Diss. ETH No. 17662

# Efficient Software Tools for Control and Analysis of Hybrid Systems

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY (ETH) ZURICH

for the degree of
Doctor of Sciences

presented by

Michal Kvasnica
ing., Slovak University of Technology in Bratislava, Slovakia
born 20.6.1977
citizen of Slovakia

accepted on the recommendation of

Prof. Dr. Manfred Morari, examiner
Prof. Ing. Vladimír Havlena, CSc., co-examiner

2008

# Acknowledgement

My time at ETH has been a unique experience. I am heavily indebted to my supervisor Professor Manfred Morari, who gave me the opportunity to join his laboratory and to work with great people. His continuous guidance and support during my studies helped me to stay focused on my journey towards this thesis. For this, I am utmost grateful.

During my stay I had the possibility to interact with many bright persons. First, I would like to acknowledge Pascal Grieder. His enthusiasm and critical attitude helped me to develop many of the ideas presented in this thesis. A special thanks goes to my roommates Valentina Sartori and Mato Baotic for creating a pleasant working atmosphere. I will never forget Mato's patience while discussing various issues. I'd also like to acknowledge Frank J. Christophersen for his invaluable feedback on just about anything I worked on.

Everyday life in the lab has been pleasant thanks to a prompt staff (Alice, Martine, Esther, Danielle), who were always helpful in solving administrative issues. I would also like to acknowledge all the people who shared the working environment with me. I shared a lot of my research with Miroslav Baric, Gianni Beccuti, Raphael Cagienard, Frank J. Christophersen, Tobias Geyer, Ioannis Fotiou, Johan Löfberg, and George Papafotiou. A special thanks goes to my "czech" friends Vaclav and Ondrej for their support during my whole stay in Zurich. Last, but not least, I would like to express my thanks to all my friends at IfA: Thomas Besselmann, Antonello Caruso, Andreas Kuhn, Marc Lawrence, Urban Mäder, Kristian Nolde, Frauke Oldewurtel, Helfried Peyrl, Philipp Rostalski, Akin Sahin, Eleonora Zanderigo, and Melanie Zeilinger.

However, even the most perfect working environment is worthless if you cannot retreat to a fulfilled private life. This need is even more amplified by the distance to the people you miss the most. Therefore I am deeply indebted to my family who have always been very supportive on my journey towards this thesis. Thank you.

# Abstract

This thesis deals with the topic of control and analysis of constrained dynamical systems.

Specifically, we consider the class of hybrid dynamical systems, i.e. systems which combine continuous dynamics with discrete logic. Such systems can efficiently describe the dynamical behavior of systems with on/off switches, gear shifts, or can be used to approximate nonlinearities by utilizing the concept of multiple linearizations.

It is well known that the task of deriving stabilizing controllers for dynamical systems subject to constraints on states and inputs can be attacked by utilizing the concept of *Receding Horizon Control* (RHC). In RHC, the sequence of manipulated variables is obtained by optimizing a given performance function subject to specified constraints. Subsequently, only the first input of that sequence is applied to the system. At the next time step, the state is measured again and the procedure is repeated. However, the computational complexity involved in solving each optimization problem significantly limits the minimal admissible sampling rate at which RHC can be applied on-line.

This problem has been alleviated to some degree by the recent introduction of *multi-parametric programming* to control theory. In this approach the given RHC optimization problem is solved off-line for all admissible initial conditions which satisfy system constraints. By solving the problem in a parametric fashion, the solution can be shown to take the form of a look-up table, which describes a piecewise affine state feedback law defined over a polyhedral partition. The on-line implementation of such feedback laws then reduces to a simple set-membership set, which can be performed very efficiently on-line, thus allowing to apply the concept of RHC to processes with fast dynamics. The main drawback of the multi-parametric technique, however, is the growing complexity of the look-up table as the problem size increases. One of the aims of this thesis is therefore to mitigate this problem.

Specifically, various schemes to speed up the calculation of the parametric solutions to RHC problems are presented in this thesis. A combination of reachability-based methods along with efficient polytope reduction techniques yields new computation algorithms which are substantially faster than other known schemes. Moreover, new algorithms are given which serve to speed up the task of finding a correct entry in the look-up table on-line.

Large part of this thesis is devoted to a description of the Multi-Parametric Toolbox (MPT), which is a novel software tool for modeling, control, and analysis of constrained dynamical systems. The main strong point of MPT is that it simplifies and automates many tasks a control engineer has to go through when designing and validating optimal control laws based on the RHC principle. The toolbox offers a broad spectrum of algorithms compiled in a user friendly and accessible format starting from different performance objectives (linear, quadratic, minimum-time) to the handling of systems with persistent additive and polytopic uncertainties. Users can add custom constraints, such as polytopic, contraction or collision avoidance constraints, or create custom objective functions. Resulting optimal control laws can either be embedded into target applications in the form of the C code, or deployed to control platforms using the Real Time Workshop.

The MPT toolbox contains all of the algorithms presented in this thesis as well as a wide range of additional algorithms and tools developed by the academic community.

# Zusammenfassung

Die vorliegende Arbeit beschäftigt sich mit der Analyse und Regelung dynamischer Systeme mit Beschränkungen.

Die untersuchte Systemklasse der hybriden Systeme zeichnet sich durch die Kombination kontinuierlicher Dynamik mit diskreter Logik aus. Diese Klasse eignet sich sowohl für eine effiziente Beschreibung schaltender, dynamischer System als auch zur Approximation nichtlinearer Systeme mit Hilfe einer stückweisen Linearisierung.

