the correctness issue. We experienced problems with this class if the derivatives of the Jacobian are small and changes in the optimization parameter sets result in changes of the function value which are lower than the tolerance level of the differential equation solver.

6.2.5.3 Class SyncOdeManager

Within the class `SimpleOdeManager` and the class `CacheOdeManager`, derivatives are computed by separately solving two differential equations based on the nominal and an appropriately changed set of parameters. This may lead to two different discretization grids. Moreover, if the change between the resulting two trajectories is smaller than the tolerance of the step–size controller, then the approximation of the derivative might be useless.
The class `SyncOdeManager` is designed to cope with this problem. The fundamental principle is to combine several (one plus the number of optimization variables) copies of the differential equation system and solve it by one call of the class `OdeSolve` object. The resulting discretization grid is identical by definition such that the correctness issue is satisfied. Similar to the class `SimpleOdeManager` and `CacheOdeManager` objects, the calls

```
IOdeManager * object_odemanager = new SyncOdeManager ;
delete object_odemanager;
```

Listing 6.27: Constructing/destructing call of a class `SimpleOdeManager` object

construct and destruct such a manager.
Within this class, the cacheing principle of class `CacheOdeManager` is applied to avoid repeated calls of the differential equation solver for identical optimization parameter sets. However, since the differential equation system is enlarged, the computational advantage is reduced.

Method	Description
`comdgl`	Combines differential equation systems and the Lagrangian part of the cost function to compute the gradient of the cost function
`diffcomdgl`	Combines differential equation systems and the Lagrangian part of the cost function to compute the Jacobian of the constraints
`compareCache`	Compares actual set of time set, initial values and control values to previous calls of the class `OdeSolve` object
`calcCache`	Calculates the value of the cost function whenever a combination of time set, initial values and control values has not been handled yet
`calcCacheDerived`	Calculates the value of the cost function whenever a combination of time set, initial values and optimization variables has not been handled yet
`calcObjectiveFunction`	Utilizes the `compareCache` and `calcCache` methods to compute the value of the cost function
`calcObjectiveFunctionGrad`	Utilizes the `compareCache` and `calcCacheDerived` methods to compute the gradient of the cost function with respect to all optimization variables
`calcRestrFunction`	Utilizes the `compareCache` and `calcCache` methods to compute the value of the restriction function at one sampling instant
`calcRestrFunctionGrad`	Computes the Jacobian update for columns indicated by the array `needc` of the optimization routine using the methods `compareCache` and `calcCacheDerived`

Table 6.9: Calculation methods within class `SyncOdeManager`

Concluding, the usage of the different implementations depends on the considered example:

- A class `SimpleOdeManager` object is efficient in terms of the internal usage of memory. However, due to unnecessary differential equation solver calls, it is computationally costy and may exhibit correctness problems.

- A class `SyncOdeManager` object is designed to treat the correctness aspect. The internal combination of the differential equation systems and the resulting effort to compute the update of the Jacobian are treated efficiently but the inherent repeated calculation of the nominal trajectory cannot be avoided. Moreover, combining the dynamics results in longer solution times of the differential equation, hence it is in general slower than a class `SimpleOdeManager` object.

- Implemented for speed, a class `CacheOdeManager` object outruns the other implementations concerning the calls of the differential equation solver and shows only small memory allocations. However, robustness problems may occur and have to be kept in mind.

6.3 Optimization

Within every step of the receding horizon control method, an optimal control problem has to be solved, see Section 2.4. Using the *PCC2* package this is done by calling a class `MinProg` object to solve the abstract minimization problem defined by a class `Discretization` object, see Section 6.2.4 for details.
In the following, we describe the purpose, implementation and setup of the optimization routines implemented within the *PCC2* package.

6.3.1 Class MinProg

Within the implementation of *PCC2* we used two different methods for nonlinear programming, so called active–set sequential quadratic programming (SQP) methods and interior point (IP) methods. The class `MinProg` is contained in the library **libminprog** and parents these solvers which are contained in the classes `SqpFortran`, `SqpNagC` and `BtIpOpt`, i.e. the libraries **libsqpf**, **libsqpnagc** and **libbtipopt**. It presents a general interface between the discretized version of the optimal control problem given by the class `Discretization` and the requested form of an optimization problem within the classes `SqpFortran`, `SqpNagC` and `BtIpOpt`, i.e. it represents an *abstract optimization problem*, see also Figure 6.8 for the interconnection of the classes defining this problem.

Remark 6.24
The class `BtIpOpt` containing a wraparound for the interior point method `IpOpt` has been implemented but is not yet optimized. For this reason, this class is not presented here.

6.3.1.1 Constructor / Destructor

Due to its parenting nature, a `MinProg` object is usually constructed by creating a class `SqpFortran`, `SqpNagC` or `BtIpOpt` object. This automatically initializes the compatible parts of these solvers. Any of the calls

```
btmb::MinProg::MinProg * object_minimizer = new SqpFortran ;
btmb::MinProg::MinProg * object_minimizer = new SqpNagC ;
btmb::MinProg::MinProg * object_minimizer = new BtIpOpt ;
```

Listing 6.28: Constructors of a class `SqpFortran`/`SqpNagC`/`BtIpOpt` object

can be used for this purpose. Similarly, the destruction of such an object also removes the class `MinProg` object.

6.3.1.2 Initialization / Calculation

The memory allocation for the internal minimization routine structures is handled upon initialization of a class `MinProg` object. In particular, the *cost function* and *restriction function structures* are created and initialized with the correct function pointers of the class `Model` provided by the calling class `Discretization` object.
Additionally, the parenting class `MinProg` offers the user the possibility to add a third stopping criterion to the optimizers. The first two criteria are standard in all implemented minimization routines and check whether a point satisfies the KKT conditions of Theorem 5.18 or if a maximal number of iterations has been reached, see also Sections 6.3.2 and 6.3.3 below. Here, a timing criterion is introduced which allows the user to stop the optimization process at real–time instant $t_0 + CT$ where t_0 is the real–time instant of the start of the optimization, $C > 0$ is a scaling factor and T represents the sampling time of the receding horizon controller. Hence, if a control shall be implemented at the time instant $T_0 + T$, the user may set the factor $C < 1$ to guarantee termination of the optimization before the real–time instant $t_0 + T$.

6.3.2 Class SqpFortran

The class `SqpFortran` contains the C++ wraparound of the FORTRAN77 SQP solver NLPQLP of K. Schittkowski, see also [47, 202]. It is included in the library **libsqpf** and the code is linked as external source in the C++ class.
The abstract optimization problem for this method is given by

$$\begin{array}{rll} \text{Minimize} & F(x) & \forall x \in \mathbb{R}^n \\ \text{ST.} & G_i(x) = 0, & i \in \mathcal{E} \\ & H_i(x) \geq 0, & i \in \mathcal{I} \end{array}$$

and the method uses the merit function

$$\widetilde{L}_\xi(x,\mu) := F(x) - \sum_{i \in J} \left(\mu_i - \frac{\xi_j}{2} A_i(x) \right) A_i(x) - \frac{1}{2} \sum_{i \in K} \frac{\mu_i^2}{\xi_j} \tag{6.3}$$

with $J := \mathcal{E} \cup \{ i : i \in \mathcal{I}, H_i(x) \leq \frac{\mu_j}{\xi_j} \}$, $K := (\mathcal{E} \cup \mathcal{I}) \setminus J$ and penalty parameter ξ controlling the degree of constraint violation. These parameters are chosen to guarantee a decrease in the merit function.
The optimization routine itself implements a modification of an active–set line search method, see Section 5.2.5, allowing for l parallel function calls. The stability and convergence issues of this algorithm have been treated in [47]. Moreover, it contains a non–monotone line search, i.e. an increase within the merit function is allowed. For further details, see Section 5.2.4.2. The update of the Jacobian of the constraints is computed using a reduced Hessian approximation, see Section 5.2.4.4.
Implementations of active–set line search methods are known to show good performance even for large–scale optimization problems with many optimization variables, cf. [203]. Within our receding horizon controller setting, this method shows outstanding performance for small optimization problems. For a detailed computing time analysis we refer to [88].

6.3.2.1 Constructors

There exist two constructors for a class `SqpFortran` object, that is

```
btmb::MinProg::MinProg * object_minimizer = new SqpFortran ;
btmb::MinProg::MinProg * object_minimizer = new SqpFortran (
   dimension_x, f, g, dimension_G, dimension_H, df, dg);
```

Listing 6.29: Constructors of a class `SqpFortran` object

The first one is the standard constructor which we use within the main programm of *PCC2*. The second one, however, is optional and can be used for third party codes to utilize this class. This constructor automatically calls the initialization routine which needs to be done manually in the first case.
The header of the second constructor already indicates that within this class the standard nomenclature of nonlinear optimization is used, see also the definition of the abstract optimization problem in Section 6.3.1 which is inherited by the class `SqpFortran`.
In particular, the function pointer `f` denotes the cost function and `g` the equality and inequality constraints. Note that for the usage of NLPQLP, `g` requires an internal ordering of the constraints, i.e. the first `dimension_G` elements are the equality constraints which are followed by the `dimension_H` inequality constraints. Moreover, the function pointer `df` represents the gradient of the cost function and `dg` the Jacobian of the (active) constraints.

6.3.2.2 Initialization

The SQP routine NLPQLP is configured using default values for all user setable options. Similar to the second constructor of this class, the initialization method is executed by calling

```
object_minimizer->init ( dimension_x, f, g, dimension_G,
   dimension_H, df, dg);
```

Listing 6.30: Initializing call of a class `SqpFortran` object

where the standard notation of nonlinear optimization applies as explained at the end of the previous section.
The problem specifications are handled by the initialization routine of class `MinProg`, see Section 6.3.1.2. Hence, we do not have to worry about handling the discretization and the function pointers anymore within this class.

Remark 6.25
The initialization of class ***`MinProg`*** *is written for any kind of optimization problem. Hence, if a standard optimization problem shall be solved outside our standard receding horizon control setting using an object of class* ***`MinProg`*** *object, the user can simply supply the size of the problem as well as cost function, restrictions and their derivatives respectively.*

Additionally, the routine allocates the memory used by the SQP algorithm NLPQLP. To this end several constants need to be set, that is

Variable	Default	Description
`dimension_x`		Number of optimization variables
`dimension_G`		Number of constraints
`dimension_H`		Number of equality constraints

Variable	Default	Description
nmax	dimension_x + 1	Row dimension of the Hessian of the cost functional
mmax	nrest + 1	Row dimension of the Jacobian of constraints
mnn2	nrest + 2dimension_x + 2	Number of Lagrangian multipliers
lwa	$9\texttt{nrest} + \lceil \frac{3}{2}\texttt{nmax}^2 \rceil + 33\texttt{nmax} + 200$	Size of working double array
lkwa	nmax + 10	Size of working integer array
lactive	2nrest + 10	Size of logical array of active constraints

Table 6.10: Constants within the class SqpFortran

Note that in our implementation the readily setable constant l which characterizes the number of parallel line searches is taken to be one.

Variable	Description
x	Initially, the first column of x has to contain starting values for the optimal solution. On return, x is replaced by the current iterate. In the driving program the row dimension of x has to be equal to nmax.
f	On return, f contains the cost function values at the final iterate x.
g	On return, g contains the constraint function values at the final iterate x. In the driving program the row dimension of g has to be equal to mmax.
gvalue1	gvalue1 is used to store the undisturbed values of the constraints to compute the Jacobian.
gvalue2	Similarly, gvalue2 is used to store the disturbed values of the constraints.
df	df contains intermediate gradients of the objective function.
dg	dg is used to store gradients of the active constraints at a current iterate x. The remaining rows are filled with previously computed gradients. In the driving program the row dimension of dg has to be equal to mmax.
u	u contains the multipliers with respect to the actual iterate stored in the first column of x. The first nrest locations contain the multipliers of the nonlinear constraints, the subsequent dimension_x locations the multipliers of the lower bounds, and the final dimension_x locations the multipliers of the upper bounds. At an optimal solution, all multipliers with respect to inequality constraints should be nonnegative.
d	The elements of the diagonal matrix of the LDL decomposition of the quasi–ewton matrix are stored in the one-dimensional array d.

Variable	Description
`c`	On return, `c` contains the last computed approximation of the Hessian matrix of the Lagrangian function stored in form of an LDL decomposition. `c` contains the lower triangular factor of an LDL factorization of the final quasi-Newton matrix (without diagonal elements, which are always one). In the driving program, the row dimension of `c` has to be equal to `nmax`.
`wa`	Working double array
`kwa`	Working integer array
`active`	Logical array of active constraints

Table 6.11: Memory allocation within class `SqpFortran`

Last, some variables for the optimization routine QPSOLVE need to be set. Note that these values massively influence the configuration and hence the result of the optimization.

Variable	Default	Description
`l`	1	Number of parallel systems, i.e. function calls during line search at predetermined iterates
`accuracy`	10^{-6}	Final accuracy of the optimizer, its value should not be smaller than the accuracy by which gradients are computed.
`accql`	10^{-8}	This constant is used by the QP solver to perform e.g. testing optimality conditions or whether a number is considered as zero. If `qccql` is less or equal to zero, then the machine precision is computed and subsequently multiplied by 10^4.
`stepmin`	accuracy	Steplength reduction factor, recommended is any value in the order of the accuracy by which functions are computed.
`maxfun`	20	This variable defines an upper bound for the number of function calls during the line search (e.g. 20). `maxfun` must not be greater than 50.
`maxit`	100	Maximum number of outer iterations, where one iteration corresponds to one formulation and solution of the quadratic programming subproblem, or, alternatively, one evaluation of gradients.
`max_nm`	0	Stack size for storing merit function values at previous iterations for non-monotone line search (e.g. 10). In case of `max_nm` = 0, monotone line search is performed. `max_nm` should not be greater than 50.
`tol_nm`	10^{-1}	Relative bound for increase of merit function value, if line search is not successful during the very first step. Must be nonnegative.
`lql`	*true*	If `lql = true`, the quadratic programming subproblem is to be solved with a full positive definite quasi-Newton matrix. Otherwise, a Cholesky decomposition is performed and updated, so that the subproblem matrix contains only an upper triangular factor.

Variable	Default	Description
`iprint`	0	Specification of the desired output level. `iprint` = 0: No output of the program. `iprint` = 1: Only a final convergence analysis is given. `iprint` = 2: One line of intermediate results is printed in each iteration. `iprint` = 3: More detailed information is printed in each iteration step, e.g. variable, constraint and multiplier values. `iprint` = 4: In addition to 'IPRINT=3', merit function and steplength values are displayed during the line search.
`iout`	6	Integer indicating the desired output unit number
`mode`	0	The parameter specifies the desired version of NLPQLP. `mode` = 0: Normal execution (reverse communication!). `mode` = 1: The user wants to provide an initial guess for the multipliers in `u` and for the Hessian of the Lagrangian function in `c` and `d` in form of an LDL decomposition.

Table 6.12: Input parameter of the class `SqpFortran`

6.3.2.3 Calculation

After creating and initializing a class `SqpFortran` object, the abstract optimization problem defined by the class `Discretization` within the class `MinProg` can now be solved by calling

```
object_minimizer->calcMin( x, fx, lb, ub);
```

Listing 6.31: Calculation call to a class `SqpFortran` object

Within the receding horizon control procedure this call is triggered by the function `calc` of a class `Discretization` object.
This command causes the routine NLPQLP to be executed solving the abstract optimization problem iteratively. In turn, this routine repeatedly demands calculations of the values of all functions, that is cost function and restrictions, and all derivatives, i.e. gradient of the cost function and Jacobian of the restrictions. These values are supplied by a class `IOdeManager` object, see Section 6.2.5 for details.
Upon termination, the flag `IFAIL` is set automatically. Here, the following outcome may occur:

Value	Description
`IFAIL` $= -2$	Compute gradient values with respect to the variables stored in `x`, and store them in `df` and `dg`. Only derivatives for active constraints `active[j]` = `true` need to be computed. Then call NLPQLP again.
`IFAIL` $= -1$	Compute objective function and all constraint values subject the variable `x` and store them in `f` and `g`. Then call NLPQLP again.
`IFAIL` $= 0$	The optimality conditions are satisfied.
`IFAIL` $= 1$	The algorithm has been stopped after `maxit` iterations.
`IFAIL` $= 2$	The algorithm computed an uphill search direction.
`IFAIL` $= 3$	Underflow occurred when determining a new approximation matrix for the Hessian of the Lagrangian.
`IFAIL` $= 4$	The line search could not be terminated successfully.
`IFAIL` $= 5$	Length of a working array is too short. More detailed error information is obtained with `iprint` > 0.
`IFAIL` $= 6$	There are false dimensions.
`IFAIL` $= 7$	The search direction is close to zero, but the current iterate is still infeasible.
`IFAIL` $= 8$	The starting point violates a lower or upper bound.
`IFAIL` $= 9$	Wrong input parameter.
`IFAIL` $= 10$	Internal inconsistency of the quadratic subproblem, division by zero.
`IFAIL` > 100	The solution of the quadratic programming subproblem has been terminated with an error message and `IFAIL` is set to `IFQL` $+ 100$, where IFQL denotes the index of an inconsistent constraint.

Table 6.13: Error flags of NLPQLP

Remark 6.26
As mentioned at the end of Section 5.2.6, the sequence of evaluations is important for the speed of the minimizer. The implementation of `NLPQLP` *demands a recomputation of parts of the variables* `df` *and* `dg` *if* `IFAIL`$= -2$. *This allows us to use information in the differential equation managers of class* ***`IOdeManager`*** *to simultaneously compute* `df` *and* `dg`.

6.3.3 Class SqpNagC

The class `SqpNagC` is, similar to the class `SqpFortran`, a wraparound of the procedure `nag_opt_nlp_solve` (`e04wdc`) of the *Numerical Algorithms Group*[2].
This method implements a modification of a trust–region SQP algorithm, see Section 5.2.6 for the general setting and [78] for details. The algorithm is designed to cope efficiently with large–scale optimization problems and few optimization variables. In [88], it has been shown that given such a situation, a class `SqpNagC` object outruns a class `SqpFortran` object in terms of speed. The algorithm itself distinguishes between certain types of constraints, that is box constraints, linear constraints and nonlinear constraints. This has

[2]Webpage: `http://www.nag.co.uk`

been introduced for reasons of computing efficiency. Moreover, upper and lower bounds have to be assigned for every constraint. Hence, the abstract optimization problem for this method is given by

$$\begin{array}{lll} \text{Minimize} & F(x) & \forall x \in \mathbb{R}^n \\ \text{ST.} & l \leq \begin{pmatrix} x \\ Ax \\ C(x) \end{pmatrix} & \leq u \end{array}$$

where $A \in \mathbb{R}^{n_l \times n_l}$ is a constant matrix and $C : \mathbb{R}^n \to \mathbb{R}^p$ represents the nonlinear constraints. The constraints are internally transformed to $G(x) \geq 0$. The merit function used by this algorithm is given by

$$\widetilde{L}_\xi(x, \mu) := F(x) + \mu^\top (G(x) - s) + \frac{1}{2} \sum_{i=1}^{r} \xi_i \left(G_i(x) - s_i\right)^2 \tag{6.4}$$

where ξ is a penalty vector and s the vector of slackness variables. The penalty vector ξ is updated using the linearly constrained least–squares problem

$$\begin{array}{lll} \text{Minimize} & \|\xi\|_2^2 & \forall \xi \in \mathbb{R}^r \\ \text{ST.} & \nabla \widetilde{L}_\xi(x, \mu) \leq -\frac{1}{2} d^{(k)} B^{(k)} d^{(k)} & \end{array}$$

defining a penalty vector $\xi^\star$ such that $\nabla \widetilde{L}_\xi(x, \mu) \leq -\frac{1}{2} d^{(k)} B^{(k)} d^{(k)}$ for any $\xi \geq \xi^\star$, i.e. the directional derivative is sufficiently negative to ensure a decrease in the merit function (6.4). The required decrease in the penalty vector to ensure convergence is implemented by setting

$$\overline{\xi}_i = \max\{\xi_i^\star, \hat{\xi}_i\} \quad \text{where } \hat{\xi}_i = \begin{cases} \xi_i & \text{if } \xi_i < 4(\xi_i^\star + \Delta_\xi) \\ \sqrt{\xi_i(\xi_i^\star + \Delta_\xi)} & \text{else} \end{cases}$$

Δ_ξ is set to one at start and is doubled whenever strict monotonicity of $\|\overline{\xi}_i\|_2$, either decreasing or increasing, after at least two consecutive steps is violated which prohibits oscillation in the penalty vector and ensures $\overline{\xi} \to \xi$.
The minimization itself is implemented as a trust–region SQP, see Algorithm 5.61, using a *reduced Hessian BFGS update* and a *watchdog algorithm*, see Sections 5.2.4.4 and 5.2.4.2 respectively. Moreover, it contains an *elastic* mode to deal with inconsistent linearizations, see Section 5.2.4.3.

6.3.3.1 Constructors

In order to be easily exchangable within the *PCC2* program, the constructors

```
btmb::MinProg::MinProg * object_minimizer = new SqpNagC ;
btmb::MinProg::MinProg * object_minimizer = new SqpNagC (
   dimension_x, f, g, dimension_G, dimension_H, df, dg);
```

Listing 6.32: Constructors of a class `SqpNagC` object

and the initialization routine

```
object_minimizer->init ( dimension_x, f, g, dimension_G,
   dimension_H, df, dg);
```

Listing 6.33: Initializing call of a class `SqpNagC` object

of class `SqpNagC` and class `SqpFortran` objects are identical, see Sections 6.3.2.1 and 6.3.2.2 for details.

6.3.3.2 Initialization

Again, pointers to the problem details such as cost functional and its gradient, constraints and the corresponding Jacobian as well as their parameter are set by the class `MinProg`, see Section 6.3.1.2.
The optimization routine of a class `SqpNagC` object, however, requires a different memory setup than a class `SqpFortran` object which has to be provided according to the discretized problem. Hence, the following variables are declared and allocated within the initialization routine:

Variable	Description
`nctotl`	Number of optimization variables
`x`	Initially, `x` contains an initial guess for the optimal solution. On return, `x` is replaced by the current iterate.
`f`	On return, `f` contains the cost function values at the final iterate `x`.
`g`	On return, `g` contains the nonlinear constraint function values at the final iterate `x`.
`lb`, `ub`	`lb` must contain the lower bounds, `ub` the upper bounds for all the constraints. Here, the first n elements of each array must contain the bounds on the variables, the next n_l elements the bounds for the general linear constraints (if any) and the next p elements the bounds for the general nonlinear constraints (if any). To specify a non-existent upper/lower bound, set `lb[j-1]` $\leq \pm$`bigbnd` where `bigbnd` is the optional argument *Infinite Bound Size* of the procedure. To specify the j–th constraint as an equality, set `lb[j-1]` = `ub[j-1]` $= a$ where $\lvert a \rvert <$ `bigbnd`.
`ccon`	If there exist nonlinear constraints then `ccon[i-1]` contains the value of the i–th constraint function $C_i(x)$.
`cjac`	This array contains the Jacobian matrix of the nonlinear constraint functions at the final iterate, i.e. `cjac[i][j]` represents the partial derivative of the i–th constraint function with respect to the j–th variable.
`clambda`	The values of the QP multipliers from the last QP subproblem are stored within this variable.
`grad`	Represents the gradient of the cost function at the last iterate.
`hess`	This array contains the Hessian of the Lagrangian at the final estimate.
`gvalue1`	`gvalue1` is used to store the undisturbed values of the constraints to compute the Jacobian.
`gvalue2`	Similarly, `gvalue2` is used to store the disturbed values of the constraints.

Variable	Description
`state`	Must not be modified in any way by the user since it contains internal information required for functions in this algorithm.
`comm`	This variable is a communication structure with a generic pointer. It can be used to communicate double and integer pointers with no casting.
`fail`	The structure `fail` is an NAG internal error structure which can be used to identify errorcodes delivered by a routine. It contains the 5 members: `code` refers to a fixed error or warning code. `print` needs to be set before calling a routine refering to `fail`. If it is set to `Nag_True` the error message `message` is printed, `Nag_False` otherwise. `message` contains the error text. `errnum` may deliver additional information about the warning or error. `(*handler)(char*,int,char*)` handles errors.

Table 6.14: Variables and arrays within the class `SqpNagC`

Moreover, the integer array `istate` needs not to be set if a cold start of the routine is performed. In case of a warm start, it needs to contain expectations whether the constraints are active as shown in Table 6.15. Note that the variable `istate` contains all restrictions in the same sequence as in `lb` and `ub`.

`istate`	Description
0	Constraint is not expected to be active.
1	Constraint is expected to be active on lower bound.
2	Constraint is expected to be active on upper bound.
3	Constraint is an equality constraint.

Table 6.15: Setting expectation of active constraints in class `SqpNagC`

On exit of the procedure, this variable contains all information about whether a constraint is active or not. This can be read out using the identification shown in Table 6.16.

`istate`	Description
-2	Lower bound is violated by more than δ.
-1	Upper bound is violated by more than δ.
0	Both bounds are satisfied with more than δ.
1	Lower bound is active (to within δ).
2	Upper bound is active (to within δ).
3	Both bounds are equal and are satisfied (to within δ).

Table 6.16: Obtaining information about active constraints in class `SqpNagC`

where δ is an internal value.

6.3.3.3 Calculation

Similar to class `SqpFortran` the optimization problem is solved using a class `SqpNagC` object by calling

```
object_minimizer->calcMin ( x, fx, lb, ub, rpar, ipar);
```

Listing 6.34: Calculation call to a class `SqpNagC` object

Within our receding horizon control procedure, this function is called by the method `calc` of a class `Discretization` object.
Similar to class `SqpFortran`, this method causes the SQP algorithm to be executed, see Section 6.3.2.3 for details. On exit, the variable `fail` contains either the code for successful termination of the algorithm or the resulting error code, that is

Code	Description
`NW_TOO_MANY_ITER`	Maximal number of iterations reached
`NE_ALLOC_FAIL`	Internal error: memory allocation failed
`NE_ALLOC_INSUFFICIENT`	Internal memory allocation was insufficient
`NE_BAD_PARAM`	Basis file dimensions do not match this problem
`NE_BASIS_FAILURE`	Error in basis package
`NE_DERIV_ERRORS`	User-supplied function computes incorrect constraint or objective derivatives
`NE_E04WCC_NOT_INIT`	Initialization function `nag_opt_nlp_init` has not been called
`NE_INT`	Error in problem setup
`NE_INT_2`	Error in problem setup
`NE_INT_3`	Error in problem setup
`NE_INTERNAL_ERROR`	Internal Error occurred, set option Print File and examine output
`NE_REAL_2`	Error in problem setup
`NE_UNBOUNDED`	Problem seems to be unbounded
`NE_USER_STOP`	User-supplied function requested termination
`NE_USRFUN_UNDEFINED`	User-supplied function not defined
`NW_NOT_FEASIBLE`	Problem appears to be infeasible

Table 6.17: Error codes of e04wdc

Internally, the algorithm repeatedly calls functions `confun` and `objfun` which in our receding horizon controller setting are linked back to a class `IOdeManager` object, a class `Model` object and a class `OdeSolve` object, compare Figure 6.8.

Function	Description
`confun`	User-supplied function to evaluate the constraints and the Jacobian of the constraints of the problem
`objfun`	User-supplied function to evaluate the cost function and its gradient

Table 6.18: Functions of e04wdc

As mentioned at the end of Section 5.2.6, these computations are in a bad order in the receding horizon context since the gradient of the cost function and the Jacobian of the constraints are computed separately. Note, however, that a class `CacheOdeManager` or `SyncOdeManager` object may compensate for this since previous evaluations of the dynamic of the system are stored.

6.4 Differential Equation Solver

Since the receding horizon controller is designed to solve a sequence of shifted optimal control problems resulting from a discretization of the dynamic of the system, its restrictions and the corresponding cost functional, the values of the state of the system at the discretization points must be known. To compute these values, differential equation solvers can be used. The solver, in turn, require initial states of the states and all parameters of the dynamic. In our receding horizon controller context, these values correspond to the optimization variable (5.10) and the initial values (5.4).
The implemented differential equation solvers are contained in the library **libodesol2**. For ease of implementation, the configuration of a solver should be stored separately which induces the classes `OdeSolve` and `OdeConfig`.

6.4.1 Class OdeSolve

The differential equation solvers are parented by the class `OdeSolve`. In the current version, the following methods are implemented:

Shortcut	Name and Order of Convergence
`DoPri5`	explicit Runge-Kutta method of order 5(4)
`DoPri853`	explicit Runge-Kutta method of order 8(5,3)
`Radau5`	implicit Runge-Kutta method of order 5
`Radau5913`	implicit Runge-Kutta method of variable order (switches automatically between orders 5, 9, and 13)
`RecurseSequence`	Implements discrete–time systems
`Euler`	explicit Euler method

Table 6.19: Classes derived from class `OdeSolve`

Within the *PCC2* package, objects of this class are constructed by class `Model` objects. Yet, they can also be constructed using one of the calls

```
OdeSolve * object_solver = new btmb::OdeSol2::DoPri5 ;
OdeSolve * object_solver = new btmb::OdeSol2::DoPri853 ;
OdeSolve * object_solver = new btmb::OdeSol2::Radau5 ;
OdeSolve * object_solver = new btmb::OdeSol2::Radau5913 ;
OdeSolve * object_solver = new btmb::OdeSol2::RecurseSequence ;
OdeSolve * object_solver = new btmb::OdeSol2::Euler ;
```

Listing 6.35: Constructing call of a class `OdeSolve` object

and destructed by calling the corresponding destructor of the chosen class

```
delete object_solver;
```

Listing 6.36: Destructing call of a class `OdeSolve` object

Internally, the class `OdeSolve` manages the common functions of all subclasses, i.e.

Function	Description
`reset( func, config )`	Sets dynamic and configuration of the object
`resize( dimension )`	Changes the dimension of the dynamic
`refreshConfig`	Adopts all changes within the `OdeConfig` object
`calc( t, rpar, ipar )`	Solves the dynamic with terminal time `t` and parameter sets `rpar` and `ipar`
`assertMemory`	Allocates the required memory
`assertConfig`	Submits changes of configuration
`assertValidDGL( dgl )`	Allows subclasses to check whether a newly added differential equation system can be solved
`abstractCalc( t, rpar, ipar )`	Abstract method which is modified by the subclasses and called by the `calc` method

Table 6.20: Functions managed within class `OdeSolve`

which are inherited or replace by the subclasses. Additionally, all shared parameters

Variable	Description
`dimension`	Dimension of the dynamic
`func`	Function structure of type `OdeFunction` representing the right hand side
`config`	Configuration object of class `OdeConfig`
`t`	Current time instant
`y`	Current state vector
`paramdata`	Structure of type `ODEPARAMS` to save the parameter of the differential equation system
`did`	Return value of the called method: `did` = 1: Successful computation `did` = 2: Successful computation interrupted by output `did` = −1: Inconsistent input `did` = −2: Maximal number of steps too small `did` = −3: Minimal step size too small `did` = −4: Problem appears to be stiff

Variable	Description
`iout`	Outputparameter: `iout` ≥ 1: Output of the state vector for every accepted step `iout` $= 0$: No output but dummy function required
`rtol`	Relative tolerance
`atol`	Absolute tolerance
`rtol_vec`	Vector of relative tolerances
`atol_vec`	Vector of absolute tolerances
`itol`	Distinguishing parameter for tolerances: `itol` $= 0$: `rtol`, `atol` are scalar `itol` ≥ 0: `rtol`, `atol` are vectors
`solout`	Function pointer for continuous output
`initial_value_set`	Boolean to check for supplied initial value
`work`	Working array of double values
`lwork`	Size of the working array `work`
`iwork`	Working array of integer values
`liwork`	Size of the working array `iwork`

Table 6.21: Parameters managed within class `OdeSolve`

are contained inside this parenting class. Within our receding horizon controller setting, these values are set by the class `Model` object which initialized such an `OdeSolve` object, either directly or indirectly using an class `OdeConfig` object.

6.4.2 Class OdeConfig

The class `OdeConfig` can be used to construct an object to set up a given differential equation solver from class `OdeSolve`. It is purely for configuration purposes and no calculations of any kind are done by the methods of such an object.
Since not all differential equation solver derived from class `OdeSolve` reveal the same parameters, the class `OdeConfig` is parenting the classes `DopriConfig`, `RadauConfig` and `Radau5913Config` and manages the following shared parameters:

Variable	Default	Description
`rtol`	10^{-6}	Relative tolerance
`atol`	10^{-6}	Absolute tolerance
`rtol_vec`		Vector of relative tolerances
`atol_vec`		Vector of absolute tolerances
`max_steps`	10^7	Maximal number of integration steps
`ss_fac1`	0.2	Security factor representing the lower bound in the relative change of the step size

Variable	Default	Description
`ss_fac2`	10	Security factor representing the upper bound in the relative change of the step size
`max_stepsize`	Length of interval	Largest allowable step size
`initial_stepsize`	Calculated numerically	Initial step size
`safety_factor`	0.9	Security factor

Table 6.22: Parameters of class `OdeConfig`

Note that the class `RecurseSequence` does not contain any parameters since it deals with discrete–time systems. Within the subclasses, the remaining parameters are stored.

Variable	Default	Description
`DoPri5` and `DoPri853`		(requires DoPriConfig)
`stiff_teststeps`	1000	Allowed number of steps to test for stiffness of the dynamic
`beta`	0.04	Stability parameter of the step size control
`Radau5` and `Radau5913`		(requires RadauConfig)
`enable_hessenberg`	0	Uses Hessenberg transformation if set to 1
`newton_steps`	7	Maximal number of Newton steps
`jac_step`	0.001	Step size bound for the Jacobian to be recomputed
`strat`	Gustaffson	Step size control strategy (Gustaffson or Classic)
`sf_fac1`	1.0	Lower bound for the relative change in the step size `sf_fac1` $\leq \frac{h_{\text{new}}}{h_{\text{old}}}$
`sf_fac2`	1.2	Upper bound for the relative change in the step size $\frac{h_{\text{new}}}{h_{\text{old}}} \leq$ `sf_fac2`
`Radau5913`		(requires Radau5913Config)
`isValidStage`	true	Validity check variable
`os_inc`	0.002	Lower bound for contraction to increase the order of the solver
`os_dec`	0.8	Upper bound for contraction to decrease the order of the solver
`os_fac1`	0.8	Order is only decreased if `os_fac1` $\leq \frac{h_{\text{new}}}{h_{\text{old}}}$
`os_fac2`	1.2	Order is only decreased if $\frac{h_{\text{new}}}{h_{\text{old}}} \leq$ `os_fac2`
`s_min`	3	Minimal number of stages
`s_max`	7	Maximal number of stages
`s_first`	3	Choice of stages on initialization

Table 6.23: Parameters of class `OdeConfig`

All these parameters can be changed to adapt the solver using the set– and get– methods of the class. For numerical results regarding those parameters which exhibit a major impact on the computing time, see Section 8.1.

6.5 Getting started

After explaining the setup and interaction of the main classes in the *PCC2* package, we now show how to implement an example. Here, we focus on explaining the essential parts required in the main program and the example class. A complete program is contained in Appendix A. Note that the program requires the libraries

- libmpc2 (containing the RHC procedures, the discretization and odemanager),
- libodesol2 (providing ODE solvers),
- libbtmbutils (for basic functions like timing or saving data) and
- libsqpf (chosen minimization routine)

to be compiled appropriately such that the classes can be used.

6.5.1 An Example Class

Since the class `Model` defines an abstract example, see Section 6.1.1, we have to define a class derived from the class `Model` to define our example. Here, the constructor has the form

```
InvertedPendulum::InvertedPendulum()
        : btmb::MPC2::Model ( new btmb::OdeSol2::DoPri853(),
                              new btmb::OdeSol2::DoPriConfig(),
                              4,
                              1,
                              1,
                              4 )
{
    setDoubleParameter ( 0, 0.007 );
    setDoubleParameter ( 1, 1.25 );
    setDoubleParameter ( 2, 9.81 );
    setDoubleParameter ( 3, 0.197 );

    getOdeConfig()->setTolerance ( 1E-10, 1E-10 );
}
```

Listing 6.37: Constructor of an example of a class `Model` object

defining the `OdeSolve` and `OdeConfig` objects and setting the size of the problem, in order of appearance, *dimension of the state*, *dimension of the control*, *number of restrictions* and *number of model parameters*.
The destructor is given by

```
InvertedPendulum::~InvertedPendulum()
{
    delete getOdeSolver();
    delete getOdeConfig();
}
```

Listing 6.38: Destructor of an example of a class `Model` object

Next, we have to define all functions stated in Table 6.1. The *dynamics* of the control system as described in (7.8) – (7.11) within our example class is implemented via

```
void InvertedPendulum::dglFunction ( double t, double * x, double
    * u, double * dx )
{
    dx[0] = x[1];
    dx[1] = - params[2] * sin ( x[0] ) / params[1] - u[0] * cos (
        x[0] ) - params[0] * atan ( 1.0e3 * x[1] ) * x[1] * x[1] -
        ( 4.0 * x[1] / ( 1.0 + 4.0 * x[1] * x[1] ) + 2.0 * atan (
       2.0 * x[1] ) / PI ) * params[3];
    dx[2] = x[3];
    dx[3] = u[0];
}
```

Listing 6.39: Implementation of the dynamic of the system

The *cost functional* (7.12) needs to be separated into two functions,

```
double InvertedPendulum::objectiveFunction ( double t, double * x
   , double * u )
{
    double sinxpi = sin ( x[0] - PI );
    double cosy = cos ( x[1] );
    double temp = ( 1.0 - cos ( x[0] - PI ) ) * ( 1.0 + cosy *
       cosy );

    return 1.0e-1 * pow ( 3.51 * sinxpi * sinxpi + ( 4.82 *
       sinxpi + 2.31 * x[1] ) * x[1] + 2.0 * temp * temp + 1.0 * x
       [2] * x[2] + 1.0 * x[3] * x[3], 2.0 );
}
```

Listing 6.40: Implementation of the Lagrangian part of the cost functional

and

```
double InvertedPendulum::pointcostFunction ( int length, int
   horizon, double * t, double * x, double * u )
{
    if ( length < horizon )
    {
        return 0.0;
    }
    else
    {
        double * lastx = &x[ ( horizon - 1 ) * 4];

        double sinxpi = sin ( lastx[0] - PI );
        double cosy = cos ( lastx[1] );
        double temp = ( 1.0 - cos ( lastx[0] - PI ) ) * ( 1.0 +
           cosy * cosy );

        return pow ( 3.51 * sinxpi * sinxpi + ( 4.82 * sinxpi +
           2.31 * lastx[1] ) * lastx[1] + 2.0 * temp * temp + 1.0
           * pow ( lastx[2], 2.0 ) + 1.0 * lastx[3] * lastx[3],
           2.0 );
```

```
    }
}
```

Listing 6.41: Implementation of the Mayer part of the cost functional

Here, the discrete part of the cost functional implementation simulates a Mayer–term. These two parts are weigthed equally using the *weighting* method

```
void InvertedPendulum::getObjectiveWeight ( double & obj_weight,
   double & pointcost_weight )
{
    obj_weight = 1.0;
    pointcost_weight = 1.0;
}
```

Listing 6.42: Weighting of the two cost functional parts

Even if there are no non–constant *restrictions*, we have to define the corresponding function

```
void InvertedPendulum::restrictionFunction ( double t, double * x
   , double * u, double * fx )
{
}
```

Listing 6.43: Implementation of the non–constant restrictions

The box constraints (7.15) and (7.16) are set within the methods

```
void InvertedPendulum::getControlBounds ( double * lb, double *
   ub )
{
    lb[0] = -5.0;
    ub[0] =  5.0;
}
```

Listing 6.44: Implementation of the constant control restrictions

and

```
void InvertedPendulum::getModelBounds ( double * lb, double * ub
   )
{
    lb[0] = -INF;
    lb[1] = -INF;
    lb[2] = -5.0;
    lb[3] = -10.0;

    ub[0] = INF;
    ub[1] = INF;
    ub[2] = 5.0;
    ub[3] = 10.0;
}
```

Listing 6.45: Implementation of the constant state restrictions

The default initial value for the inverted pendulum is defined in

```
void InvertedPendulum::getDefaultState ( double * x )
{
    x[0] = 0.0;
    x[1] = 0.0;
    x[2] = 0.0;
    x[3] = 0.0;
}
```

Listing 6.46: Setting default initial values of the state

and similarly the default control vector is given in

```
void InvertedPendulum::getDefaultControl ( double * u )
{
    u[0] = 0.0;
}
```

Listing 6.47: Memory allocation and definition of the initial guess for the control

Here, we use two multiple shooting points which are situated on the first dimension of the state at sampling instants 13 and 15. The corresponding function are given by

```
int InvertedPendulum::getShootingDataLength ( int horizon )
{
    return 2;
}

int InvertedPendulum::getMaxShootingDataLength ( int maxhorizon )
{
    return 2;
}

void InvertedPendulum::getShootingDataInfo ( int horizon, btmb::
   MPC2::STARTDATA * sdata )
{
    sdata[0].horizontindex = 13;
    sdata[0].varindex = 0;
    sdata[1].horizontindex = 15;
    sdata[1].varindex = 0;
}
```

Listing 6.48: Definition and setting of the multiple shooting nodes

To set the values of these points, we use the function

```
void InvertedPendulum::eventBeforeMPC ( int horizon, double * t,
   double * x, double * sdatavalues )
{
    sdatavalues[0] = 3.1415;
    sdatavalues[1] = 3.1415;
}
```

Listing 6.49: Automated resets before each step in the RHC algorithm

6.5.2 A Main Program

The control problem itself is defined within the *main program*, cf. Figure 6.2. According to the mentioned hierarchy, a *minimization routine* derived from the class `MinProg` and a linking *differential equation manager* method from class `IOdeManager` are required to construct an object of class `MPC`.

Remark 6.27
The differential equation solver object and its configuration are predefined by the constructor of the example derived from the class `Model`*, see Listing 6.37. Hence, only the optimizer, the manager and the example must be known at this stage.*

```
Model * object_model = new InvertedPendulum();
IOdeManager * object_odemanager = new SimpleOdeManager();
btmb::MinProg::MinProg * object_minimizer = new SqpFortran();
InvertedPendulum * object_modelspecify = ( ( InvertedPendulum
    * ) object_model );

btmb::MPC2::MPC * mpc_problem = new MPC ( INF );
mpc_problem->reset ( object_odemanager, object_minimizer,
   object_model, HORIZON );
```

Listing 6.50: Definition and Initialization of a class `MPC` object

The construction of the controller object from class `MPC` configures the minimizer object implicitly using standard values, e.g. for optimality tolerance and iteration bounds. These settings may be specified within the main program via

```
SqpFortran * object_minimizerSpecify = ( ( SqpFortran * )
   object_minimizer );
object_minimizerSpecify->setAccuracy ( 1E-8 );
object_minimizerSpecify->setMaxFun ( 20 );
object_minimizerSpecify->setMaxIterations ( 1000 );
object_minimizerSpecify->setLineSearchTol ( 0.1 );
```

Listing 6.51: Configuration of a class `MinProg` object

Since the example does neither provide *initial values* for the state, nor a *time grid* for the discretization nor an *initial guess* of the optimal control, these arrays have to be defined within the main program. Here, the user can utilize methods predefined in the class `MPC` and the class `Model` objects to avoid segmentation faults.

```
double * u, * next_u, * t, * x;
next_u = ( double * ) malloc ( sizeof ( double ) *
   object_model->getDimensionControl() );
x = ( double * ) malloc ( sizeof ( double ) * object_model->
   getDimension() );
object_modelspecify->getDefaultState ( x );
mpc_problem->allocateMemory ( t, u );
mpc_problem->initCalc ( t, u );
```

Listing 6.52: Memory allocation of the state, control and time variable

Setting appropriate values may now be performed using external sources. For simulation purposes, as in our example, the default methods of the example class can be used.

```
    for ( int i = 0; i < HORIZON; i++ )
    {
        object_model->getDefaultControl ( & u[i * object_model->
           getDimensionControl() ] );
    }
```

Listing 6.53: Setting initial state and control values

Moreover, the initial time grid needs to be set

```
    for ( int i = 0; i < HORIZON + 1; i++ )
    {
        t[i] = i * H_NEW;
    }
```

Listing 6.54: Implementation of the time grid

Now, the *computation of an optimal control* given the time grid and the initial values reduces to the command

```
        try
        {
            mpc_problem->calc ( x );
        }
        catch ( btmb::MinProg::sqpException e )
        {
            cout << e.what() << endl;
        }
```

Listing 6.55: Solving the optimal control problem

where the try-and-catch block reveals possible errors or warnings produced by the underlying instances. The *time shift* is done via

```
        mpc_problem->shiftHorizon ( x, next_u, H_NEW, mstep );
```

Listing 6.56: Calling the receding horizon step

and resulting values can be stored using an additional class `SaveData` object from the library **libbtmbutils** or transmitted to the actuator implementing the control signal at the plant.

```
    SaveData * save = new SaveData ( object_model->getDimension()
       , object_model->getDimensionControl() );
```

```
        save->save2Files ( t, x, u );
```

Listing 6.57: Saving results using internal routines of a class `SaveData` object

Finally, the allocated memory should be freed and all objects should be destructed.

Chapter 7

Examples

Within our analysis of the receding horizon controller and its performance, we consider three types of problems. First, shown in Section 7.1, a one–dimensional heat equation is presented. In Chapter 8, we use this problem to demonstrate the influence of the chosen differential equation solver on the computing time. In particular, computing tolerance, stiffness of the dynamic and of the number of dimensions are considered, see Section 8.1. Moreover, using this benchmark problem, the different linking methods of the minimization and the differential equation solver are tested in Section 8.2.

Our second model is the inverted pendulum on a cart problem which we describe in Section 7.2. Using this particular example, we show that our implementation of the receding horizon controller *PCC2* is real–time applicable in Section 8.3. Moreover, the effects of the chosen optimality tolerance, the number of steps within the optimization routine and the length of the optimization horizon are discussed. Additionally, the influence of the initial guess of the control and of the multiple shooting knots is depicted.

Our last model is a sampled–data redesign problem. Here, we derive a continuous–time feedback for an arm–rotor–platform model, see Section 7.3. The resulting trajectory is then used as a reference for the receding horizon control problem to design a sampled–data control law. These tracking type problems allow for a wide variety of optimization horizons N since close to the reference very small horizons are sufficient for tracking. However, if the trajectory under control is far from the reference, then usually large horizons are required. Hence, this type of problem is well suited for analyzing the suboptimality degree of the resulting closed–loop control. Additionally, it allows us to effectively test adaptation strategies for the horizon length. These issues are treated in Sections 8.4 and 8.5 respectively.

7.1 Benchmark Problem — 1D Heat Equation

In order to analyze how the speed of elements of the package *PCC2* depend on the size of a problem, we consider the one–dimensional heat equation

$$\lambda \triangle y(x,t) - y_t(x,t) = u, \qquad y(x,t) \in \Omega = [0,1]$$

with distributed control, Dirichlet boundary conditions

$$y(x,t) = 0, \qquad x \in \Gamma = \{0,1\}$$

and variable stiffness parameter λ.

Remark 7.1
During the development of PCC2 more difficult partial differential equations have been treated as well, see e.g. [88]. Here, we choose this simple example since all effects of size and stiffness on the computing time can be experienced.

For our implementation we use the initial values

$$y(x,0) = -(x-0.5)^2 + 0.25. \tag{7.1}$$

We solve this partial differential equation by applying the method of lines, see e.g. [130]. For this purpose, we discretize the state space Ω using the grid

$$G_M := \left\{ \frac{i}{M-1} \,\middle|\, i = \{0,\dots,M-1\} \right\} \tag{7.2}$$

with gridwidth $w = 1/(M-1)$ and $M \geq 3$. The resulting system of ordinary differential equations is given by:

$$i = 1: \qquad \dot{y}(x_i,t) = \lambda \frac{y(x_{i+1},t) - 2y(x_i,t)}{h^2} + u_i(t) \tag{7.3}$$

$$i = 2,\dots,M-1: \qquad \dot{y}(x_i,t) = \lambda \frac{y(x_{i+1},t) - 2y(x_i,t) + y(x_{i-1},t)}{h^2} + u_i(t) \tag{7.4}$$

$$i = M: \qquad \dot{y}(x_i,t) = \lambda \frac{-2y(x_i,t) + y(x_{i-1},t)}{h^2} + u_i(t) \tag{7.5}$$

In order to analyze the implemented optimization routines, we use the cost functional

$$J_N(x_0,u) := \sum_{i=0}^{N-1} l(x(i),u(i)) + F(x(N)) \tag{7.6}$$

with stage cost $l(x(i),u(i)) := \int_{t_i}^{t_{i+1}} \|x(t)\|_2^2 dt$ and terminal cost $F(x(N)) := \|x(t_N)\|_2^2$. Additionally, the set of restrictions

$$R_p := \{x_i(\cdot) \geq 0 \,\|\, i \in \{1,\cdots,p\}\} \quad \text{with} \quad R_0 = \emptyset \tag{7.7}$$

is considered where the number of restrictions $p \in \mathbb{N}_0$ is variable.
Results using this example can be found in Section 8.1 and 8.2.

7.2 Real–time Problem — Inverted Pendulum on a Cart

Our second example is the standard "inverted pendulum on a cart" problem shown in Figure 7.1. This problem has two categories of equilibria, the stable downright position and the unstable upright position. Here, the task is to stabilize one of the unstable upright equilibria.
Our aim using this problem, however, is not only to stabilize an upright position but also to compute the control online. Therefore, the calculation of a control, which shall be applied at a sampling instant $t_i = t_0 + iT$, has to be finished before the real–time t_{real} equals t_i. In Section 8.3, we analyze the impact of all modifiable parameters of the receding horizon controller and illustrate how these parameters can be adjusted to speed up the solution process.

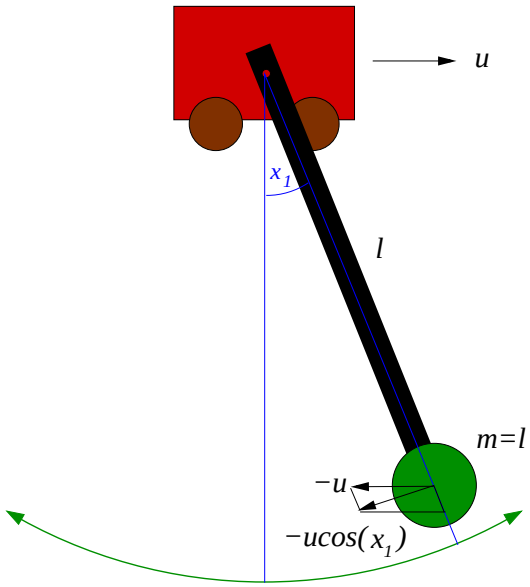


Figure 7.1: Inverted Pendulum on a Cart

The dynamic of the inverted pendulum on a cart can be expressed via the system of ordinary differential equations

$$\begin{aligned}
\dot{x}_1(t) &= x_2(t) \\
\dot{x}_2(t) &= -\frac{g}{l}\sin(x_1(t)) - \frac{k_L}{l}x_2(t)|x_2(t)| - u(t)\cos(x_1(t)) - k_R\mathrm{sgn}(x_2(t)) \\
\dot{x}_3(t) &= x_4(t) \\
\dot{x}_4(t) &= u(t)
\end{aligned}$$

with parameters $g = 9.81$, $l = 1.25$, $k_L = 0.007$ and $k_R = 0.197$. Here, $x_1(\cdot)$ denotes the angle of the pendulum, $x_2(\cdot)$ the angular velocity of the pendulum, $x_3(\cdot)$ the position and $x_4(\cdot)$ the velocity of the cart. Within this system the second equation is numerically problematic for an adaptive differential equation solver:

$$\dot{x}_2(t) = -\frac{g}{l}\sin(x_1(t)) - \underbrace{\frac{k_L}{l}x_2(t)|x_2(t)|}_{\text{non diff'able}} - \underbrace{u(t)\cos(x_1(t))}_{\text{measurable}} - \underbrace{k_R\mathrm{sgn}(x_2(t))}_{\text{discontinuous}}.$$

Hence, this differential equation causes a reduction of the order of consistence of the differential equation solver to $\mathcal{O}(n)$ at points where the right hand side is not differentiable. Moreover, stability of the solver is lost at the discontinuity points.
Since we are going to use a receding horizon controller, i.e. a sample–and–hold strategy for the contol $u(\cdot)$, we do not have to modify the term concerning the control since within the receding horizon controller setting the points of discontinuity are known in advance and can be added to the discretization grid. Hence, the order of consistence is preserved although $u(\cdot)$ is discontinuous.
For both other terms we use smooth approximations which allow us to mimic the behaviour of the original equation but retain the oder of consistence of the differential equation solver:

$$\dot{x}_1(t) = x_2(t) \tag{7.8}$$

$$\begin{aligned}
\dot{x}_2(t) = &-\frac{g}{l}\sin(x_1(t)) - \frac{k_L}{l}\arctan(1000x_2(t))x_2^2(t) - u(t)\cos(x_1(t)) \\
&- k_R\left(\frac{4ax_2(t)}{1+4(ax_2(t))^2} + \frac{2\arctan(bx_2(t))}{\pi}\right)
\end{aligned} \tag{7.9}$$

$$\dot{x}_3(t) = x_4(t) \tag{7.10}$$
$$\dot{x}_4(t) = u(t). \tag{7.11}$$

For this system the upright unstable equilibria are given by the set $\mathcal{S} = \{((k+1)\pi, 0, 0, 0)^\top \mid k \in 2\mathbb{Z}\}$. To develop a cost functional we make use of the geometry of the dynamic of the system. To stabilize an arbitrary point of this set, we consider the cost functional

$$J_N(x_0, u) := 0.1 \sum_{i=0}^{N-1} l(x(i), u(i)) + F(x(N)) \tag{7.12}$$

where the stage costs are given by

$$l(x(i), u(i)) := 0.1 \int_{t_i}^{t_{i+1}} \Big(3.51 \sin(x_1(t) - \pi)^2 + 4.82 \sin(x_1(t) - \pi) x_2(t) + 2.31 x_2(t)^2 + 2\left((1 - \cos(x_1(t) - \pi)) \cdot (1 + \cos(x_2(t))^2)\right)^2 + x_3(t)^2 + x_4(t)^2\Big)^2 dt \tag{7.13}$$

and the terminal cost is defined via

$$F(x(N)) := \Big(3.51 \sin(x_1(t_N) - \pi)^2 + 4.82 \sin(x_1(t_N) - \pi) x_2(t_N) + 2.31 x_2(t_N)^2 + 2\left((1 - \cos(x_1(t_N) - \pi)) \cdot (1 + \cos(x_2(t_N))^2)\right)^2 + x_3(t_N)^2 + x_4(t_N)^2\Big)^2. \tag{7.14}$$

Moreover, we impose constraints on the state and the control vectors by defining

$$\mathbb{X} := \mathbb{R} \times \mathbb{R} \times [-5, 5] \times [-10, 10] \tag{7.15}$$
$$\mathbb{U} := [-5, 5]. \tag{7.16}$$

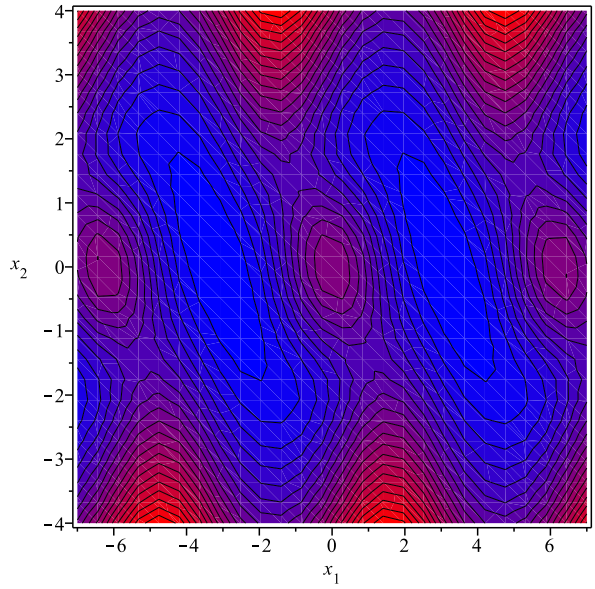


Figure 7.2: Contourplot of the cost functional (7.12) considering only x_1 and x_2. The minima of this function are $((k+1)\pi, 0, 0, 0)^\top$, $k \in 2\mathbb{Z}$ whereas points $(k\pi, 0, 0, 0)^\top$, $k \in 2\mathbb{Z}$ are local maxima located at the downward stable equilibria.

Remark 7.2

Note that every point in the set $\mathcal{S}$ is treated equally by this cost functional. Hence, the outcome of the receding horizon control problem is depending on the initial value x_0 of the system, but also on the optimization horizon N. While the first dependency is due to the periodicity of the cost functional, the latter one follows from possible changes of the optimum if different horizon lengths are considered.

7.3 Tracking Problem — Arm-Rotor-Platform Model

In contrast to the inverted pendulum example, we construct the control problem for the arm–rotor–platform model as illustrated in Figure 7.3 as a digital redesign problem. The redesign approach consists of the two separate steps of constructing a continuous–time feedback for the continuous–time plant without consideration of sampling effects and modifying the resulting control law to be applicable in the digital setting.
Within this work, we described the so called emulation design previously in Section 1.3.3, see also [8,40,70,134]. For the present example, we follow the optimization based approach proposed in [173, 179]. To this end, we require the following:

(i) A continuous–time control law $u = u(x)$ can be designed which (globally) asymptotically stabilizes the continuous–time closed-loop system $\dot{x}(t) = f(x(t), u(x(t)))$

(ii) The control law is implemented in a sample–and–hold fashion, i.e. $u(t) = u(t_k) =$ const for all $t \in [t_k, t_{k+1})$, $k \in \mathbb{N}$ where $t_k = kT$ are the sampling instances.

Now, our aim is to use the continuous–time control law to redesign it in a digital way. For the receding horizon control setting described in Chapter 2, this can be done straight forward by incorporating the continuous–time closed–loop system as a reference trajectory for the state of the system within our receding horizon controller setup. To this end, we use the tracking cost functional (2.4)

$$J_\infty(x_0, u) := \int_0^\infty L\left(x_u(t, x_0) - x_{\mathrm{ref}}(t), u(t)\right) dt.$$

Minimizing this function causes our algorithm to compute a sampled–data feedback law which keeps the solution of the digitally controlled system $x_u(\cdot, x_0)$ as close as possible to the reference system $x_{\mathrm{ref}}(\cdot)$. In particular, we do not consider the continuous–time control law to be re–evaluated for the predicted digitally controlled state trajectories at the sampling instances on the optimization horizon during the optimization. Instead, we compute the continuous–time closed–loop trajectory $x_{\mathrm{ref}}(\cdot)$ on the entire horizon once and try to compensate for the sampling effects using an optimization of the digital control.

Remark 7.3
The emulation approach requires some robustness with respect to the sampling error caused by the discretization. Since this error depends on the length of the sampling interval, the typical way to treat this issue is to sufficiently reduce the sampling period, see also Remark 1.42. Since the receding horizon controller takes the sampling effects directly into account and minimizes the resulting deviation, we expect not only better results but also larger sampling periods than using the Emulation approach.

Remark 7.4
For our implementation the redesign idea results in doubling the differential equation system if no closed formula for the reference trajectory can be computed. Hence, this may cause increased computing times for our differential equation solver. Upon implementation, however, this approach turned out to be very effective since the given reference trajectory allowed for a significant reduction of the optimization horizon, see e.g. [121, 226] for tracking results and [88] for standard implementation.

The following theorem is proved in [173] and shows that this redesign approach recovers the asymptotic stability property of the continuously controlled system up to a practical region, compare also Definitions 1.35 and 1.44. Moreover, this property is obtained in a sub–optimal way in the sense of Theorem 3.50.

Theorem 7.5
Suppose that the following conditions hold:

- *The running cost $l(\cdot,\cdot)$ (and the terminal cost $F(\cdot)$) are continuous;*
- *$\mathbb{U}$ is bounded;*
- *The continuous–time system $\dot{x}(t) = f(x(t), u(x(t)))$ is globally asymptotically stable;*
- *There exists $r_0 > 0$ and $\gamma \in \mathcal{K}_\infty$ with*

$$l(y,u) \geq \max\left\{ \max_{\|x\| \leq 2\|y\|} \|f(x,u)\|, \gamma(\|y\|) \right\} \qquad \forall \|y\| \geq r_0;$$

- *$f(\cdot,\cdot)$ and $u(\cdot)$ are locally Lipschitz in their arguments;*
- *The value function is such that for some $\alpha \in \mathcal{K}_\infty$ we have that $V(y) \leq \alpha(\|y\|)$ for all $y = (x^\top, x_{ref}^\top)^\top \in \mathbb{R}^{2n}$.*

Then, there exists a function $\beta \in \mathcal{KL}$ such that for each pair of strictly positive real numbers (Δ, δ) there exists $N_1 \in \mathbb{Z}_{\geq 1}$ such that for all $y = (x^\top, x_{ref}^\top)^\top \in B_\Delta(0)$ and $N \geq N_1$ the solutions (2.16) satisfies

$$\|y(t)\| \leq \max\{\beta(\|y(0)\|, t), \delta\} \qquad \forall t \geq 0.$$

Remark 7.6
Different to the emulation design, the optimization based approach is designed to incorporate and automatically compensate the intersampling error. Hence, also long sampling periods may be considered without loosing stability if the requirements for the system and the receding horizon controller stated in Theorem 7.5 hold.

Here, we start by defining the system under control, and then derive a continuous–time control law which allows us to track a given function. Hence, the arm–rotor–platform model in the receding horizon control implementation is a double tracking example since first the continuous–time control law tracks some given external signal. Secondly, the redesigned digital control law tries to keep the state trajectories close to the continuously controlled ones.

Within this example, we consider a robot arm (A) which is driven by a motor (R) via a flexible joint. This motor is mounted on a platform (P) which is again flexibly connected to a fixed base (B). Moreover, we assume that there is no vertical force. This system has five degrees of freedom. Three of those degrees are for the platform, translational positions x and y as well as rotational position β. The fourth degree is the rotational position α of the motor and last the rotational position θ of the arm itself. As shown in Figure 7.3 all coordinates are defined relative to the fixed base and all angles are measured with respect to the x–axis of the base B.
In order to design a continuous–time static state feedback we follow the approach shown in Chapter 7 of [72]. Our design goal is to stear the system such that the position of the

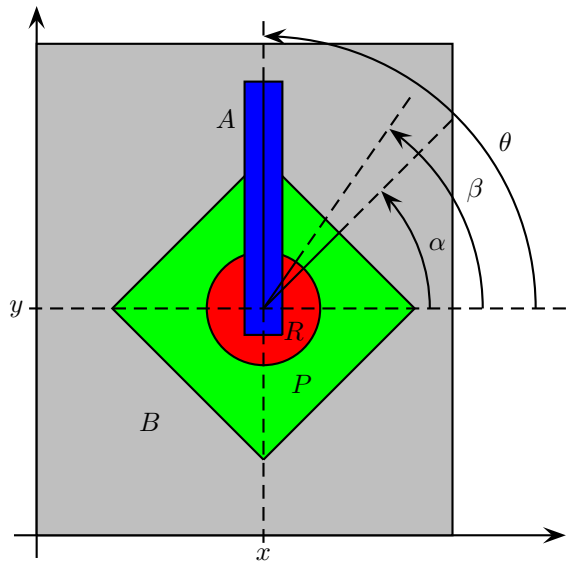


Figure 7.3: Graphical illustration of the ARP problem

arm relative to the platform, i.e. the angle $\zeta = \theta - \beta$, tracks a given reference signal. Note that this task is not simple since both connections of the rotor are flexible. Here, we assume that the reference signal and its derivatives are known and available to the controller. Moreover, we assume that the relative positions and velocities $(\theta-\beta)$, $(\dot\theta-\dot\beta)$, $(\alpha-\beta)$ and $(\dot\alpha-\dot\beta)$ are supplied to the controller.
Using the Lagrange formulation we obtain the model

$$M\ddot{x} - mr\left[\ddot{\theta}\sin(\theta) + \dot{\theta}^2\cos(\theta)\right] + b_1\dot{x} + k_1 x = 0 \tag{7.17}$$

$$M\ddot{y} - mr\left[\ddot{\theta}\cos(\theta) + \dot{\theta}^2\sin(\theta)\right] + b_1\dot{y} + k_1 y = 0 \tag{7.18}$$

$$I\ddot{\theta} + mr\left[-\ddot{x}\sin(\theta) + \ddot{y}\cos(\theta)\right] + b_3(\dot\theta - \dot\alpha + k_3(\theta-\alpha) = 0 \tag{7.19}$$

$$J\ddot{\alpha} - b_3(\dot\theta - \dot\alpha + k_3(\theta-\alpha) + b_4(\dot\alpha-\dot\beta) = u \tag{7.20}$$

$$D\ddot{\beta} + b_2\dot\beta + k_2\beta - b_4(\dot\alpha-\dot\beta) = -u. \tag{7.21}$$

To compute the stabilizing control law we assume that the rotational motion of the platform is not present. This simplification is justified since the rotational spring constant k_2 of the platform/base connection is large and the viscous friction coefficient b_4 of the rotor/platform connection is small in comparison to k_2.
Due to this assumption we can neglect (7.21). We use the change of coordinates

$$\begin{pmatrix}\eta_1\\ \eta_2\\ \eta_3\\ \eta_4\end{pmatrix} = \begin{bmatrix}\sin(\theta) & 0 & -\cos(\theta) & 0\\ 0 & \sin(\theta) & 0 & -\cos(\theta)\\ \cos(\theta) & 0 & \sin(\theta) & 0\\ 0 & \cos(\theta) & 0 & \sin(\theta)\end{bmatrix}\begin{pmatrix}x\\ \dot x\\ y\\ \dot y\end{pmatrix} + \frac{mr}{M}\begin{pmatrix}0\\ -\dot\theta\\ 1\\ 0\end{pmatrix} \tag{7.22}$$

with inverse transformation

$$\begin{pmatrix}x\\ \dot x\\ y\\ \dot y\end{pmatrix} = \begin{bmatrix}\sin(\theta) & 0 & \cos(\theta) & 0\\ 0 & \sin(\theta) & 0 & -\cos(\theta)\\ -\cos(\theta) & 0 & \sin(\theta) & 0\\ 0 & -\cos(\theta) & 0 & \sin(\theta)\end{bmatrix}\begin{pmatrix}\eta_1\\ \eta_2 + \frac{mr}{M}\dot\theta\\ \eta_3 - \frac{mr}{M}\\ \eta_4\end{pmatrix}.$$

Now, we can differentiate (7.22) using (7.17), (7.18). Hence, we obtain the following first set of differential equations:

$$\begin{pmatrix} \dot{\eta}_1 \\ \dot{\eta}_2 \\ \dot{\eta}_3 \\ \dot{\eta}_4 \end{pmatrix} = \begin{bmatrix} 0 & 1 & \dot{\theta} & 0 \\ -\frac{k_1}{M} & -\frac{b_1}{M} & 0 & \dot{\theta} \\ -\dot{\theta} & 0 & 0 & 1 \\ 0 & -\dot{\theta} & -\frac{k_1}{M} & -\frac{b_1}{M} \end{bmatrix} \begin{pmatrix} \eta_1 \\ \eta_2 \\ \eta_3 \\ \eta_4 \end{pmatrix} + \frac{mr}{M} \begin{pmatrix} 0 \\ -b_1\dot{\theta} \\ 0 \\ k_1 \end{pmatrix}$$

Moreover, we get

$$\left[I - \frac{(mr)^2}{M}\right] \ddot{\theta} + \left[b_3 + b_1(\frac{(mr)^2}{M}\right] \dot{\theta} + \frac{mr}{M}(k_1\eta_i + b_1\eta_2) = k_3\alpha + b_3\dot{\alpha} \tag{7.23}$$

by reformulating (7.19) using the changed coordinates. Hence, by (7.20), (7.23) we obtain the second set of differential equations:

$$\begin{pmatrix} \dot{\theta} \\ \ddot{\theta} \\ \dot{\alpha} \\ \ddot{\alpha} \end{pmatrix} = \begin{bmatrix} 0 & 1 & \dot{\theta} & 0 \\ -a_1 & -a_2 & a_1 & a_3 \\ 0 & 0 & 0 & 1 \\ a_4 & a_5 & -a_4 & -a_5 - a_6 \end{bmatrix} \begin{pmatrix} \theta \\ \dot{\theta} \\ \alpha \\ \dot{\alpha} \end{pmatrix} - \begin{pmatrix} 0 & 0 \\ p_1 & p_2 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \eta_1 \\ \eta_2 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ \frac{1}{J} \end{pmatrix} u$$

Renaming the coordinates $(\eta_1, \eta_2, \eta_3, \eta_4, \theta, \dot{\theta}, \alpha, \dot{\alpha}) =: (x_1, \ldots, x_8)$, the model can be described by the differential equation system

$$\dot{x}_1(t) = x_2(t) + x_6(t)x_3(t) \tag{7.24}$$

$$\dot{x}_2(t) = -\frac{k_1}{M}x_1(t) - \frac{b_1}{M}x_2(t) + x_6(t)x_4(t) - \frac{mr}{M^2}b_1x_6(t) \tag{7.25}$$

$$\dot{x}_3(t) = -x_6(t)x_1(t) + x_4(t) \tag{7.26}$$

$$\dot{x}_4(t) = -x_6(t)x_2(t) - \frac{k_1}{M}x_3(t) - \frac{b_1}{M}x_4(t) + \frac{mr}{M^2}k_1 \tag{7.27}$$

$$\dot{x}_5(t) = x_6(t) \tag{7.28}$$

$$\dot{x}_6(t) = -a_1x_5(t) - a_2x_6(t) + a_1x_7(t) + a_3x_8(t) - p_1x_1(t) - p_2x_2(t) \tag{7.29}$$

$$\dot{x}_7(t) = x_8(t) \tag{7.30}$$

$$\dot{x}_8(t) = a_4x_5(t) + a_5x_6(t) - a_4x_7(t) - (a_5 + a_6)x_8(t) + \frac{1}{J}u(t). \tag{7.31}$$

In order to design the feedback law, we shorten notation by defining the vectors and matrices

$$[\eta(t)] := \begin{pmatrix} x_1(t) & x_2(t) & x_3(t) & x_4(t) \end{pmatrix}^T, \quad [\chi(t)] := \begin{pmatrix} x_5(t) & x_6(t) & x_7(t) & x_8(t) \end{pmatrix}^T,$$

$$[F(x_6(t))] := \begin{pmatrix} 0 & 1 & x_6(t) & 0 \\ -\frac{k_1}{M} & -\frac{b_1}{M} & 0 & x_6(t) \\ -x_6(t) & 0 & 0 & 1 \\ 0 & -x_6(t) & -\frac{k_1}{M} & -\frac{b_1}{M} \end{pmatrix}, \; [G(x_6(t))] := \begin{pmatrix} 0 \\ -\frac{mrb_1}{M^2}x_6(t) \\ 0 \\ \frac{mrk_1}{M^2} \end{pmatrix},$$

$$[A] := \begin{pmatrix} 0 & 1 & 0 & 0 \\ -a_1 & -a_2 & a_1 & a_3 \\ 0 & 0 & 0 & 1 \\ a_4 & a_5 & -a_4 & -(a_5 + a_6) \end{pmatrix}, \; [E] := \begin{pmatrix} 0 & 0 \\ -p_1 & -p_2 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}, \; [B] := \begin{pmatrix} 0 \\ 0 \\ 0 \\ \frac{1}{J} \end{pmatrix}$$

Hence, equations (7.24)-(7.31) can be rewritten as

$$[\dot{\eta}(t)] = [F(x_6(t))] \cdot [\eta(t)] + [G(x_6(t))] \tag{7.32}$$
$$[\dot{\chi}(t)] = [A] \cdot [\chi(t)] + [E] \cdot [\eta(t)] + [B] \cdot [u]. \tag{7.33}$$

Moreover, we introduce the row vector $[p] := \begin{pmatrix} p_1 & p_2 & 0 & 0 \end{pmatrix}$. Within the vectors and matrices the following shortcuts are used:

$$a_1 = \frac{k_3 M}{MI - (mr)^2} \qquad a_4 = \frac{k_3}{J} \qquad p_1 = \frac{mr}{MI - (mr)^2} k_1$$
$$a_2 = \frac{b_3 M^2 - b_1 (mr)^2}{M[MI - (mr)^2]} \qquad a_5 = \frac{b_3}{J} \qquad p_2 = \frac{mr}{MI - (mr)^2} b_1$$
$$a_3 = \frac{b_3 M}{MI - (mr)^2} \qquad a_6 = \frac{b_4}{J}$$

For this particular example, we use the parameter data shown in Table 7.1:

M	5.0	Total mass of arm, rotor and platform
m	0.5	Mass of arm
r	0.3	Distance from the A/R joint to arm cener of mass
I	0.06	Moment of inertia of arm about A/R joint
J	0.005	Moment of inertia of rotor
D	0.5	Moment of inertia of platform
k_1	64.0	Translational spring constant of P/B connection
k_2	3600	Rotational spring constant of P/B connection
k_3	8.0	Rotational spring constant of A/R joint
b_1	1.6	Translational friction coefficient of P/B connection
b_2	12.0	Rotational friction coefficient of P/B connection
b_3	0.04	Rotational friction coefficient of A/R connection
b_4	0.007	Rotational friction coefficient of R/P connection

Table 7.1: Parameters of the arm–rotor–platform model

Here, we follow the approach shown in Chapter 7 of [72] and choose not to track $\zeta(t) = x_5(t)$ but the modified output

$$\zeta(t) = x_5(t) - \frac{a_3}{a_1 - a_2 a_3} \left[x_6(t) - a_3 x_7(t) \right]. \tag{7.34}$$

Remark 7.7
The chosen output $\zeta(t)$ is close to $x_5(t)$ since we have $a_1 \approx 140$, $a_2 \approx a_3 \approx 0.7$.

This output has relative degree 4, that is the control $u(t)$ appears explicitly within the fourth derivative of $\zeta(t)$:

$$\begin{aligned}
\dot{\zeta}(t) &= x_6(t) - \frac{a_3}{a_1 - a_2 a_3} \Big(- a_1 x_5(t) - a_2 x_6(t) + a_1 x_7(t) - [p] \cdot [\eta(t)] \Big) \\
&= \frac{a_1}{a_1 - a_2 a_3} x_6(t) - \frac{a_3}{a_1 - a_2 a_3} \Big(- a_1 x_5(t) + a_1 x_7(t) - [p] \cdot [\eta(t)] \Big) \qquad (7.35) \\
\ddot{\zeta}(t) &= \frac{a_1}{a_1 - a_2 a_3} \Big(- a_1 x_5(t) - a_2 x_6(t) + a_1 x_7(t) + a_3 x_8(t) - [p] \cdot [\eta(t)] \Big)
\end{aligned}$$

$$
\begin{aligned}
&- \frac{a_3}{a_1 - a_2 a_3}\Big(- a_1 x_6(t) + a_1 x_8(t) - [p] \cdot [[F(x_6(t))] \cdot [\eta(t)] + [G(x_6(t))]] \Big) \\
&= \frac{a_1}{a_1 - a_2 a_3}\Big(- a_1 x_5(t) - a_2 x_6(t) + a_1 x_7(t) - [p] \cdot [\eta(t)]\Big) \\
&\quad - \frac{a_3}{a_1 - a_2 a_3}\Big(- a_1 x_6(t) - [p] \cdot [[F(x_6(t))] \cdot [\eta(t)] + [G(x_6(t))]] \Big) \qquad (7.36) \\
\zeta^{(3)}(t) &= \frac{a_1}{a_1 - a_2 a_3}\Big(- a_1 x_6(t) - a_2\Big(- a_1 x_5(t) - a_2 x_6(t) + a_1 x_7(t) + a_3 x_8(t) \\
&\quad - [p] \cdot [\eta(t)]\Big) + a_1 x_8(t) - [p] \cdot [[F(x_6(t))] \cdot [\eta(t)] + [G(x_6(t))]] \Big) \\
&\quad - \frac{a_3}{a_1 - a_2 a_3}\Bigg(- a_1\Big(- a_1 x_5(t) - a_2 x_6(t) + a_1 x_7(t) + a_3 x_8(t) - [p] \cdot [\eta(t)]\Big) \\
&\quad - [p] \cdot \Bigg(\Bigg[\Bigg[\frac{\partial F(x_6(t))}{\partial x_6(t)}\Bigg] \cdot [\eta(t)] + \Bigg[\frac{\partial G(x_6(t))}{\partial x_6(t)}\Bigg]\Bigg] \\
&\quad \Big(- a_1 x_5(t) - a_2 x_6(t) + a_1 x_7(t) + a_3 a_8(t) - [p] \cdot [\eta(t)]\Big) \\
&\quad + [F(x_6(t))] \cdot [[F(x_6(t))] \cdot [\eta(t)] + [G(x_6(t))]] \Bigg)\Bigg) \\
&= \frac{1}{a_1 - a_2 a_3}\Bigg(- a_1^2\Big(x_6(t) - x_8(t)\Big) \\
&\quad + \Big(- a_1 x_5(t) - a_2 x_6(t) + a_1 x_7(t) + a_3 x_8(t) - [p] \cdot [\eta(t)]\Big) \\
&\quad \Bigg(- a_1(a_2 - a_3) + a_3[p] \cdot \Bigg[\Bigg[\frac{\partial F(x_6(t))}{\partial x_6(t)}\Bigg] \cdot [\eta(t)] + \Bigg[\frac{\partial G(x_6(t))}{\partial x_6(t)}\Bigg]\Bigg] \Bigg) \\
&\quad + \Big(- a_1[p] + a_3[p] \cdot [F(x_6(t))] \Big) \cdot [[F(x_6(t))] \cdot [\eta(t)] + [G(x_6(t))]] \Bigg) \qquad (7.37) \\
\zeta^{(4)}(t) &= \frac{1}{a_1 - a_2 a_3}\Bigg(- a_1^2\Big(- a_1 x_5(t) - a_2 x_6(t) + a_1 x_7(t) + a_3 x_8(t) - [p] \cdot [\eta(t)] \\
&\quad - a_4 x_5(t) - a_5 x_6(t) + a_4 x_7(t) + (a_5 + a_6) x_8(t) + \frac{1}{J} u(t)\Big) \\
&\quad + \Big(- a_1 x_6(t) - a_2\Big(- a_1 x_5(t) - a_2 x_6(t) + a_1 x_7(t) + a_3 x_8(t) - [p] \cdot [\eta(t)]\Big) \\
&\quad + a_1 x_8(t) + a_3\Big(a_4 x_5(t) + a_5 x_6(t) - a_4 x_7(t) - (a_5 + a_6) x_8(t) + \frac{1}{J} u(t)\Big) \\
&\quad - [p] \cdot [[F(x_6(t))] \cdot [\eta(t)] + [G(x_6(t))]] \Big) \\
&\quad \Bigg(- a_1(a_2 - a_3) + a_3[p] \cdot \Bigg[\Bigg[\frac{\partial F(x_6(t))}{\partial x_6(t)}\Bigg] \cdot [\eta(t)] + \Bigg[\frac{\partial G(x_6(t))}{\partial x_6(t)}\Bigg]\Bigg] \Bigg) \\
&\quad + 2\Big(- a_1 x_5(t) - a_2 x_6(t) + a_1 x_7(t) + a_3 x_8(t) - [p] \cdot [\eta(t)]\Big) \\
&\quad \Bigg(a_3[p] \cdot \Bigg[\frac{\partial F(x_6(t))}{\partial x_6(t)}\Bigg] \cdot [[F(x_6(t))] \cdot [\eta(t)] + [G(x_6(t))]] \Bigg)
\end{aligned}
$$

$$
\begin{aligned}
&+\Big(-a_1[p]+a_3[p]\cdot[F(x_6(t))]\Big)\cdot\Bigg(\Big[\Big[\frac{\partial F(x_6(t))}{\partial x_6(t)}\Big]\cdot[\eta(t)]+\Big[\frac{\partial G(x_6(t))}{\partial x_6(t)}\Big]\Big]\\
&\Big(-a_1x_5(t)-a_2x_6(t)+a_1x_7(t)+a_3x_8(t)-[p]\cdot[\eta(t)]\Big)\\
&+[F(x_6(t))]\cdot[[F(x_6(t))]\cdot[\eta(t)]+[G(x_6(t))]]\Bigg)\Bigg)\\
=\;&\frac{1}{a_1-a_2a_3}\Bigg(\Big(-a_1x_5(t)-a_2x_6(t)+a_1x_7(t)+a_3x_8(t)-[p]\cdot[\eta(t)]\Big)\\
&\Bigg(-a_1^2+a_1a_2(a_2-a_3)+\Big(-(a_1+a_2a_3)[p]+a_3[p]\cdot[F(x_6(t))]\Big)\\
&\cdot\Big[\Big[\frac{\partial F(x_6(t))}{\partial x_6(t)}\Big]\cdot[\eta(t)]+\Big[\frac{\partial G(x_6(t))}{\partial x_6(t)}\Big]\Big]\\
&+2a_3[p]\Big[\frac{\partial F(x_6(t))}{\partial x_6(t)}\Big]\cdot[[F(x_6(t))]\cdot[\eta(t)]+[G(x_6(t))]]\Bigg)\\
&+\Big(a_4x_5(t)+a_5x_6(t)-a_4x_7(t)-(a_5+a_6)x_8(t)+\frac{1}{J}u(t)\Big)\\
&\Bigg(a_1^2+a_3\Bigg(-a_1(a_2-a_3)+a_3[p]\cdot\Big[\Big[\frac{\partial F(x_6(t))}{\partial x_6(t)}\Big]\cdot[\eta(t)]+\Big[\frac{\partial G(x_6(t))}{\partial x_6(t)}\Big]\Big]\Bigg)\Bigg)\\
&+\Big(-a_1[p]+a_3[p]\cdot[F(x_6(t))]\Big)\cdot[F(x_6(t))]\cdot[[F(x_6(t))]\cdot[\eta(t)]+[G(x_6(t))]]\\
&+\Big(-a_1(x_6(t)-x_8(t))-[p]\cdot[[F(x_6(t))]\cdot[\eta(t)]+[G(x_6(t))]]\Big)\\
&\Bigg(-a_1(a_2-a_3)+a_3[p]\cdot\Big[\Big[\frac{\partial F(x_6(t))}{\partial x_6(t)}\Big]\cdot[\eta(t)]+\Big[\frac{\partial G(x_6(t))}{\partial x_6(t)}\Big]\Big]\Bigg)\Bigg).
\end{aligned}
\tag{7.38}
$$

Note that the coefficient of $u(\cdot)$ is affine in $\eta(\cdot)$ and vanishes only for large values of $\eta(\cdot)$. Therefore, the model has relative output degree four with respect to the output variable $\zeta(\cdot)$ over the expected operating range of $\eta(\cdot)$. Hence, we can solve (7.38) for $u(\cdot)$ and obtain the control law

$$
\begin{aligned}
u(t)=&\frac{J}{a_1^2+a_3[p]\cdot\Big[\Big[\frac{\partial F(x_6(t))}{\partial x_6(t)}\Big]\cdot[\eta(t)]+\Big[\frac{\partial G(x_6(t))}{\partial x_6(t)}\Big]\Big]}\\
&\Bigg(-\Big(-a_1x_5(t)-a_2x_6(t)+a_1x_7(t)+a_3x_8(t)-[p]\cdot[\eta(t)]\Big)\\
&\Bigg(-a_1^2+a_1a_2(a_2-a_3)+\Big(-(a_1+a_2a_3)[p]+a_3[p]\cdot[F(x_6(t))]\Big)\\
&\cdot\Big[\Big[\frac{\partial F(x_6(t))}{\partial x_6(t)}\Big]\cdot[\eta(t)]+\Big[\frac{\partial G(x_6(t))}{\partial x_6(t)}\Big]\Big]\\
&+2a_3[p]\Big[\frac{\partial F(x_6(t))}{\partial x_6(t)}\Big]\cdot[[F(x_6(t))]\cdot[\eta(t)]+[G(x_6(t))]]\Bigg)\\
&-\Big(a_4x_5(t)+a_5x_6(t)-a_4x_7(t)-(a_5+a_6)x_8(t)\Big)
\end{aligned}
$$

$$
\begin{aligned}
&\Bigg(a_1^2 + a_3\Bigg(- a_1(a_2 - a_3) + a_3[p] \cdot \Bigg[\Bigg[\frac{\partial F(x_6(t))}{\partial x_6(t)}\Bigg] \cdot [\eta(t)] + \Bigg[\frac{\partial G(x_6(t))}{\partial x_6(t)}\Bigg]\Bigg] \Bigg)\Bigg) \\
&- \Big(- a_1[p] + a_3[p] \cdot [F(x_6(t))] \Big) \cdot [F(x_6(t))] \cdot [[F(x_6(t))] \cdot [\eta(t)] + [G(x_6(t))]] \\
&- \Big(- a_1(x_6(t) - x_8(t)) - [p] \cdot [[F(x_6(t))] \cdot [\eta(t)] + [G(x_6(t))]] \Big) \\
&\Bigg(- a_1(a_2 - a_3) + a_3[p] \cdot \Bigg[\Bigg[\frac{\partial F(x_6(t))}{\partial x_6(t)}\Bigg] \cdot [\eta(t)] + \Bigg[\frac{\partial G(x_6(t))}{\partial x_6(t)}\Bigg]\Bigg] \Bigg) \\
&+ (a_1 - a_2 a_3) v(t) \Bigg)
\end{aligned} \tag{7.39}
$$

where $v(t)$ is the external command signal. Here, we think of this command signal as a reference for the modified output (7.34) for the *continuously controlled system*. To obtain the required continuous–time control law, we combine (7.39) and (7.38) and obtain the chain integrator $\zeta^{(4)}(t) = v(t)$. This allows us to design $v(t)$ such that $\zeta(\cdot)$ asymptotically tracks any given reference signal $\zeta_{\text{ref}}(\cdot)$ by defining

$$
\begin{aligned}
v(t) := \zeta_r^{(4)}(t) &- c_3(\zeta^{(3)}(t) - \zeta_{\text{ref}}^{(3)}(t)) - c_2(\ddot{\zeta}(t) - \ddot{\zeta}_{\text{ref}}(t)) \\
&- c_1(\dot{\zeta}(t) - \dot{\zeta}_{\text{ref}}(t)) - c_0(\zeta(t) - \zeta_{\text{ref}}(t))
\end{aligned} \tag{7.40}
$$

with design parameters $c_i \in \mathbb{R}$, $c_i \geq 0$. Here, we set $(c_0, c_1, c_2, c_3) = (10000, 3500, 500, 35)$ to obtain short transient times.