Die Berechnung stabilisierender Regelgesetze für solche dynamischen Systeme unter zusätzlichen Stell- und Zustandsbeschränkungen wird häufig mittels Optimierungsverfahren durchgeführt, wobei das Konzept *Receding Horizon Control* (RHC) zur Anwendung kommt. Hierbei wird, durch Prädiktion der Zustände und unter Berücksichtigung von Nebenbedingungen, ein Gütefunktional minimiert, um eine optimale Stellgrössensequenz zu generieren. Von dieser Sequenz wird der erste Eingangsvektor auf das System gegeben. Im nächsten Zeitschritt initiiert die Messung des neuen Systemzustands eine neuerliche Optimierung. Die Komplexität dieser einzelnen Optimierungsprobleme limitiert massgeblich die maximal erreichbare Abtastrate der RHC.

Durch die Einführung der *multi-parametrischen Programmierung* in der Regelungstechnik konnte diese Grenze deutlich verschoben werden. Bei diesem neuartigen Ansatz wird das Optimierungsproblem der RHC offline, d.h. für alle möglichen Zustände des Systems, gelöst. Das Ergebnis ist eine parametrische Darstellung des optimalen Eingangs und des Gütefunktionswertes als Funktion dieser Anfangszustände. Es handelt sich dabei um eine stückweis-affine Funktion, deren Partitionierung durch lineare Ungleichungen beschrieben wird. Durch die effiziente Darstellung des Regelgesetzes in Form einer Look-up Tabelle reduziert sich die online Auswertung auf das Auffinden der aktiven Region des gemessenen Zustandes und das Auswerten eines affinen Regelungsgesetzes. Dies erlaubt die Anwendung der Modelprädiktiven RHC für Systeme, die eine hohe Abtastrate benötigen. Ein wesentlicher Nachteil dieser Technik ist die stark steigende Komplexität der Lösung bei wachsender Problemgrösse.

Ein Ziel dieser Arbeit ist es, dieses Problem zu entschärfen.

Es werden verschiedene Verfahren zur Beschleunigung der Lösungsberechnung für das RHC Problem vorgestellt. Dabei kommen eine Kombination aus Erreichbarkeitsmethoden und effizienten Verfahren zur Polytopreduktion zur Anwendung. Dies führt auf einen neuen, deutlich verbesserten Algorithmus, der eine schnellere Berechnung der Regelgesetze erlaubt. Darüber hinaus werden neue Algorithmen für das Auffinden der aktiven Region und damit das Auswerten der Look-up Tabelle präsentiert.

Ein grosser Teil der Arbeit befasst sich mit der Beschreibung der Multi-Parametrischen Toolbox (MPT), einem neuen Programm zur Modellierung, Regelung und Analyse dynamischer Systeme mit Stell- und Zustandsbeschränkungen. Die Stärke dieser Software ist eine deutlich vereinfachte und automatisierte Prozedur zum Entwurf optimaler Regelungen und dessen Validierung. Die Toolbox bietet ein breites Spektrum von benutzerfreundlichen Algorithmen angefangen bei der Unterstützung für verschiedene Gütefunktionale (linear, quadratisch, zeit-minimal, benutzerdefiniert), über die Berücksichtigung additiver und polytopischer Störungen bis hin zu verschiedenartigen Beschränkungen (polytopisch, kontrahierend oder kollisionsvermeidend).

Das berechnete, optimale Regelgesetz kann automatisch entweder in C-Code exportiert und auf die Zielplattform heruntergeladen werden, oder mittels Real-Time Workshop auf eine Regelungsplattform portiert werden.

Die MPT Toolbox enthält alle in dieser Arbeit vorgestellten Algorithmen, sowie eine Vielzahl weiterer Algorithmen und Werkzeuge, die weltweit an Universitäten entwickelt wurden.

# Contents

# NOTATION

## Logic Operators and Functions

| | |
|---|---|
| $A \Rightarrow B$ | A implies B, i.e. if A *true* then B *true* |
| $A \Leftrightarrow B$ | A implies B and B implies A, i.e. A *true* if and only if B *true* |

## Sets

| | |
|---|---|
| $\mathbb{R}$ $(\mathbb{R}_+)$ | Set of (non-negative) real numbers |
| $\mathbb{N}$ | Set of non-negative integers |
| $\mathbb{R}^n$ | Set of real vectors with $n$ elements |
| $\mathbb{R}^{n\times m}$ | Set of real matrices with $n$ rows and $m$ columns |

## Algebraic Operators

| | |
|---|---|
| $A^T$ | Transpose of matrix $A$ |
| $A^{-1}$ | Inverse of matrix $A$ |
| $\det(A)$ | Determinant of matrix $A$ |
| $A(\succeq) \succ 0$ | $A$ positive (semi)definite matrix, $x^T Ax(\geq) > 0,\ \forall x \neq 0$ |
| $A(\preceq) \prec 0$ | $A$ negative (semi)definite matrix, $x^T Ax(\leq) < 0,\ \forall x \neq 0$. |
| $A_{(i)}$ | $i$-th row of matrix $A$ |
| $x_{(i)}$ | $i$-th element of the vector $x$ |
| $\lvert x \rvert$ | Element wise absolute value |
| $\lvert\lvert x \rvert\rvert$ | Any vector norm of $x$ |
| $\lvert\lvert x \rvert\rvert_2$ | Euclidian norm of vector $x$ |
| $\lvert\lvert x \rvert\rvert_1$ | Sum of absolute elements of vector $x \in \mathbb{R}^n$, $\lvert\lvert x \rvert\rvert_1 := \sum_{i=1}^{n} \lvert x_{(i)} \rvert$ |
| $\lvert\lvert x \rvert\rvert_\infty$ | Largest absolute value of the vector $x \in \mathbb{R}^n$, $\lvert\lvert x \rvert\rvert_\infty := \max_{i\in\{1,\dots,n\}} \lvert x_{(i)} \rvert$ |
| $\lvert\lvert x \rvert\rvert_p$ | $p$-norm of a vector $x \in \mathbb{R}^n$, $\lvert\lvert x \rvert\rvert_p := \sqrt[p]{\sum_{i=1}^{n} \lvert x_{(i)} \rvert^p}$ |