In order to obtain the sampled–data control law, we double the dynamic of the system and use the derived continuous–time control law (7.39) together with (7.40) to obtain the reference trajectory $x_{\text{ref}}(\cdot)$. In Section 8.4, we analyze the suboptimality estimates described in Chapter 3. Within this analysis, we consider two cost functionals for the computation of the sampled–data control, that is

$$
J(x_0, u) = \sum_{j=0}^{N} \int_{t_j}^{t_{j+1}} |x_{5,u}(t, x_0) - x_{5,\text{ref}}(t)| dt. \tag{7.41}
$$

and

$$
J(x_0, u) = \sum_{j=0}^{N} \int_{t_j}^{t_{j+1}} \|x_u(t, x_0) - x_{\text{ref}}(t)\|_2^2 dt \tag{7.42}
$$

The latter one is the standard functional to minimize the deviation between the digitally controlled system $x(\cdot)$ and the continuously controlled system $x_{\text{ref}}(\cdot)$. The first cost functional inherits the design task of the continuous–time feedback, that is to stear the system such that the position of the arm relative to the platform tracks a given command signal. Within the receding horizon controller setting we define the sampling time $T = 0.2$ and fix the initial value to

$$
x(t_0) = (0, 0, 0, 0, 10, 0, 0, 0) \tag{7.43}
$$

for both the continously and sampled–data controlled systems to make our results comparable.

Chapter 8

Numerical Results and Effects

Having described the mathematical and implementational background of our receding horizon controller in the previous chapters, we now state and discuss numerical results of both the elements of the controller and their interaction.

In particular, we analyze the impact of the *computing tolerances*, *stiffness of the dynamic* and of the *number of dimensions* on the computing time of a differential equation solver of class `OdeSolve` in Section 8.1 using the modifiable one dimensional heat equation example from Section 7.1. In the following Section 8.2 we present results concerning the different class `IOdeManager` objects and show their influence as *link between the minimization and differential equation solver* on the computing time.

Thereafter, we change our goal from a complete analysis to a guideline for developing a stabilizing and yet fast receding horizon controller. To this end, we consider the real–time inverted pendulum example stated in Section 7.2 and present effects of the main parameters of the receding horizon controller in Section 8.3, that is the chosen *optimality tolerance*, the *number of steps within the optimization routine*, the *length of the optimization horizon* and the influence of the *initial guess of the control* and of the *multiple shooting nodes*.

Having shown how a setup of a standard receding horizon controller should look like, we use our *suboptimality estimates* from Chapter 3 to analyze the stability and the degree of suboptimality of such a controller in Section 8.4. Here, we utilize our third example, the arm–rotor–platform model presented in Section 7.3, since its redesign nature allows us to create different standard situations which a receding horizon controller usually faces. In the last Section 8.5 we make use of this possibility and define a constant setpoint tracking and a switching setpoint tracking scenario. For both settings, we compare the different *adaptation strategies* stated in Chapter 4 where we consider the standard receding horizon controller as our benchmark. Moreover, we state results for the parameter required in the closed–loop suboptimality estimate of Theorems 4.4 and 4.6 and draw some conclusions.

Throughout this chapter, all the shown data has been computed on a machine with 2 Dual Core AMD Opteron 265 processors with 1800MHz each using the fortran compiler G77.

Remark 8.1

We also examined different compilers, that is GFortran and F77, and experienced deviations from the computing times using the G77 compiler. However, our experiments indicate that non of the mentioned compilers is preferable in general since the deviations in the computing times seem to appear randomly.

8.1 Comparison of the Differential Equation Solvers

The chosen ordinary differential equation solver (short ODE solver) of class `OdeSolve` is the subroutine within the RHC algorithm which is called the most, i.e. to compute the value of the cost function, the values of the restrictions and their derivatives respectively which are required by the optimization routine of class `MinProg`, see Chapter 6 for details on the interaction of the classes. Hence, it is probably the most crucial part of the RHC algorithm and needs to be chosen carefully.
We consider the explicit ODE solver DoPri5 and DoPri853 as well as the implicit methods Radau5 and Radau5913, see [108, 109], which are implemented in the derived subclasses `DoPri5`, `DoPri853`, `Radau5` and `Radau5913` of class `OdeSolve`.
There are three aspects which have a major impact on the computing time necessary to solve a given ODE over a fixed interval and therefore have to be kept in mind:

- stiffness of the considered ODE
- dimension of the problem
- chosen tolerances (absolute and relative) of the solver

In this section, we consider the one dimensional heat equation stated in Section 7.1 and compare the implemented solver with respect to these issues on the interval $I = [0, 1]$ where the control is set to zero. Since computing times are often very small and may be influenced by background processes, the presented data is computed as the mean over 2000 iterations.

8.1.1 Effect of Stiffness

First, we analyze the impact of stiffness on the computation time by varying the stiffness parameter $\lambda \in [0.01, 100]$ within the line discretization (7.3) – (7.5) of the one dimensional heat equation. To make results comparable, we consider $M = 25$ lines.
Figures 8.1 and 8.2 show calculation times for $\lambda \in [0.01, 100]$ in milliseconds. We first analyze the nonstiff case.

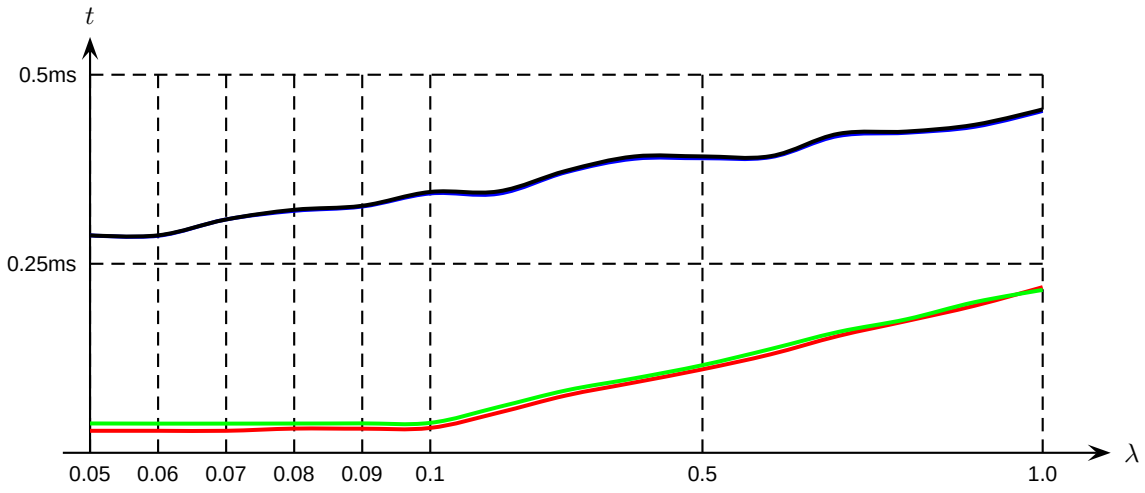


Figure 8.1: Graph of the computing times of the ODE solver routines DoPri5 (red), DoPri853 (green), Radau5 (blue) and Radau5913 (black) for small stiffness parameter λ

From the numerical experience and the data it is clear that one of the explicit methods should be chosen if the system is nonstiff. However, we cannot conclude which of the explicit solvers is preferable. DoPri853 is the more sophisticated one but its advantage of adapting the order of accuracy might be lost if the integration interval is small. Hence, if an application appears to be nonstiff, both methods should be tested.

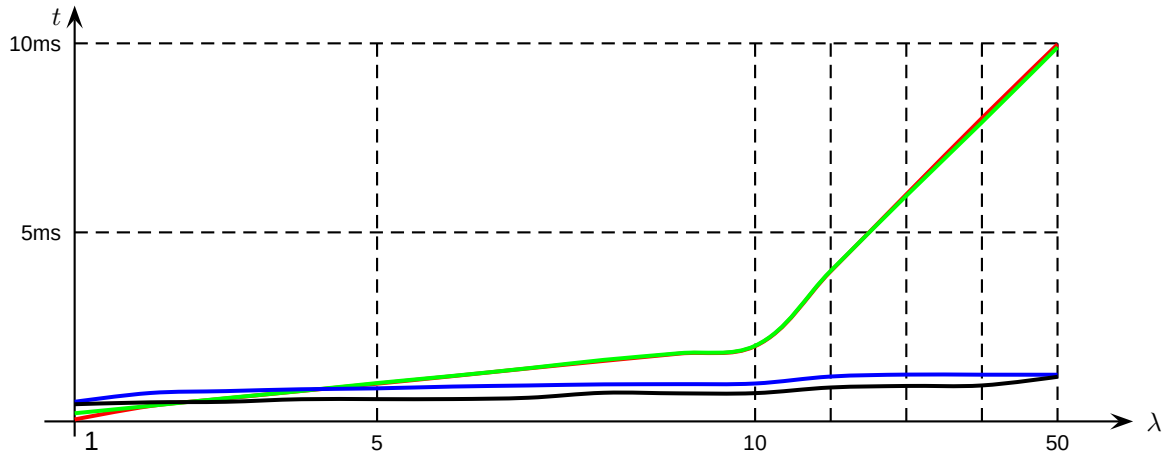


Figure 8.2: Graph of the computing times of the ODE solver routines DoPri5 (red), DoPri853 (green), Radau5 (blue) and Radau5913 (black) for large stiffness parameter λ

Similarly, an implicit solver should be chosen if the problem is stiff. For DoPri5 and DoPri853 the computational cost grows rapidly, cf. Figure 8.2. In contrast to the nonstiff case, we can explicitly recommend the usage of Radau5913 since computation times are significantly smaller compared to Radau5. This was also confirmed by our numerical experiments using the implemented receding horizon controller.

8.1.2 Curse of Dimensionality

We now fix the stiffness parameter $\lambda = 1$ and analyze the computing times necessary to solve the problem (7.3) – (7.5) for different space discretizations. Here, we consider the ODE systems for $M \in \{3, \ldots, 150\}$ and, again, solve the resulting problems on the time interval $I = [0, 0.1]$. Since the computing times vary massively we present the data in two figures.

Analyzing Figure 8.3, we observe that for lower dimensions explicit solvers should be preferred. Furthermore, Figures 8.3 and 8.4 also indicate that there exists a $\overline{M}$ such that the implicit solvers Radau5 and Radau5913 show a better performance if $M \geq \overline{M}$.

Moreover, we can compare the computing times for $M = 25$ concerning the stiffness of a problem presented in the previous Section 8.1.1 to those computing times for varied number of dimensions of the problem. We observe the following: If we increase/decrease in the stiffness parameter λ then the separating bound $\overline{M}$ decreases/increases. Hence, an implicit solver should be chosen if the dimension of a system is large and the problem appears to be stiff. Conversely, explicit methods are preferable if the dimension of a system is small and the problem not considered to be stiff. In any other case, one should test the methods against each other.

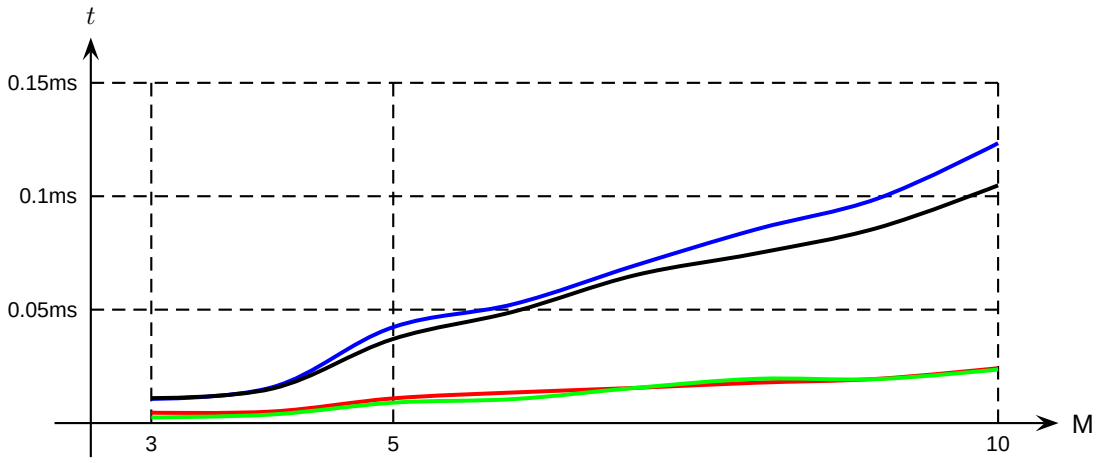


Figure 8.3: Graph of the computing times of the ODE solver routines DoPri5 (red), DoPri853 (green), Radau5 (blue) and Radau5913 (black) for lower dimensions

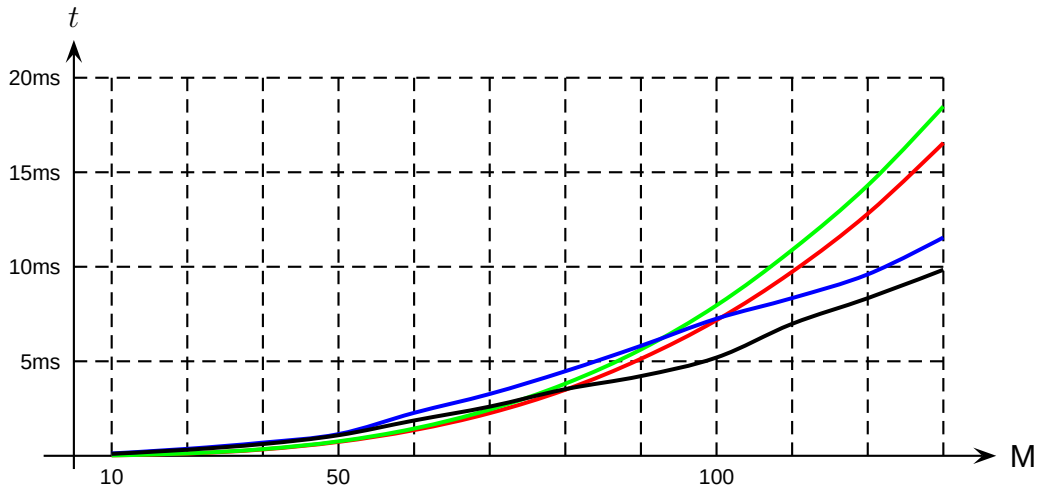


Figure 8.4: Graph of the computing times of the ODE solver routines DoPri5 (red), DoPri853 (green), Radau5 (blue) and Radau5913 (black) for higher dimensions

	System nonstiff	System stiff
Dimension small	Explicit Solver	Test
Dimension large	Test	Implicit Solver

Remark 8.2

For every newly derived class `Model` *problem various tests should be run using the optimization routines as well. In particular, one has to keep in mind that using the control parameter u within our problem RHC_N, the optimization may cause a uncontrolled nonstiff system, that is a system which is nonstiff for $u \equiv 0$, to become stiff for certain choices of the control. Here, the internal exceptions can be used to circumvent this problem by*

exchanging or adapting the differential equation solver.

8.1.3 Effects of absolute and relative Tolerances

From the previous Section 8.1.2, we know that if the number of dimensions is large and the tolerances are fixed, then the implicit solvers Radau5 and Radau5913 show better performance than the explicit methods DoPri5 and DoPri853. Here, we focus on the question which solver or type of solver should be preferred if we vary tolerance values.

Remark 8.3
Increasing numerical tolerances is non standard in optimal control since the structure of the resulting control function massively depends on the state trajectory. In particular if we consider constraints, switching points of the control may vary since we may incorrectly compute those time instances when a trajectory hits a boundary.

Since the switching structure of the control is known a priori for our problem RHC_N, we can neglect the structural issue of the resulting control function. Moreover, we focus on satisfying possibly existent constraints at the sampling instances only. Hence, if we consider slightly tighter bounds, we may ease the need of highly accurate solutions of the differential equation system.

Remark 8.4
Even if we choose to solve the differential equation system using a class `OdeSol` *object with extremely small tolerances, the calling class* `MinProg` *object will in almost no case satisfy all constraints exactly, see e.g. Sections 6.3.2.2 and 6.3.3.2 for details on the methods* `SqpFortran` *and* `SqpNagC` *respectively.*

Within the RHC setting, our aim is to speed up the solution of differential equation system and hence the solution of every single optimization problem in the receding horizon control process by enlarging the tolerance levels of the differential equation solvers. Yet, as a side effect, the search direction of the SQP routine may deviate, compare Algorithms 5.59 and 5.61, and we have to keep in mind that this possibly leaves the output to be far from optimal.

Remark 8.5
From our numerical experience, the inherited error in the search direction becomes dominant if the solution is already close to the optimum. Hence, the tolerance of the differential equation solver should match the tolerance of the optimization routine to obtain acceptable results.
If one primarily aims at stabilizing a system, it often suffices to choose quite large optimization tolerances which allows us to speed up the computation significantly by increasing the tolerances of the differential equation solver. For exemplary results of this connection, see Section 8.3.1.

In the following, we analyze the impact of variations in the tolerances of the ODE solvers on the computing times. Again, we use the example of a discretized one dimensional heat equation, cf. (7.3) – (7.5). Moreover, the dependency of the resulting computing times on stiffness and dimension of the system are considered.
From Figure 8.5 we can see that for large tolerances the implicit solvers are faster than the explicit ones. For very small tolerances we observe that the computing time necessary to solve the problem increases dramatically for Radau5 and Radau5913. For the

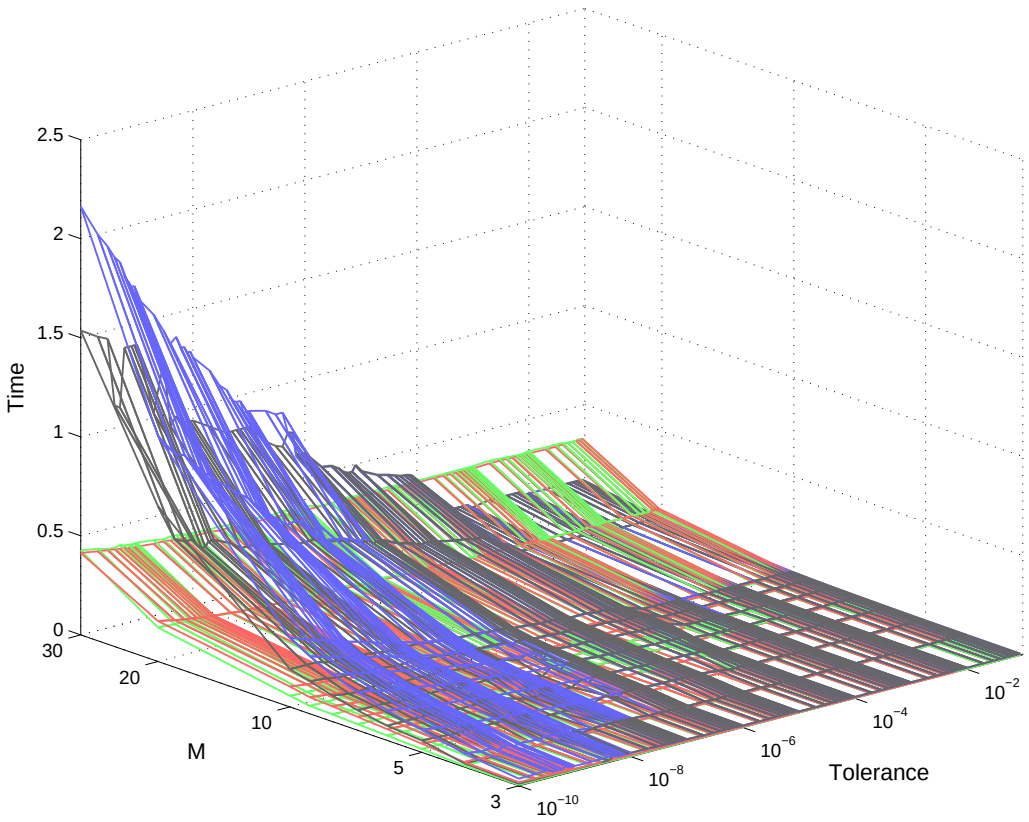
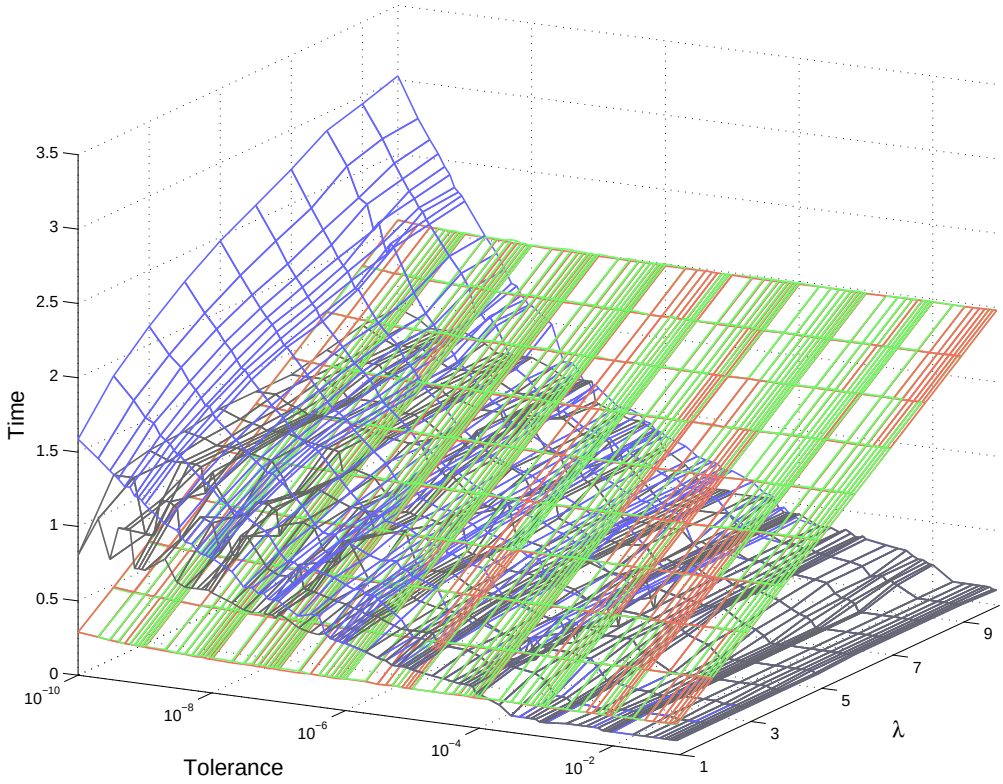
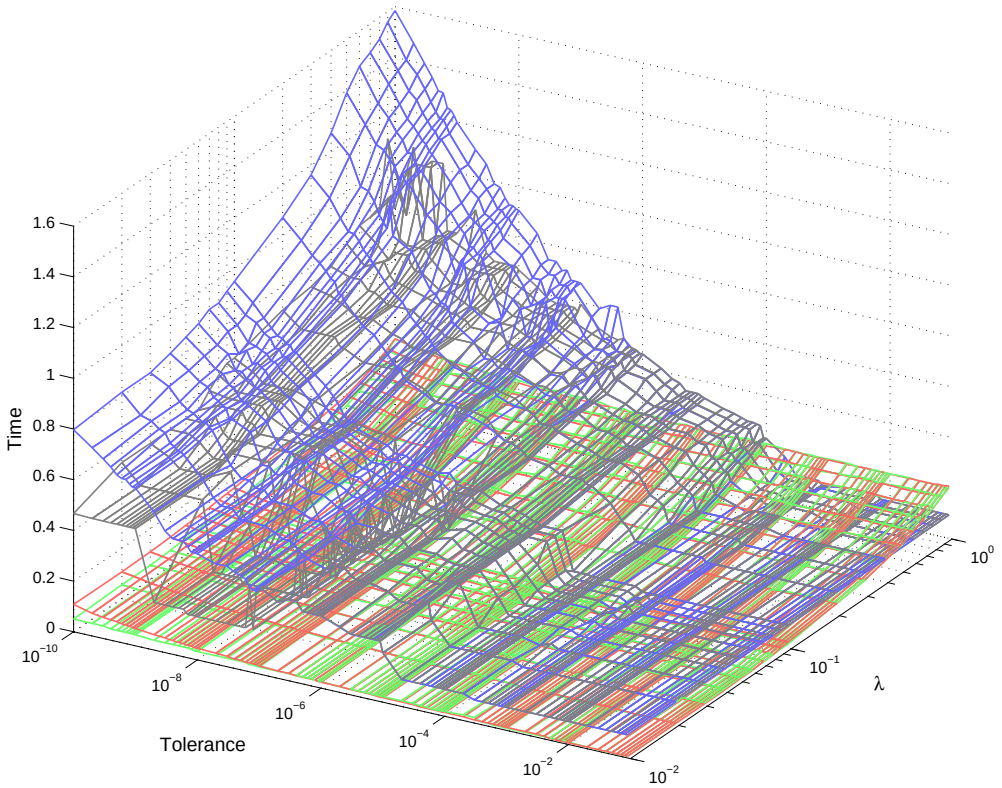


Figure 8.5: Influence of number of dimensions and tolerance on the necessary computing time of the ODE solver routines DoPri5 (red), DoPri853 (green), Radau5 (blue) and Radau5913 (black)

explicit DoPri methods, however, the effort stays almost the same. Yet, considering higher dimensions leads to a change in favor of the implicit solvers within the ranking.
Similar effects can be observed if we vary stiffness instead of the size of the problem, see Figure 8.6 . Here, we again fix $M = 25$.
In particular, the explicit solvers show almost constant computing times if the tolerance level is enlarged. However, the effort for implicit solvers decreases significantly. Moreover, we make the same observation as in Section 8.1.1, that is for very small λ explicit and for large λ implicit solvers are preferable.
Since implicit methods are designed for treating stiff differential equations, this is what we expected to happen. Surprisingly, we also see that even in the nonstiff case, implicit solvers may outrun explicit ones if the tolerance levels are chosen large.
However, in our examples we cannot say in advance which ODE solver is the best. Since we deal with control systems the control u influences the stiffness property. Hence, solving the optimal control problem ($\mathrm{RHC_N}$) defined in Section 2.4 might change a nonstiff system associated to the initial guess into a stiff one for the outcome of the optimization. This phenomenon is not a problem in simulations but for real–time applications it has to be avoided since it may render the available computing time to be insufficient. Moreover, u depends on the current state and we may face regions in the state space which are stiff. This can be explicitly tricky if the equilibrium one tries to stabilize is contained in such a region. Possible solutions are an automated change of the solver or of the tolerance levels.



(a) Influence of Stiffness vs. Tolerance for large λ



(b) Influence of Stiffness vs. Tolerance for small λ

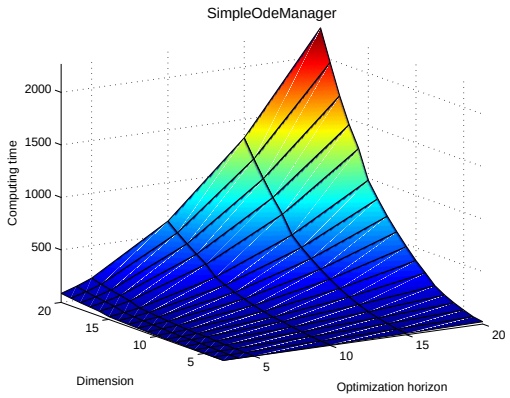
Figure 8.6: Influence of stiffness and tolerance on the necessary computing time of the ODE solver routines DoPri5 (red), DoPri853 (green), Radau5 (blue) and Radau5913 (black)

8.2 Comparison of Differential Equation Manager

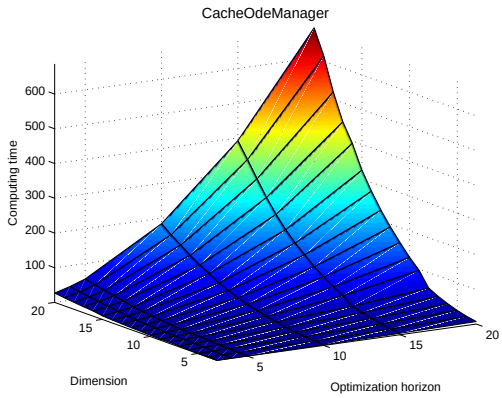
The aim of the following section is to show the impact of the different implemented links between optimization routines and differential equation solver, the so called differential equation manager, on the computing time. To this end, we consider the setting of a single SQP step and ten consecutive steps and discuss the results as well as the implementational background.
Here, we again utilize the discretized one dimensional heat equation (7.3) – (7.5) stated in Section 7.1 with $\lambda = 1$. To make the differential equation managers from class `IOdeManager` comparable, we use the differential equation solver Radau5913 exclusively and fix the absolute and relative tolerances to 10^{-10}.

8.2.1 Effects for a single SQP Step

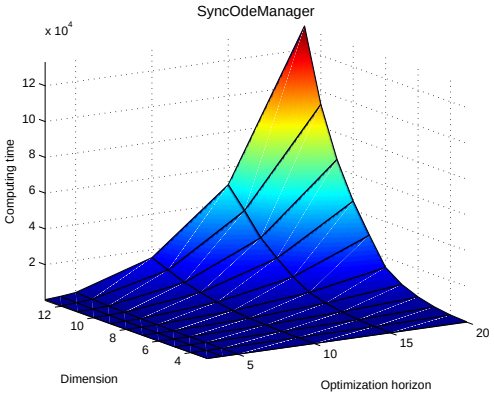
In the following Figure 8.7, we consider computing times for one SQP step using a class `SqpFortran` object, i.e. we stop the minimizer after computing the gradient of the cost function and the Jacobian of the constraints once.



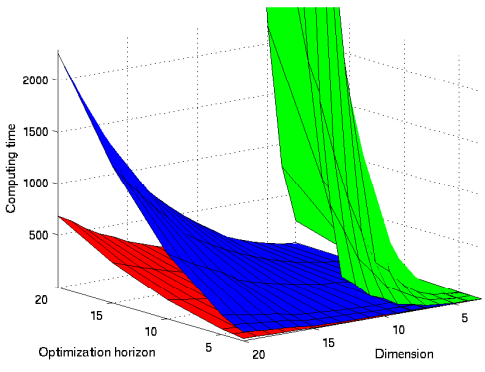
(a) Computing times `SimpleOdeManager`



(b) Computing times `CacheOdeManager`



(c) Computing times `SyncOdeManager`



(d) Comparison of `IOdeManager` classes

Figure 8.7: Influence of number of dimensions and horizon length on the necessary computing time of the `IOdeManager` classes `SimpleOdeManager` (blue), `CacheOdeManager` (red) and `SyncOdeManager` (green) for an initial SQP step

The optimization horizon is given by $H = N \cdot T$ where $T = 0.1$ is fixed for this comparison while the parameter N and M characterizing the optimization length and the dimension

of the system are varied. Here, we start by analyzing the unconstrained case, i.e. we set $p = 0$ in (7.7).
For the initial step of the SQP routine we can see that the class `CacheOdeManager` object outruns both the `SimpleOdeManager` and the `SyncOdeManager` object. Moreover, computations using the `SyncOdeManager` object are clearly the slowest ones. In Table 8.1, we display the absolute and relative computing times of all three `IOdeManager` objects where $N = 10$ is fixed.

M	Simple	Cache	Sync	$\frac{\text{Simple}}{\text{Cache}}$	$\frac{\text{Sync}}{\text{Simple}}$	$\frac{\text{Sync}}{\text{Cache}}$
3	2.9588	1.6732	18.9482	1.76	6.40	11.32
4	4.9362	2.7603	63.3587	1.79	12.84	22.95
5	10.2527	5.0513	224.0629	2.03	21.85	44.35
6	16.1331	7.5556	520.8210	2.14	32.28	68.93
7	22.8421	10.6945	958.7065	2.14	41.97	89.64
8	31.9308	14.3154	1777.4462	2.23	55.67	124.16
9	37.6704	19.9755	2929.4702	1.88	77.76	146.65
10	47.9160	25.4729	4354.8941	1.88	90.88	170.96
11	60.5290	32.0525	6253.1095	1.88	103.31	195.09
12	74.4925	39.1798	8946.3031	1.90	120.10	228.34
13	89.6633	46.7690	12225.0225	1.92	136.34	261.39
14	105.7580	55.2267	16360.8011	1.91	154.70	296.25
15	123.1933	64.8654	20925.7137	1.90	169.86	322.60
16	165.9090	77.7999	31561.0758	2.13	190.23	405.67
17	194.5727	90.6133	—	2.15	—	—
18	225.3715	106.7261	—	2.11	—	—
19	260.4182	120.3044	—	2.16	—	—
20	299.1995	136.9109	—	2.19	—	—

Table 8.1: Comparison of calculation times for one SQP step

Here, one can see the advantage of a class `CacheOdeManager` objects which in principle halves the computing time required by a class `SimpleOdeManager` object. This is due to the reuse of precomputed trajectory values which results in a triangular structure of the computations shown in Figure 8.8. Within this figure, we display those state values which are recomputed with a modified control to evaluate the difference quotient (6.2) of the cost function.
A class `CachOdeManager` object uses the fact that only endpieces of the control are varied and hence the frontpieces of the trajectories are not computed again.
Note that here the unconstrained case is considered and hence no Jacobian of the constraints has to be computed. Once this matrix is required as well, we expect a class `CacheOdeManager` object to outrun a class `SimpleOdeManager` object by more than a factor of two since apart from evaluating the fraction of the difference quotient (6.2) no recalculation of the trajectory has to be performed. Within a class `SimpleOdeManager` object, however, the evaluation of a single column of the Jacobian requires a complete recalculation of the trajectory of the system.
Moreover, we observe that the performance of the class `SyncOdeManager` object is poor in the unconstrained case. The additional effort is caused by the coupling of the systems dynamics which results in a significant increase of the computing time necessary to solve the resulting differential equation, cf. Section 8.1.2.

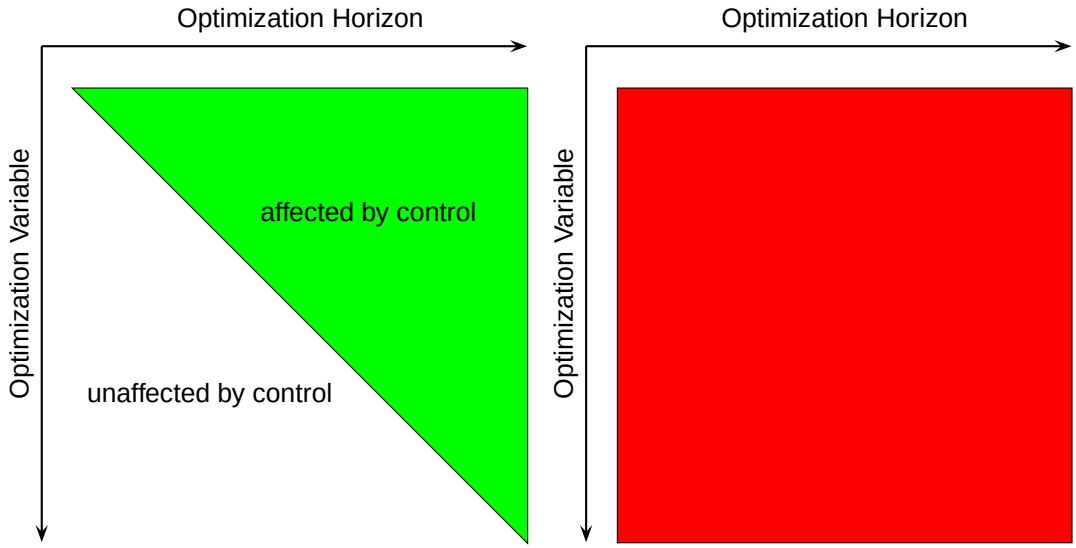


Figure 8.8: Computing structures of a class `CacheOdeManager` (left) and a class `SimpleOdeManager` object (right)

Now, we add costraints to the problem and re–evaluate all data. Here, we choose the set of constraints (7.7) with $p = M$ while leaving the rest of the problem unchanged. The following Table 8.2 shows calculation times for $M = 10$:

N	Simple	Cache	Sync	$\frac{\text{Simple}}{\text{Cache}}$
3	11.2973	4.0352	81.4827	2.43
4	36.3457	7.6900	415.5835	4.72
5	54.6212	10.6218	765.1625	5.14
6	76.7315	13.8552	1312.8389	5.53
7	102.5981	18.4680	2114.3480	5.55
8	132.1269	23.8587	3116.4190	5.53
9	166.1137	30.8212	4330.0736	5.39
10	202.7110	37.3903	5985.3090	5.42
11	243.7564	46.0038	7839.1500	5.30
12	289.4485	55.3483	10110.5106	5.23
13	337.3657	64.6308	12529.8389	5.22
14	388.8956	75.0318	15596.5177	5.18
15	445.1079	86.3597	19276.9178	5.15
16	511.7867	96.5668	23837.6270	5.30
17	570.9604	109.4915	27727.1883	5.21
18	638.9876	124.1468	33215.7455	5.15
19	713.0454	139.2447	39145.8085	5.12
20	789.6375	155.5406	46003.3345	5.08

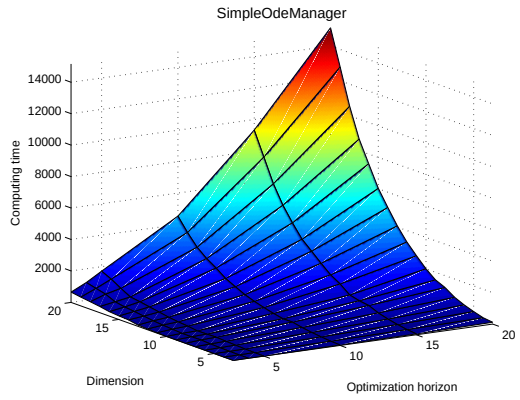
Table 8.2: Comparison of calculation times for one SQP step of a constrained problem

We observe that the `CacheOdeManager` object shows outstanding performance compared to both the `SimpleOdeManager` and the `SyncOdeManager` object which is what we expected. Moreover, the acceleration factor of the `CacheOdeManager` object relative to the `SimpleOdeManager` object seems to be stay between 5 and 5.5 which has also been observed in other examples as well.

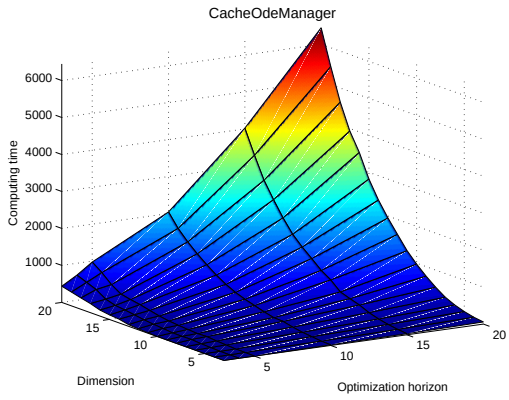
Remark 8.6
If the horizon length N and the dimension of the system M are small, then a class `SyncOdeManager` *object may exhibit smaller computing times than a class* `SimpleOdeManager` *object. However, this link between minimizer and differential equation solver is originally designed for analytical purposes only, see Section 6.2.5.3 for details.*

8.2.2 Effects for multiple SQP Steps

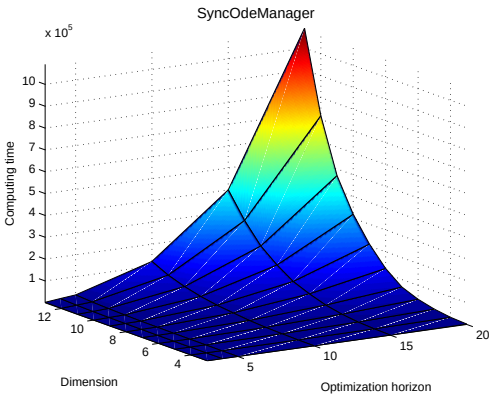
Until now we analyzed one step of the SQP method only. Since SQP routines use several steps based on the same dynamic but for changing control variables, already computed state values may reoccur. Hence, since a class `CacheOdeManager` object exploits this property of the problem, we expect this method to show an even better performance if consecutive steps are considered.
In the unconstrained case the updated control is usually changed in every component after each SQP step since no bounds have to be satisfied. Hence, the computing times are a multiple of the computing times required for a single step for the selected class `IOdeManager` object. This is illustrated in Figure 8.9 where we display results for the setting we used in Figure 8.7 but execute ten steps of the SQP method `SqpFortran`.



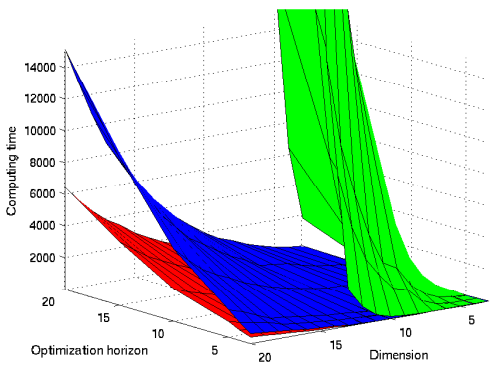
(a) Computing times `SimpleOdeManager`



(b) Computing times `CacheOdeManager`



(c) Computing times `SyncOdeManager`



(d) Comparison of `IOdeManager` classes

Figure 8.9: Influence of number of dimensions and horizon length on the necessary computing time of the `IOdeManager` classes `SimpleOdeManager` (blue), `CacheOdeManager` (red) and `SyncOdeManager` (green) for ten SQP steps

Again, we obtain the same tendency as from Figure 8.7, i.e. the class `CacheOdeManager`

object outruns both other objects and a `SimpleOdeManager` object shows better performance than a class `SyncOdeManager` object.

However, if we impose constraints similar to Section 8.2.1, i.e. using the set of constraints (7.7) with $p = M$, we expect the class `CacheOdeManager` and `SyncOdeManager` objects to exhibit computing times which are much lower than the multiple computing time of a single step of the SQP algorithm. Still, we have to keep in mind that a consecutive execution of several SQP steps generates problems which are not identical and hence the computing times for each single step vary in general. For this setting, the results are contained in the following Table 8.3. Moreover, we compare the resulting computing times for the constrained and unconstrained case.
We first observe that the computing times in the constrained case are lower than in the unconstrained case although we additionally have to compute the Jacobian of the constraints. Taking a closer look at the outcome of the optimization in each step, we see that large control values are considered in the unconstrained case which results in a reduction of the costs for the first sampling points but leads to large deviations from the desired equilibrium for the state at the last sampling points. Moreover, the differential equation becomes stiff in this region. Since the class `SyncOdeManager` object combines all sampling intervals, this slowdown affects all parts of the horizon which explains the observed bad performance. For the class `SimpleOdeManager` and `CacheOdeManager` objects, comparing the unconstrained and the constrained case we obtain that the reduced computing times caused by the nonstiffness of the system compensate for the additional cost for calculation the Jacobian of the constraints only for small horizons. Still, one has to keep in mind that the Jacobian of the constraints grows quadratically in the horizon length N. Hence, for all class `IOdeManager` objects, there exists some horizon length N such that the computing times for the constrained problems are larger than in the unconstrained case. For a class `SimpleOdeManager` object, this can be observed in Table 8.3.
If we compare the displayed measurement for one and ten SQP steps, we observe the increase in the computing times of the class `SimpleOdeManager` is quite small. As mentioned in Section 6.2.5.1, the `SimpleOdeManager` class transforms the active–set Quasi–Newton approximation of the optimization method to a Newton method with full BFGS–update. For this particular example, this results in just two evaluations of the Jacobian matrix while for both `CacheOdeManager` and `SyncOdeManager` eight evaluations of the reduced Hessian and one of the complete Jacobian are performed. Moreover, we observe that the increase in the computing time for the class `CacheOdeManager` and `SyncOdeManager` objects is growing in N but smaller than eight. This is due to two sources: for one, the computational effort for obtaining the Jacobian is reduced by recomputing required updates of parts of the matrix only, see also Section 6.2.5.2, and secondly, the reuse of known information on the state trajectory speeds up the computation.

From this analysis, we see that the class `CacheOdeManager` significantly improves the speed of the computing process and is therefore preferable. Here, we again remind that class `CacheOdeManager` objects are not as robust as class `SimpleOdeManager` or `SyncOdeManager` objects regarding the correctness issue, see also Section 6.2.5.2.

Remark 8.7
For higher dimensions or longer horizons, the method used by the class ***`SyncOdeManager`*** *requires too much memory such that swaping memory becomes unavoidable. Once this happens, the computing times explode.*

	unconstrained case, 10 SQP steps			constrained case, 10 SQP steps			constrained case, 1 SQP step		
N	Simple	Cache	Sync	Simple	Cache	Sync	Simple	Cache	Sync
3	13.1505	4.7632	1099.1316	18.3812	4.4733	84.9232	11.2973	4.0352	81.4827
4	70.4794	42.1296	2663.7205	50.6497	14.9453	501.6929	36.3457	7.6900	415.5835
5	97.3493	76.7835	5058.5770	87.5432	22.2208	1170.0020	54.6212	10.6218	765.1625
6	102.1465	105.3355	8687.7953	131.0241	34.1699	2301.1349	76.7315	13.8552	1312.8389
7	266.2635	140.0757	14140.9442	224.0904	67.8548	4902.5388	102.5981	18.4680	2114.3480
8	344.7292	178.4512	21212.2974	315.7195	69.3653	9892.6243	132.1269	23.8587	3116.4190
9	433.9290	222.1279	29828.8681	427.1371	113.8952	10684.1498	166.1137	30.8212	4330.0736
10	533.4967	270.0599	41553.8968	582.0995	160.4895	19638.1373	202.7110	37.3903	5985.3090
11	637.8299	319.2618	55705.3349	664.8787	195.3231	25244.7710	243.7564	46.0038	7839.1500
12	610.8299	379.4902	72414.2295	701.0420	246.7882	31704.9278	289.4485	55.3483	10110.5106
13	881.6595	381.5603	91061.1325	1084.7242	296.2977	45384.2946	337.3657	64.6308	12529.8389
14	757.3996	502.1227	112645.7515	1127.8375	366.6607	49624.3013	388.8956	75.0318	15596.5177
15	1025.8416	523.1227	140511.9187	1279.1668	427.6893	58659.8644	445.1079	86.3597	19276.9178
16	1314.0926	644.8333	171741.8670	1382.2241	476.2329	82156.4235	511.7867	96.5668	23837.6270
17	1479.4552	717.2727	200889.4977	1542.8805	541.2652	81396.7569	570.9604	109.4915	27727.1883
18	1652.7445	797.0910	239424.4523	1680.0273	659.3980	91415.9953	638.9876	124.1468	33215.7455
19	1835.9452	885.4684	283490.5182	2311.3047	822.9271	146763.2249	713.0454	139.2447	39145.8085
20	2028.6359	978.0781	332925.6442	2633.1941	909.0864	184021.5298	789.6375	155.5406	46003.3345

Table 8.3: Comparison of calculation times for ten SQP steps of the unconstrained and constrained problem and for one SQP step of the constrained problem

8.3 Tuning of the Receding Horizon Controller

In this section, we characterize the impact of the main parameter of the controller on the computing time, that is the *tolerance of the optimization routine* as well as the *tolerances of the differential equation solver*, the *length of the horizon*, the *stopping criteria* of the optimization method, the *initial guess* of the control and the choice of the *multiple shooting nodes* as depicted in Figure 8.10. Our aim in this section is not to give a complete analysis which might be impossible due to the complexity of the problem. Instead, we consider the real–time problem of the inverted pendulum on a cart described in Section 7.2 and show how the receding horizon control algorithm can be tuned.

The following sections are governed by one issue each and are not sorted deliberately but can be seen as a guideline for the tuning. Moreover, these parameters are not (or at least not only) sorted by their impact on the computing time. Note that the parameter also depend on each other as shown in Figure 8.10. Hence, some parameters may have to be considered repeatedly.

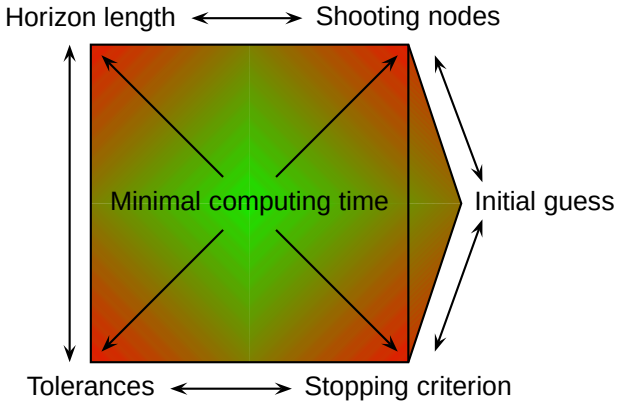


Figure 8.10: Effects and interaction of implementation parameters on the computing time

We start by considering the tolerances of the minimizer and the differential equation solver in Section 8.3.1. Since these algorithms cause the major computational cost, we aim at choosing large tolerances. However, this may lead to difficulties, even instability, since the allowed errors interact in the receding horizon controller setting. For the parameters, a good compromise between speed and accuracy cannot be rigorously described in a mathematical way, hence we can only describe how such a compromise should look like.

In Section 8.3.2, we consider the horizon length as the parameter of choice. We investigate the differences of closed–loop solutions and closed–loop costs for different horizon lengths. Since the size of the discretized optimal control problem depends massively on this parameter, we want to choose the horizon length as small as possible.

The following Sections 8.3.3 and 8.3.4 deal with different aspects of the initial guess problem of the control. Since there exist no methods for computing a good initial guess of the optimization variable, we shortly describe various heuristics in Section 8.3.3. Thereafter, we change the discretization of the optimal control problem by introducing multiple shooting points in Section 8.3.4, see also Section 5.1.3. These additional optimization variables may allow us to improve the initial guess and even reduce the minimal length of the optimization horizon. Yet, difficulties such as instability of the closed–loop may occur if one uses this approach which we discuss as well.

Last, since we aim at real–time applicability of the controller, we also consider the effects of stopping criteria of the optimization method. This issue is treated in Section 8.3.5 where we analyze the impact of a low number of maximal iterations of the optimization method on the resulting closed–loop behaviour on the one hand, and of a time dependent break of the optimization. Both methods are motivated by the goal of real–time applicability of the controller but may cause unwanted effects.

8.3.1 Effects of Optimality and Computing Tolerances

Within this section, we analyze the impact of the user dependent choices of tolerance values for the optimization routine and for the differential equation solver on the *stability of the closed–loop system* on the one hand, and on the *resulting computing time* on the other hand. To this end, we use the grid of parameter

$$(\mathrm{tol}_{\mathrm{SQP}}, \mathrm{tol}_{\mathrm{ODE}}) \in \{10^{-8}, 10^{-7}, \ldots, 10^{-1}\}^2$$

and generate a sequence of receding horizon controllers. Then, we compute the closed–loop solution (2.16) on the interval $[0, 50]$ with sampling length $T = 0.1$, initial value $x_0 = (2, 2, 0, 0)$ and $\mathrm{tol}_{\mathrm{ODE}} = 10^{-10}$.

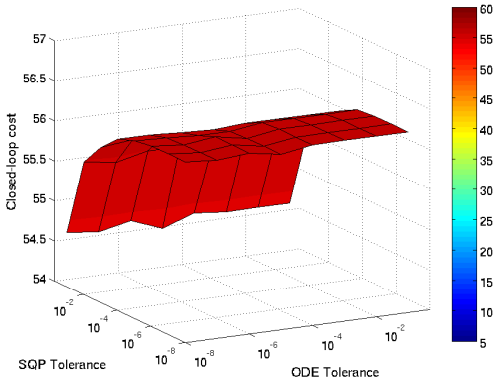
Remark 8.8
In this section, we use the following strategy to obtain a real–time implementable receding horizon controller: We combine suitable cost functionals, one for the swing–up which induces enough potential and kinetic energy in the system, and one for stabilization purposes. Here, we only consider the second and more interesting part and use the initial value $x_0 = (2, 2, 0, 0)$ for our receding horizon controller. Note that if we started in the downward position $(0, 0, 0, 0)$, then the horizon would have to be chosen larger than $N = 65$ given a good initial guess to obtain a stabilizing control. Such a long horizon, however, is not real–time solvable by now.

Remark 8.9
Since we analyze the impact of stopping criteria in Section 8.3.5 separately, we display and discuss summarized computing times only.
Note that stability of the resulting closed–loop trajectory depends massively on the horizon length N. Hence, we consider different horizon lengths in this section as well. For a more detailed discussion of the effects of the horizon length, we refer to Section 8.3.2.

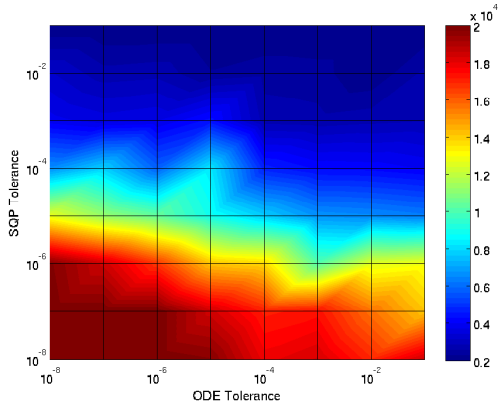
Here, we consider the simple criterion of the closed–loop costs to distinguish between stable and unstable closed–loop trajectories. In order to be easily comparable, the following figures are generated using identical colorbars for both the closed–loop costs and the computing times. In Figure 8.11, we start with the limit cases $N = 15$ and $N = 16$ for the optimization horizon.
Comparing the costs for $N = 15$ and $N = 16$ in Figures 8.11(a) and (c), we can estimate that all combinations $(\mathrm{tol}_{\mathrm{SQP}}, \mathrm{tol}_{\mathrm{ODE}}) \in \{10^{-8}, 10^{-7}, \ldots, 10^{-1}\}^2$ exhibit unstable closed–loop trajectories for $N = 15$. Yet, the corresponding computing times for $N = 15$ shown in 8.11(b) are higher than the required times for $N = 16$ displayed in 8.11(d). This difference is due to the instability of the closed–loop for $N = 15$.

Remark 8.10
Here, we do not display the corresponding solutions for all receding horizon controller settings. For a particular choice of the pair (tol_{SQP}, tol_{ODE}), however, the $x_1(\cdot)$ trajectories are shown in Figures 8.16, 8.17, 8.18 and 8.19.

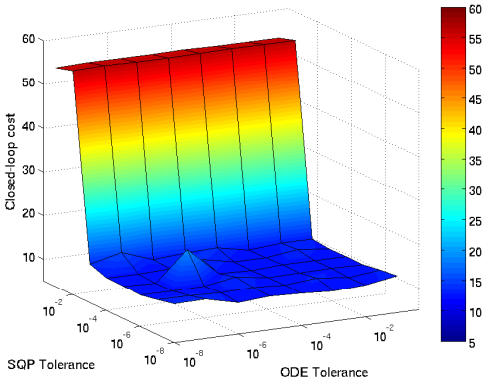
Figures 8.11(c) and (d) allow us to conclude that tolerance levels may be eased without changing the closed–loop behaviour or closed–loop cost significantly while the computing times are lowered. A standard initial setting for $(\mathrm{tol}_{\mathrm{SQP}}, \mathrm{tol}_{\mathrm{ODE}})$ is the pair $(10^{-6}, 10^{-8})$ which here results in a total computing time of 9.943s. For $(\mathrm{tol}_{\mathrm{SQP}}, \mathrm{tol}_{\mathrm{ODE}}) = (10^{-3}, 10^{-4})$, this is reduced to 7.479s saving approximately 25% of the computing time.



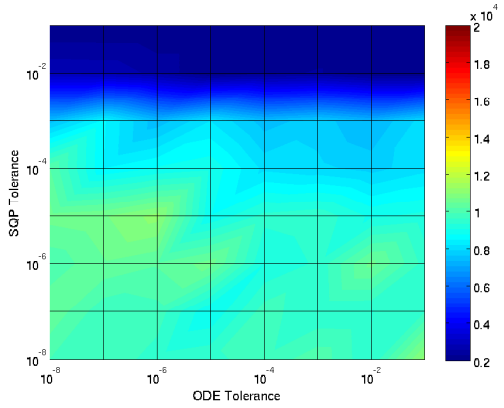
(a) Closed–loop costs for $N = 15$



(b) Computing times for $N = 15$



(c) Closed–loop costs for $N = 16$

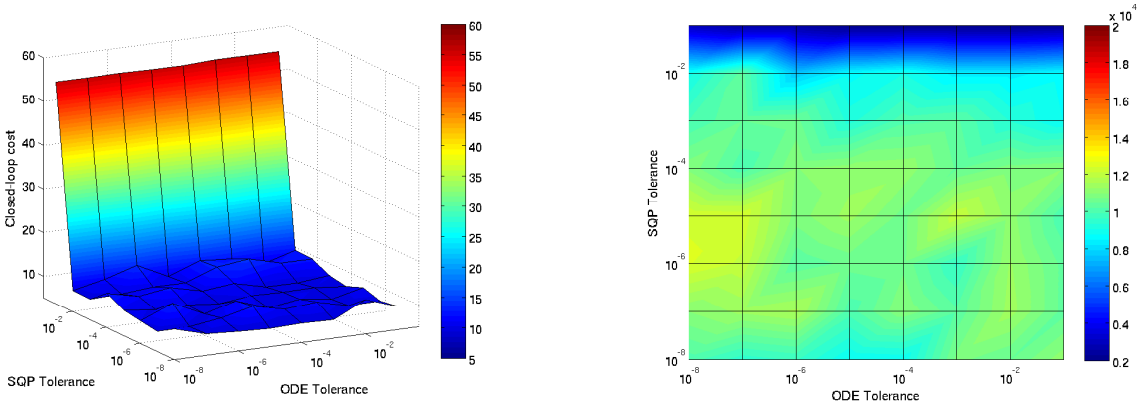


(d) Computing times for $N = 16$

Figure 8.11: Comparison of closed–loop costs and computing times for the limit cases of optimization horizons N and various optimization and differential equation solver tolerances

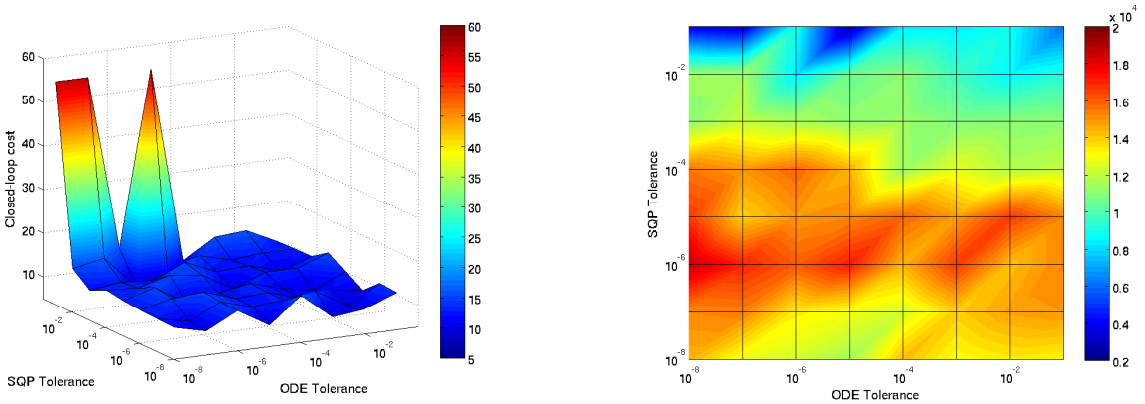
Remark 8.11
The tolerance level of the differential equation solver should not be chosen larger than the tolerance level of the optimization routine. In any other case, the search direction may be heavily corrupted by errors in the numerical difference quotients used to compute this vector and the optimization iteration may be unable to discover a point satisfying the KKT conditions of Theorem 5.18 with accuracy less than tol_{SQP}. For this reason, we consider only tolerance pairs satisfying $tol_{SQP} \geq tol_{ODE}$ for our analysis.

In Figures 8.12 and 8.13, the optimization horizon N is increased further. Similar to our previous analysis of the impact of the horizon length, see Section 8.2, this results in enlarged computing times.
We also experience an improvement in the closed–loop cost for $\text{tol}_{\text{ODE}} = 10^{-2}$ in Figure 8.12(a) and additionally for $\text{tol}_{\text{ODE}} = 10^{-1}$ in Figure 8.13(a). This corresponds to receding horizon controllers which now stabilizes the system in an upright equilibrium. Hence, enlarging the horizon, i.e. increasing the complexity of each optimal control problem in the receding horizon control process, may enable us to ease up the tolerance parameter. For the setting here, the effects on the computing time almost cancel out. For $N = 17$, we obtain a total computing time 10.243s for our standard choice $(\text{tol}_{\text{SQP}}, \text{tol}_{\text{ODE}}) = (10^{-6}, 10^{-8})$ which can be improved by $\approx 18\%$ to 8.353s for $(\text{tol}_{\text{SQP}}, \text{tol}_{\text{ODE}}) = (10^{-2}, 10^{-2})$.



(a) Closed–loop costs (b) Computing times

Figure 8.12: Comparison of closed–loop costs and computing times for optimization horizon $N = 17$ and various optimization and differential equation solver tolerances

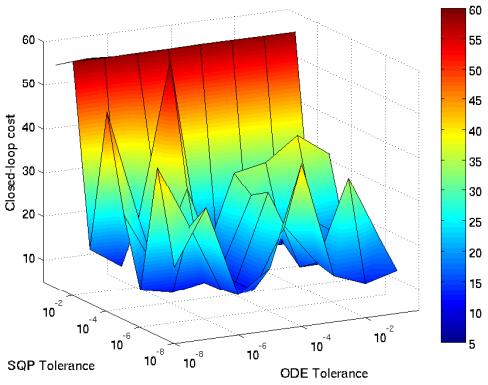


(a) Closed–loop costs (b) Computing times

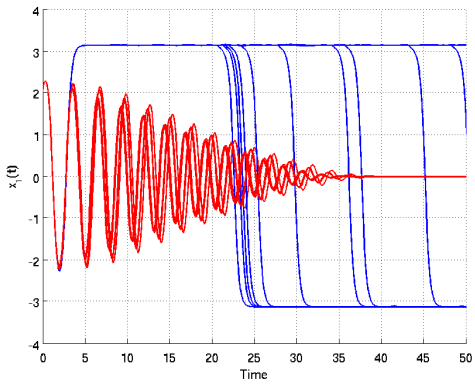
Figure 8.13: Comparison of closed–loop costs and computing times for optimization horizon $N = 20$ and various optimization and differential equation solver tolerances

For $N = 20$, the corresponding times are 18.669s in the standard case and 8.612s for $(\text{tol}_{\text{SQP}}, \text{tol}_{\text{ODE}}) = (10^{-2}, 10^{-2})$, i.e. a decrease of $\approx 53\%$. Yet, the computing time for $N = 16$ is the smallest one. This tendency can also be experienced for other example which leads to the general guideline to choose the lowest optimization horizon possible which still guarantees stability of the closed–loop. Note that this is exactly the idea of the adaptation strategies presented in Chapter 4.

Last, the two examples shown in Figures 8.14 and 8.15 illustrate that the parameter $(\text{tol}_{\text{SQP}}, \text{tol}_{\text{ODE}})$ should be chosen carefully due to their effects on the closed–loop.
Here, Figure 8.14(a) shows that for some pairs $(\text{tol}_{\text{SQP}}, \text{tol}_{\text{ODE}})$ the closed–loop costs are significantly higher. If we analyze the corresponding closed–loop trajectories, the system is driven to the downward position $(0, 0, 0, 0)$ for $\text{tol}_{\text{SQP}} = 10^{-1}$, $\text{tol}_{\text{SQP}} = 10^{-2}$ and all considered values for tol_{ODE}. All other pairs, however, do "mainly" stabilize the pendulum in an upright position. However, these positions vary in two ways. For one, the $x_1(\cdot)$ trajectory does not converge to the same upright position. And more importantly, once an upright position is reached, it may be switched to another upright position. Here, the

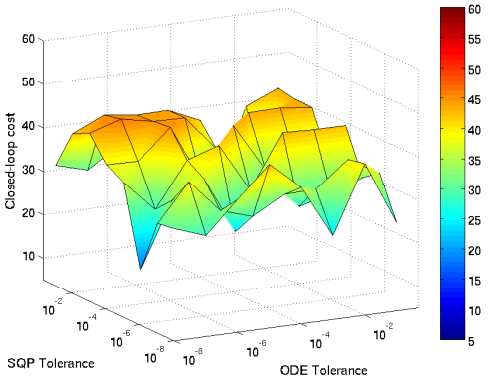


(a) Closed–loop cost

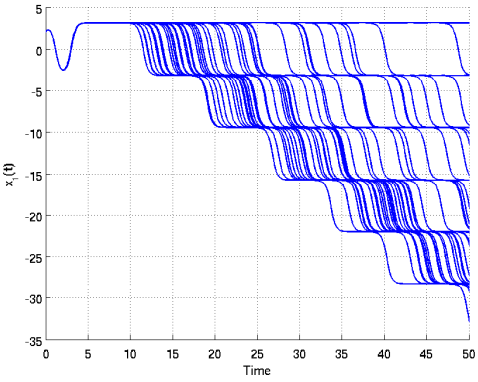


(b) $x_1(\cdot)$ trajectories

Figure 8.14: Exceptional closed–loop costs for $N = 25$ and various optimization and differential equation solver tolerances



(a) Closed–loop costs



(b) $x_1(\cdot)$ trajectories

Figure 8.15: Exceptional closed–loop costs for $N = 30$ and various optimization and differential equation solver tolerances

observed closed–loop costs arise due to the repeated transitions upright positions. For the controller shown in Figure 8.15(a) the situation is even worse since such a random behaviour of the closed–loop solution is obtained for all considered pairs $(\text{tol}_{\text{SQP}}, \text{tol}_{\text{ODE}})$. The reason for this unpredictable behaviour can be found in our implementation: Since the closed–loop trajectory is always computed with accuracy $\text{tol}_{\text{ODE}} = 10^{-10}$ and the open–loop trajectories are computed with different tolerance levels of the differential equation solver, the solutions deviate slightly. If the pendulum is close in the upright position, then this deviation may shift the closed–loop state vector to the other side of this position. Now, the initial guess of the control points in the wrong direction and the optimization routine identifies a different upright position as the local optimum since it can be reached by one swing–over before the end of the optimization horizon is reached.

Remark 8.12

There are different ways to cope with this problem. For example, additional constraints can be added once a trajectory is close to one target in order to exclude other target points. Alternatively, the initial guess of the control may be changed on start of the optimization. A third and very sophisticated possibility is to add shooting points and set their values to

the desired target. This is less restrictive than adding constraints and incorporates the aspect of modifying the initial guess at the same time.

Hence, as a general guideline, the tolerance levels of both the optimization routine and the differential equation solver should be chosen as large as possible without compromising stability of the closed–loop. For analytical reasons, the accuracy of the differential equation solver tol_{ODE} should not be chosen larger than tol_{SQP}, cf. Remark 8.11. For the considered inverted pendulum example, the choice $(\text{tol}_{\text{SQP}}, \text{tol}_{\text{ODE}}) = (10^{-4}, 10^{-4})$ appears to be appropriate since the additional accuracy does not cause a significant increase in the computing times.

8.3.2 Effects of the Horizon Length

As we have seen from the previous Sections 8.2 and 8.3.1, the horizon length must be chosen as short as possible to reduce the required computing time. However, the problem may become unstable if the horizon length is too short. Here, we consider the inverted pendulum example from Section 7.2 and, as in Section 8.3.1, we compute the closed–loop solution (2.16) on the interval $[0, 50]$ with sampling length $T = 0.1$ and initial value $x_0 = (2, 2, 0, 0)$ but fix the tolerance levels to $\text{tol}_{\text{ODE}} = 10^{-4}$ and $\text{tol}_{\text{SQP}} = 10^{-4}$.
As expected, the desired upright equilibrium is not stabilized for small horizon length N as shown in Figure 8.16.

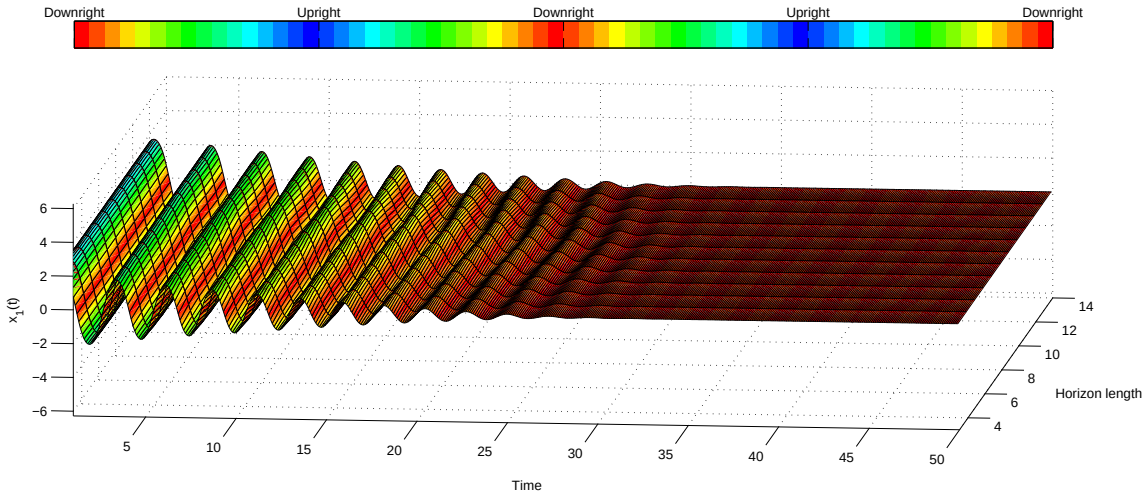


Figure 8.16: Comparison of closed–loop trajectories $x_1(\cdot)$ for optimization horizons N which are too small

For larger optimization horizons, the controller suddenly stabilizes the upright equilibrium $(-\pi, 0, 0, 0)$ as shown in Figure 8.17.
As shown in Section 8.3.1, there exists a range of optimization horizons N such that the corresponding receding horizon controller almost randomly switches the point to be stabilized, see also the discussion after Figure 8.15 for the reason of this behaviour and Remark 8.12 for methods to avoid this effect. In Figure 8.18, we display the corresponding $x_1(\cdot)$ trajectories.
The phenomenon of repeatingly switched target point is not restricted to the small range of optimization horizons N considered in Figure 8.18, but can also be experienced for some larger optimization horizons N as shown in Figure 8.19.

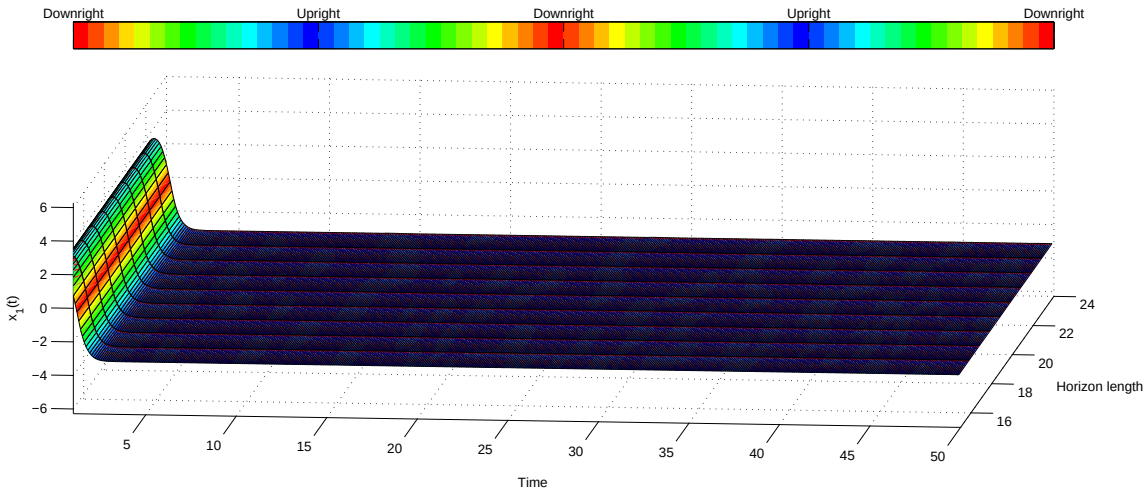


Figure 8.17: Comparison of closed–loop trajectories $x_1(\cdot)$ for various small optimization horizons N

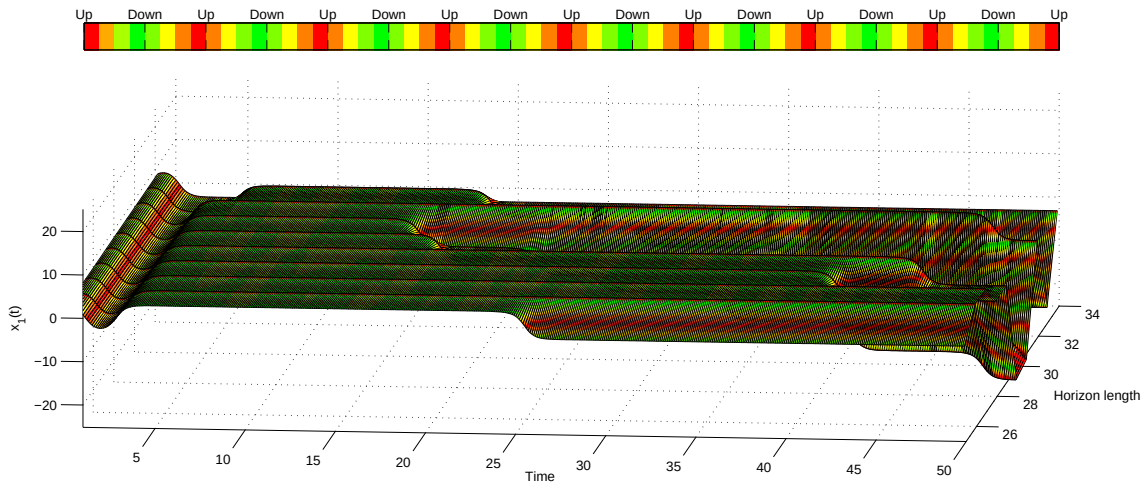


Figure 8.18: Exceptional "almost" stable closed–loop trajectories $x_1(\cdot)$ for various optimization horizons N

Within this figure, green represents the current state to be close to an upright position whereas red corresponds to the downward position. Here, we present these results for reasons of completeness, for our task of real–time applicability of the receding horizon controller, we stick to short horizons.

As a guideline, we obtain the set of interest $\mathcal{S}$ for the optimization horizon N. This set is bounded from below by the minimal optimization horizon which stabilizes the closed–loop system and from above by those horizons which exhibit unwanted estimation effects. Unfortunately, it is a priori not clear if $\mathcal{S}$ is nonempty and in many cases we have to deal with the mentioned estimation effects. Moreover, obtaining this set is heuristic and until now it is based on trial–and–error. For our inverted pendulum example, we obtain $\mathcal{S} := \{16, \ldots, 24\}$.

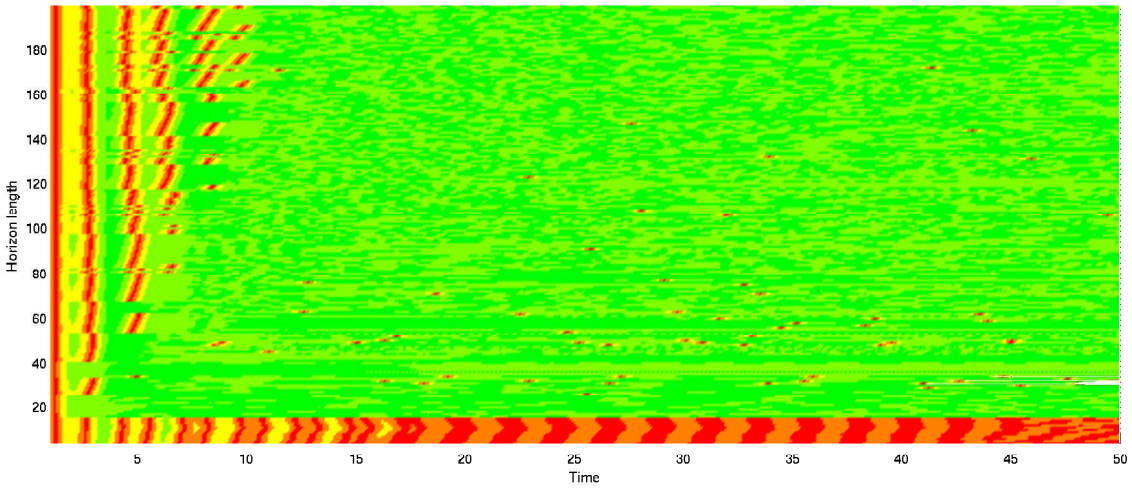


Figure 8.19: Comparison of closed–loop trajectories $x_1(\cdot)$ for various optimization horizons

8.3.3 Effects of Initial Guess

The problem of obtaining a good initial guess of the optimization variable appears twice in the receding horizon control setup and needs to be addressed in two ways:
First, as known from standard optimization theory, the optimization routine requires an initial guess which is sufficiently close to the optimum of the discretized first optimal control problem. Unfortunately, there does not exist a general recipe to develop such a control guess which is why we leave it to the user to supply the first initial guess.

Remark 8.13
An initial guess can be obtained heuristically, e.g. by knowledge and intuition of the considered problem. Different to that, in the case that a stabilizing continuous–time control law $u(\cdot)$ is known, then the emulated version of the control $u_T(\cdot)$ according to (1.20) *can be used as an initial guess. Another way of improving this first initial guess is by using multiple shooting nodes which we consider in the following Section 8.3.4.*
We also like to mention that combining the solutions of consecutive but shorter optimal control problems does not only not provide a good initial guess but also destabilizes the system in most cases. This can be seen directly from Figure 8.16: If the first optimization horizon is chosen too small, then the trajectory of the first optimal control problem does not converge to the desired equilibrium. Hence, the initial value for the consecutive optimal control problem is even further away from this equilibrium and the resulting control is highly unlikely to suit the stabilization task.