## Set Operators and Functions

| | |
|---|---|
| $\emptyset$ | The empty set |
| $\mathcal{P} \cap \mathcal{Q}$ | Set intersection $\mathcal{P} \cap \mathcal{Q} = \{x \mid x \in \mathcal{P} \text{ and } x \in \mathcal{Q}\}$ |
| $\mathcal{P} \cup \mathcal{Q}$ | Set union $\mathcal{P} \cup \mathcal{Q} = \{x \mid x \in \mathcal{P} \text{ or } x \in \mathcal{Q}\}$ |
| $\bigcup_{r\in\{1,\dots,R\}} \mathcal{P}_r$ | Union of $R$ sets $\mathcal{P}_r$, i.e. $\bigcup_{r\in\{1,\dots,R\}} \mathcal{P}_r = \{x \mid x \in \mathcal{P}_0 \text{ or } \dots \text{ or } x \in \mathcal{P}_R\}$ |
| $\mathcal{P}^c$ | Complement of the set $\mathcal{P}$, $\mathcal{P}^c = \{x \mid x \notin \mathcal{P}\}$ |
| $\mathcal{P} \setminus \mathcal{Q}$ | Set difference $\mathcal{P} \setminus \mathcal{Q} = \{x \mid x \in \mathcal{P} \text{ and } x \notin \mathcal{Q}\}$ |
| $\mathcal{P} \subseteq \mathcal{Q}$ | The set $\mathcal{P}$ is a subset of $\mathcal{Q}$, $x \in \mathcal{P} \Rightarrow x \in \mathcal{Q}$ |
| $\mathcal{P} \subset \mathcal{Q}$ | The set $\mathcal{P}$ is a strict subset of $\mathcal{Q}$, $x \in \mathcal{P} \Rightarrow x \in \mathcal{Q}$ and $\exists x \in (\mathcal{Q} \setminus \mathcal{P})$ |
| $\mathcal{P} \supseteq \mathcal{Q}$ | The set $\mathcal{P}$ is a superset of $\mathcal{Q}$ |
| $\mathcal{P} \supset \mathcal{Q}$ | The set $\mathcal{P}$ is a strict superset of $\mathcal{Q}$ |
| $\mathcal{P} \ominus \mathcal{Q}$ | Pontryagin difference $\mathcal{P} \ominus \mathcal{Q} = \{x \mid x + q \in \mathcal{P},\ \forall q \in \mathcal{Q}\}$ |
| $\mathcal{P} \oplus \mathcal{Q}$ | Minkowski sum $\mathcal{P} \oplus \mathcal{Q} = \{x + q \mid x \in \mathcal{P},\ q \in \mathcal{Q}\}$ |
| $\partial\mathcal{P}$ | The boundary of $\mathcal{P}$ |
| $\text{int}(\mathcal{P})$ | The interior of $\mathcal{P}$, i.e. $\text{int}(\mathcal{P}) = \mathcal{P} \setminus \partial\mathcal{P}$ |
| $\lvert\mathcal{I}\rvert$ | The cardinality of the discrete set $\mathcal{I}$ |
| $\mathcal{B}^{(d)}$ | The projection of the set $\mathcal{B}$ onto the $d$-th dimension |

## Dynamical Systems

| | |
|---|---|
| $x(k)$ | Measurement of state $x$ at time $k$ |
| $x_k$ | Predicted value of state $x$ at time $k$, given a measurement $x(0)$ |
| $x^+$ | Successor of vector $x$, i.e. if $x = x(k)$ then $x^+ = x(k+1)$ |
| $\mathcal{F}_\infty$ | Minimal robust positive invariant set |
| $\mathcal{O}_\infty$ | Maximal robust positive invariant set |
| $\mathcal{C}_\infty$ | Maximal robust control invariant set |
| $\mathcal{K}_\infty$ | Maximal robust stabilizable set |
| $\mathcal{O}_\infty^{\text{LQR}}$ | Maximal positive invariant set $\mathcal{O}_\infty$ for LTI systems subject to the Riccati LQR controller |

## Others

| | |
|---|---|
| $\mathbf{I}$ | Identity matrix |
| $\mathbf{1}$ | Vector of ones, $\mathbf{1} = [1\ 1\ \dots\ 1]^T$ |
| $\mathbf{0}$ | Vector of zeros, $\mathbf{0} = [0\ 0\ \dots\ 0]^T$ |

## Acronyms

| | |
|---|---|
| ARE | Algebraic Riccati Equation |
| CFTOC | Constrained Finite Time Optimal Control |
| CITOC | Constrained Infinite Time Optimal Control |
| DP | Dynamic Program(ming) |
| LMI | Linear Matrix Inequality |
| LP | Linear Program(ming) |
| LQR | Linear Quadratic Regulator |
| LTI | Linear Time Invariant |
| MILP | Mixed Integer Linear Program |
| MIQP | Mixed Integer Quadratic Program |
| MLD | Mixed Logical Dynamical |
| MPC | Model Predictive Control |
| mp-LP | multi-parametric Linear Program |
| mp-QP | multi-parametric Quadratic Program |
| PWA | Piecewise Affine |
| PWP | Piecewise Polynomial |
| PWQ | Piecewise Quadratic |
| QP | Quadratic Program(ming) |
| RHC | Receding Horizon Control |
| SDP | Semi Definite Program(ming) |

# 1

# INTRODUCTION

## Outline

The focus of this thesis is on Receding Horizon Control (RHC) and Model Predictive Control (MPC) of discrete-time linear time invariant (LTI) and piecewise-affine (PWA) systems. PWA systems represent a powerful modelling tool to capture nonlinear and hybrid behavior of dynamical systems and have therefore received great interest in academia and industry. Therefore we present in this thesis a novel software tool which addresses the problems of modeling of hybrid systems, control synthesis, analysis and deployment of control laws to target platforms.