The second issue arises during the shift of the optimal control problem. Similar to the first point, an initial guess is required for the new (shifted) discretized problem. But now, we can exploit the structure of the receding horizon control problem and reuse the computed optimal solution of the previous problem as an initial guess of the actual problem by using its endpiece and append a suitable control vector to its end.
Considering the structure of the problem, our implementation contains three different strategies to obtain the required new control vector, see Section 6.2.1.5 for details. Here, we exclusively consider the cases `optimize`= 0 and `optimize`= 1. The option `optimize`= 2 is designed for m–step feedback laws and only of academic use since it causes a large additional computational effort.

To compare these two strategies, we consider the inverted pendulum problem with horizon lengths $N \in \mathcal{S} = \{16, \dots, 24\}$ and compute the closed–loop solution on the interval $[0, 50]$. In Table 8.4, the summarized results from this experiment are displayed where the closed–loop solution has been computed 200 times each to obtain reliable computing times.

N	`optimize`= 0				`optimize`= 1			
	t_{average}	$t_{\max}$	$t_{\min}$	$V_{500}^{\mu_N}$	t_{average}	$t_{\max}$	$t_{\min}$	$V_{500}^{\mu_N}$
16	5.7127	44.4997	2.9918	9.4096	4.5451	76.1882	2.8862	11.6181
17	7.0676	345.6728	3.2172	9.1515	3.9272	46.0116	3.1527	9.2273
18	6.6571	46.9765	3.4768	11.0712	5.0737	42.4183	3.5139	14.9645
19	8.1253	200.7184	3.7279	9.1481	6.6153	307.3953	3.6240	13.8994
20	8.1175	84.6425	4.0064	9.5446	6.6531	55.9213	3.8609	17.6396
21	9.1212	80.4660	4.2891	9.2145	8.1731	59.8523	4.1438	18.5700
22	9.3382	88.1437	4.6020	10.2978	9.3808	53.1171	4.5159	23.5261
23	9.6738	67.6229	4.9009	10.9821	8.3203	56.2793	4.7123	24.9199
24	10.2203	68.8702	5.2126	13.7050	6.2717	68.5017	5.0568	14.9428

Table 8.4: Comparison of shift routines

From Table 8.4 we obtain that the procedure using an optimization (`optimize`= 1) instead of a simple copy of the last control vector (`optimize`= 0) reduces the average computing time in every step of the RHC scheme by approximately 15%. Note that the resulting closed–loop costs are significantly higher. Here, this does not correspond to a repeated change of the point to be stabilized but is due to solutions shaking around the optimum which is induced by the overshoot of the added optimization.

8.3.4 Effects of Multiple Shooting Nodes

In the case of long optimization horizons N, solving the discretized optimal control problem SDOCP$_N$ becomes hard since the initial guess is difficult to obtain in general and usually renders the state trajectory to be far from optimal. Additionally, the endpieces of the state trajectories are sensitive to changes of the control at the beginning of the optimization horizon.
However, if we consider a setpoint regulation problem, see Remark 1.40, we have not used the information on the target point within the optimization process yet. In particular, defining multiple shooting points, see Definition 5.5, within endpieces of the state trajectories allows us to reduce the sensitivity of the solution trajectory on initial controls. Moreover, we can improve the initial guess of the control by setting these multiple shooting nodes to the value of the target setpoint.

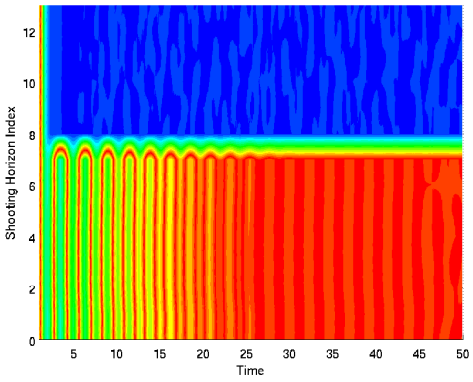
Remark 8.14
Setting multiple shooting nodes to the target value allows us to utilize local knowledge of the system, i.e. a local feedback. Hence, the trajectory emanating from this shooting node remains close to the target and the optimization is eased.

For the inverted pendulum example, the most critical trajectory component is the angle of the pendulum $x_1(\cdot)$. In the following, we illustrate the efficient use and effect of multiple shooting nodes for this variable for several reasons:
First of all, defining every sampling point in every dimension of the problem to be a sampling point slows down the optimization process significantly due to the enlarged

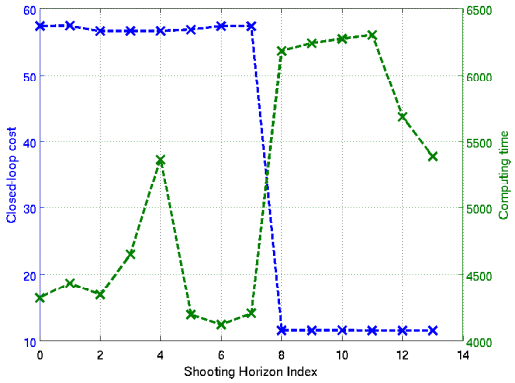
dimension of the optimization variable, see also Section 5.1. Hence, this number should be chosen as small as possible.
Secondly, for slow parts of the system, that is state trajectories which do not change their values quickly, the use of shooting points may obstruct the optimization since it is possible that there does not exist a control which links two consecutive shooting nodes. In this case, the optimization routine wastes iteration steps trying to adapt the value of the shooting nodes and is often unable to find a solution. However, setting sampling points to be multiple shooting points may result in significantly improving the initial guess of the control. Still, one has to keep in mind that this causes the optimization variable to be enlarged. Hence, computation of the gradient of the cost function as well as the Jacobian of the constraints require additional effort. Therefore, a balance between improving the initial guess of the control and the number of multiple shooting points must be found.
And finally, the use of multiple shooting nodes may also lead to unstable solutions even if the problem without shooting nodes is stable.

Remark 8.15
For our inverted pendulum example, the full discretization leads to unstable solutions while Figure 8.17 shows that we can stabilize the system for $N \geq 16$ using no shooting nodes.

In the following, we consider the horizons $N = 15$, 16 and 17 and solve the resulting receding horizon control problems on the interval $[0, 50]$ while setting one shooting node to a variable position $\varsigma(1)$ on the sampling grid of the first state dimension $x_1(\cdot)$, that is $\varsigma(1) \in \{0, \ldots, N-2\}$, and the corresponding initial value to $x_1 = \pi$. Here, the computing times are obtained by repeatedly solving each problem 200 times.



(a) $x_1(\cdot)$ trajectories



(b) Closed–loop costs and computing times

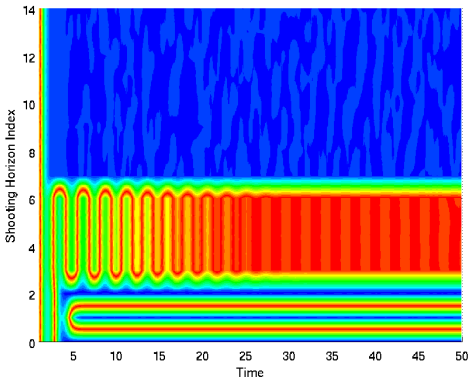
Figure 8.20: Results for a varying single shooting node for horizon length $N = 15$

As we can see from Figure 8.20(a), the state trajectory is stabilized if we add a shooting node for the first differential equation at position $\varsigma(1) \in \{t_9, \ldots, t_{14}\}$ in the time grid. Hence, using a single shooting node, we are now able to stabilize the problem for a reduced optimization horizon N.

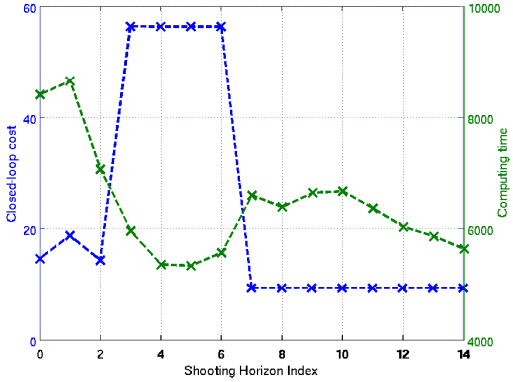
Remark 8.16
Note that the notation for positions in the time grid and the number of the shooting horizon index differ by one, see Remark 5.6 for details on this issue.

In Figure 8.20(b) the caused closed–loop costs and the required computing times of the different problems are displayed. Here, we can identify the stable solutions via the significantly reduced closed–loop costs which again serves as our indicator for stability in

the case of two varying shooting nodes. Additionally, we see that the computing costs are falling if the shooting horizon index of the shooting node is close to the end of the optimization horizon.
As mentioned earlier, adding shooting nodes may also lead to instability of the closed–loop. Such a situation is shown in Figure 8.21.



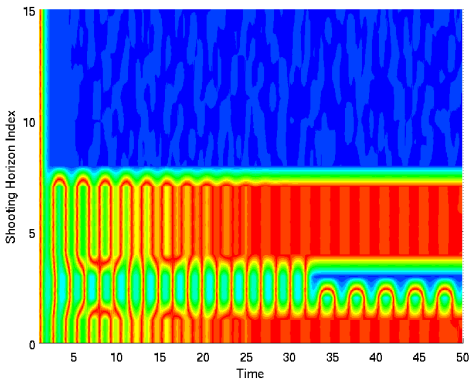
(a) $x_1(\cdot)$ trajectories



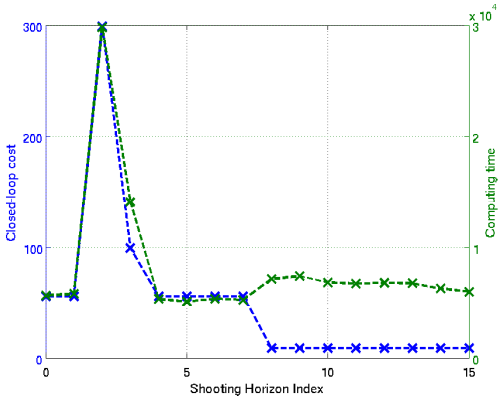
(b) Closed–loop costs and computing times

Figure 8.21: Results for a varying single shooting node for horizon length $N = 16$

Here, choosing shooting horizon index $\varsigma(1) \in \{3, \ldots, 6\}$ results in instability of the closed–loop. Moreover, $\varsigma(1) \in \{0, 2\}$ causes the controller to stabilize the upright position $(-\pi, 0, 0, 0)$ while for $\varsigma(1) \in \{1, 7, 8, \ldots, 14\}$ this changes to $(\pi, 0, 0, 0)$. Hence, by our numerical experience, the effect of stabilizing a chosen equilibrium by adding a shooting node can only be confirmed if $\varsigma(1)$ is set to a time instant in the last third of the time grid. This also holds true for Figure 8.22 which illustrates the results for $N = 17$.
Note that although the optimization variable has been enlarged by using a single shooting node, the computing times stay almost unchanged.



(a) $x_1(\cdot)$ trajectories



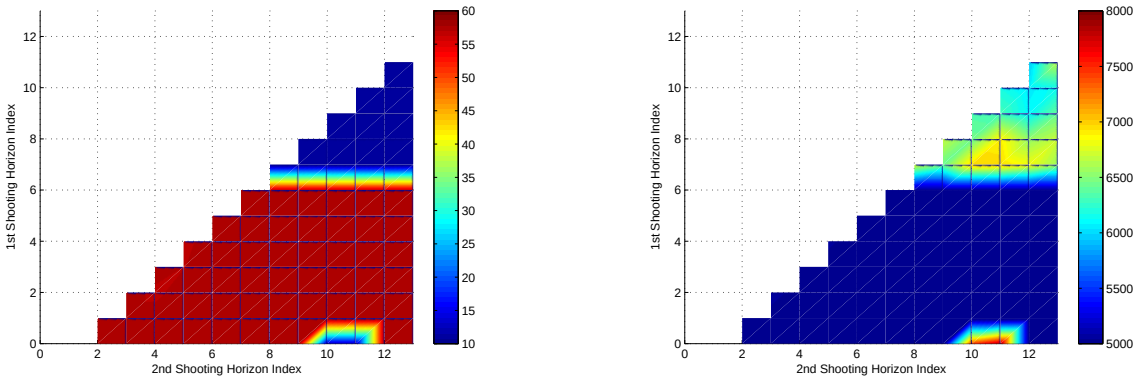
(b) Closed–loop costs and computing times

Figure 8.22: Results for a varying single shooting node for horizon length $N = 17$

Remark 8.17
Note that adding shooting nodes to a problem may cause the optimization routine to stabilize a different equilibrium. For our inverted pendulum example the choice of setting

$x_1 = \pi$ highlights the upright position $(\pi, 0, 0, 0)$ in the set of equilibrium points. Hence, this power has to used carefully since other choices may results in lower closed–loop costs and computing times.

Next, we add a second shooting node to our receding horizon control problem. Since our implementation expects the time indices of these points to be in ascending order, see Section 5.1.3, the following Figures 8.23 and 8.24 show closed–loop costs and computing times only for feasible combinations $(\varsigma(1), \varsigma(2)) \in \{0, \dots, N-2\}^2$.
Similar to the case with just one shooting node, the problem can be stabilized for a reduced horizon $N = 15$ as shown in Figure 8.23. Again, we see that — apart from a few exceptions — the time indices of the shooting nodes should be contained in the last third of the time grid.
For this example every added shooting node increases the computing times by around $10 - 15\%$. From our numerical experience and the definition of the optimization variable in Problem 5.7 we know that this percentage evolves indirectly proportional to the actual size of the optimization vector.



(a) Closed–loop costs

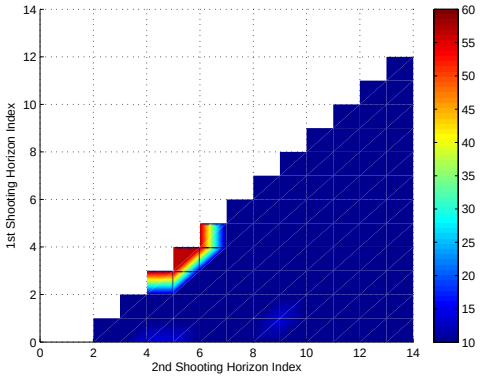
(b) Computing times

Figure 8.23: Results for two varying single shooting node for horizon length $N = 15$

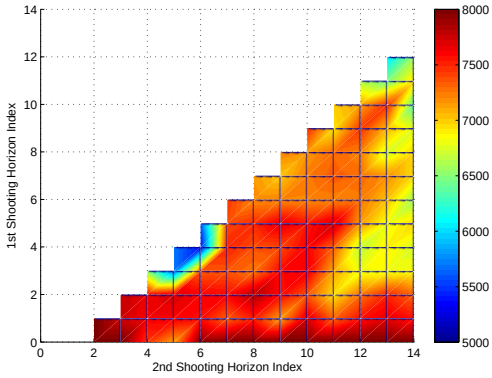
From Figures 8.24 (and 8.21, 8.22 respectively) to we see that this proportion is reduced as the horizon length N is increased. The smaller increase corresponds to the reduced percentage of shooting node variables in the total number of optimization variables and the constant number of equality constraints.
Moreover, we see that the closed–loop is stable for most combinations $(\varsigma(1), \varsigma(2))$. However, there are some exceptions which give us another nice heuristical insight: As Figures 8.24(a) and (c) show we obtain unstable closed–loops only for $\varsigma(1) \geq \varsigma(2) - 2$, i.e. if the two consecutive shooting nodes are close to each other in time.
All mentioned points strengthen the following heuristic:

- Keep the number of shooting nodes as small as possible
- Place the shooting node in the last third of the time grid
- Choose the time distance between two shooting nodes sufficiently large

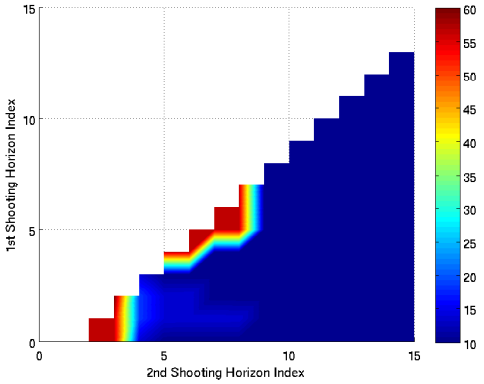
If these three issues are considered in the presented order, then it allows us to reduce the computing time by



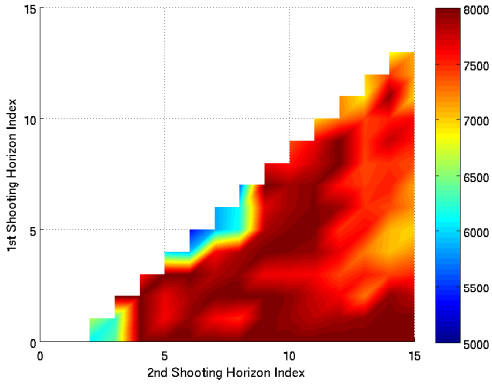
(a) Closed–loop costs for $N = 16$



(b) Computing times for $N = 16$



(c) Closed–loop costs for $N = 17$



(d) Computing times for $N = 17$

Figure 8.24: Results for two varying single shooting node for horizon length $N = 16, 17$

- improving the initial guess which results in a lower number of required optimization steps and by
- reducing the optimization horizon.

Remark 8.18
If the horizon length N is reduced further, there exists no setting of multiple shooting nodes which allows us to stabilize the closed–loop for this particular example.

Remark 8.19
Within the receding horizon control context, we also have to consider that information about the previous optimal control can be used to improve the initial guess of the actual optimization problem. Hence, the number of shooting points should be reconsidered in every step of the receding horizon control problem. To this end, the function `eventBeforeMPC()` *can be used, see Section 6.1.1.2. Here, however, we neither change the number nor the positions of the implemented shooting nodes.*

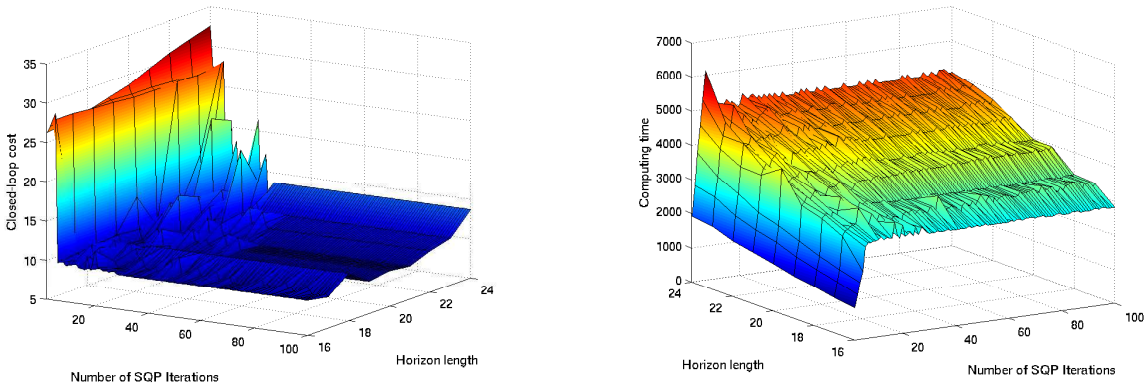
8.3.5 Effects of Stopping Criteria

Our last setscrew to be analyzed is the stopping criterion of the optimization routine. At present, there exist three different criteria in our implementation of the receding horizon control algorithm which can be used for this purpose. These are the already discussed

tolerance level of the optimizer, a maximal number of iterations and a time dependent method. Since we discussed the tolerance level before in Section 8.3.1, we do not consider this method again here. Note that the tolerance level and maximal iteration stopping mechanism are incorporated within the optimization routines itself and therefore always present while the latter method is supplementary imposed in the receding horizon control setting.

Here, we focus on the breaking method via fixing a maximal number of iterations of the optimization routine. There are several reasons for this choice: Most importantly, if we use the time dependent method to stop the optimization, the algorithm is no longer deterministic since random events in the background of the CPU interfere with the calculation and the closed–loop trajectories may vary drastically. Secondly, all effects can be noticed in the fixed case as well. Note that in both cases, the optimization can be terminated by the tolerance level criterion as well.

The following Figure 8.25 shows results for our setting of the inverted pendulum example. In particular, we consider the set of interesting horizon lengths $N \in \mathcal{S} = \{16, \ldots, 24\}$ for the set of maximal numbers of iterations $\texttt{maxiter} \in \{2, \ldots, 100\}$. Here, $\texttt{maxiter} = 1$ is excluded since it corresponds to evaluating the gradient of the cost function and the Jacobian of the constraints only, i.e. no actual computation of a search direction is performed.



(a) Closed–loop costs (b) Computing times

Figure 8.25: Closed–loop costs and computing times for various maximal numbers of SQP iterations and horizon lengths N

From Figure 8.25 we obtain that if $\texttt{maxiter}$ is chosen to be small, here $\texttt{maxiter} \in \{2, 3\}$, then the closed–loop system does not converge towards an upright position which corresponds to the large closed–loop costs displayed in Figure 8.25(a). Moreover, if the horizon N is chosen large, the maximal number of iteration is required to be larger than for small horizons. For our example and $N = 16$ the receding horizon controller using $\texttt{maxiter} = 4$ stabilizes the closed–loop in an upright position. For $N = 24$, however, $\texttt{maxiter}= 7$ is required.

Hence, the parameter N should be chosen as small as possible to accelerate the receding horizon control algorithm: For one, small horizon lengths require less SQP steps in the optimization and therefore the computing time is reduced, see Figure 8.25(b). Additionally, the dynamic of the system can be solved faster as shown in Section 8.3.2.

Remark 8.20

The time dependent breaking criterion may be critical upon implementation since a priori

it is not clear how many iterations can be executed by the optimization routine. Moreover, the optimization routines allow for increases in the cost function to compensate for the Maratos effect, see Section 5.2.4.2. Hence, even if a number of iterations is performed, the actually best computed solution may still be the initial solution.
This also holds for the maximal number criterion. Here, however, we can choose the maximal number of iterations to be larger than the allowed number of trial steps. Hence, the outcome of the optimization is either an enhanced solution or the optimization algorithm is unable to improve the initial guess even if no additional breaking criterion is imposed.

Remark 8.21
Note that in this work we do not compare the different implemented optimization routines `NLPQLP`, `e04wdc` *and* `IpOpt`. *For the first two methods, a comparison can be found in [88]. These results confirm that in case of a large–scale optimization problem with only few optimization variables the method* `e04wdc`, *i.e. a class* `SqpNagC` *object, shows better performance than the routine* `NLPQLP`. *For real–time applications, however, mainly small optimization problem are considered. For these problems, a class* `SqpFortran` *object shows outstanding performance, see also Remark 5.63 for details.*

8.4 Suboptimality Results

From the previous Section 8.3 we know how a receding horizon controller can be configured to stabilize the system under control and at the same time to show low computational costs. Yet, at runtime we do not know if the resulting closed–loop is controlled to the desired equilibrium and if yes, how large the additional cost for using the truncated optimization horizon N is compared to the infinite one. Our aim in this section is to show and check the applicability of the suboptimality estimates of Propositions 3.3, 3.28 as well as Theorems 3.15, 3.22, 3.34 and 3.39 which are designed for this purpose. In order to illustrate our results from Chapter 3, we consider the digital redesign problem of the arm/rotor/platform (ARP) model as shown in Section 7.3.
Note that, in contrast to the previous Section 8.3 which aims for speed of the receding horizon control algorithm, we now want to compute a suboptimality estimate α which characterizes the decrease of the function $V_N(\cdot)$ along the closed–loop trajectory in the nonlinear distance defined by the stage cost $l(\cdot,\cdot)$.

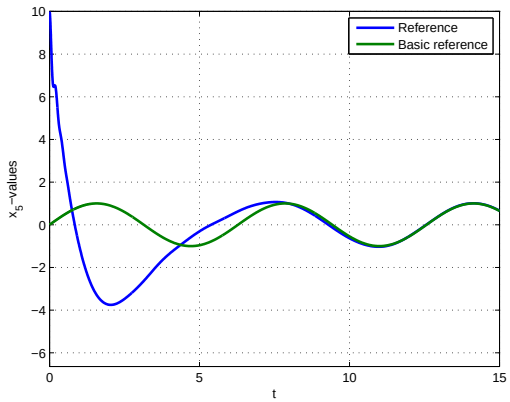


Figure 8.26: Reference function of the arm–rotor–platform model

Here, we use the best numerically possible solutions and we fix the absolute and relative tolerances for the solver of the differential equation to 10^{-10} and set the optimality tolerance of the SQP solver to 10^{-8}. Additionally, we define the length of the open–loop horizon within the receding horizon control algorithm to $N = 10$ and consider the reference function $v(t) = \sin(t)$. Last, if not stated differently, we consider the cost functional (7.41)

$$J(x_0, u) = \sum_{j=0}^{N} \int_{t_j}^{t_{j+1}} |x_{5,u}(t, x_0) - x_{5,\mathrm{ref}}(t)| dt.$$

The corresponding x_5–component of the trajectory is shown in Figure 8.26. We like to point out that the basic assumptions

$$V_N(x) \geq V_N(f(x, \mu_N(x))) + \alpha l(x, \mu_N(x))$$

or

$$V_N(x(n)) \geq V_N(x(n+1)) + \min\left\{\alpha\left(l(x(n), \mu_N(x(n))) - \varepsilon\right), l(x(n), \mu_N(x(n))) - \varepsilon\right\}$$

of Propositions 3.2 and 3.28 respectively do not rely on the definition $V_N(\cdot)$ as in Definition 2.15, that is

$$V_N(x(n)) := \inf_{u_N \in \mathbb{U}^N} J_N(x(n), u_N).$$

Within this section, we implicitly make use of this freedom of choice. In particular, we compute $V_N(x(\cdot))$ and $l(x(\cdot), \mu_N(x(\cdot)))$ numerically along the trajectory and use these approximations in the suboptimality estimates.
In the following Section 8.5, we relax the tolerance levels of both the optimizer and the differential equation solver and show applicability of the suboptimality estimates within the adaptation strategies presented in Chapter 4.

8.4.1 A posteriori Suboptimatity Estimate

Starting with our a posteriori estimate from Proposition 3.3 we obtain the data shown in Table 8.5 for the arm–rotor–platform example using stated setting of the receding horizon controller:

n	α	$V_N(x(n))$	$l(x(n), \mu_N(x(n)))$	Violation
0	0.99999631	0.55917846	0.47991166	
1	0.99998840	0.07926857	0.06836975	
2	0.99991231	0.01089961	0.00890766	
3	1.00000000	0.00199273	0.00169242	(KKT)
4	0.99739334	0.00030002	0.00023833	
5	0.99630676	0.00006231	0.00005041	
6	0.80184613	0.00001209	0.00000816	
7	-0.1941001	0.00000555	0.00000077	(3.8)
8	1.00000000	0.00000569	0.00000047	(3.8), (KKT)
9	0.76238278	0.00000450	0.00000206	
10	0.56527463	0.00000293	0.00000042	

Table 8.5: α values from Proposition 3.3

Note that the values of $V_N(x(n+1))$ which are needed to calculate α from Proposition 3.3 are obtained at no cost in the $(n+1)$th step of the receding horizon controller. According to this, α can be computed after the second step of the controller for the first time. Moreover, violations of the relaxed Lyapunov inequality (3.8) as well as errors coming from our optimization routine which are indicated by (KKT) are displayed in Table 8.5. In our simulation, the exact α-values from Proposition 3.3 are close to one for the first iteration steps indicating that the feedback is almost infinite horizon optimal. However,

from iteration step six onwards the values become smaller and even negative which shows that optimality is lost here. One possible reason for this is that here the values of $V_N(\cdot)$ are close to the accuracy of the optimization routine and the tolerances of the solver of the differential equations, hence numerical errors become dominant. Nevertheless, the measured values of α in conjunction with the values of $V_N(\cdot)$ show that the closed–loop system behaves "almost optimal" until a very small neighborhood of the reference trajectory is reached which is exactly what we expected to happen, see the introduction of Section 3.3.

Remark 8.22
One has to keep in mind that the minimal α is giving an estimate for the degree of suboptimality along the trajectory. Here, we mentioned all α values to show the progress within the calculation.

Remark 8.23
Although it is not our primary interest one can observe the influence of the choice of the cost function using the calculated α values. To show this we test the second cost functional (7.42)

$$J(x_0, u) = \sum_{j=0}^{N} \int_{t_j}^{t_{j+1}} \|x_u(t, x_0) - x_{ref}(t)\|_2^2 dt$$

which gives us the estimates for α_2 stated in Table 8.6.
This difference can also be illustrated by plotting the trajectories of the state $x_5(\cdot)$ and its tracking reference $x_{ref}(\cdot)$. Note that the estimate α is the degree of suboptimality for a fixed cost functional and hence it has got no meaning for different ones. This is due to the fact that we cannot qualify one resulting receding horizon control law to be superior since the usage of different stage costs $l(\cdot,\cdot)$ results in uncomparable closed–loop costs $V_\infty(\cdot)$. Still, one can see that the closed–loop trajectory using the first running cost converges faster towards the tracking signal and we also obtain larger α values and particularly small values of the optimal value function $V_N(\cdot)$.
Although the observed differences between the suboptimality estimate α and the value function $V_N(x(0))$ may give us an intuition of how to construct cost functionals which are appropriate for our control task, a design method is not at hand.

n	α	α_2	$V_N(x(n))$	$l(x(n), \mu_N(x(n)))$
0	0.99999631	0.99985713	5044.50874629	4389.32095396
1	0.99998840	0.99940034	655.81491586	626.11058543
2	0.99991231	0.99181745	30.07978235	27.39755105
3	1.00000000	0.89245119	2.90641322	1.28994349
4	0.99630676	0.87663185	1.75520163	0.78653864
5	0.96778356	0.86449625	1.06569680	0.46395693
6	0.80184613	0.88072356	0.66460777	0.28546064
7	-0.1941001	0.87676324	0.41319585	0.17505245
8	1.00000000	0.81896807	0.25971630	0.09739556
9	0.76238278	0.82787192	0.17995245	0.07273550
10	0.56527463	0.84014543	0.11973677	0.05141013

Table 8.6: Comparison of α values from Proposition 3.3 for different cost functionals

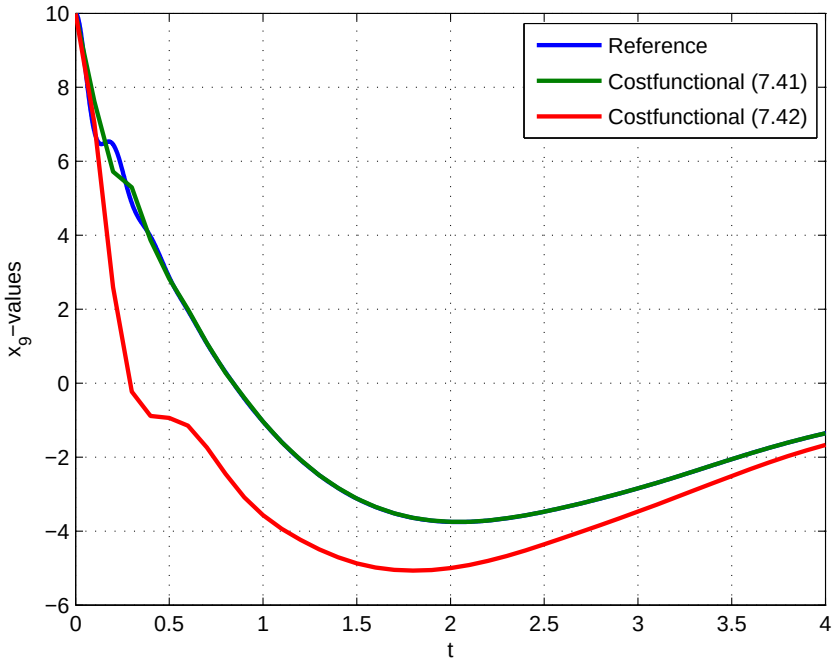


Figure 8.27: Comparison of trajectories for cost functionals (7.41) and (7.42)

8.4.2 A priori Suboptimality Estimate

Now we compare our results from Proposition 3.3 and Theorem 3.15 to see how conservative the a priori estimate actually is. Again, we display errors of the optimization routine by (KKT) and additionally indicate violations of Assumption 3.11 in this table.

n	α (Prop. 3.3)	α (Theorem 3.15)	γ (Ass. 3.11)	Violations
0	0.99999631	0.99997706	0.46683396	
1	0.99998840	-264.92387	19.8500131	
2	0.99991231	-2182.4879	50.5375020	
3	1.00000000	-1214.4208	38.6167593	(KKT)
4	0.99739334	-1068.8198	36.4481006	
5	0.99630676	-139.27856	15.2657865	
6	0.80184613	-571.23289	27.5835655	
7	-0.1941001	-565.22399	27.4561781	(3.12)
8	2.53347027	-300.29165	20.9219290	
9	0.76238278	-19.854158	7.52312945	
10	0.56527463	-132.33098	14.9583235	

Table 8.7: Comparing results from Proposition 3.3 and Theorem 3.15

As shown in Table 8.7, almost none of the computed local suboptimality values α is positive. Hence, for this example, the a priori estimate from Theorem 3.15 is too conservative and we are unable to guarantee stability of the closed–loop system if we consider the inequalities from Assumption 3.11 required for Theorem 3.15.

Remark 8.24
If one prolongates the optimization horizon one may expect better results. However, numerical experiments have shown that γ is growing significantly if N is enlarged since the trajectory is much closer to the steady state and computational errors come into play. Unfortunately, (3.18) *cannot be used to directly recognize this interaction.*
Conversely, if the horizon length is too short, there may not be enough extractable information about stability and suboptimality within the open–loop solution.

If we apply Theorem 3.22 for various N_0 we expect better results due to the theoretical background stated in Section 3.2. This can be confirmed by numerical experiments. Here, we compare our results from Proposition 3.3 and Theorems 3.15 and 3.22 in Table 8.8.

	$N_0 = 2$		$N_0 = 5$		$N_0 = 9$	
n	α	γ	α	γ	α	γ
0	0.99997706	0.46683395	0.99998313	0.24294141	0.99644951	0.16030633
1	-264.92387	19.8500134	0.99452510	0.69188060	0.99118052	0.22081641
2	-2182.4879	50.5375029	0.99960136	0.41987289	0.99453994	0.18641408
3	-1214.4208	38.6167594	0.40089028	2.07184767	0.98784725	0.24749807
4	-1068.8198	36.4481006	-1159.3136	36.4481004	0.98790473	0.24707919
5	-139.27856	15.2657861	-168.69106	15.2657865	0.89210579	0.55107453

Table 8.8: Comparing results from Theorem 3.22 for various N_0

From Table 8.8 one can clearly see that the enhanced a priori estimate given by Theorem 3.22 is by far not as conservative as the a priori estimate given by Theorem 3.15 which does not even guarantee stability. Compared to results for Proposition 3.3 we obtain very good estimates. Table 8.7 also shows that one cannot specify a "best" N_0. However, since all necessary values are computable or can be estimated easily, one can use a simple maximization to obtain the "best" possible estimate α.

Remark 8.25
Using a maximization over all possible values of N_0 is applicable for academic use since we do not have to care about computing times. If, however, we aim at reducing the computing time for the receding horizon controller by applying our adaptation strategies presented in Chapter 4, this time issue becomes critical if the a priori estimates are used, see Section 8.5 below.

8.4.3 A posteriori practical Suboptimality Estimate

In Table 8.5 we observed that our estimates deteriorate when the values of $V_N(x(n))$ or $l(x(n), \mu_n(x(n)))$ are very small, implying that optimality of the trajectories is lost in a small neighborhood of the reference solution. Note that this is also true for Tables 8.7 and 8.8 which we truncated for reasons of clarity. One can only speculate whether this is due to sampling or due to numerical optimization errors, most likely the effect is caused by a combination of both. In either case, one would expect to obtain better estimates when considering practical optimality.
In this section, we apply Proposition 3.28 to our setting of the receding horizon controller for the arm–rotor–platform model and obtain the following values for α for horizon length $N = 10$ and $\varepsilon = 10^{-6}$.

n	α	$V_N(x(n))$	$l(x(n), \mu_N(x(n)))$
0	0.99999840	0.55917846	0.47991166
1	1.00000000	0.07926857	0.06836975
2	1.00000000	0.01089961	0.00890766
3	1.00000000	0.00199273	0.00169242
4	1.00000000	0.00030002	0.00023833
5	1.00000000	0.00006231	0.00005041
6	0.91383624	0.00001209	0.00000816
7	1.00000000	0.00000555	0.00000077
8	1.00000000	0.00000569	0.00000047
9	1.00000000	0.00000450	0.00000206
10	1.00000000	0.00000293	0.00000042

Table 8.9: α values from Proposition 3.28

Here, we do not compare results for suboptimality and practical suboptimality since it is clear from the inequalities that the estimates for practical suboptimality always yield better α values.
We also like to point out that for $n = 7$ and 8 we have $l(x(n), \mu_N(x(n))) < \varepsilon$. Since this corresponds to the chosen ε-region we have $\alpha = 1$ by definition. For $n = 9$, however, we leave the ε-region but the optimization routine takes us back there in only one step. Hence, using the notation from Proposition 3.28, we obtain two intervals $I_1 = \{0, 6\}$ and $I_2 = \{9\}$ for this example.

Remark 8.26
If one wants to obtain a feasible and in some sense optimal ε, a nonlinear optimization problem can be generated where the constraints are given by (3.27) *and the cost function can be chosen freely. Note that this optimization problem is growing during the receding horizon control process since starting from the second iteration for every single iteration one nonlinear constraint has to be added.*
We like to point out that the choice of the cost function is critical for the optimization itself since in general α is forced to be 1 whereas ε is almost 0. Hence, since numerical errors occur within the receding horizon control algorithm, it is sometimes rather tricky to obtain any result different from 1 and 0 respectively.

Remark 8.27
Again, note that the values of α are computed separately for each point. To obtain the closed–loop suboptimality estimate, the minimum of the computed local suboptimality estimates has to be considered.

8.4.4 A priori practical Suboptimality Estimate

Last, we compare the results from Theorem 3.39 and all the previously mentioned results.

n	α (Prop. 3.3)	α (Theorem 3.22)	α (Prop. 3.28)	α (Theorem 3.39)
0	0.99999631	0.99998316	0.99999840	0.99999994
1	0.99998840	0.99982736	1.00000000	0.99999822
2	0.99991231	0.99899284	1.00000000	0.99894664
3	1.00000000	0.99775792	1.00000000	0.99995965
4	0.99739334	0.98790477	1.00000000	0.99999651

n	α (Prop. 3.3)	α (Theorem 3.22)	α (Prop. 3.28)	α (Theorem 3.39)
5	0.99630676	0.96963602	1.00000000	0.99999264
6	0.80184613	0.97978377	0.91383624	0.99999198
7	-0.1941000	-0.6194016	1.00000000	1.00000000
8	1.00000000	-300.29165	1.00000000	1.00000000
9	0.76238278	0.97854192	1.00000000	0.99967378
10	0.56527463	0.91644361	1.00000000	1.00000000

Table 8.10: Comparing results from Propositions 3.3, 3.28 and Theorems 3.22, 3.39

In Table 8.9 we used the same setting as in Section 8.4.3. To compute the results for Theorem 3.39 we solved all possible N_0–subproblems and took the maximal value of α. We may have stated the results for each N_0 separately but this will give us no more insight information than we already obtained in the non-practical case.

Remark 8.28
If exact solutions are at hand then the a priori estimate is always more conservative than the a posteriori estimate. In Table 8.9, however, this is not the case for the practical a posteriori and a priori suboptimality estimates for $n = 6$ due to numerical errors. Since we deal with numerical approximations such deviations cannot be prevented.

We observe that using the practical a posteriori or a priori estimates from Proposition 3.3 or Theorem 3.39, we can guarantee stability of the closed–loop by combining our local suboptimality estimates to a closed–loop suboptimality estimate. For the a posteriori and a priori estimates of Proposition 3.28 and Theorem 3.22, however, this is not possible since negative α values occur. These may be due to numerical errors which occur at runtime but it is also not clear whether the system is asymptotically stabilizable using a class of piecewise constant control functions. For an adaptation algorithm as proposed in Chapter 4, this is critical since negative values of α cause a prolongation of the optimization horizon.

Remark 8.29
If conditions (3.6) *or* (3.27) *can be satisfied in every step of the receding horizon control algorithm, then they may also be used as a stopping criterion for the optimization routine. Unfortunately, we observed that the applicability assumption does not hold in general. One possibility to circumvent this problem is to apply our adaptation strategies from Chapter 4 which are designed guarantee* (3.6) *or* (3.27) *to hold. Advantages and disadvantages of this stopping criterion in comparison to other criteria have not been tested by now.*

8.5 Adaptivity

Knowing the properties and weaknesses of our suboptimality estimate from Chapter 3 and the previous Section 8.4, we again consider the arm–rotor–platform model described in Section 7.3 in numerical experiments to analyze our adaptation strategies of Chapter 4 within this last section. This example is appropriate for this test since, using tracking examples, one requires long optimization horizons to handle the transient behaviour while close to the desired equilibrium short horizons are sufficient to stabilize the system.
To this end, we design simulation settings of the ARP model in Section 8.5.1 which are typical for a RHC application. For these situations, we state results and computing times for a standard receding horizon controller with minimal but fixed optimization horizon

which still guarantees a fixed suboptimality degree $\overline{\alpha}$. These results serve us as benchmark for our adaptation strategies. Starting with our simple implementation in Section 8.5.2 and the more sophisticated fixed point and monotone prolongation strategies in Section 8.5.3, we present trajectories and computing times for our competitive adaptive receding horizon controller implementation. Last, in Section 8.5.4, we use Theorems 4.4 and 4.6 to interpret the local suboptimality estimates along the closed–loop solution. Moreover, we draw some conclusions concerning the practicability of the considered adaptation strategies.

8.5.1 Setup of the Simulations

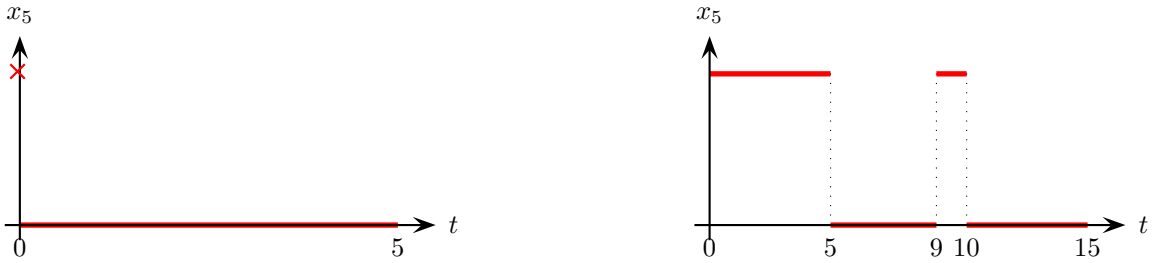
For our simulation, we create two different reference functions $v(\cdot)$ of the form (7.40) for the continuous–time controller. The resulting two simulations are typical for a receding horizon controller setting.

(1) Constant setpoint tracking: The problem is initialized far from the desired equilibrium. Hence, the continuous–time controller causes a typical transient behaviour of the reference state trajectory $x_{\mathrm{ref}}(\cdot)$. Here, we consider the reference function $v_1(t) \equiv 0$, compare Figure 8.28(a).

(2) Switching setpoint tracking: Upon initialization, the problem is operating at the desired setpoint. During the runtime, this setpoint is changed several times. The recovery times are not equidistant and the switching times are not required to be sampling instances. For our simulation, we consider the reference function

$$v_2(t) = \begin{cases} 10, & t \in [0,5) \cup [9,10) \\ 0, & t \in [5,9) \cup [10,15) \end{cases},$$

see also Figure 8.28(b).



(a) Reference function $v_1(t)$ (b) Reference function $v_2(t)$

Figure 8.28: Typical tracking functions for a receding horizon controller

For these two situations, we now develop reference values using a standard receding horizon controller without adaptation of the optimization horizon.
Considering the first situation, we use the horizon length $N = 6$ which guarantees a local suboptimality degree $\overline{\alpha} = 0.5$ and is minimal, i.e. choosing $N < 6$ results in a closed–loop suboptimality degree $\alpha < \overline{\alpha}$. Here, we measure the local suboptimality degree by using the practical a posteriori estimate of Proposition 3.28. Moreover, we set the tolerance levels of both the minimizer and the differential equation solver to 10^{-6} and the truncation constant of the running costs to $\varepsilon = 10^{-5}$. For this setup of the receding

horizon controller, we obtain the results and computing times displayed in Table 8.11 for our first simulation setting which serve us as a benchmark for the different adaptive receding horizon controller implementations.

$t\ [s]$	$x_{5,\mathrm{ref}}(t)$	$x_5(t)$	α	Time $[10^{-3}s]$
0.00	10.00000	10.00000	1.00000	124.97
0.20	-3.31615	-0.61539	0.99984	85.67
0.40	-5.82954	-7.14547	1.00000	32.77
0.60	-1.82597	-1.52865	1.00000	47.82
0.80	-0.24045	-0.26322	1.00000	25.24
1.00	0.03912	0.05703	1.00000	24.48
1.20	0.01567	-0.00547	1.00000	20.50
1.40	-0.01729	-0.01245	1.00000	10.26
1.60	-0.01926	-0.02112	1.00000	23.01
1.80	-0.00666	-0.00550	1.00000	23.50
2.00	0.00678	0.01020	1.00000	20.38
2.20	0.01538	0.01804	1.00000	4.94
2.40	0.01660	0.01688	1.00000	18.81
2.60	0.01023	0.00800	1.00000	20.14
2.80	-0.00077	-0.00356	1.00000	17.74
3.00	-0.01088	-0.01316	1.00000	9.17
3.20	-0.01504	-0.01501	1.00000	4.71
3.40	-0.01178	-0.00954	1.00000	18.86
3.60	-0.00363	-0.00157	1.00000	19.26
3.80	0.00515	0.00664	1.00000	17.30
4.00	0.01087	0.01206	1.00000	4.92
4.20	0.01137	0.01069	1.00000	8.55
4.40	0.00664	0.00524	1.00000	18.67
4.60	-0.00099	-0.00173	1.00000	9.18
4.80	-0.00771	-0.00792	1.00000	18.60
5.00	-0.01026	-0.00984	1.00000	8.61
				628.52

Table 8.11: Results for a standard RHC in a steady state situation

Remark 8.30
Within this and the following tables, we only show results for the state $x_5(\cdot)$ since the aim of the receding horizon control law is to keep the distance $x_5(\cdot)$ and $x_{5,ref}(\cdot)$ as small as possible.

From the computing times (and the suboptimality estimate α) we can see that the transient phase of the state trajectory is computationally demanding. Here, computing times are usually larger than 30ms. Once the state trajectory is close to the steady state given by $v_1(\cdot)$, the computing times shrink drastically. This indicates that only small adjustments have to be made during the optimization process which may allow us to reduce the length of the optimization horizon in this time phase of the control process.

Remark 8.31
Here, we only display the first 26 steps of the receding horizon control process. The aim of the considered exemplary setting, however, is to analyze the impact of using adaptation

algorithms on computing times for long–term applications. Hence, we do not consider the time interval $[0, 5]$ *for our comparison of the standard receding horizon controller and the various adaptive implementations but the larger interval* $[0, 50]$.

For our second standard situation, we consider a similar setting for the receding horizon controller. Again, we set $\varepsilon = 10^{-5}$ and both optimality and differential equation solver tolerances to 10^{-6}. Given the reference function $v_2(\cdot)$, we can guarantee local suboptimality degree $\overline{\alpha} = 0.5$ for a horizon length $N = 6$ using the standard receding horizon controller without adaptation of the horizon length. For this setting, we obtain the following benchmark results:

t $[s]$	$x_{5,\mathrm{ref}}(t)$	$x_5(t)$	α	Time $[10^{-3}s]$
0.00	10.00000	10.00000	1.00000	117.60
0.20	-1.58301	1.36121	0.99978	86.50
0.40	0.86187	-0.67806	1.00000	56.09
0.60	7.53857	7.92565	1.00000	27.86
0.80	9.72219	9.69652	1.00000	26.71
1.00	9.99944	10.01864	1.00000	24.26
1.20	9.97356	9.95028	1.00000	21.36
1.40	9.94905	9.95711	1.00000	5.41
1.60	9.94825	9.94541	1.00000	6.94
1.80	9.95790	9.95897	1.00000	16.66
2.00	9.96823	9.96937	1.00000	16.62
2.20	9.97513	9.97506	1.00000	6.63
2.40	9.97666	9.97625	1.00000	8.00
2.60	9.97242	9.97088	1.00000	7.97
2.80	9.96432	9.96343	1.00000	15.79
3.00	9.95634	9.95693	1.00000	8.48
3.20	9.95256	9.95304	1.00000	8.58
3.40	9.95453	9.95589	1.00000	4.38
3.60	9.96052	9.96216	1.00000	9.11
3.80	9.96726	9.96654	1.00000	9.60
4.00	9.97188	9.96909	1.00000	7.89
4.20	9.97270	9.96988	1.00000	7.78
4.40	9.96951	9.96747	1.00000	4.19
4.60	9.96384	9.96317	1.00000	7.45
4.80	9.95851	9.95952	1.00000	7.87
5.00	9.95617	9.95820	1.00000	83.69
5.20	8.23689	8.07522	1.00000	51.24
5.40	3.63383	3.74642	1.00000	43.14
5.60	0.70561	0.67299	0.98167	21.23
5.80	-0.01646	-0.02819	1.00000	41.15
6.00	-0.06209	-0.05688	1.00000	23.70
6.20	0.01084	0.00917	1.00000	12.86
6.40	0.02821	0.03045	1.00000	22.75
6.60	0.00426	0.00231	1.00000	23.38
6.80	-0.01988	-0.02589	1.00000	24.05
7.00	-0.03267	-0.03846	1.00000	21.37
7.20	-0.03075	-0.03546	1.00000	9.43

t $[s]$	$x_{5,\mathrm{ref}}(t)$	$x_5(t)$	α	Time $[10^{-3}s]$
7.40	-0.01394	-0.01482	1.00000	20.34
7.60	0.01013	0.01285	1.00000	21.60
7.80	0.02806	0.03249	1.00000	20.46
8.00	0.03036	0.03430	1.00000	17.94
8.20	0.01864	0.02039	1.00000	11.11
8.40	0.00116	0.00078	1.00000	22.41
8.60	-0.01458	-0.01669	1.00000	22.99
8.80	-0.02307	-0.02697	1.00000	21.15
9.00	-0.02087	-0.02422	1.00000	81.88
9.20	1.73013	1.93590	1.00000	54.71
9.40	6.50679	6.37147	1.00000	27.32
9.60	9.38915	9.41591	1.00000	3.23
9.80	9.92415	9.95285	1.00000	11.44
10.00	9.93538	9.92177	1.00000	10.68
10.20	9.93837	9.93994	1.00000	83.06
10.40	8.24106	8.16091	0.99142	74.01
10.60	3.95424	3.95015	1.00000	20.75
10.80	0.82267	0.83935	1.00000	42.20
11.00	0.07257	0.03354	1.00000	44.22
11.20	-0.08727	-0.07722	1.00000	42.61
11.40	0.00089	-0.00799	1.00000	13.15
11.60	0.05221	0.05321	1.00000	5.50
11.80	0.02750	0.02721	1.00000	40.26
12.00	-0.01136	-0.01213	1.00000	40.19
12.20	-0.04392	-0.04059	1.00000	22.05
12.40	-0.05830	-0.05940	1.00000	10.06
12.60	-0.04345	-0.04732	1.00000	35.16
12.80	-0.00432	-0.00537	1.00000	24.83
13.00	0.03525	0.03666	1.00000	38.96
13.20	0.05172	0.05341	1.00000	12.77
13.40	0.04277	0.04109	1.00000	5.34
13.60	0.01957	0.01880	1.00000	37.44
13.80	-0.00830	-0.00772	1.00000	23.60
14.00	-0.03170	-0.03311	1.00000	21.28
14.20	-0.04034	-0.04609	1.00000	5.33
14.40	-0.02832	-0.03037	1.00000	20.29
14.60	-0.00138	0.00257	1.00000	22.54
14.80	0.02426	0.02836	1.00000	22.64
15.00	0.03476	0.03936	1.00000	21.10
			0.98167	1946.72

Table 8.12: Results for a standard RHC in a switching steady state situation

For this example, we can clearly identify the switching instances of the reference function $v_2(\cdot)$ from the values of $x_{5,\mathrm{ref}}(\cdot)$ and $x_5(\cdot)$. Additionally, we observe drastical increases in the computing times once a switch in the reference function occurs. Similar to the first example, calculation times during the transient phases are significantly larger than during the recovery phases. Moreover, we only experience suboptimality estimates α

which are smaller than one during these phases. These two facts indicate that during the transient phase an adaptation algorithm will in general have to increase the length of the optimization horizon. For the recovery phases, we expect the horizon length to be shortened since by the low computing times we can conclude that only minor changes of the open–loop control are made by the optimization routine.

Remark 8.32
In contrast to the first situation which deals with long–term aspects of the comparison of standard and adaptive receding horizon controllers, this second one is designed to exhibit advantages and disadvantages of the presented adaptation strategies given a specific switching of steady states. In particular, we want to clarify whether the additional effort for enlarging the optimization horizon pays off during shortening phases.

8.5.2 Simple Shortening and Prolongation

Having the benchmark results using the standard receding horizon controller for both situations stated in the previous Section 8.5.1, we now present results for receding horizon controllers which are based on

- the simple shortening and prolongation strategies 4.10, 4.14 using the a posteriori estimates of Propositions 3.3 or 3.28 or on
- the simple shortening and prolongation strategies 4.19, 4.21 using the a poriori estimates of Theorems 3.22 or 3.39.

In either case, we use a setup of the receding horizon controller which is — apart from the adaptation of the optimization horizon — identical to the benchmark standard receding horizon controller, i.e. we set both the optimizer and differential equation solver tolerances to 10^{-6} and the truncation constant of the running costs to $\varepsilon = 10^{-5}$. Moreover, we initialize the controller with a horizon length $N = 6$ and set $\overline{\alpha} = 0.5$.

Reference		Proposition 3.3				Proposition 3.28			
t $[s]$	$x_{5,\mathrm{ref}}(t)$	$x_5(t)$	α	Time $[10^{-3}s]$	N	$x_5(t)$	α	Time $[10^{-3}s]$	N
0.00	10.00000	10.00000	0.99996	498.72	6	10.00000	0.99997	489.29	6
0.20	-3.31615	-0.61541	1.00000	161.96	5	-0.61541	1.00000	163.14	5
0.40	-5.82954	-7.14043	1.00000	90.85	4	-7.14043	1.00000	91.81	4
0.60	-1.82597	-1.52993	1.00000	48.48	3	-1.52993	1.00000	48.55	3
0.80	-0.24045	-0.26170	0.93143	20.66	2	-0.26170	0.93871	20.23	2
1.00	0.03912	0.04677	0.70421	5.94	2	0.04677	0.73161	6.01	2
1.20	0.01567	-0.00454	0.73270	6.40	2	-0.00454	0.81033	6.07	2
1.40	-0.01729	-0.00049	1.00000	15.14	3	-0.00049	1.00000	15.45	3
1.60	-0.01926	-0.03437	0.72729	10.35	2	-0.03437	1.00000	10.29	2
1.80	-0.00666	-0.00367	1.00000	10.03	3	-0.00367	1.00000	2.54	2
2.00	0.00678	0.00804	1.00000	164.93	8	0.00698	1.00000	2.38	2
2.20	0.01538	0.01636	1.00000	383.49	12	0.01695	1.00000	2.28	2
2.40	0.01660	0.01607	1.00000	658.44	16	0.02079	1.00000	2.35	2
2.60	0.01023	0.00996	1.00000	261.27	16	0.01574	1.00000	2.37	2
2.80	-0.00077	-0.00012	1.00000	375.33	17	0.00424	1.00000	2.30	2
3.00	-0.01088	-0.00983	1.00000	344.48	18	-0.00807	1.00000	1.44	2
3.20	-0.01504	-0.01444	1.00000	316.03	17	-0.01508	1.00000	2.19	2

Reference		Proposition 3.3				Proposition 3.28			
t $[s]$	$x_{5,\mathrm{ref}}(t)$	$x_5(t)$	α	Time $[10^{-3}s]$	N	$x_5(t)$	α	Time $[10^{-3}s]$	N
3.40	-0.01178	-0.01174	1.00000	347.72	18	-0.01276	1.00000	2.27	2
3.60	-0.00363	-0.00497	1.00000	440.56	19	-0.00483	1.00000	2.13	2
3.80	0.00515	0.00330	0.99327	322.08	18	0.00370	1.00000	2.19	2
4.00	0.01087	0.00931	1.00000	258.22	18	0.01108	1.00000	2.15	2
4.20	0.01137	0.00987	1.00000	423.02	19	0.01331	1.00000	2.31	2
4.40	0.00664	0.00609	1.00000	1878.04	24	0.00940	1.00000	2.28	2
4.60	-0.00099	-0.00059	0.77468	2106.05	28	0.00133	1.00000	2.22	2
4.80	-0.00771	-0.00638	0.97135	1327.45	30	-0.00684	1.00000	2.07	2
5.00	-0.01026	-0.00805	0.51607	2780.64	34	-0.01109	1.00000	2.18	2
				13215.00				872.38	

Table 8.13: Results for the ARP model using Algorithms 4.8, 4.10, 4.14 based on estimates of Propositions 3.3 or 3.28 given a steady state situation

Considering the results based on the practical estimate of Proposition 3.28, we observe exactly those results we expected: During the transient phase the required computing times are large and due to the additional optimal control problem to be solved to obtain the required suboptimality estimate $\alpha(N)$, these computing times exceed those of caused by the standard receding horizon controller. Yet, we also obtain that the optimization horizon N is shrinking during this phase such that for $t \geq 0.8$ the required computing times of the adaptive receding horizon controller in each step are smaller than those of our benchmark, cf. Table 8.11.
The results which are computed using the estimate of Proposition 3.3, on the other hand, show not only large horizons but also large computing times. Moreover, this implementation of the adaptive receding horizon controller fails to guarantee a local suboptimality degree $\overline{\alpha} = 0.5$ on the complete considered interval $[0, 50]$. Since it is not clear whether the sampled–data problem is asymptotically stabilizable and since we use numerical approximations to solve the optimal control problem, we cannot expect this approximation to reveal good results.

Remark 8.33
Note that the optimization horizon is not shrinking to $N = 1$ which we explicitly excluded for our controller. From an analytical point of view this should not be a problem, cf. Propositions 3.3 and 3.28. During our numerical tests, however, $N = 1$ in almost every case leads to instability of the closed–loop. Unfortunately, this instability does not appear in our suboptimality degree $\alpha(N)$ for several iterations and stays therefore uncompensated. If we exclude the case $N = 1$, however, this problem never occured during our tests of the algorithm.

Coming to our a priori estimates of Theorems 3.22 and 3.39, we again use an identical setup of the controller as in the benchmark case. Here, we always consider $N_0 = N - 1$ which has shown to be an effective choice during our test runs.

Reference		Theorem 3.22				Theorem 3.39			
t $[s]$	$x_{5,\mathrm{ref}}(t)$	$x_5(t)$	α	Time $[10^{-3}s]$	N	$x_5(t)$	α	Time $[10^{-3}s]$	N
0.00	10.00000	10.00000	0.98802	462.29	6	10.00000	0.98803	443.22	6

Reference		Theorem 3.22				Theorem 3.39			
t $[s]$	$x_{5,\mathrm{ref}}(t)$	$x_5(t)$	α	Time $[10^{-3}s]$	N	$x_5(t)$	α	Time $[10^{-3}s]$	N
0.20	-3.31615	-0.61789	0.99760	223.42	5	-0.61789	0.99760	222.94	5
0.40	-5.82954	-7.14123	0.99663	75.28	4	-7.14123	0.99670	76.17	4
0.60	-1.82597	-1.52114	0.99002	22.15	3	-1.52114	0.99116	22.27	3
0.80	-0.24045	-0.27003	0.94674	9.85	3	-0.27003	0.95112	9.87	3
1.00	0.03912	0.05914	0.83568	10.23	3	0.05914	1.00000	10.36	3
1.20	0.01567	0.00837	1.00000	23.92	4	0.00837	1.00000	8.83	3
1.40	-0.01729	-0.01208	0.98841	18.83	4	-0.00713	1.00000	9.18	3
1.60	-0.01926	-0.01468	1.00000	17.91	3	-0.01796	1.00000	7.19	3
1.80	-0.00666	0.00158	1.00000	10.77	4	-0.00530	1.00000	3.61	3
2.00	0.00678	0.00267	1.00000	7.01	3	0.00391	1.00000	3.50	3
2.20	0.01538	0.01609	0.53451	2.66	3	0.01178	1.00000	3.10	3
2.40	0.01660	0.01823	0.56256	3.51	3	0.02103	1.00000	5.40	3
2.60	0.01023	0.01043	0.69038	3.23	3	0.01329	1.00000	3.68	3
2.80	-0.00077	0.00131	0.77679	3.13	3	-0.00119	1.00000	3.41	3
3.00	-0.01088	-0.01106	0.75806	7.03	3	-0.00834	1.00000	3.21	3
3.20	-0.01504	-0.01547	1.00000	79.67	6	-0.01799	1.00000	6.97	3
3.40	-0.01178	-0.01009	1.00000	59.27	5	-0.01399	1.00000	3.58	3
3.60	-0.00363	-0.00596	1.00000	16.63	4	-0.00425	1.00000	3.43	3
3.80	0.00515	0.00487	1.00000	7.87	3	0.00037	1.00000	2.45	3
4.00	0.01087	0.01307	0.58641	3.50	3	0.00973	1.00000	2.89	3
4.20	0.01137	0.00945	0.91588	2.47	3	0.01680	1.00000	5.31	3
4.40	0.00664	0.00613	1.00000	41.79	6	0.00958	1.00000	2.75	3
4.60	-0.00099	-0.00094	1.00000	20.30	5	-0.00233	1.00000	2.68	3
4.80	-0.00771	-0.00840	1.00000	9.88	4	-0.00761	1.00000	3.10	3
5.00	-0.01026	-0.00960	1.00000	5.61	3	-0.01396	1.00000	5.19	3
				1138.29				864.83	

Table 8.14: Results for the ARP model using Algorithms 4.16, 4.19, 4.21 based on estimates of Theorems 3.22 or 3.39 given a steady state situation

Compared to the a posteriori estimates, the a priori estimate designed for the non–practical case shows better performance. However, it is also unable to compute a closed–loop solution with guaranteed local suboptimality estimate $\overline{\alpha} = 0.5$ on the complete considered interval $[0, 50]$.

Using the practical estimate of Theorem 3.39, we observe that the computing times during the transient phase are larger than in Table 8.11 for standard RHC but smaller than using the a posteriori estimate, cf. Table 8.13. Since the additional computational cost for the a posteriori estimate originate from solving twice as many optimal control problems as in the standard case and the a priori estimate is designed to reduce this effort, this reduction is not surprising. More surprisingly, the smallest optimization horizon for this problem is $N = 3$. One reason for this choice may be that the estimate for $N_0 = 2$ is very conservative, see also Tables 8.7 and 8.8. Due to this (internal) choice of the algorithm, the computing times once the state trajectory is close to the desired steady state are larger than using the a posteriori estimate. Still, the total computing time for this exemplary setting as displayed at the bottom of Tables 8.13 and 8.14 show that the a priori estimate reveals an effective adaptation strategy.

Since our second situation, the switching setpoint tracking, is based on the same model, we can expect that the non–practical estimates exhibit an adaptation algorithm with poor performance. Hence, we only consider the practical estimate of Proposition 3.28 and Theorem 3.39. In order to be comparable to the benchmark results of the standard receding horizon controller, we use an identical setup, i.e. we set the tolerances levels to 10^{-6} and $\varepsilon = 10^{-5}$. Moreover, we initialize the controller with a horizon length $N = 6$ and set $\overline{\alpha} = 0.5$.

Reference		Proposition 3.28				Theorem 3.39			
t $[s]$	$x_{5,\mathrm{ref}}(t)$	$x_5(t)$	α	Time $[10^{-3}s]$	N	$x_5(t)$	α	Time $[10^{-3}s]$	N
0.00	10.00000	10.00000	0.99996	479.44	6	10.00000	0.98611	461.94	6
0.20	-1.58301	1.36119	1.00000	183.19	5	1.36494	0.99721	226.10	5
0.40	0.86187	-0.67151	1.00000	93.82	4	-0.66782	0.99585	111.68	4
0.60	7.53857	7.92313	1.00000	50.43	3	7.92586	0.97686	51.73	3
0.80	9.72219	9.69712	0.92963	20.93	2	9.69940	0.96025	21.43	3
1.00	9.99944	10.00726	0.71314	6.66	2	10.02277	1.00000	64.88	5
1.20	9.97356	9.94931	0.77431	6.11	2	9.94895	1.00000	24.25	4
1.40	9.94905	9.96907	1.00000	12.08	3	9.96158	1.00000	9.50	3
1.60	9.94825	9.92436	1.00000	9.53	2	9.94130	1.00000	3.06	3
1.80	9.95790	9.95685	1.00000	1.68	2	9.95880	1.00000	3.56	3
2.00	9.96823	9.96938	1.00000	2.46	2	9.97193	1.00000	3.36	3
2.20	9.97513	9.97495	1.00000	2.21	2	9.98037	1.00000	6.17	3
2.40	9.97666	9.97737	1.00000	2.17	2	9.97848	1.00000	3.50	3
2.60	9.97242	9.97457	1.00000	2.25	2	9.97291	1.00000	3.51	3
2.80	9.96432	9.96671	1.00000	2.22	2	9.97117	1.00000	2.30	3
3.00	9.95634	9.95817	1.00000	2.13	2	9.96476	1.00000	2.33	3
3.20	9.95256	9.95322	1.00000	2.07	2	9.95871	1.00000	2.27	3
3.40	9.95453	9.95402	1.00000	2.05	2	9.95675	1.00000	2.15	3
3.60	9.96052	9.95945	1.00000	2.09	2	9.95946	1.00000	2.82	3
3.80	9.96726	9.96635	1.00000	1.30	2	9.96503	1.00000	2.83	3
4.00	9.97188	9.97156	1.00000	1.31	2	9.97066	1.00000	2.74	3
4.20	9.97270	9.97339	1.00000	1.96	2	9.97382	1.00000	2.12	3
4.40	9.96951	9.96956	1.00000	2.25	2	9.97315	1.00000	2.25	3
4.60	9.96384	9.96349	1.00000	2.08	2	9.96904	1.00000	2.26	3
4.80	9.95851	9.95928	1.00000	7.09	2	9.96354	1.00000	9.39	3
5.00	9.95617	9.95638	0.93036	41.44	3	9.95937	0.96739	69.25	4
5.20	8.23689	8.08290	0.98630	27.94	2	8.07478	0.96229	44.74	3
5.40	3.63383	3.72689	0.97849	8.94	2	3.74030	0.85671	21.44	3
5.60	0.70561	0.65233	0.98991	7.41	2	0.66903	0.53104	15.28	3
5.80	-0.01646	-0.02347	0.91672	2.68	2	-0.01702	1.00000	29.51	4
6.00	-0.06209	-0.06160	1.00000	3.02	2	-0.05325	1.00000	20.65	3
6.20	0.01084	0.01625	1.00000	16.65	3	0.00363	1.00000	4.50	3
6.40	0.02821	0.03695	1.00000	10.69	2	0.03232	1.00000	7.02	3
6.60	0.00426	0.01318	1.00000	2.69	2	0.00506	1.00000	3.69	3
6.80	-0.01988	-0.01674	1.00000	1.52	2	-0.01771	1.00000	3.53	3
7.00	-0.03267	-0.03111	1.00000	2.20	2	-0.02647	0.84307	3.27	3
7.20	-0.03075	-0.03365	1.00000	2.77	2	-0.03341	1.00000	6.35	3
7.40	-0.01394	-0.02124	1.00000	16.65	3	-0.01641	1.00000	8.42	3

Reference		Proposition 3.28				Theorem 3.39			
t $[s]$	$x_{5,\mathrm{ref}}(t)$	$x_5(t)$	α	Time $[10^{-3}s]$	N	$x_5(t)$	α	Time $[10^{-3}s]$	N
7.60	0.01013	0.01130	1.00000	9.58	2	0.01331	1.00000	4.08	3
7.80	0.02806	0.03258	1.00000	2.46	2	0.02421	1.00000	3.39	3
8.00	0.03036	0.02818	1.00000	2.33	2	0.02748	1.00000	6.39	3
8.20	0.01864	0.02064	1.00000	2.31	2	0.01860	1.00000	6.83	3
8.40	0.00116	0.00695	1.00000	2.44	2	-0.00075	1.00000	3.50	3
8.60	-0.01458	-0.00871	1.00000	2.34	2	-0.01093	1.00000	3.38	3
8.80	-0.02307	-0.01903	1.00000	5.98	2	-0.01590	1.00000	7.70	3
9.00	-0.02087	-0.02000	0.93499	43.98	3	-0.02035	0.97887	72.43	4
9.20	1.73013	1.92919	0.97768	35.27	2	1.93631	0.97696	46.92	3
9.40	6.50679	6.38804	0.95393	9.36	2	6.37470	0.89778	28.79	3
9.60	9.38915	9.43669	0.99774	3.11	2	9.41498	0.98502	13.70	3
9.80	9.92415	9.94472	1.00000	3.55	2	9.95211	1.00000	10.46	3
10.00	9.93538	9.92173	1.00000	9.31	2	9.92691	1.00000	17.32	3
10.20	9.93837	9.93531	0.56752	8.81	2	9.94172	0.70144	24.46	3
10.40	8.24106	8.18114	0.94721	8.90	2	8.15265	0.61188	19.71	3
10.60	3.95424	3.98536	1.00000	27.96	3	3.95759	0.97954	46.89	4
10.80	0.82267	0.84555	1.00000	25.45	2	0.84762	0.87465	51.46	5
11.00	0.07257	0.03571	0.81082	15.52	3	0.02282	0.99208	34.82	4
11.20	-0.08727	-0.07157	1.00000	109.04	6	-0.06908	1.00000	10.67	3
11.40	0.00089	-0.01035	1.00000	45.99	5	-0.00059	1.00000	4.42	3
11.60	0.05221	0.04969	1.00000	21.89	4	0.04750	1.00000	7.99	3
11.80	0.02750	0.03031	1.00000	16.03	3	0.02493	1.00000	8.63	3
12.00	-0.01136	-0.01327	1.00000	5.55	2	-0.01433	1.00000	3.62	3
12.20	-0.04392	-0.04314	1.00000	2.23	2	-0.03997	1.00000	3.52	3
12.40	-0.05830	-0.05803	1.00000	2.86	2	-0.05222	1.00000	7.44	3
12.60	-0.04345	-0.04973	1.00000	16.71	3	-0.04089	1.00000	8.72	3
12.80	-0.00432	-0.00509	1.00000	6.58	2	-0.00053	1.00000	8.91	3
13.00	0.03525	0.03709	1.00000	2.50	2	0.03616	1.00000	3.97	3
13.20	0.05172	0.04796	1.00000	2.22	2	0.05222	1.00000	6.61	3
13.40	0.04277	0.04587	1.00000	2.45	2	0.04176	1.00000	7.65	3
13.60	0.01957	0.02729	1.00000	13.59	3	0.01631	1.00000	8.06	3
13.80	-0.00830	-0.00651	1.00000	5.03	2	-0.01018	1.00000	3.53	3
14.00	-0.03170	-0.03354	1.00000	2.37	2	-0.02858	1.00000	3.52	3
14.20	-0.04034	-0.03894	1.00000	3.09	2	-0.03610	1.00000	7.26	3
14.40	-0.02832	-0.03178	1.00000	2.97	2	-0.02684	1.00000	8.48	3
14.60	-0.00138	-0.00984	1.00000	2.92	2	0.00110	1.00000	7.79	3
14.80	0.02426	0.01445	1.00000	2.36	2	0.02440	1.00000	3.87	3
15.00	0.03476	0.02829	1.00000	1.62	2	0.03471	1.00000	6.48	3
			0.56752	1484.05			0.53104	1771.40	

Table 8.15: Results for the ARP model using Algorithms 4.16, 4.19, 4.21 based on estimates of Theorems 3.22 or 3.39 given a switching steady state situation

From Table 8.15 we obtain that almost all observations made from Tables 8.13 and 8.14 carry over to the second situation. The remarkable difference here is that the total time required by the a posteriori based adaptation algorithm is significantly lower than for the

one based on the a priori estimate.
Apart from this difference we see that both implementations react quickly to the changed circumstances and enlarge the optimization horizon once a switch in the function $v_2(\cdot)$ occurs. Moreover, these prolongations are small for either of the estimates and the optimization horizon is reduced just as in the case of the first exemplary setting. Still, we have to keep in mind that a prolongation from $N = 3$ to $N = 6$ requires in total four calls of computing the underlying suboptimality estimate $\alpha(N)$, $N \in \{3, 4, 5, 6\}$. Hence, a large number of optimal control problems need to be solved. To reduce the corresponding computational effort, we now analyze more sophisticated prolongation strategies.

8.5.3 Fixed Point and Monotone Iteration

In the previous Section 8.5.2 we observed that prolongating the optimization horizon is computationally costy. To deal with this issue, we introduced prolongation strategies which compute a horizon length N which guarantees a given lower suboptimality bound $\overline{\alpha} \in (0, 1)$ in Sections 4.3.1, 4.3.2 and 4.3.3. Here, our hope is that using these strategies not only reduces the number of optimal control problems to be solved during the iteration process of the prolongation strategies, but also that these strategies do not significantly overestimate the required horizon length. If such an overestimates occurs, then the computing times necessary to obtain the suboptimality degree $\alpha(N)$ explode which is the reason for introducing upper bounds σ on the increase of the optimization horizon.
For the given two typical situations described in Section 8.5.1, we now use the prolongation strategies of Algorithms 4.23 or 4.29 instead of the simple prolongation strategy of Algorithm 4.21 in our adaptive receding horizon controller to compute a closed–loop solution.

Remark 8.34
Since in the constant setpoint tracking setting no prolongations occur, we skip the close analysis of results for this situation here. Yet, we state summarized results for this example in Table 8.18 at the end of this section.

We analyze results for our second standard situation using the a priori estimate of Proposition 3.28. Similar to the previous Section 8.5.2, we set tolerances levels to 10^{-6} and $\varepsilon = 10^{-5}$. Moreover, we initialize the controller with $\sigma = 5$, horizon length $N = 6$ and set $\overline{\alpha} = 0.5$ which renders our results comparable to the benchmark results from Table 8.11 and the results of the simple adaptation strategy from Tables 8.13 and 8.14.

Reference		Fixed Point Strategy				Monotone Iteration Strategy			
t $[s]$	$x_{5,\mathrm{ref}}(t)$	$x_5(t)$	α	Time $[10^{-3}s]$	N	$x_5(t)$	α	Time $[10^{-3}s]$	N
0.00	10.00000	10.00000	0.98716	444.39	6	10.00000	0.98716	448.09	6
0.20	-1.58301	1.36494	0.99791	219.32	5	1.36494	0.99791	219.44	5
0.40	0.86187	-0.66782	0.99940	109.17	4	-0.66782	0.99940	108.75	4
0.60	7.53857	7.92586	1.00000	50.86	3	7.92586	1.00000	50.69	3
0.80	9.72219	9.69940	1.00000	20.90	3	9.69940	1.00000	20.69	3
1.00	9.99944	10.02277	1.00000	13.05	3	10.02277	1.00000	12.97	3
1.20	9.97356	9.94058	1.00000	10.06	3	9.94058	1.00000	9.95	3
1.40	9.94905	9.95398	1.00000	8.47	3	9.95398	1.00000	8.38	3
1.60	9.94825	9.94354	1.00000	6.86	3	9.94354	1.00000	6.81	3
1.80	9.95790	9.95718	1.00000	3.35	3	9.95718	1.00000	3.33	3

Reference		Fixed Point Strategy				Monotone Iteration Strategy			
t $[s]$	$x_{5,\mathrm{ref}}(t)$	$x_5(t)$	α	Time $[10^{-3}s]$	N	$x_5(t)$	α	Time $[10^{-3}s]$	N
2.00	9.96823	9.97071	1.00000	6.06	3	9.97071	1.00000	6.00	3
2.20	9.97513	9.97672	1.00000	2.56	3	9.97672	1.00000	2.56	3
2.40	9.97666	9.97765	1.00000	6.06	3	9.97765	1.00000	6.00	3
2.60	9.97242	9.97410	1.00000	3.40	3	9.97410	1.00000	3.38	3
2.80	9.96432	9.96488	1.00000	2.43	3	9.96488	1.00000	2.39	3
3.00	9.95634	9.96113	1.00000	2.21	3	9.96113	1.00000	2.20	3
3.20	9.95256	9.95633	1.00000	1.47	3	9.95633	1.00000	1.45	3
3.40	9.95453	9.95461	1.00000	2.11	3	9.95461	1.00000	2.09	3
3.60	9.96052	9.95736	1.00000	2.20	3	9.95736	1.00000	2.18	3
3.80	9.96726	9.96304	1.00000	2.22	3	9.96304	1.00000	2.20	3
4.00	9.97188	9.96871	1.00000	1.54	3	9.96871	1.00000	1.51	3
4.20	9.97270	9.97212	1.00000	2.00	3	9.97212	1.00000	1.99	3
4.40	9.96951	9.97174	1.00000	2.09	3	9.97174	1.00000	2.06	3
4.60	9.96384	9.96763	1.00000	2.14	3	9.96763	1.00000	2.13	3
4.80	9.95851	9.96221	1.00000	9.30	3	9.96221	1.00000	9.28	3
5.00	9.95617	9.95820	1.00000	93.09	5	9.95820	1.00000	93.59	5
5.20	8.23689	8.07627	1.00000	80.09	4	8.07627	1.00000	79.78	4
5.40	3.63383	3.74506	1.00000	26.89	3	3.74506	1.00000	27.37	3
5.60	0.70561	0.66530	1.00000	8.90	3	0.66530	1.00000	8.88	3
5.80	-0.01646	-0.02483	1.00000	11.84	3	-0.02483	1.00000	11.74	3
6.00	-0.06209	-0.04484	1.00000	10.56	3	-0.04484	1.00000	10.47	3
6.20	0.01084	0.01071	1.00000	4.23	3	0.01071	1.00000	4.22	3
6.40	0.02821	0.02928	1.00000	6.64	3	0.02928	1.00000	6.58	3
6.60	0.00426	0.00533	1.00000	3.63	3	0.00533	1.00000	3.61	3
6.80	-0.01988	-0.01728	1.00000	3.57	3	-0.01728	1.00000	3.53	3
7.00	-0.03267	-0.02483	1.00000	3.28	3	-0.02483	1.00000	3.25	3
7.20	-0.03075	-0.03130	1.00000	6.27	3	-0.03130	1.00000	6.19	3
7.40	-0.01394	-0.01565	1.00000	8.11	3	-0.01565	1.00000	8.15	3
7.60	0.01013	0.01293	1.00000	3.96	3	0.01293	1.00000	4.02	3
7.80	0.02806	0.02449	1.00000	2.62	3	0.02449	1.00000	2.62	3
8.00	0.03036	0.02815	1.00000	6.25	3	0.02815	1.00000	6.21	3
8.20	0.01864	0.01894	1.00000	6.69	3	0.01894	1.00000	6.65	3
8.40	0.00116	-0.00088	1.00000	3.45	3	-0.00088	1.00000	3.41	3
8.60	-0.01458	-0.01086	1.00000	3.29	3	-0.01086	1.00000	3.28	3
8.80	-0.02307	-0.01584	1.00000	7.47	3	-0.01584	1.00000	7.48	3
9.00	-0.02087	-0.02021	0.99994	175.81	6	-0.02021	0.99957	61.21	4
9.20	1.73013	1.93424	1.00000	134.34	5	1.93630	1.00000	46.25	3
9.40	6.50679	6.38096	1.00000	55.31	4	6.37466	1.00000	28.19	3
9.60	9.38915	9.41895	1.00000	21.66	3	9.41499	1.00000	13.49	3
9.80	9.92415	9.94756	1.00000	10.96	3	9.95210	1.00000	10.24	3
10.00	9.93538	9.92633	0.84837	19.56	3	9.92690	1.00000	17.09	3
10.20	8.20383	8.09010	1.00000	26.37	3	9.94172	1.00000	23.99	3
10.40	3.87739	3.87063	1.00000	20.53	3	8.15263	1.00000	19.34	3
10.60	0.78059	0.83502	1.00000	25.91	3	3.95760	1.00000	24.84	3
10.80	0.22849	0.16371	1.00000	27.73	3	0.84434	1.00000	15.61	3
11.00	0.01671	0.05055	1.00000	15.39	3	0.03000	1.00000	11.52	3

Reference		Fixed Point Strategy				Monotone Iteration Strategy			
t $[s]$	$x_{5,\mathrm{ref}}(t)$	$x_5(t)$	α	Time $[10^{-3}s]$	N	$x_5(t)$	α	Time $[10^{-3}s]$	N
11.20	-0.01193	-0.02744	1.00000	10.06	3	-0.05878	1.00000	11.27	3
11.40	0.03573	0.03096	1.00000	7.23	3	-0.00103	1.00000	4.45	3
11.60	0.03929	0.03435	1.00000	9.85	3	0.05418	1.00000	8.91	3
11.80	0.01125	0.00754	1.00000	8.38	3	0.02844	1.00000	8.79	3
12.00	-0.01994	-0.02310	1.00000	3.56	3	-0.01625	1.00000	8.07	3
12.20	-0.04373	-0.04052	1.00000	3.53	3	-0.04474	1.00000	3.42	3
12.40	-0.04791	-0.04467	1.00000	8.08	3	-0.06046	1.00000	7.28	3
12.60	-0.02611	-0.02528	1.00000	9.07	3	-0.04282	1.00000	9.10	3
12.80	0.01051	0.01362	1.00000	4.01	3	0.00036	1.00000	8.51	3
13.00	0.03800	0.03371	1.00000	3.56	3	0.03599	1.00000	3.87	3
13.20	0.04255	0.03903	1.00000	6.43	3	0.05213	1.00000	6.53	3
13.40	0.02953	0.02928	1.00000	8.36	3	0.04164	1.00000	7.51	3
13.60	0.00818	0.00533	1.00000	7.89	3	0.01638	1.00000	8.03	3
13.80	-0.01481	-0.01612	1.00000	3.41	3	-0.01021	1.00000	3.46	3
14.00	-0.03119	-0.03015	1.00000	3.39	3	-0.02850	1.00000	3.46	3
14.20	-0.03246	-0.03444	1.00000	8.09	3	-0.03582	1.00000	7.15	3
14.40	-0.01634	-0.01765	1.00000	9.01	3	-0.02671	1.00000	8.31	3
14.60	0.00803	0.01090	1.00000	3.94	3	0.00105	1.00000	7.63	3
14.80	0.02559	0.02233	1.00000	3.35	3	0.02443	1.00000	3.84	3
15.00	0.02861	0.02585	1.00000	7.90	3	0.03462	1.00000	6.33	3
			0.84837	1866.56			0.98716	1617.99	