It is well known, that optimal state feedback controllers for the class of linear and hybrid systems can be computed by applying multi-parametric programming techniques. The resulting controller then takes the form of a feedback law which is affine over polyhedral sets, such that the optimal input becomes a piecewise affine function of the current state. The necessary on-line effort thus reduces to identifying which polyhedral set contains the current state and evaluating the associated affine feedback law. The advantage of this scheme is that no time consuming on-line optimization is necessary and the control input can be computed with low hardware cost and small computation time. However, there is a drawback: in the worst case, the number of control laws grows exponentially with the size of the control problem and may quickly reach a prohibitive number of elements.

In this thesis, these complexity issues are investigated and methods for reducing complexity are presented. Specifically, we attack the two main factors which incur complexity concerns: (1) the computation of the control laws and (2) the process of on-line implementation of said controllers. In this thesis we present new approaches to addressing these two issues for PWA systems.

The thesis is subdivided into four main parts, whereby each part is written to be self-contained. Hence, certain key theorems and definitions are stated more than once throughout the thesis.

In the first part of the thesis, the necessary background from the field of optimal control and computational geometry is summarized. We also present there a novel way of speeding up the calculation of parametric solutions to mathematical programs.

Part II of this thesis deals with controller construction for PWA systems. It shows how to obtain stabilizing feedback laws for PWA systems along with the associated terminal penalties. Based on these results, it is then shown how to formulate control problems which yield feedback controllers of very low complexity. Finally, a search algorithm based on bounding boxes that speeds up the on-line application of PWA controllers is illustrated. The results in this part are based on [SLG+04, GKBM04, GKBM05, CKJM07]

In Part III, the Multi-Parametric Toolbox (MPT) is presented. The MPT toolbox for MATLAB contains all the algorithms presented in this thesis as well as a wide range of additional algorithms and tools developed by the international academic community. This part of the thesis will introduce the reader to MPT, describe the software framework and provide examples. The content in this part is based on [KGB04].

Finally, several case studies are presented in Part IV which illustrate the potential and capabilities of the Multi-Parametric Toolbox. Specifically, we show how MPT was used to design an optimal infusion policy of intravenous Morphine and Ketamine during the anaesthesia process. Secondly, control design for mechanical systems with backlash is presented. The content of this part is based on [SZKM05] and [RBBM06].

## Contributions

The main contribution of this thesis may be looked at from two perspectives. From the theoretical point of view we propose new ways of dealing with optimal control of hybrid systems with constraints. Specifically, we show that stability of the closed-loop system can be enforced if a suitable terminal set constrain together with an appropriate terminal penalty are used in the MPC-based control design.

These issues are reviewed in Chapter 7 where we provide a simple algorithm to calculate the stabilizing piecewise linear feedback laws along with a Lyapunov-type terminal penalty matrix for the class of PWA systems. We show that the search for these two entities can be formulated as a positive semidefinite program which can be solved efficiently using state-of-the-art optimization techniques. The results of this chapter are exploited in Chapter 8 where we show how the concept of minimum-time control [GKBM05] may be used to derive feedback controllers for the class of PWA systems. The idea is based on steering the system states into an appropriately chosen terminal set in the least possible number of steps. Once the state is contained in the terminal set, a stabilizing feedback law drives the states towards the origin. We show that if such a procedure is used to construct the controller, usually the control law will be of lower complexity compared to other optimization-based strategies. Equally important is the fact that the construction of the controller is shown to be fast. The resulting controller guarantees closed-loop stability and constraint satisfaction.

We further extend this framework in Section 8.3 to show that if one drops the requirements of exponential stability when constructing the control laws and only deals with constraint satisfaction, very simple control laws can be obtained. The stability then needs to be checked a-posteriori by searching for a suitable Lyapunov function.

In Chapter 9 we then address the problem of on-line region identification, which is the procedure one needs to perform in order to apply the resulting PWA optimal control law on-line. We show that exploiting certain geometric structure a so-called *interval search tree* [CKJM07] can be constructed with very low effort. The tree serves to speed up the region identification step by eliminating elements which are of no interest at a particular stage.

The second main contribution presented in Part III [KGBM03a, KGBC06] is the description of a software tool – the Multi-Parametric Toolbox – which is a MATLAB-based toolbox which simplifies and automates many tasks a control engineer has to go through when designing and validating optimal control laws based on the MPC principle. As the name of the tool hints, its primal objective is to address the problem of multi-parametric programming described in Chapter 4. Based on this principle, the toolbox allows to formulate, solve, analyze and deploy feedback controllers for linear and hybrid systems with constraints. The toolbox offers a broad spectrum of algorithms compiled in a user friendly and accessible format starting from different performance objectives (linear, quadratic, minimum-time) to the handling of systems with persistent additive and polytopic uncertainties. Users can add custom constraints, such as polytopic, contraction or collision avoidance constraints, or create custom objective functions. Resulting optimal control laws can either be embedded into target applications in the form of the C code, or deployed to control platforms using the Real Time Workshop.

Note that many results in this thesis have been obtained in close collaboration with various colleagues based on the papers listed in Chapter 27. Moreover, not all of the results which were obtained during my graduate studies are contained in this thesis.

# Part I

# BACKGROUND

# 2

# Standard Optimization Problems

For the sake of completeness, some standard optimization problems and definitions will first be introduced. For a detailed reference, we refer the reader to the excellent book [BV04].

A generic optimization problem can be described by the following set of equations.

$$\min_{x} \quad f_0(x) \tag{2.1a}$$

$$\text{subj. to} \quad f_i(x) \leq 0, \qquad i = 1, \dots, q, \tag{2.1b}$$

$$g_j(x) = 0, \qquad j = 1, \dots, q_{eq}, \tag{2.1c}$$

with an objective function $f_0 : \mathbb{R}^n \to \mathbb{R}$ and constraint functions $f_i : \mathbb{R}^n \to \mathbb{R}$, $g_j : \mathbb{R}^n \to \mathbb{R}$. The variable $x$ is the *optimization variable* and the solution $x^*$ to optimization problem (2.1) is referred to as *optimizer*.

**Definition 2.1.1 (Convex Function, [Wei])** *A convex function is a continuous function whose value at the midpoint of every interval in its domain does not exceed the average of its values at the ends of the interval. In other words, a function $f(x)$ is convex on an interval $[a, b]$ if for any two points $x_1$ and $x_2$ in $[a, b]$,*

$$f(\frac{1}{2}(x_1 + x_2)) \leq \frac{1}{2}(f(x_1) + f(x_2))$$

*If $f(x)$ has a second derivative in $[a, b]$, then a necessary and sufficient condition for it to be convex on that interval is that the second derivative $f''(x) \geq 0$ for all $x$ in $[a, b]$.*

If the objective function $f_0$ and the constraint functions $f_i(x)$ are convex and the equality constraints $g_j$ are all affine (i.e. $A_{eq}x = B_{eq}$), problem (2.1) is a convex optimization problem. Although general convex optimization problems can be solved relatively efficiently it is always advantageous to use dedicated solvers for specific problems. A number of specific convex optimization problems for which such solvers exist will be discussed in the following.

**Linear Program (LP)**

$$\begin{aligned} \min_{x} \quad & c^T x \\ \text{subj. to} \quad & Ax \leq B, \\ & A_{eq}x = B_{eq}. \end{aligned}$$

A practical algorithm to solve an LP with $n$ variables and $s$ constraints requires roughly $O((n^3+n^2s)\sqrt{s})$ operations on average (see Section 4.3, page 36). There are two fundamentally different types of algorithms for solving LPs: *simplex* and *interior-point* solvers. The runtime for the simplex method is exponential in the worst case, while interior-point algorithms have a worst-case polynomial bound. However, this worst-case bound has little relevance for practical problems and both schemes are competitive in practice.

**Quadratic Program (QP)**

$$\begin{aligned} \min_{x} \quad & \frac{1}{2}x^T Q x + c^T x \\ \text{subj. to} \quad & Ax \leq B, \\ & A_{eq}x = B_{eq}. \end{aligned}$$

When referring to QPs it is generally assumed that $Q \succeq 0$, such that the resulting optimization problem is convex. QPs can be solved with roughly the same efficiency as LPs, but on average the solvers are approximately 5-times slower than LP solvers [Neu04, page 37].

**Linear Matrix Inequality (LMI)** The semidefinite cone $F(x) \succeq 0$ can be described with LMIs according to

$$F(x) = F_0 + \sum_{i=0}^{q} x_{(i)} F_i \succeq 0, \quad x \in \mathbb{R}^q, \ F_i = F_i^T \in \mathbb{R}^{n \times n},$$

where $x_{(i)}$ denotes the $i$-th element of the vector $x$. LMIs are generally used when searching for a matrix, for which some *linear* combination of the matrix is required to be positive definite, hence the term *Linear Matrix Inequality* (LMI). In control for example, LMIs are often used to obtain Lyapunov functions and stabilizing feedback laws [BGFB94]. Note that an LMI defines a feasible set and is not an optimization problem as (2.1).

# 3

# Polytopes

Polytopic (or, more general, polyhedral) sets are an integral part of most standard constrained control problems. For this reason we present some definitions and fundamental operations with polytopes. For additional details on polytope computation we refer the reader to [Zie94, Grü00, Fuk00].

## 3.1 Definitions

Some basic definitions in computational geometry will be introduced in this section.

**Definition 3.1.1 (Convex Set, [BV04])** *A set $\mathcal{C}$ is convex if the line segment between any two points in $\mathcal{C}$ lies in $\mathcal{C}$, i.e., if for any $x_1, x_2 \in \mathcal{C}$ and any real scalar $\theta$ with $0 \leq \theta \leq 1$, we have $\theta x_1 + (1-\theta)x_2 \in \mathcal{C}$.*

**Definition 3.1.2 (Neighborhood, [Wei])** *The neighborhood of a point $x \in \mathbb{R}^n$ (also called an epsilon-neighborhood or infinitesimal open set) is the set of points inside an $n$-ball with center $x$ and radius $\epsilon > 0$.*

**Definition 3.1.3 (Closed Set, [Wei])** *A set $\mathcal{S}$ is closed if every point outside $\mathcal{S}$ has a neighborhood disjoint from $\mathcal{S}$.*

**Definition 3.1.4 (Bounded Set, [Wei])** *A set in $\mathbb{R}^n$ is bounded if it is contained inside some ball $\mathcal{B}_R = \{x \in \mathbb{R}^n \mid \|x\|_2 \leq R\}$ of finite radius $R$.*

**Definition 3.1.5 (Compact Set, [Wei])** *A set in $\mathbb{R}^n$ is compact if it is bounded and closed.*

**Definition 3.1.6 (Polyhedron, [Grü00])** *A convex set $\mathcal{S} \subseteq \mathbb{R}^n$ given as an intersection of a finite number of closed half-spaces*

$$\mathcal{S} = \{x \in \mathbb{R}^n \mid S^x x \leq S^o\}, \tag{3.1}$$

*is called a polyhedron. Here, $S^o \in \mathbb{R}^q$, $S^x \in \mathbb{R}^{q \times n}$ where $q$ denotes the number of half-spaces defining $\mathcal{S}$ and the operator $\leq$ denotes an element-wise comparison of two vectors.*

**Definition 3.1.7 (Polytope, [Grü00])** *A bounded polyhedron $\mathcal{P} \subset \mathbb{R}^n$*

$$\mathcal{P} = \{x \in \mathbb{R}^n \mid P^x x \leq P^o\}, \tag{3.2}$$

*is called a polytope. Here, $P^o \in \mathbb{R}^q$, $P^x \in \mathbb{R}^{q \times n}$ where $q$ denotes the number of half-spaces defining $\mathcal{P}$ and the operator $\leq$ denotes a element-wise comparison of two vectors.*

A polytope defined by half-spaces is depicted in Figure 3.1(a).