Table 8.16: Results for the ARP model using Algorithms 4.16, 4.19 and Algorithms 4.23 or 4.29 based on estimates of Theorem 3.39 given a switching steady state situation

Before we state a complete comparison of all adaptation methods, we use our practical a posteriori estimate of Proposition 3.28 within these more sophisticated prolongation strategies by utilizing Newton's method as described in Section 4.3.3. This gives us the following data:

Reference		Fixed Point Strategy				Monotone Iteration Strategy			
t $[s]$	$x_{5,\mathrm{ref}}(t)$	$x_5(t)$	α	Time $[10^{-3}s]$	N	$x_5(t)$	α	Time $[10^{-3}s]$	N
0.00	10.00000	10.00000	1.00000	482.50	6	10.00000	1.00000	471.51	6
0.20	-1.58301	1.36119	1.00000	184.79	5	1.36119	1.00000	183.16	5
0.40	0.86187	-0.67151	1.00000	94.85	4	-0.67151	1.00000	94.40	4
0.60	7.53857	7.92313	1.00000	50.93	3	7.92313	1.00000	50.39	3
0.80	9.72219	9.69712	1.00000	20.80	2	9.69712	0.97629	20.56	2
1.00	9.99944	10.00726	1.00000	6.68	2	10.00726	0.93421	6.61	2
1.20	9.97356	9.94931	1.00000	6.18	2	9.94931	1.00000	6.15	2
1.40	9.94905	9.96907	1.00000	3.43	2	9.96907	1.00000	3.42	2
1.60	9.94825	9.93649	1.00000	3.16	2	9.93649	1.00000	3.15	2
1.80	9.95790	9.97901	1.00000	3.01	2	9.97901	1.00000	2.99	2
2.00	9.96823	9.98603	1.00000	3.03	2	9.98603	1.00000	3.01	2
2.20	9.97513	9.98900	1.00000	2.92	2	9.98900	1.00000	2.91	2
2.40	9.97666	9.98903	1.00000	3.03	2	9.98903	1.00000	2.97	2

Reference		Fixed Point Strategy				Monotone Iteration Strategy			
t $[s]$	$x_{5,\mathrm{ref}}(t)$	$x_5(t)$	α	Time $[10^{-3}s]$	N	$x_5(t)$	α	Time $[10^{-3}s]$	N
2.60	9.97242	9.98366	1.00000	2.89	2	9.98366	1.00000	2.88	2
2.80	9.96432	9.97427	1.00000	2.80	2	9.97427	1.00000	2.77	2
3.00	9.95634	9.96454	1.00000	2.26	2	9.96454	1.00000	2.25	2
3.20	9.95256	9.95845	1.00000	2.30	2	9.95845	1.00000	2.25	2
3.40	9.95453	9.95750	1.00000	2.25	2	9.95750	1.00000	2.24	2
3.60	9.96052	9.96141	1.00000	2.10	2	9.96141	1.00000	2.08	2
3.80	9.96726	9.96793	1.00000	2.04	2	9.96793	1.00000	2.04	2
4.00	9.97188	9.97293	1.00000	2.14	2	9.97293	1.00000	2.12	2
4.20	9.97270	9.97424	1.00000	2.16	2	9.97424	1.00000	2.15	2
4.40	9.96951	9.97142	1.00000	2.22	2	9.97142	1.00000	2.21	2
4.60	9.96384	9.96577	1.00000	2.13	2	9.96577	1.00000	2.12	2
4.80	9.95851	9.95996	1.00000	7.05	2	9.95996	1.00000	7.04	2
5.00	9.95617	9.95680	1.00000	75.09	4	9.95680	1.00000	75.21	4
5.20	8.23689	8.07466	1.00000	41.92	3	8.07466	1.00000	41.56	3
5.40	3.63383	3.74436	1.00000	22.30	2	3.74436	0.97020	22.27	2
5.60	0.70561	0.66432	1.00000	3.67	2	0.66432	0.93042	3.64	2
5.80	-0.01646	-0.03230	1.00000	5.97	2	-0.03230	1.00000	6.01	2
6.00	-0.06209	-0.07331	1.00000	5.90	2	-0.07331	1.00000	5.92	2
6.20	0.01084	0.00027	1.00000	3.27	2	0.00027	1.00000	3.29	2
6.40	0.02821	0.03661	1.00000	2.66	2	0.03661	1.00000	2.68	2
6.60	0.00426	-0.00639	1.00000	2.44	2	-0.00639	1.00000	2.44	2
6.80	-0.01988	-0.02585	1.00000	2.40	2	-0.02585	1.00000	2.38	2
7.00	-0.03267	-0.03726	1.00000	2.83	2	-0.03726	1.00000	2.85	2
7.20	-0.03075	-0.03794	1.00000	2.91	2	-0.03794	1.00000	2.93	2
7.40	-0.01394	-0.02486	1.00000	2.98	2	-0.02486	1.00000	2.99	2
7.60	0.01013	-0.00327	1.00000	2.90	2	-0.00327	1.00000	2.91	2
7.80	0.02806	0.01609	1.00000	2.40	2	0.01609	1.00000	2.41	2
8.00	0.03036	0.02337	1.00000	1.49	2	0.02337	1.00000	1.50	2
8.20	0.01864	0.01789	1.00000	2.20	2	0.01789	1.00000	2.18	2
8.40	0.00116	0.00339	1.00000	2.34	2	0.00339	1.00000	2.41	2
8.60	-0.01458	-0.01215	1.00000	2.22	2	-0.01215	1.00000	2.22	2
8.80	-0.02307	-0.02117	1.00000	5.88	2	-0.02117	1.00000	6.25	2
9.00	-0.02087	-0.02169	1.00000	100.98	5	-0.02169	1.00000	102.64	5
9.20	1.73013	1.93557	1.00000	60.63	4	1.93557	1.00000	61.24	4
9.40	6.50679	6.37381	1.00000	21.52	3	6.37381	1.00000	21.73	3
9.60	9.38915	9.40940	1.00000	14.62	2	9.40940	1.00000	14.74	2
9.80	9.92415	9.95993	1.00000	8.60	2	9.95993	1.00000	79.39	5
10.00	9.93538	9.95569	0.64306	8.77	2	9.94343	0.96083	98.39	4
10.20	8.20383	8.13191	1.00000	8.80	2	8.10270	0.75029	50.73	4
10.40	3.87739	3.90372	1.00000	8.26	2	3.87779	1.00000	61.33	3
10.60	0.78059	0.80230	1.00000	6.24	2	0.83116	1.00000	31.53	2
10.80	0.22849	0.16078	1.00000	6.51	2	0.17868	1.00000	6.59	2
11.00	0.01671	0.06359	1.00000	6.52	2	0.05029	1.00000	54.75	5
11.20	-0.01193	-0.03681	1.00000	3.42	2	-0.04120	1.00000	16.53	4
11.40	0.03573	0.04643	1.00000	3.26	2	0.03899	1.00000	9.40	3
11.60	0.03929	0.01836	1.00000	2.40	2	0.04169	1.00000	10.23	2

Reference		Fixed Point Strategy				Monotone Iteration Strategy			
t $[s]$	$x_{5,\mathrm{ref}}(t)$	$x_5(t)$	α	Time $[10^{-3}s]$	N	$x_5(t)$	α	Time $[10^{-3}s]$	N
11.80	0.01125	0.00254	1.00000	2.30	2	0.01018	1.00000	1.48	2
12.00	-0.01994	-0.02183	1.00000	2.18	2	-0.02085	1.00000	2.11	2
12.20	-0.04373	-0.04439	1.00000	2.62	2	-0.04287	1.00000	2.65	2
12.40	-0.04791	-0.05163	1.00000	2.84	2	-0.05014	1.00000	2.86	2
12.60	-0.02611	-0.03587	1.00000	2.97	2	-0.03539	1.00000	3.01	2
12.80	0.01051	-0.00444	1.00000	2.94	2	-0.00395	1.00000	3.02	2
13.00	0.03800	0.02382	1.00000	2.41	2	0.02432	1.00000	2.45	2
13.20	0.04255	0.03471	1.00000	2.34	2	0.03515	1.00000	2.32	2
13.40	0.02953	0.02887	1.00000	2.32	2	0.02918	1.00000	2.38	2
13.60	0.00818	0.01206	1.00000	2.33	2	0.01222	1.00000	2.37	2
13.80	-0.01481	-0.00948	1.00000	2.27	2	-0.00941	1.00000	2.32	2
14.00	-0.03119	-0.02685	1.00000	2.25	2	-0.02685	1.00000	2.25	2
14.20	-0.03246	-0.03176	1.00000	2.77	2	-0.03177	1.00000	2.80	2
14.40	-0.01634	-0.02027	1.00000	2.90	2	-0.02023	1.00000	2.87	2
14.60	0.00803	0.00081	1.00000	2.43	2	0.00091	1.00000	2.40	2
14.80	0.02559	0.01866	1.00000	2.27	2	0.01880	1.00000	2.28	2
15.00	0.02861	0.02602	1.00000	2.25	2	0.02614	1.00000	2.24	2
			0.64306	1348.87			0.75029	1692.62	

Table 8.17: Results for the ARP model using Algorithms 4.16, 4.19 and Algorithms 4.23 or 4.29 based on estimates of Proposition 3.28 given a switching steady state situation

Summarizing our results for both situations we obtain the following Tables 8.18 and 8.19. Note that in case of Table 8.18, we consider the interval $[0, 50]$ to obtain long–term conclusions.

Adaptive RHC		Computing times in $[10^{-3}s]$				
Implementation	Estimate	max	min	∅	$\varnothing_{[0,1)}$	$\varnothing_{[1,50)}$
Standard RHC	—	85.67	4.71	4.66	63.29	3.95
Simple Prolongation	Prop. 3.28	163.14	1.12	2.54	80.93	1.26
Simple Prolongation	Thm. 3.39	222.94	1.14	2.67	82.81	1.37
Fixed Point	Prop. 3.28	164.56	1.35	2.80	81.59	1.52
Fixed Point	Thm. 3.39	225.58	1.23	2.82	83.40	1.50
Monotone	Prop. 3.28	162.18	1.33	2.82	80.70	1.55
Monotone	Thm. 3.39	221.39	1.17	2.79	82.28	1.50

Table 8.18: Comparison of RHC results in a steady state situation

For our first situation, the constant setpoint tracking, we observe that all six adaptive receding horizon controllers show an improved performance compared to the standard receding horizon controller. Upon intialization, the first RHC iterate is compromised by the startup of the minimizer which is why we exclude this step from our considerations. Still, the first few iterations are computationally more expensive if we use the adaptation strategies. For this example, the cost for the first RHC step rise by about 250% if we use the a posteriori estimate and by about 200% if the adaptation is based on the a priori estimate. During the transient phase $[0, 1)$ of this example, this reduces to approximately

130% for all strategies, i.e. the shortening of the optimization horizon slowly starts to pay off. Once the solution is close to the steady state, the average computing times of the adaptive receding horizon controller are 60–70% lower than in the standard case.
Comparing the different implementations, we see that the implementations using the simple prolongation strategies of Algorithms 4.14 and 4.21 exhibit lower computing times once the trajectory is close to the steady state. Considering the underlying estimations, the a priori estimate of Theorem 3.39 shows better performance than the a posteriori estimate of Proposition 3.28 for the fixed point and monotone strategy. However, this does not hold for the simple strategy.
Hence, at least for this example, the simple strategy combined with the a posteriori estimate from Proposition 3.28 shows best performance. For a general example, the "best" adaptation strategy (among the presented ones) is not clear. This is due to two different and model inherent properties:

- If, for example, the minimal optimization horizon which guarantees at least local suboptimality degree $\overline{\alpha} \in (0,1)$ is large, then the a posteriori estimates becomes more efficient since only one additional optimal control problem needs to be solved. If, however, some knowledge on N_0 can be used, then the a priori estimate may be the better choice.

- Secondly, if large jumps within the minimal length of the optimization horizons occur, then the fixed point and monotone strategies start to pay off.

For the first issue, it is currently unknown how a good parameter N_0 can be characterized. The second issue, however, corresponds to the appropriate choice of the parameter σ. In particular, σ bounds the increase of the optimization horizon for both the fixed point and the monotone strategy. From numerical results, we know that this parameter is required since the prolongation formulae typically overestimate the required optimization horizon if the current suboptimality estimate α is much smaller than $\overline{\alpha}$.

For our second standard situation, the switching setpoint tracking problem, we choose $\sigma = 5$ since according to the outcome of the simple strategies no bigger jumps occur during the simulation. For a detailed comparison of the corresponding experiments, the data is contained in the following Table 8.19.

Adaptive RHC		Time in $[10^{-3}s]$			Horizon length		
Implementation	Estimate	max	min	∅	max	min	∅
Standard RHC	—	86.50	3.23	23.14	6	6	6
Simple Prolongation	Prop. 3.28	183.19	1.30	13.37	6	2	2.39
Simple Prolongation	Thm. 3.39	226.10	2.12	17.46	6	3	3.21
Fixed Point	Prop. 3.28	184.79	1.49	11.55	6	2	2.25
Fixed Point	Thm. 3.39	219.32	1.45	18.96	6	3	3.19
Monotone	Prop. 3.28	183.16	1.48	16.28	6	2	2.43
Monotone	Thm. 3.39	219.44	1.47	15.60	6	3	3.13

Table 8.19: Comparison of RHC results in a switching steady state situation

We obtain that, similar to the first experiment, the adaptive receding horizon controllers require much larger computing times on initialization of the problem. Again, we exclude the first RHC step from our considerations since it is compromised by the startup of the

minimizer. From the average horizon length, however, all six implementations operate with much smaller optimization horizons in general. The payoff can be seen from the average computing times required by these methods. For this setting, we again obtain that all six adaptive receding horizon controllers outrun the standard implementation. Conclusions about effectiveness of the suboptimality estimate and the implementation methods, however, are problematic.
Here, we obtain the best result if we consider the fixed point prolongation strategy of Algorithm 4.23 combined with our a posteriori estimate from Proposition 3.28. Yet, the worst outcome uses the same implementation but our a priori estimate from Theorem 3.39. This mismatch is most probably related to the contraction requirement (4.12) of the fixed point strategy and we experienced this problem for other examples and settings as well. The monotone strategy of Algorithm 4.29 uses more iteration steps than the fixed point iteration which is confirmed by other numerical experiments. Yet, using the a priori estimate, the average computing time of the simple implementation is improved for this setting.

In general, the ability of these more sophisticated strategies to outrun the simple method is directly proportional to the minimal horizon length N at those closed–loop points where enlarging jumps of the horizon length are required to sustain the local suboptimality degree $\overline{\alpha}$, and to the size of these jumps. Hence, these two parameter have to be used to define the bound σ. Moreover, two effects need to be waged to optimally set this bound: For one, an increase of the optimization horizon by one leaves the optimal solution relatively untouched, hence the computing time is fairly small. If the optimization horizon is increased by a large number (relative to the current length of the horizon), then the solution of the new optimal control problem is costy. For this reason, the choice of the prolongation method is problem dependent.

8.5.4 Computing the Closed–loop Suboptimality Degree

Using the computed values for both the constant setpoint and the switching setpoint tracking example from the last sections, we now want to draw conclusion on the closed–loop. To this end, we compute the closed–loop suboptimality degree α from the local data according to Theorem 4.6. Note that we can restrict ourselves to those time instances where $l(\cdot,\cdot) > \varepsilon$ holds, i.e. when the state trajectory is outside the practical area around the target.
The following Tables 8.20 and 8.21 show the corresponding values α_C for both situations and all previously discussed adaptive receding horizon controller implementations, see Sections 8.5.2 and 8.5.3.

Remark 8.35
Here, we set $N^\star = 6$ and recompute each open–loop solution for identical base points along the closed–loop trajectory. This allows us to consider local information as described in Theorem 4.6 and we do not have to compute C_l and C_α for all $x \in \mathbb{X}$.

Adaptive RHC		C_l		C_α		α_C
Implementation	Estimate	min	max	min	max	
All strategies	Prop. 3.28	0.97029	0.99981	0.99406	1.05840	0.50017
All strategies	Thm. 3.39	0.99710	1.04115	1.00043	1.05140	0.50360

Table 8.20: Values of C_l, C_α and α_C of Theorem 4.6 for the setpoint stabilization example

Remark 8.36
For the constant setpoint tracking situation, we combined the results for the different adaptation algorithms apart from the suboptimality estimation since we obtained identical results down to the state trajectory level.

Adaptive RHC		C_l		C_α		α_C
Implementation	Estimate	min	max	min	max	
Simple Prolongation	Prop. 3.28	0.99595	1.12821	1.00000	1.05063	0.44318
Simple Prolongation	Thm. 3.39	0.88244	1.14606	1.00102	1.83214	0.47516
Fixed Point	Prop. 3.28	1.10786	1.10786	1.00000	1.00000	0.45132
Fixed Point	Thm. 3.39	0.94800	1.53346	0.98164	1.64574	0.33668
Monotone	Prop. 3.28	0.96389	1.17606	1.00000	1.10885	0.42918
Monotone	Thm. 3.39	0.88354	2.03629	0.98564	1.83608	0.25354

Table 8.21: Values of C_l, C_α and α_C of Theorem 4.6 for the tracking example

Note that according to Theorem 4.6, α_C is given by combinations of $C_l^{(i)}$ and $C_\alpha^{(i)}$ for $i \in I$, while Tables 8.21 and 8.21 show minimal and maximal values of C_l and C_α.
For both situations, we observe that using the a priori estimate from Theorem 3.39 exhibits satisfactory closed–loop suboptimality estimates while results based on the a posteriori estimate from Proposition 3.28 show very good performance. Indeed, the minimal local suboptimality degree $\overline{\alpha} = 0.5$ is almost completely carried over to the closed–loop suboptimality degree. In either case, the presented adaptation strategies guarantee stability of the closed–loop, show a good local and closed–loop suboptimality degree and still offer lower computing times than the standard receding horizon controller implementation.

Concluding, in Chapters 5 and 6 we presented an implementation of a receding horizon controller described in the earlier Chapter 2. In Sections 8.1 and 8.2 we showed that this implementation is efficient and illustrated the interaction of various components of the method in Section 8.3 using a real–time example. In Section 8.4 we observed that our practical suboptimality estimates from Chapter 3 are applicable in the context of adaptation strategies for the optimization horizon. Moreover, we saw in Section 8.5 that these methods interact nicely with our presented adaptation strategies from Chapter 4. In total, we presented an efficient and stability guaranteeing receding horizon algorithm. Future work concerns many parts of this method. Probably the most important point is to improve the a priori estimate presented in Theorem 3.39 by a more detailed analysis of the parameter N_0 and to develop other efficiently computable suboptimality estimates. By this, we hope to reduce the required computing time significantly. Moreover, development and investigation of alternatives to prolongate or shorten the optimization horizon will be an issue. In particular, combinations of iterates may allow for further insight of the process under control. Upon implementation, the integration of the suboptimality estimate into the optimization routine, e.g. as a breaking criterion, will be a field of interest. Furthermore, extensions to cover partial differential equations are planned.

Appendix A

An Implementation Example

The following program code represents an implementation of the inverted pendulum on a cart problem, see Section 7.2, within *PCC2*.

```
#include "InvertedPendulum.h"

#include <odesol2/dopri853.h>
#include <iostream>
#include <cmath>

using namespace std;

#define PI 3.141592654
#define INF 1.0E19

InvertedPendulum::InvertedPendulum()
        : btmb::MPC2::Model ( new btmb::OdeSol2::DoPri853(),
                              new btmb::OdeSol2::DoPriConfig(),
                              4,
                              1,
                              1,
                              4 )
{
    setDoubleParameter ( 0, 0.007 );
    setDoubleParameter ( 1, 1.25 );
    setDoubleParameter ( 2, 9.81 );
    setDoubleParameter ( 3, 0.197 );

    getOdeConfig()->setTolerance ( 1E-10, 1E-10 );
}


InvertedPendulum::~InvertedPendulum()
{
    delete getOdeSolver();
    delete getOdeConfig();
}

void InvertedPendulum::dglFunction ( double t, double * x, double
    * u, double * dx )
```

```
{
    dx[0] = x[1];
    dx[1] = - params[2] * sin ( x[0] ) / params[1] - u[0] * cos (
        x[0] ) - params[0] * atan ( 1.0e3 * x[1] ) * x[1] * x[1] -
        ( 4.0 * x[1] / ( 1.0 + 4.0 * x[1] * x[1] ) + 2.0 * atan (
       2.0 * x[1] ) / PI ) * params[3];
    dx[2] = x[3];
    dx[3] = u[0];
}

double InvertedPendulum::objectiveFunction ( double t, double * x
   , double * u )
{
    double sinxpi = sin ( x[0] - PI );
    double cosy = cos ( x[1] );
    double temp = ( 1.0 - cos ( x[0] - PI ) ) * ( 1.0 + cosy *
       cosy );

    return 1.0e-1 * pow ( 3.51 * sinxpi * sinxpi + ( 4.82 *
       sinxpi + 2.31 * x[1] ) * x[1] + 2.0 * temp * temp + 1.0 * x
       [2] * x[2] + 1.0 * x[3] * x[3], 2.0 );
}

double InvertedPendulum::pointcostFunction ( int length, int
   horizon, double * t, double * x, double * u )
{
    if ( length < horizon )
    {
        return 0.0;
    }
    else
    {
        double * lastx = &x[ ( horizon - 1 ) * 4];

        double sinxpi = sin ( lastx[0] - PI );
        double cosy = cos ( lastx[1] );
        double temp = ( 1.0 - cos ( lastx[0] - PI ) ) * ( 1.0 +
           cosy * cosy );

        return pow ( 3.51 * sinxpi * sinxpi + ( 4.82 * sinxpi +
           2.31 * lastx[1] ) * lastx[1] + 2.0 * temp * temp + 1.0
           * pow ( lastx[2], 2.0 ) + 1.0 * lastx[3] * lastx[3],
           2.0 );
    }
}

void InvertedPendulum::getObjectiveWeight ( double & obj_weight,
   double & pointcost_weight )
{
    obj_weight = 1.0;
    pointcost_weight = 1.0;
}
```

```
void InvertedPendulum::restrictionFunction ( double t, double * x
    , double * u, double * fx )
{
}

void InvertedPendulum::getControlBounds ( double * lb, double *
    ub )
{
     lb[0] = -5.0;
     ub[0] =  5.0;
}

void InvertedPendulum::getModelBounds ( double * lb, double * ub
    )
{
     lb[0] = -INF;
     lb[1] = -INF;
     lb[2] = -5.0;
     lb[3] = -10.0;

     ub[0] = INF;
     ub[1] = INF;
     ub[2] = 5.0;
     ub[3] = 10.0;
}


void InvertedPendulum::getDefaultState ( double * x )
{
     x[0] = 0.0;
     x[1] = 0.0;
     x[2] = 0.0;
     x[3] = 0.0;
}

void InvertedPendulum::getDefaultControl ( double * u )
{
     u[0] = 0.0;
}

int InvertedPendulum::getShootingDataLength ( int horizon )
{
     return 2;
}

int InvertedPendulum::getMaxShootingDataLength ( int maxhorizon )
{
     return 2;
}

void InvertedPendulum::getShootingDataInfo ( int horizon, btmb::
```

```
    MPC2::STARTDATA * sdata )
{
    sdata[0].horizontindex = 13;
    sdata[0].varindex = 0;
    sdata[1].horizontindex = 15;
    sdata[1].varindex = 0;
}

void InvertedPendulum::eventBeforeMPC ( int horizon, double * t,
    double * x, double * sdatavalues )
{
    sdatavalues[0] = 3.1415;
    sdatavalues[1] = 3.1415;
}
```

Listing A.1: Header file of the Inverted Pendulum example using *PCC2*

```
#include "InvertedPendulum.h"

#include <odesol2/dopri853.h>
#include <iostream>
#include <cmath>

using namespace std;

#define PI 3.141592654
#define INF 1.0E19

InvertedPendulum::InvertedPendulum()
        : btmb::MPC2::Model ( new btmb::OdeSol2::DoPri853(),
                              new btmb::OdeSol2::DoPriConfig(),
                              4,
                              1,
                              1,
                              4 )
{
    setDoubleParameter ( 0, 0.007 );
    setDoubleParameter ( 1, 1.25 );
    setDoubleParameter ( 2, 9.81 );
    setDoubleParameter ( 3, 0.197 );

    getOdeConfig()->setTolerance ( 1E-10, 1E-10 );
}


InvertedPendulum::~InvertedPendulum()
{
    delete getOdeSolver();
    delete getOdeConfig();
}
```

```
void InvertedPendulum::dglFunction ( double t, double * x, double
    * u, double * dx )
{
    dx[0] = x[1];
    dx[1] = - params[2] * sin ( x[0] ) / params[1] - u[0] * cos (
        x[0] ) - params[0] * atan ( 1.0e3 * x[1] ) * x[1] * x[1] -
        ( 4.0 * x[1] / ( 1.0 + 4.0 * x[1] * x[1] ) + 2.0 * atan (
       2.0 * x[1] ) / PI ) * params[3];
    dx[2] = x[3];
    dx[3] = u[0];
}

double InvertedPendulum::objectiveFunction ( double t, double * x
   , double * u )
{
    double sinxpi = sin ( x[0] - PI );
    double cosy = cos ( x[1] );
    double temp = ( 1.0 - cos ( x[0] - PI ) ) * ( 1.0 + cosy *
       cosy );

    return 1.0e-1 * pow ( 3.51 * sinxpi * sinxpi + ( 4.82 *
       sinxpi + 2.31 * x[1] ) * x[1] + 2.0 * temp * temp + 1.0 * x
       [2] * x[2] + 1.0 * x[3] * x[3], 2.0 );
}

double InvertedPendulum::pointcostFunction ( int length, int
   horizon, double * t, double * x, double * u )
{
    if ( length < horizon )
    {
        return 0.0;
    }
    else
    {
        double * lastx = &x[ ( horizon - 1 ) * 4];

        double sinxpi = sin ( lastx[0] - PI );
        double cosy = cos ( lastx[1] );
        double temp = ( 1.0 - cos ( lastx[0] - PI ) ) * ( 1.0 +
           cosy * cosy );

        return pow ( 3.51 * sinxpi * sinxpi + ( 4.82 * sinxpi +
           2.31 * lastx[1] ) * lastx[1] + 2.0 * temp * temp + 1.0
           * pow ( lastx[2], 2.0 ) + 1.0 * lastx[3] * lastx[3],
           2.0 );
    }
}

void InvertedPendulum::getObjectiveWeight ( double & obj_weight,
   double & pointcost_weight )
{
    obj_weight = 1.0;
```

```
    pointcost_weight = 1.0;
}

void InvertedPendulum::restrictionFunction ( double t, double * x
   , double * u, double * fx )
{
}

void InvertedPendulum::getControlBounds ( double * lb, double *
   ub )
{
    lb[0] = -5.0;
    ub[0] =  5.0;
}

void InvertedPendulum::getModelBounds ( double * lb, double * ub
   )
{
    lb[0] = -INF;
    lb[1] = -INF;
    lb[2] = -5.0;
    lb[3] = -10.0;

    ub[0] = INF;
    ub[1] = INF;
    ub[2] = 5.0;
    ub[3] = 10.0;
}


void InvertedPendulum::getDefaultState ( double * x )
{
    x[0] = 0.0;
    x[1] = 0.0;
    x[2] = 0.0;
    x[3] = 0.0;
}

void InvertedPendulum::getDefaultControl ( double * u )
{
    u[0] = 0.0;
}

int InvertedPendulum::getShootingDataLength ( int horizon )
{
    return 2;
}

int InvertedPendulum::getMaxShootingDataLength ( int maxhorizon )
{
    return 2;
}
```

```
void InvertedPendulum::getShootingDataInfo ( int horizon, btmb::
   MPC2::STARTDATA * sdata )
{
    sdata[0].horizontindex = 13;
    sdata[0].varindex = 0;
    sdata[1].horizontindex = 15;
    sdata[1].varindex = 0;
}

void InvertedPendulum::eventBeforeMPC ( int horizon, double * t,
   double * x, double * sdatavalues )
{
    sdatavalues[0] = 3.1415;
    sdatavalues[1] = 3.1415;
}
```

Listing A.2: Implementation file of the Inverted Pendulum using *PCC2*

```
#include <iostream>
#include <cmath>
#include <iomanip>
#include <cstdio>
#include <unistd.h>
#include <cstdlib>

#include "InvertedPendulum.h"
#include <mpc2/mpc.h>
#include <mpc2/simpleodemanager.h>
#include <sqpf/sqpfortran.h>
#include <btmbutils/exception.h>
#include <btmbutils/rtclock.h>
#include <btmbutils/optimaldiff.h>
#include <btmbutils/savedata.h>

#define HORIZON 50
#define ITERATIONS 1000
#define H_NEW 0.1
#define INF 1.0E19

using namespace btmb::MPC2;
using namespace btmb::SqpFortran;
using namespace btmb::Utils;
using namespace std;

int main()
{
    OptimalDiff::calc();
    Exception::enableDebugMessage ( true );

//  --- Setting up MPC ---
    Model * object_model = new InvertedPendulum();
```

```
    IOdeManager * object_odemanager = new SimpleOdeManager();
    btmb::MinProg::MinProg * object_minimizer = new SqpFortran();
    InvertedPendulum * object_modelspecify = ( ( InvertedPendulum
        * ) object_model );

    btmb::MPC2::MPC * mpc_problem = new MPC ( INF );
    mpc_problem->reset ( object_odemanager, object_minimizer,
       object_model, HORIZON );

//  --- Setting up optimizer ---
    SqpFortran * object_minimizerSpecify = ( ( SqpFortran * )
       object_minimizer );
    object_minimizerSpecify->setAccuracy ( 1E-8 );
    object_minimizerSpecify->setMaxFun ( 20 );
    object_minimizerSpecify->setMaxIterations ( 1000 );
    object_minimizerSpecify->setLineSearchTol ( 0.1 );

//  --- Memory Allocation ---
    double * u, * next_u, * t, * x;
    next_u = ( double * ) malloc ( sizeof ( double ) *
       object_model->getDimensionControl() );
    x = ( double * ) malloc ( sizeof ( double ) * object_model->
       getDimension() );
    object_modelspecify->getDefaultState ( x );
    mpc_problem->allocateMemory ( t, u );
    mpc_problem->initCalc ( t, u );

//  --- Initial Values ---
    for ( int i = 0; i < HORIZON; i++ )
    {
        object_model->getDefaultControl ( & u[i * object_model->
           getDimensionControl() ] );
    }
    for ( int i = 0; i < HORIZON + 1; i++ )
    {
        t[i] = i * H_NEW;
    }

//  --- Setting up output ---
    SaveData * save = new SaveData ( object_model->getDimension()
       , object_model->getDimensionControl() );

//  --- MPC Iteration ---
    mpc_problem->resizeHorizon ( 17, H_NEW );
    int mstep = 1;
    for ( int j = 0; j < ITERATIONS; j++ )
    {
        RTClock timer;
        try
        {
            mpc_problem->calc ( x );
        }
```

```
        catch ( btmb::MinProg::sqpException e )
        {
            cout << e.what() << endl;
        }

        save->save2Files ( t, x, u );

        mpc_problem->shiftHorizon ( x, next_u, H_NEW, mstep );
    }

//  --- Free variables ---
    delete object_odemanager;
    delete object_minimizer;
    delete object_model;
    delete mpc_problem;
    free ( t );
    free ( u );

    return 0;
}
```

Listing A.3: Main function using the Inverted Pendulum example using *PCC2*

Note that the program requires the libraries

- libmpc2 (containing the RHC procedures, the discretization and odemanager),
- libodesol2 (providing ODE solvers),
- libbtmbutils (for basic functions like timing or saving data) and
- libsqpf (chosen minimization routine)

to be compiled appropriately such that the classes can be used.

Glossary

RHC

α	Degree of suboptimality
$\alpha(N)$	Degree of suboptimality for horizon N
γ	Auxilliary characteristic for degree of suboptimality
$\gamma(N)$	Auxilliary characteristic for degree of suboptimality for horizon N
$\mathcal{I}_u$	Index set for open–loop control in RHC optimization
$\mathcal{I}_x$	Index set for open–loop states in RHC optimization
$\mu_N(\cdot,\cdot)$	RHC closed–loop control law
$\mu_{N,m}(\cdot,\cdot)$	m–step RHC closed–loop control
$\mathbb{U}$	Control value space
$\mathbb{X}$	State space
$c(\cdot,\cdot)$	Vector of constraints
$c_k(\cdot)$	State constraint
$c_k(\cdot,\cdot)$	Mixed constraint
$F(\cdot)$	Terminal cost function
$f(\cdot,\cdot)$	Dynamic of the system
$J_\infty(\cdot,\cdot)$	Infinite horizon cost functional
$J_N(\cdot,\cdot)$	Finite horizon cost functional
$L(\cdot,\cdot)$	Continuous–time running cost function
$l(\cdot,\cdot)$	Discrete–time running cost function
T	Sampling time
$u(\cdot)$	Open–loop control law
$u(x_0,\cdot)$	Closed–loop control law for initial value x_0
$u^\star(\cdot)$	Optimal feedback control law
$u_N(x_0,\cdot)$	RHC open–loop control law
$u_T(\cdot)$	Sample and hold control law
$V(\cdot)$	Lyapunov function
$V_\infty(\cdot)$	Infinite horizon optimal value function
$V_N(\cdot)$	Finite horizon optimal value function
$V_T(\cdot)$	Sampling Lyapunov function
$x(\cdot)$	State trajectory
$x^\star$	Equilibrium or target values
x_0	Initial value $x_0 \in \mathbb{X}$
$x_T(\cdot,x_0,u)$	Continuous–time sampling solution with sampling time T, initial value x_0 and control function u
$x_u(\cdot)$	Trajectory for control sequence $u(\cdot)$
$x_u(\cdot,x_0)$	Trajectory for control sequence $u(\cdot)$ and initial value x_0

$x_u^{(a)}(\cdot, x_0)$	Approximated open–loop trajectory for control sequence $u(\cdot)$ and initial value x_0
$x_{\mu_N}(\cdot, x_0)$	RHC closed–loop solution with control function $\mu_N(\cdot,\cdot)$ and initial value x_0
$x_{\text{ref}}(\cdot)$	Reference trajectory
$x_{u_N}(\cdot, x_0)$	RHC open–loop trajectory

NLP

$\alpha^{(k)}$	Step length in search direction $d^{(k)}$
$\Delta^{(k)}$	Radius for trust–region algorithm
η	Decrease parameter for optimization routine
γ	Contraction parameter for trust–region
ι	Index mapping of dimension for a shooting node
λ	Langrange multiplier
$\mathcal{A}(x)$	Index set of active constraints at x
$\mathcal{E}$	Index set of equality constraints
$\mathcal{E}_g$	Set of discretized equality constraints
$\mathcal{I}$	Index set of inequality constraints
$\mathcal{I}_h$	Set of discretized inequality constraints
$\mathcal{X}$	Feasible set of optimization variables
$\nabla^2_{xx} L^p(\cdot,\cdot)$	Projected Hessian of the Lagrangian of NLP
$\Phi(\cdot,\cdot)$	Solution operator of a ODE solver
$\mathbb{T}_k$	Sampling grid
ε	Optimality threshold for optimization
$\subsetneq$	Index mapping of time step for a shooting node
$\widetilde{L}(\cdot,\cdot)$	Merit function
$A(x)$	Set of active constraints at x
$B^{(k)}$	BFGS approximation of the Hessian of the constraints
$d^{(k)}$	Search direction of optimization algorithm
$F(\cdot)$	Cost function
$G(\cdot)$	Set of equality constraints
$H(\cdot)$	Set of inequality constraints
$L(\cdot,\cdot)$	Lagrangian of the NLP
s_x	Vector of multiple shooting nodes
x	Optimization variable

Problems

ECQP	Equality constrained QP
QP	Quadratic Program
ECNLP	Equality constrained NLP
NLP	Nonlinear optimization problem
OCP_∞	Continuous–time infinite horizon optimal control problem
QP	Quadratic subproblem
RHC_N	RHC optimal control problem
$\text{RHC}_\text{N}^\text{B}$	RHC optimal control problem with Bolza type cost functional
SDOCP_∞	Sampled–data infinite horizon optimal control problem
SDOCP_N	Sampled–data finite horizon optimal control problem

SQP	Sequential quadratic program

System Theory

$\alpha(\cdot)$	Class $\mathcal{K}_\infty$ comparison function
$\beta(\cdot,\cdot)$	Class $\mathcal{KL}$ comparison function
$\mathcal{G}$	Set of all class $\mathcal{G}$ functions
$\mathcal{KL}$	Set of all class $\mathcal{KL}$ functions
$\mathcal{K}$	Set of all class $\mathcal{K}$ functions
$\mathcal{K}_\infty$	Set of all class $\mathcal{K}_\infty$ functions
$\mathcal{L}$	Set of all class $\mathcal{L}$ functions
$\mathbb{R}$	Set of real numbers
$\mathbb{T}$	Time set
$\mathbb{U}$	Control value space
$\mathbb{U}^I$	Set of all control functions $u : I \to \mathbb{U}$
$\mathcal{U}$	Set of all control functions $u : \mathbb{T} \to \mathbb{U}$
$\mathbb{X}$	State space

Bibliography

[1] *The history of technology - irrigation*, 1994.

[2] G.B. Airy, *On the regulator of the clock-work for effecting uniform movement of equatoreals*, R. Astr. Soc. **11** (1840), 249–268 (English).

[3] ———, *On a method of regulating the clock-work for equatoreals*, R. Astr. Soc. **20** (1850), 115–120 (English).

[4] M. Alamir, *Stabilization of nonlinear systems using receding-horizon control schemes*, Lecture Notes in Control and Information Sciences, vol. 339, Springer-Verlag London Ltd., London, 2006 (English).

[5] F. Allgöwer, T.A. Badgwell, J.S. Qin, J.B. Rawlings, and S.J. Wright, *Nonlinear predictive control and moving horizon estimation — An introductory review*, Advances in Control, Highlights of ECC'99, Springer Verlag, Berlin, 1999, pp. 391–449 (English).

[6] F. Allgöwer, R. Findeisen, and C. Ebenbauer, *Nonlinear model predictive control*, Encyclopedia for Life Support Systems (EOLSS),, 2003 (English).

[7] B.D.O. Anderson, *Controller design: moving from theory to practice*, IEEE Control Systems Magazine **13** (1993), no. 4, 16–25 (English).

[8] M. Arcak and D. Nešić, *A framework for nonlinear sampled-data observer design via approximate discrete-time models and emulation*, Automatica J. IFAC **40** (2004), no. 11, 1931–1938 (English).

[9] S. Arimoto, *Control theory of non-linear mechanical systems: A passivity-based and circuit-theoretic approach*, Clarendon Press, Oxford and New York, 1996 (English).

[10] M. Arioli and L. Baldini, *A backward error analysis of a null space algorithm in sparse quadratic programming*, SIAM J. Matrix Anal. Appl. **23** (2001), no. 2, 425–442 (electronic) (English).

[11] Z. Artstein, *Stabilization with relaxed controls*, Nonlinear Anal. **7** (1983), no. 11, 1163–1173 (English).

[12] J. Aseltine, A. Mancini, and C. Sarture, *A survey of adaptive control systems*, IRE Trans. Automat. Contr. **6** (1958), no. 1, 102–108 (English).

[13] K.J. Aström, U. Borisson, L. Ljung, and B. Wittenmark, *Theory and applications of selftuning regulators*, Automatica **13** (1977), 457–476 (English).

[14] O. Axelsson, *Solution of linear systems of equations: iterative methods*, Sparse matrix techniques (Adv. Course, Technical Univ. Denmark, Copenhagen, 1976), Springer, Berlin, 1977, pp. 1–51. Lecture Notes in Math., Vol. 572 (English).

[15] T.A. Badgwell and S.J. Qin, *An overview of industrial model predictive control technology*, in F. Allgöwer and A. Zheng (eds), Nonlinear model predictive control (2000), 232–256 (English).

[16] ———, *A survey of industrial model predictive control technology*, Control Engineering Practice **11** (2003), 733–764 (English).

[17] R. Bellman, *Dynamic programming*, Princeton University Press, 1957 (English).