**Definition 3.1.8 (Dimension of Polytope)** *A polytope $\mathcal{P} \subset \mathbb{R}^n$ is of dimension $d \leq n$, if there exists a $d$-dimensional ball with radius $\epsilon > 0$ contained in $\mathcal{P}$ and there exists no $(d+1)$-dimensional ball with radius $\epsilon > 0$ contained in $\mathcal{P}$.*

**Definition 3.1.9 (Face,Vertex, Edge, Ridge, Facet, [Zie94])** *A linear inequality $a^T x \leq b$ is called valid for a polyhedron $\mathcal{P}$ if $a^T x \leq b$ holds for all $x \in \mathcal{P}$. A subset $\mathcal{F}$ of a polyhedron is called a face of $\mathcal{P}$ if it can be represented as*

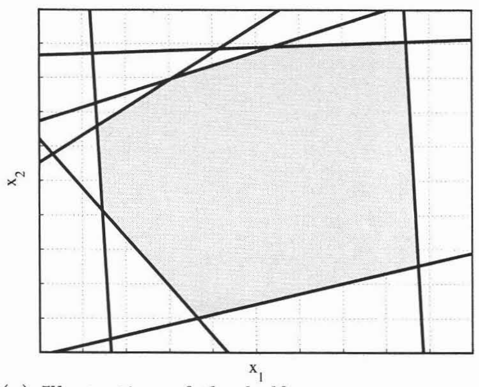
$$\mathcal{F} = \mathcal{P} \cap \{x \in \mathbb{R}^n \mid a^T x = b\}, \tag{3.3}$$

*for some valid inequality $a^T x \leq b$. The faces of a polyhedron $\mathcal{P}$ of dimension 0, 1, $(n-2)$ and $(n-1)$ are called vertices, edges, ridges and facets, respectively.*

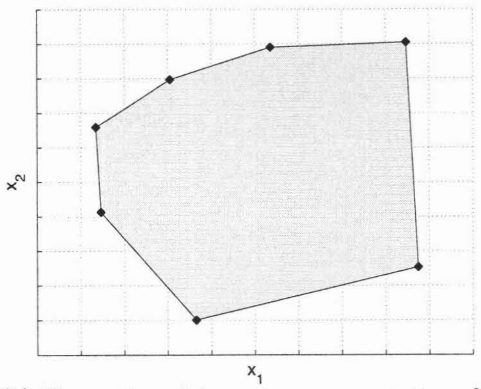
Note that $\emptyset$ and $\mathcal{P}$ itself are also faces of $\mathcal{P}$ [Fuk00].

One of the fundamental properties of a polytope is that it can be described in half-space representation as in Definition 3.2 or in vertex presentation, as given below,

$$\mathcal{P} = \{x \in \mathbb{R}^n \mid x = \sum_{i=1}^{v_P} \alpha_i V_P^{(i)},\ 0 \leq \alpha_i \leq 1,\ \sum_{i=1}^{v_P} \alpha_i = 1\}, \tag{3.4}$$



(a) Illustration of the half-space representation of a polytope $\mathcal{P}$. The half-spaces $P^x_{(i)}x \leq P^o_{(i)}$, $i = 1, \dots, 7$ are depicted in bold.



(b) Illustration of the vertex representation of a polytope $\mathcal{P}$. The vertices $V_P^{(1)}, \dots, V_P^{(7)}$ are depicted in bold.

Figure 3.1: Illustration of a polytope $\mathcal{P}$ in half-space and vertex representation.

where $V_P^{(i)}$ denotes the $i$-th vertex of $\mathcal{P}$, and $v_P$ is the total number of vertices of $\mathcal{P}$ (see Figure 3.1(b)).

It is obvious from the above definitions that every polytope represents a convex and compact set. We say that a polytope $\mathcal{P} \subset \mathbb{R}^n$, $\mathcal{P} = \{x \in \mathbb{R}^n \mid P^x x \leq P^o\}$ is *full dimensional* if $\exists x \in \mathbb{R}^n$, $\epsilon \in \mathbb{R}$ such that $\epsilon > 0$ and $P^x(x + \delta) \leq P^o$, $\forall\, \delta \in \mathbb{R}^n$ subject to $\|\delta\| \leq \epsilon$, i.e., it is possible to fit a $n$-dimensional ball inside the polytope $\mathcal{P}$. A polytope is referred to as *empty* if $\nexists x \in \mathbb{R}^n$ such that $P^x x \leq P^o$. Furthermore, if $\|P^x_{(i)}\| = 1$, where $P^x_{(i)}$ denotes $i$-th row of a matrix $P^x$, we say that the polytope $\mathcal{P}$ is *normalized.*

**Remark 3.1.10** *Note that the MPT toolbox (see Part III or [KGB04]) only deals with full dimensional polytopes. Polyhedra and lower dimensional polytopes are not considered, since they are not necessary to formulate realistic control problems, i.e. it is always possible to formulate the problems using full dimensional polytopic sets only.*

We say that a polytope $\mathcal{P} \subset \mathbb{R}^n$, $\mathcal{P} = \{x \in \mathbb{R}^n \mid P^x x \leq P^o\}$ is in a *minimal representation* if the removal of any of the rows in $P^x x \leq P^o$ would change it (i.e., there are no redundant half-spaces). The computation of a minimal representation (henceforth referred to as *polytope reduction*) of polytopes is discussed in Section 4.3 and generally requires to solve one LP for each half-space defining the non-minimal representation of $\mathcal{P}$ [Bon83]. It is straightforward to see that a normalized, full dimensional polytope $\mathcal{P}$

has a *unique* minimal representation. This fact is very useful in practice. Normalized, full dimensional polytopes in a minimal representation allow us to avoid any ambiguity when comparing them and very often speed up other polytope manipulations.