[18] R. Bellman and S.E. Dreyfus, *Applied dynamic programming*, Princeton University Press, Princeton, N.J., 1962 (English).

[19] D.P. Bertsekas, *Constrained optimization and Lagrange multiplier methods*, Computer Science and Applied Mathematics, Academic Press Inc. [Harcourt Brace Jovanovich Publishers], New York, 1982 (English).

[20] N.P. Bhatia and G.P. Szegő, *Stability theory of dynamical systems*, Classics in Mathematics, Springer-Verlag, Berlin, 2002 (English), Reprint of the 1970 original.

[21] L.T. Biegler, J. Nocedal, and C. Schmid, *A reduced Hessian method for large-scale constrained optimization*, SIAM J. Optim. **5** (1995), no. 2, 314–347 (English).

[22] R.R. Bitmead, M. Gevers, and V. Wertz, *Adaptive optimal control: The thinking man's GPC.*, International Series in Systems and Control Engineering, Prentice-Hall, New York, 1990 (English).

[23] G. Bleisteiner and W. Mangoldt, *Handbuch der Regelungstechnik*, Springer Verlag, Berlin, 1961 (German).

[24] A.M. Bloch, *Nonholonomic mechanics and control*, Interdisciplinary Applied Mathematics, vol. 24, Springer-Verlag, New York, 2003 (English).

[25] H.W. Bode, *A general theory of electric wave filters*, Bell Syst. Tech. J. **14** (1933), 211–214 (English).

[26] ———, *A general theory of electric wave filters*, J. Math. Phys. **13** (1934), 275–362 (English).

[27] ———, *Network analysis and feedback amplifier design*, Van Nostrand, New York, 1945 (English).

[28] P.T. Boggs and J.W. Tolle, *Sequential quadratic programming*, Acta numerica, 1995, Acta Numer., Cambridge Univ. Press, Cambridge, 1995, pp. 1–51 (English).

[29] P.T. Boggs, J.W. Tolle, and P. Wang, *On the local convergence of quasi-Newton methods for constrained optimization*, SIAM J. Control Optim. **20** (1982), no. 2, 161–171 (English).

[30] R.W. Brockett, *The status of stability theory for deterministic systems*, IEEE Internat. Convention Record **1966** (1966), no. pt. 6, 125–142 (English).

[31] ———, *Lie algebras and lie groups in control theory.*, Geom. Methods Syst. Theory, Proc. NATO advanced Study Inst., London, 1973, pp. 43–82.

[32] ———, *New issues in the mathematics of control*, Mathematics unlimited—2001 and beyond, Springer, Berlin, 2001, pp. 189–219 (English).

[33] C.G. Broyden, *The convergence of a class of double-rank minimization algorithms. II. The new algorithm*, J. Inst. Math. Appl. **6** (1970), 222–231 (English).

[34] J.R. Bunch and B.N. Parlett, *Direct methods for solving symmetric indefinite systems of linear equations*, SIAM J. Numer. Anal. **8** (1971), 639–655 (English).

[35] R.H. Byrd, J. Nocedal, and Y.X. Yuan, *Global convergence of a class of quasi-Newton methods on convex problems*, SIAM J. Numer. Anal. **24** (1987), no. 5, 1171–1190 (English).

[36] R.H. Byrd, R.B. Schnabel, and G.A. Shultz, *A trust region algorithm for nonlinearly constrained optimization*, SIAM J. Numer. Anal. **24** (1987), no. 5, 1152–1170 (English).

[37] C.C. Chen and L. Shaw, *On receding horizon feedback control*, Automatica—J. IFAC **18** (1982), no. 3, 349–352 (English).

[38] H. Chen, *Stability and Robustness Considerations in Nonlinear Predictive Control*, PhD–thesis in Engineering, University of Stuttgart, 1997.

[39] H. Chen and F. Allgöwer, *Nonlinear model predictive control schemes with guaranteed stability*, Nonlinear Model Based Process Control, Kluwer Academic Publishers, Dodrecht, 1999, pp. 465–494 (English).

[40] T. Chen and B. Francis, *Optimal sampled-data control systems*, Communications and Control Engineering Series, Springer-Verlag London Ltd., London, 1996 (English).

[41] F.H. Clarke, Y.S. Ledyaev, E.D. Sontag, and A.I. Subbotin, *Asymptotic controllability implies feedback stabilization*, IEEE Trans. Automat. Control **42** (1997), no. 10, 1394–1407 (English).

[42] F.H. Clarke, Y.S. Ledyaev, R.J. Stern, and P.R. Wolenski, *Nonsmooth analysis and control theory*, Graduate Texts in Mathematics, vol. 178, Springer-Verlag, New York, 1998 (English).

[43] F. Colonius and W. Kliemann, *The dynamics of control*, Systems & Control: Foundations & Applications, Birkhäuser Boston Inc., Boston, MA, 2000 (English).

[44] A.R. Conn, N.I.M. Gould, and P.L. Toint, *Testing a class of methods for solving minimization problems with simple bounds on the variables*, Math. Comp. **50** (1988), no. 182, 399–430 (English).

[45] ———, *Trust-region methods*, MPS/SIAM Series on Optimization, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000 (English).

[46] R.L. Cosgriff, *Nonlinear control systems*, McGraw Hill, New York, 1958 (English).

[47] Y.-H. Dai and K. Schittkowski, *A sequential quadratic programming algorithm with non-monotone line search*, Pac. J. Optim. **4** (2008), no. 2, 335–351 (English).

[48] W.C. Davidon, *Variable metric method for minimization*, Research and Development Report ANL-5990 (Rev.), Argonne National Laboratory, Argonne, Illinois (1959) (English).

[49] G. De Nicolao, L. Magni, and R. Scattolini, *Stability and robustness of nonlinear receding horizon control*, Nonlinear Predictive Control, Birkhäuser, 2000, pp. 3–23 (English).

[50] D. DeHaan and M. Guay, *A real-time framework for model-predictive control of continuous-time nonlinear systems*, IEEE Trans. Automat. Control **52** (2007), no. 11, 2047–2057 (English).

[51] P. Deuflhard and A. Hohmann, *Numerische Mathematik. I*, second ed., de Gruyter Textbook, Walter de Gruyter & Co., Berlin, 2003 (German).

[52] F. Di Palma and L. Magni, *On optimality of nonlinear model predictive control*, Systems Control Lett. **56** (2007), no. 1, 58–61 (English).

[53] M. Diehl, *Real-Time Optimization for Large Scale Nonlinear Processes*, PhD–Thesis in Mathematics, University of Heidelberg, 2001.

[54] R. Dittmar and B.-M. Pfeiffer, *Modellbasierte prädiktive Regelung in der industriellen Praxis*, at – Automatisierungstechnik **54** (2006), 590–601 (German).

[55] D. Driankov, H. Hellendoorn, and M. Reinfrank, *An introduction to fuzzy control*, Springer-Verlag, Berlin, 2001 (English).

[56] O. Exler and K. Schittkowski, *MISQP: A Fortran implementation of a trust region SQP algorithm for mixed–integer nonlinear programming – user's guide, version 1.1*, 2005.

[57] H.O. Fattorini, *On complete controllability of linear systems*, J. Differential Equations **3** (1967), 391–402 (English).

[58] H.J. Ferreau, P. Ortner, P. Langthaler, L. Re, and M. Diehl, *Predictive control of a real-world Diesel engine using an extended online active set strategy*, Annual Reviews in Control **31** (2007), no. 2, 293 – 301 (English).

[59] R. Findeisen, *Nonlinear Model Predictive Control: A Sampled–Data Feedback Perspective*, Fortschr.–Ber. VDI Reihe 8 Nr. 1087, VDI Verlag, Düsseldorf, 2005 (English).

[60] R. Findeisen and F. Allgöwer, *The quasi-infinite horizon approach to nonlinear model predictive control*, 2003.

[61] R. Findeisen, Allgöwer F., and L.T. Biegler (eds.), *Assessment and future directions of nonlinear model predictive control*, Lecture Notes in Control and Information Sciences, no. 358, Berlin: Springer, 2007 (English).

[62] R. Findeisen, L. Imsland, F. Allgöwer, and B. Foss, *State and output feedback nonlinear model predictive control: An overview*, Europ. J. Contr. **9** (2003), no. 2-3, 190–207 (English).

[63] R. Fletcher, *A new approach to variable metric algorithms*, Computer J. **13** (1970), 317–322 (English).

[64] ———, *Practical methods of optimization*, second ed., Wiley-Interscience [John Wiley & Sons], New York, 2001 (English).

[65] R. Fletcher, S. Leyffer, and P.L. Toint, *On the global convergence of a filter-SQP algorithm*, SIAM J. Optim. **13** (2002), no. 1, 44–59 (electronic) (English).

[66] R. Fletcher and M. J. D. Powell, *A rapidly convergent descent method for minimization*, Comput. J. **6** (1963/1964), 163–168 (English).

[67] O. Föllinger, *Laplace- und Fourier-Transformation*, Elitera, Berlin, 1977 (German).

[68] F.A.C.C. Fontes, *A general framework to design stabilizing nonlinear model predictive controllers*, Systems Control Lett. **42** (2001), no. 2, 127–143 (English).

[69] ———, *Discontinuous feedbacks, discontinuous optimal controls, and continuous-time model predictive control*, Internat. J. Robust Nonlinear Control **13** (2003), no. 3-4, 191–209 (English).

[70] G.F. Franklin, J.D. Powell, and M.L. Workman, *Digital control of dynamic systems. 2nd ed.*, Amsterdam etc: Addison-Wesley Publishing Company, 1990 (English).

[71] H.I. Freedman, *Deterministic mathematical models in population ecology*, Monographs and Textbooks in Pure and Applied Mathematics, vol. 57, Marcel Dekker Inc., New York, 1980 (English).

[72] R.A. Freeman and P.V. Kokotovic, *Robust nonlinear control design*, Systems & Control: Foundations & Applications, Birkhäuser Boston Inc., Boston, MA, 1996 (English).

[73] J.B. Froisy, *Model Predictive Control: Past, Present and Future*, ISA Transactions **33** (1994), 235–243 (English).

[74] A. T. Fuller, *The early development of control theory. II*, Trans. ASME Ser. G. J. Dynamic Systems, Measurement and Control **98** (1976), no. 3, 224–235 (English).

[75] A.T. Fuller, *The early development of control theory*, Trans. ASME Ser. G. J. Dynamic Systems, Measurement and Control **98** (1976), no. 2, 109–118 (English).

[76] C.E. García, D.M. Prett, and M. Morari, *Model predictive control: Theory and practice - a survey*, Automatica **25** (1989), no. 3, 335–348 (English).

[77] M. Gerdts, *Optimal Control of Ordinary Differential Equations and Differential-Algebraic Equations*, Ph.D. thesis, University of Bayreuth, Germany, 2006, Habitilation–Thesis in Mathematics, pp. iv+235.

[78] P.E. Gill, W. Murray, and M.A. Saunders, *SNOPT: An SQP algorithm for large-scale constrained optimization*, SIAM Rev. **47** (2005), no. 1, 99–131 (electronic) (English).

[79] P.E. Gill, W. Murray, M.A. Saunders, and M.H. Wright, *A Schur-complement method for sparse quadratic programming*, Reliable numerical computation, Oxford Sci. Publ., Oxford Univ. Press, New York, 1990, pp. 113–138 (English).

[80] P.E. Gill, W. Murray, and M.H. Wright, *Practical Optimization*, Academic Press Inc., London, 1981 (English).

[81] D. Goldfarb, *A family of variable metric methods derived by variational means*, Math. Computation **24** (1970), 23–26 (English).

[82] N.I.M. Gould, *On practical conditions for the existence and uniqueness of solutions to the general equality quadratic programming problem*, Math. Programming **32** (1985), no. 1, 90–99 (English).

[83] ———, *On the accurate determination of search directions for simple differentiable penalty functions*, IMA J. Numer. Anal. **6** (1986), no. 3, 357–372 (English).

[84] ———, *An algorithm for large-scale quadratic programming*, IMA J. Numer. Anal. **11** (1991), no. 3, 299–324 (English).

[85] N.I.M. Gould, D. Orban, and P.L. Toint, *Numerical methods for large-scale nonlinear optimization*, Acta Numer. **14** (2005), 299–361 (English).

[86] N.I.M. Gould and P.L. Toint, *Numerical methods for large-scale non-convex quadratic programming*, Trends in industrial and applied mathematics (Amritsar, 2001), Appl. Optim., vol. 72, Kluwer Acad. Publ., Dordrecht, 2002, pp. 149–179 (English).

[87] G. Grimm, M.J. Messina, S.E. Tuna, and A.R. Teel, *Model predictive control: for want of a local control Lyapunov function, all is not lost*, IEEE Trans. Automat. Control **50** (2005), no. 5, 546–558 (English).

[88] A. Grötsch and S. Trenz, *Modellprädiktive Regelung für nichtlineare evolutionäre partielle Differentialgleichungen*, Diploma Thesis in Mathematics, University of Bayreuth, Germany, 2008.

[89] L. Grüne, *Asymptotic behavior of dynamical and control systems under perturbation and discretization*, Lecture Notes in Mathematics, vol. 1783, Springer-Verlag, Berlin, 2002 (English).

[90] ———, *Computing stability and performance bounds for unconstrained NMPC schemes*, Proceedings of the 46th IEEE Conference on Decision and Control, New Orleans, Louisiana, 2007, pp. 1263–1268 (English).

[91] ———, *Optimization based stabilization of nonlinear control systems*, Large-Scale Scientific Computations (LSSC07), Lecture Notes in Computer Science, vol. 4818, Springer, 2008, pp. 52–65 (English).

[92] ———, *Analysis and design of unconstrained nonlinear MPC schemes for finite and infinite dimensional systems*, SIAM Journal on Control and Optimization **48** (2009), 1206–1228 (English).

[93] L. Grüne, D. Nešić, and J. Pannek, *Model predictive sampled–data redesign for nonlinear systems*, Proceedings of the 44th IEEE Conference on Decision and Control and European Control Conference ECC2005 (Sevilla, Spain) (New York), IEEE, 2005, pp. 36–41 (English).

[94] ———, *Redesign techniques for nonlinear sampled-data control*, Oberwolfach Reports 2 (2005), no. 1, 584–586 (English).

[95] ———, *Model predictive control for nonlinear sampled-data systems*, Assessment and future directions of nonlinear model predictive control, Lecture Notes in Control and Inform. Sci., vol. 358, Springer, Berlin, 2007, pp. 105–113 (English).

[96] L. Grüne, D. Nešić, J. Pannek, and K. Worthmann, *Redesign techniques for nonlinear sampled-data systems*, at – Automatisierungstechnik **1** (2008), no. 1, 38–47 (English).

[97] L. Grüne and J. Pannek, *Trajectory based suboptimality estimates for receding horizon controllers*, Proceedings of MTNS 2008, Blacksburg, Virginia, 2008, pp. CD–ROM, paper125.pdf (English).

[98] ———, *Practical NMPC suboptimality estimates along trajectories*, Sys. & Contr. Lett. **58** (2009), no. 3, 161–168 (English).

[99] L. Grüne, J. Pannek, M. Seehafer, and K. Worthmann, *Analysis of unconstrained nonlinear MPC schemes with varying control horizon*, submitted (2009) (English).

[100] L. Grüne, J. Pannek, and K. Worthmann, *A networked unconstrained nonlinear MPC scheme*, Proceedings of the European Control Conference ECC2009 (Budapest, Hungary), 2009, pp. 371–376 (English).

[101] ———, *A prediction based control scheme for networked systems with delays and packet dropouts*, Proceedings of the 48th IEEE Conference on Decision and Control, Shanghai, China (accepted), 2009 (English).

[102] L. Grüne and A. Rantzer, *On the infinite horizon performance of receding horizon controllers*, IEEE Trans. Automat. Control **53** (2008), no. 9, 2100–2111.

[103] É. Gyurkovics and A.M. Elaiw, *Stabilization of sampled-data nonlinear systems by receding horizon control via discrete-time approximations*, Automatica J. IFAC **40** (2004), no. 12, 2017–2028 (2005) (English).

[104] E. Gyurkovics and A.M. Elaiw, *Conditions for MPC based stabilization of sampled-data nonlinear systems via discrete-time approximations*, Assessment and future directions of nonlinear model predictive control, Lecture Notes in Control and Inform. Sci., vol. 358, Springer, Berlin, 2007, pp. 35–48 (English).

[105] W.M. Haddad and V. Chellaboina, *Nonlinear dynamical systems and control*, Princeton University Press, Princeton, NJ, 2008 (English).

[106] W. Hahn, *Theorie und Anwendung der direkten Methode von Ljapunov*, Ergebnisse der Mathematik und ihrer Grenzgebiete. Neue Folge, Heft 22. Berlin-Göttingen-Heidelberg: Springer-Verlag, 1959 (German).

[107] ———, *Stability of motion*, Die Grundlehren der mathematischen Wissenschaften in Einzeldarstellungen. 138. Berlin-Heidelberg-New York: Springer-Verlag, 1967 (English).

[108] E. Hairer, S.P. Nørsett, and G. Wanner, *Solving ordinary differential equations. I*, second ed., Springer Series in Computational Mathematics, vol. 8, Springer-Verlag, Berlin, 1993 (English).

[109] E. Hairer and G. Wanner, *Solving ordinary differential equations. II*, second ed., Springer Series in Computational Mathematics, vol. 14, Springer-Verlag, Berlin, 1996 (English).

[110] J.K. Hale and S.M. Verduyn Lunel, *Introduction to functional-differential equations*, Applied Mathematical Sciences, vol. 99, Springer-Verlag, New York, 1993 (English).

[111] S.P. Han, *Superlinearly convergent variable metric algorithms for general nonlinear programming problems*, Math. Programming **11** (1976/77), no. 3, 263–282 (English).

[112] ———, *A globally convergent method for nonlinear programming*, J. Optimization Theory Appl. **22** (1977), no. 3, 297–309 (English).

[113] S. Haykin, *Kalman Filtering and Neural Networks (Adaptive and Learning Systems for Signal Processing, Communications and Control)*, Wiley & Sons, 2001 (English).

[114] M.R. Hestenes, *Optimization theory*, Wiley-Interscience [John Wiley & Sons], New York, 1975 (English).

[115] A. Isidori, *Nonlinear control systems*, third ed., Communications and Control Engineering Series, Springer-Verlag, Berlin, 1995 (English).

[116] D.H. Jacobson, *Extensions of linear-quadratic control, optimization and matrix theory*, Mathematics in Science and Engineering, vol. 133, Academic Press, London, 1977 (English).

[117] A. Jadbabaie, J. Yu, and J. Hauser, *Unconstrained receding-horizon control of nonlinear systems*, IEEE Trans. Automat. Control **46** (2001), no. 5, 776–783 (English).

[118] Z.-P. Jiang and Y. Wang, *A converse Lyapunov theorem for discrete-time systems with disturbances*, Systems Control Lett. **45** (2002), no. 1, 49–58 (English).

[119] V. Jurdjevic and H. Sussmann, *Control systems on Lie groups*, J. Differential Equations **12** (1972), 313–329 (English).

[120] J. Kahlert and H. Frank, *Fuzzy logic and fuzzy control*, Vieweg, Braunschweig, 1994 (English).

[121] N. Kaiser-Hugel, *Modellprädiktive Regelung bei Optimalsteuerungsproblemen mit partiellen Differentialgleichungen: Parabolische Optimalsteuerungsprobleme mit verteilter Steuerung und hyperbolische Optimalsteuerungsprobleme*, Diploma Thesis in Mathematics, University of Bayreuth, Germany, 2006.

[122] R.E. Kalman, *On the general theory of control systems*, Automatic and remote control, vol. 1, Oldenbourg–Verlag, München, 1961, pp. 481–492 (English).

[123] R.E. Kalman and R.S. Bucy, *New results in linear filtering and prediction theory*, Trans. ASME Ser. D. J. Basic Engrg. **83** (1961), 95–108 (English).

[124] A. Kandel and G. Langholz, *Fuzzy control systems*, CRC Press, Boca Raton, FL, 1994 (English).

[125] P. Katz, *Digital control using microprocessors*, Prentice-Hall, Englewood Cliffs, NJ, 1981 (English).

[126] S.S. Keerthi and E.G. Gilbert, *Optimal infinite-horizon feedback laws for a general class of constrained discrete-time systems: stability and moving-horizon approximations*, J. Optim. Theory Appl. **57** (1988), no. 2, 265–293 (English).

[127] C.M. Kellett, *Advances in converse and control lyapunov functions*, PhD–Thesis in Mathematics, University of California, Santa Barbara, 2000.

[128] H.F. Khalfan, R.H. Byrd, and R.B. Schnabel, *A theoretical and experimental study of the symmetric rank-one update*, SIAM J. Optim. **3** (1993), no. 1, 1–24 (English).

[129] H.K. Khalil, *Nonlinear systems. 3rd edition*, Upper Saddle River, NJ: Prentice Hall, 2002 (English).

[130] P. Knabner and L. Angermann, *Numerical methods for partial differential equations: An application-oriented introduction (Numerik partieller Differentialgleichungen: Eine anwendungsorientierte Einführung)*, Springer, Berlin, 2000 (German).

[131] A.A. Krasovsky, *A new solution to the problem of a control system analytical design*, Automatica—J. IFAC **7** (1971), 45–50 (English).

[132] M. Krstic, I. Kanellakopoulos, and P.V. Kokotovic, *Nonlinear and adaptive control design*, Wiley, New York, 1995 (English).

[133] K. Küpfmüller, *Über die Dynamik der selbsttätigen Verstärkungsregler*, Elektr. Nachrichtentechnik **5** (1928), 459–467 (German).

[134] D.S. Laila, D. Nešić, and A.R. Teel, *Open and closed loop dissipation inequalities under sampling and controller emulation*, Europ. J. Contr. **8** (2002), no. 2, 109–125 (English).

[135] J.P. LaSalle, *Stability theory for ordinary differential equations*, J. Differential Equations **4** (1968), 57–65 (English).

[136] ———, *Stability theory for difference equations*, Studies in ordinary differential equations, Math. Assoc. of America, Washington, D.C., 1977, pp. 1–31. Stud. in Math., Vol. 14 (English).

[137] R.S. Ledley, *Digital computational methods in symbolic logic, with examples in biochemistry*, Proc. Natl. Acad. Sci. USA **41** (1955), 498–511 (English).

[138] ———, *Programming and utilizing digital computers*, McGraw-Hill Book Company, Inc, New York, NY, 1962 (English).

[139] E. B. Lee and L. Markus, *Foundations of optimal control theory*, John Wiley & Sons Inc., New York, 1967 (English).

[140] J. Lee and B. Cooley, *Recent advances in model predictive control and other related areas*, in J. Kantor, C. Garcia and B. Carnahan (eds), Fifth International Conference on Chemical Process Control **V** (1996), 201–216 (English).

[141] A. Leonhard, *Die selbsttätige Regelung*, Springer Verlag, Berlin, 1940 (German).

[142] Y. Lin, E.D. Sontag, and Y. Wang, *A smooth converse Lyapunov theorem for robust stability*, SIAM J. Control Optim. **34** (1996), no. 1, 124–160 (English).

[143] Y.D. Lin and E.D. Sontag, *A universal formula for stabilization with bounded controls*, Systems Control Lett. **16** (1991), no. 6, 393–397 (English).

[144] B. Lincoln and A. Rantzer, *Relaxing dynamic programming*, IEEE Trans. Automat. Control **51** (2006), no. 8, 1249–1260 (English).

[145] J.-L. Lions, *Optimal control of systems governed by partial differential equations*, Translated from the French by S. K. Mitter. Die Grundlehren der mathematischen Wissenschaften, Band 170, Springer-Verlag, New York, 1971 (English).

[146] ———, *Exact controllability, stabilization and perturbations for distributed systems*, SIAM Rev. **30** (1988), no. 1, 1–68 (English).

[147] J. Lunze, *Control theory 1. System-theoretic foundations. Analysis and design of one-loop control. (Regelungstechnik 1. Systemtheoretische Grundlagen, Analyse und Entwurf einschleifiger Regelungen.) 6th revised ed.*, Springer, Berlin, 2007 (German).

[148] ———, *Control theory 2. Multivariable systems, digital control. (Regelungstechnik 2. Mehrgrößensysteme, digitale Regelung.) 5th revised ed.*, Springer, Berlin, 2007 (German).

[149] A.M. Lyapunov, *The general problem of the stability of motion*, Internat. J. Control **55** (1992), no. 3, 521–790 (English), Translated by A. T. Fuller from Édouard Davaux's French translation (1907) of the 1892 Russian original, With an editorial (historical introduction) by Fuller, a biography of Lyapunov by V. I. Smirnov, and the bibliography of Lyapunov's works collected by J. F. Barrett, Lyapunov centenary issue.

[150] L. Magni and R. Sepulchre, *Stability margins of nonlinear receding-horizon control via inverse optimality*, Systems Control Lett. **32** (1997), no. 4, 241–245 (English).

[151] O.L. Mangasarian, *Nonlinear programming*, McGraw-Hill Book Co., New York, 1969 (English).

[152] N. Maratos, *Exact penalty function algorithms for finite dimensional and control optimization problems*, PhD–Thesis in Mathematics, University of London, 1978.

[153] L. Markus, *A brief history of control*, Differential equations, dynamical systems, and control science, Lecture Notes in Pure and Appl. Math., vol. 152, Dekker, New York, 1994, pp. xxv–xl (English).

[154] H.J. Marquez, *Nonlinear control systems: Analysis and design*, Chichester: Wiley, 2003 (English).

[155] J.L. Massera, *On Liapounoff's conditions of stability*, Ann. of Math. (2) **50** (1949), 705–721 (English).

[156] H. Maurer and J. Zowe, *First and second order necessary and sufficient optimality conditions for infinite-dimensional programming problems*, Math. Programming **16** (1979), no. 1, 98–110 (English).

[157] J.C. Maxwell, *On Governors*, Proceedings of the Royal Society, vol. 16, 1868, pp. 270–283 (English).

[158] D.Q. Mayne and H. Michalska, *Receding horizon control of nonlinear systems*, Proceedings of the 28th IEEE Conference on Decision and Control, Vol. 1–3 (Tampa, FL, 1989) (New York), IEEE, 1989, pp. 107–108 (English).

[159] ———, *Receding horizon control of nonlinear systems*, IEEE Trans. Automat. Control **35** (1990), no. 7, 814–824 (English).

[160] D.Q. Mayne, J.B. Rawlings, C.V. Rao, and P.O.M. Scokaert, *Constrained model predictive control: Stability and optimality*, Automatica **36** (2000), no. 6, 789–814 (English).

[161] O. Mayr, *The origins of feedback control*, Cambridge, Mass.-London: The M.I.T. Press, 1970 (English).

[162] E.S. Meadows, M.A. Henson, J.W. Eaton, and J.B. Rawlings, *Receding horizon control and discontinuous state feedback stabilization*, Internat. J. Control **62** (1995), no. 5, 1217–1229 (English).

[163] J.M. Mendel and K.S. Fu (eds.), *Adaptive learning and pattern recognition systems: Theory and applications*, New York etc.: Academic Press, New York, 1970 (English).

[164] H. Michalska and D.Q. Mayne, *Robust receding horizon control of constrained nonlinear systems*, IEEE Trans. Automat. Control **38** (1993), no. 11, 1623–1633 (English).

[165] E. Mishkin and L. Braun, Jr., *Adaptive control systems*, Brooklyn Polytechnic Institute Series. McGraw-Hill Electrical and Electronic Engineering Series, McGraw-Hill, New York, 1961 (English).

[166] H. Monteleone, M.C. Yeung and R. Smith, *A review of Ancient Roman water supply exploring techniques of pressure reduction*, Water Science & Technology: Water Supply **7** (2007), no. 1, 113–120 (English).

[167] M. Morari and J.H. Lee, *Model predictive control: past, present and future*, Computers & Chemical Engineering **23** (1999), 667–682 (English).

[168] D. Nešić and A.R. Teel, *Input-output stability properties of networked control systems*, IEEE Trans. Automat. Control **49** (2004), no. 10, 1650–1667 (English).

[169] ———, *Input-to-state stability of networked control systems*, Automatica J. IFAC **40** (2004), no. 12, 2121–2128 (English).

[170] D. Nešić, A.R. Teel, and P.V. Kokotović, *Sufficient conditions for stabilization of sampled-data nonlinear systems via discrete-time approximations*, Systems Control Lett. **38** (1999), no. 4-5, 259–270 (English).

[171] D. Nešić, A.R. Teel, and E.D. Sontag, *Formulas relating $\mathcal{KL}$ stability estimates of discrete-time and sampled-data nonlinear systems*, Systems Control Lett. **38** (1999), no. 1, 49–60 (English).

[172] K. Neumann and M. Morlock, *Operations research*, Carl Hanser Verlag, Munich, 1993 (German).

[173] D. Nešić and L. Grüne, *A receding horizon control approach to sampled-data implementation of continuous-time controllers*, Sys. & Contr. Lett. **55** (2006), 660–672 (English).

[174] H. Nijmeijer and A. van der Schaft, *Nonlinear dynamical control systems*, Springer-Verlag, New York, 1990 (English).

[175] J. Nocedal and S.J. Wright, *Numerical optimization*, second ed., Springer Series in Operations Research and Financial Engineering, Springer, New York, 2006 (English).

[176] H. Nyquist, *Regeneration theory*, Bell Syst. Tech. J. **11** (1932), 126–147 (English).

[177] R. Oldenbourg and H. Sartorius, *Dynamik selbsttätige Regelungen*, Oldenbourg–Verlag, München, 1944 (German).

[178] R. Ortega, A. Loria, P.J. Nicklasson, and H. Sira-Ramirez, *Passivity-based control of euler-lagrange systems: Mechanical, electrical and electromechanical applications*, Springer, Berlin, 1998 (English).

[179] J. Pannek, *Modellprädiktive Regelung nichtlinearer sampled–data Systeme*, Diploma Thesis in Mathematics, University of Bayreuth, Germany, 2005.

[180] J.-B. Pomet and L. Praly, *Adaptive nonlinear regulation: estimation from the Lyapunov equation*, IEEE Trans. Automat. Control **37** (1992), no. 6, 729–740 (English).

[181] L.S. Pontryagin, V.G. Boltyanskii, R.V. Gamkrelidze, and E.F. Mishchenko, *The mathematical theory of optimal processes*, Translated by D. E. Brown, A Pergamon Press Book, The Macmillan Co., New York, 1964 (English).

[182] M.J.D. Powell, *Some global convergence properties of a variable metric algorithm for minimization without exact line searches*, Nonlinear programming (Proc. Sympos., New York, 1975), Amer. Math. Soc., Providence, R. I., 1976, pp. 53–72. SIAM–AMS Proc., Vol. IX (English).

[183] ———, *Algorithms for nonlinear constraints that use Lagrangian functions*, Math. Programming **14** (1978), no. 2, 224–248 (English).

[184] ———, *A fast algorithm for nonlinearly constrained optimization calculations*, Numerical analysis (Proc. 7th Biennial Conf., Univ. Dundee, Dundee, 1977), Springer, Berlin, 1978, pp. 144–157. Lecture Notes in Math., Vol. 630 (English).

[185] ———, *Variable metric methods for constrained optimization*, Mathematical programming: the state of the art (Bonn, 1982), Springer, Berlin, 1983, pp. 288–311 (English).

[186] ———, *Convergence properties of algorithms for nonlinear optimization*, SIAM Rev. **28** (1986), no. 4, 487–500 (English).

[187] L. Praly, G. Bastin, J.-B. Pomet, and Z.-P. Jiang, *Adaptive stabilization of nonlinear systems*, Foundations of adaptive control (Urbana, IL, 1990), Lecture Notes in Control and Inform. Sci., vol. 160, Springer, Berlin, 1991, pp. 347–433 (English).

[188] D.M. Prett and C.E. Garcia, *Fundamental process control*, Butterworths series in chemical engineering, Boston, 1988 (English).

[189] A.I. Propoĭ, *Application of linear programming methods for the synthesis of automatic sampled-data systems*, Avtomat. i Telemeh. **24** (1963), 912–920 (Russian).

[190] M.H. Protter and H.F. Weinberger, *Maximum principles in differential equations*, Prentice-Hall Inc., Englewood Cliffs, N.J., 1967 (English).

[191] P. Pucci and J. Serrin, *The maximum principle*, Progress in Nonlinear Differential Equations and their Applications, 73, Birkhäuser Verlag, Basel, 2007 (English).

[192] A. Rantzer, *A Separation Principle for Distributed Control*, Proceedings of the 43rd IEEE Conference on Decision and Control, vol. 6, 2006, pp. 3609–3613 (English).

[193] J.B. Rawlings, *Tutorial overview of model predictive control*, IEEE Contr. Syst. Magazine **20** (2000), no. 3, 38–52 (English).

[194] J. Richalet, A. Rault, J.L. Testud, and J. Papon, *Model predictive heuristic control: Applications to industrial processes*, Automatica **14** (1978), 413–428 (English).

[195] S.M. Robinson, *Perturbed Kuhn-Tucker points and rates of convergence for a class of nonlinear-programming algorithms*, Math. Programming **7** (1974), 1–16 (English).

[196] R.T. Rockafellar, *The multiplier method of Hestenes and Powell applied to convex programming*, J. Optimization Theory Appl. **12** (1973), 555–562 (English).

[197] R. Rojas, *Theorie der neuronalen Netze: Eine systematische Einführung, Reprint*, Springer-Verlag, Berlin, 1996 (German).

[198] N. Rouche, P. Habets, and M. Laloy, *Stability theory by Liapunov's direct method*, Springer-Verlag, New York, 1977 (English).

[199] E.J. Routh, *A Treatise on the Stability of a Given State of Motion, Particularly Steady Motion: Particularly Steady Motion*, Macmillan and co., 1877 (English).

[200] K. Schittkowski, *The nonlinear programming method of Wilson, Han, and Powell with an augmented Lagrangian type line search function. I. Convergence analysis*, Numer. Math. **38** (1981/82), no. 1, 83–114 (English).

[201] ———, *On the convergence of a sequential quadratic programming method with an augmented Lagrangian line search function*, Math. Operationsforsch. Statist. Ser. Optim. **14** (1983), no. 2, 197–216 (English).

[202] ———, *NLPQL: A FORTRAN subroutine solving constrained nonlinear programming problems*, Ann. Oper. Res. **5** (1986), no. 1-4, 485–500 (English).

[203] ———, *Solving nonlinear programming problems with very many constraints*, Optimization **25** (1992), no. 2-3, 179–196 (English).

[204] ———, *Numerical data fitting in dynamical systems*, Applied Optimization, vol. 77, Kluwer Academic Publishers, Dordrecht, 2002 (English).

[205] J.S. Shamma and D. Xiong, *Linear nonquadratic optimal control*, IEEE Trans. Automat. Control **42** (1997), no. 6, 875–879 (English).

[206] D.F. Shanno, *Conditioning of quasi-Newton methods for function minimization*, Math. Comp. **24** (1970), 647–656 (English).

[207] E.D. Sontag, *A Lyapunov-like characterization of asymptotic controllability*, SIAM J. Control Optim. **21** (1983), no. 3, 462–471 (English).

[208] ———, *Smooth stabilization implies coprime factorization*, IEEE Trans. Automat. Control **34** (1989), no. 4, 435–443 (English).

[209] ———, *Feedback stabilization of nonlinear systems*, Robust control of linear systems and nonlinear control (Amsterdam, 1989), Progr. Systems Control Theory, vol. 4, Birkhäuser Boston, Boston, MA, 1990, pp. 61–81 (English).

[210] ———, *Mathematical control theory*, second ed., Texts in Applied Mathematics, vol. 6, Springer-Verlag, New York, 1998 (English).

[211] A.B. Stodola, *Über die Regelung von Turbinen*, Schweizer Bauzeitung **22** (1893), 27–30 (German).

[212] ———, *Über die Regelung von Turbinen*, Schweizer Bauzeitung **23** (1894), 17–18.

[213] J. Stoer and R. Bulirsch, *Introduction to numerical analysis*, third ed., Texts in Applied Mathematics, vol. 12, Springer-Verlag, New York, 2002 (English), Translated from the German by R. Bartels, W. Gautschi and C. Witzgall.

[214] A.M. Stuart and A.R. Humphries, *Dynamical systems and numerical analysis*, Cambridge Monographs on Applied and Computational Mathematics, vol. 2, Cambridge University Press, Cambridge, 1996 (English).

[215] Y.A. Thomas, *Linear quadratic optimal estimation and control with receding horizon*, Electronics Letters **11** (1975), no. 1, 19–21 (English).

[216] Y.A. Thomas and A. Barraud, *Commande optimale à horizon fuyant*, Rev. Française Automat. Informat. Recherche Opérationelle Sér. Jaune **8** (1974), no. J–1, 126–140 (French).

[217] M. Tolle, *Regelung der Kraftmaschinen, Berechnung und Konstruktion der Schwungräder, des Massenausgleiches und der Kraftmaschinenregler in elementarer Behandlung*, Springer Verlag, Berlin, 1905 (German).

[218] F. Tröltzsch, *Optimal control of partial differential equations. Theory, procedures, and applications. (Optimale Steuerung partieller Differentialgleichungen. Theorie, Verfahren und Anwendungen)*, Wiesbaden: Vieweg, 2005 (German).

[219] J.G. Truxal, *Automatic feedback control system synthesis*, McGraw Hill, New York, 1955 (English).

[220] ———, *The concept of adaptive control*, Adaptive control systems, McGraw-Hill, New York, 1961, pp. 1–19 (English).

[221] P. Turchin, *Complex population dynamics: a theoretical/empirical synthesis*, Monographs in Population Biology, vol. 35, Princeton University Press, Princeton, NJ, 2003 (English).

[222] H. Unbehauen, *Control theory 1: Classical methods for the analysis and synthesis of linear continuous control systems, fuzzy control systems. (Regelungstechnik 1: Klassische Verfahren zur Analyse und Synthese linearer kontinuierlicher Regelsysteme, Fuzzy-Regelsysteme.) 10., vollst. überarb. Aufl.*, Vieweg, Braunschweig, 2000 (German).

[223] ———, *Control theory 2: State control, digital and nonlinear control systems. (Regelungstechnik 2: Zustandsregelungen, digitale und nichtlineare Regelsysteme.) 8., vollst. überarb. u. erweit. Aufl.*, Vieweg, Braunschweig, 2000 (German).

[224] ———, *Control theory 3: Identification, adaptation, optimization. (Regelungstechnik 3: Identifikation, Adaption, Optimierung.) 6., verbess. Aufl.*, Vieweg, Braunschweig, 2000 (German).

[225] A. Vardi, *A trust region algorithm for equality constrained minimization: convergence properties and implementation*, SIAM J. Numer. Anal. **22** (1985), no. 3, 575–591 (English).

[226] S. Vogel, *Modellprädiktive Regelung bei Optimalsteuerungsproblemen mit partiellen Differentialgleichungen: Parabolische Optimalsteuerungsprobleme mit Randsteuerungen und hyperbolische Optimalsteuerungsprobleme*, Diploma Thesis in Mathematics, University of Bayreuth, Germany, 2006.

[227] A. Wächter and L.T. Biegler, *On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming*, Math. Program. **106** (2006), no. 1, Ser. A, 25–57 (English).

[228] W. Walter, *Ordinary differential equations*, Springer-Lehrbuch. Berlin: Springer, 2000 (German).

[229] N. Wiener, *The theory of statistical extrapolation*, Bol. Soc. Mat. Mexicana **2** (1945), 37–42 (English).

[230] ———, *Cybernetics, or Control and Communication in the Animal and the Machine*, Actualités Sci. Ind., no. 1053, Hermann et Cie., Paris, 1948 (French).

[231] ———, *Extrapolation, Interpolation, and Smoothing of Stationary Time Series. With Engineering Applications*, The Technology Press of the Massachusetts Institute of Technology, Cambridge, Mass, 1949 (English).

[232] B. Wittenmark, *Stochastic adaptive control methods: a survey*, Internat. J. Control **21** (1975), 705–730 (English).

[233] T. Yoshizawa, *Stability theory by Liapunov's second method*, Publications of the Mathematical Society of Japan, No. 9, The Mathematical Society of Japan, Tokyo, 1966 (English).

[234] Y. Yuan, *On the superlinear convergence of a trust region algorithm for nonsmooth optimization*, Math. Programming **31** (1985), no. 3, 269–285 (English).

[235] L.A. Zadeh, *Fuzzy sets*, Information and Control **8** (1965), 338–353 (English).

[236] H. Zhang and D. Liu, *Fuzzy modeling and fuzzy control*, Control Engineering, Birkhäuser Boston Inc., Boston, MA, 2006 (English).

Ehrenwörtliche Erklärung

Hiermit erkläre ich, dass ich die von mir vorgelegte Dissertation zum Thema

"Receding Horizon Control: A Suboptimality–based Approach"

selbstständig angefertigt habe und alle benutzten Quellen und Hilfsmittel vollständig angegeben habe.

Die Arbeit wurde in gleicher oder ähnlicher Form in keinem anderen Prüfungsverfahren zur Erlangung eines akademischen Grades eingereicht. Ich habe weder diese noch eine gleichartige Promotionsprüfung an einer anderen Hochschule endgültig nicht bestanden.

Bayreuth, den 19. November 2009

.....
Jürgen Pannek