**Definition 3.1.11 (P-collection)** *A P-collection is the (possibly non-convex) union of a finite number of $R$ polytopes $\mathcal{R}_r$, i.e. $\mathcal{R} = \bigcup_{r\in\{1,\dots,R\}} \mathcal{R}_r$.*

Note that the polytopes $\mathcal{R}_r$ defining the P-collection $\mathcal{R}$ can be disjoint and/or overlapping.

**Remark 3.1.12** *Algorithms for all operations and functions described in this chapter are contained in the MPT toolbox (see Part III or [KGB04]).*

## 3.2 Operations on Polytopes

In this section, some of the basic manipulations on polytopes will be defined.

**Chebychev Ball:** The Chebychev Ball of a polytope $\mathcal{P} = \{x \in \mathbb{R}^n \mid P^x x \leq P^o\}$ corresponds to the largest radius ball $\mathcal{B}_R(x_c) = \{x \in \mathbb{R}^n \mid \|x - x_c\|_2 \leq R\}$, such that $\mathcal{B}_R \subset \mathcal{P}$, see Figure 3.2(a). The center and radius of the Chebychev ball can be easily found by solving the following LP [BV04]

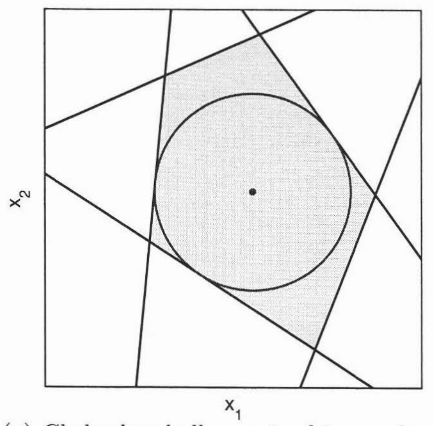
$$\max_{x_c,R} \; R \tag{3.5a}$$

$$\text{subj. to} \;\; P^x_{(i)} x_c + R\|P^x_{(i)}\| \leq P^o_{(i)}, \quad \forall i \in \{1,\dots,q\}. \tag{3.5b}$$

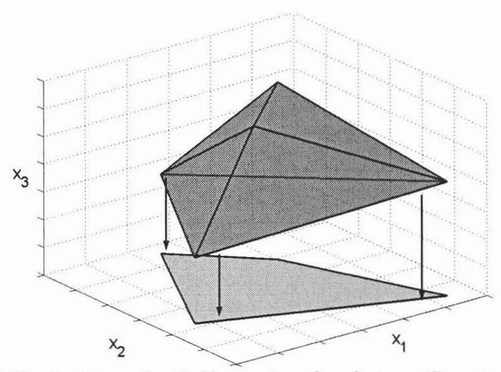
The subindex $(i)$ in (3.5) denotes the $i$-th row of $P^x_{(i)}$ and $P^o_{(i)}$, respectively and $\mathcal{P}$ is defined by the intersection of $q$ half-spaces. If the obtained radius $R = 0$, then the polytope is lower dimensional; if $R < 0$, then the polytope is empty. Note that the center of the Chebychev Ball is not unique, in general, i.e. there can be multiple solutions (e.g. for rectangles).

**Projection:** Given a polytope $\mathcal{P} = \{x \in \mathbb{R}^n, y \in \mathbb{R}^m \mid P^x x + P^y y \leq P^o\} \subset \mathbb{R}^{n+m}$ the orthogonal projection onto the $x$-space $\mathbb{R}^n$ is defined as

$$\text{proj}_x(\mathcal{P}) \triangleq \{x \in \mathbb{R}^n \mid \exists y \in \mathbb{R}^m \text{subj. to } P^x x + P^y y \leq P^o\}. \tag{3.6}$$



(a) Chebychev ball contained in a polytope $\mathcal{P}$.



(b) Projection of a 3-dimensional polytope $\mathcal{P}$ onto a plane.

Figure 3.2: Illustration of the projection operation and the Chebychev ball.

An illustration of a projection operation is given in Figure 3.2(b). Current projection methods can be grouped into four classes: Fourier elimination [Cer63, KS90], block elimination [Bal98], vertex based approaches and wrapping-based techniques [JKM04]. For a good introduction to projection, we refer the reader to [JKM04] and the references therein.

**Set-Difference:** The Set-Difference of two polytopes $\mathcal{P}$ and $\mathcal{Q}$

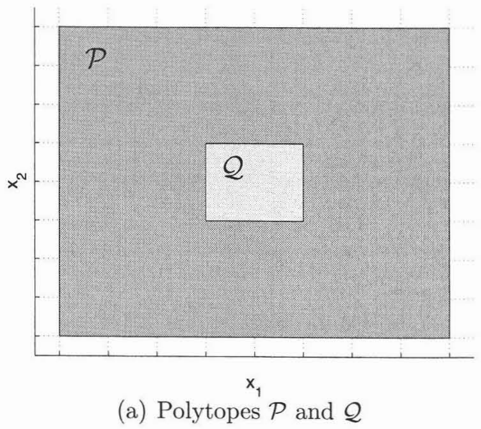
$$\mathcal{R} = \mathcal{P} \setminus \mathcal{Q} \triangleq \{x \in \mathbb{R}^n \mid x \in \mathcal{P}, x \notin \mathcal{Q}\}, \tag{3.7}$$

is a P-collection $\mathcal{R} = \bigcup_i \mathcal{R}_i$, which is easily computed by consecutively inverting the half-spaces defining $\mathcal{Q}$ as described in [BMDP02] (see Figure 3.3). The set difference between two P-collections $\mathcal{C}$ and $\mathcal{D}$ can be computed as described in [BT03, GKBM05, RKM03]. Checking whether $\mathcal{C} \subseteq \mathcal{D}$ is easily implemented since $\mathcal{C} \subseteq \mathcal{D} \Leftrightarrow \mathcal{C} \setminus \mathcal{D} = \emptyset$. Similarly $\mathcal{C} = \mathcal{D}$ is also easily verified since $\mathcal{C} = \mathcal{D} \Leftrightarrow (\mathcal{C} \setminus \mathcal{D} = \emptyset$ and $\mathcal{D} \setminus \mathcal{C} = \emptyset)$.

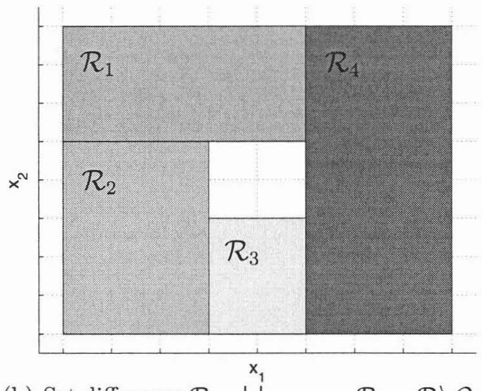
**Remark 3.2.1** *The set difference of two closed sets $\mathcal{C}$ and $\mathcal{D}$ is an open set, if $\mathcal{C} \cap \mathcal{D} \neq \emptyset$. In this thesis, we will henceforth only consider the closure of $\mathcal{C} \setminus \mathcal{D}$.*

**Convex Hull:** The convex hull of a set of points $V = [v_1, \dots, v_P]$ is defined as

$$\text{hull}(V) = \{x \in \mathbb{R}^n \mid x = \sum_{i=1}^{v_P} \alpha_i v_i,\ 0 \leq \alpha_i \leq 1,\ \sum_{i=1}^{v_P} \alpha_i = 1\}. \tag{3.8}$$



(a) Polytopes $\mathcal{P}$ and $\mathcal{Q}$



(b) Set difference $\mathcal{R} = \bigcup_{i\in\{1,\dots,4\}} \mathcal{R}_i = \mathcal{P}\setminus\mathcal{Q}$.

Figure 3.3: Illustration of the set-difference operation.

The convex hull operation is used to switch between half-space and vertex representations. The convex hull of a union of polytopes (referred to as *Extended Convex Hull*, [FLL00]) $\mathcal{R}_r \subset \mathbb{R}^n$, $r = 1, \dots, R$, is a polytope

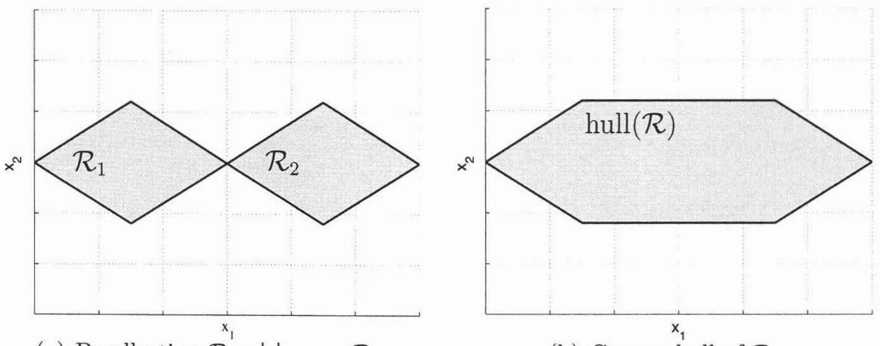
$$\text{hull}\left(\bigcup_{r=1}^{R} \mathcal{R}_r\right) \triangleq \{x \in \mathbb{R}^n \mid \exists x_r \in \mathcal{R}_r,\ x = \sum_{r=1}^{R} \alpha_r x_r,\ 0 \le \alpha_r \le 1,\ \sum_{r=1}^{R} \alpha_r = 1\}. \tag{3.9}$$

An illustration of the convex hull operation is given in Figure 3.4.

**Envelope:** The envelope of two polyhedra $\mathcal{P} = \{x \in \mathbb{R}^n \mid P^x x \le P^o\}$ and $\mathcal{Q} = \{x \in \mathbb{R}^n \mid Q^x x \le Q^o\}$ is given by

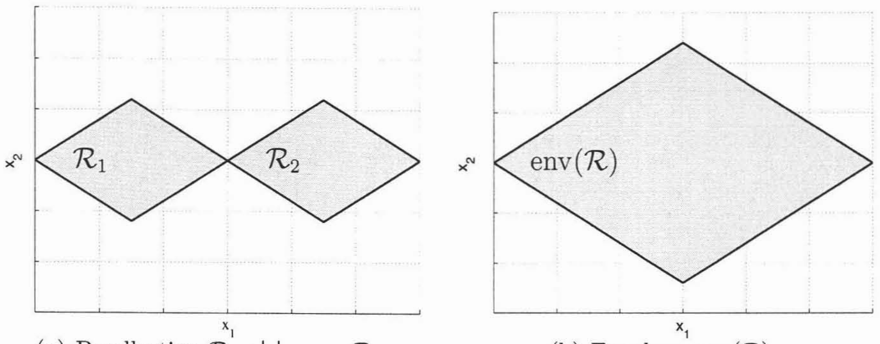
$$\text{env}(\mathcal{P}, \mathcal{Q}) = \{x \in \mathbb{R}^n \mid \bar{P}^x x \le \bar{P}^o,\ \bar{Q}^x x \le \bar{Q}^o\}, \tag{3.10}$$

where $\bar{P}^x x \le \bar{P}^o$ is the subsystem of $P^x x \le P^o$ obtained by removing all the inequalities not valid for the polyhedron $\mathcal{Q}$, and $\bar{Q}^x x \le \bar{Q}^o$ are defined in a similar way with respect to $Q^x x \le Q^o$ and $\mathcal{P}$ [BFT01]. The envelope can analogously be computed for a P-collection or a complex. An illustration of the envelope operation is depicted in Figure 3.5. The envelope can be computed by solving $c \cdot d$ LPs where $c$ is the number of input polytopes (here: $\mathcal{P}$ and $\mathcal{Q}$, i.e. $c = 2$) and $d$ is the total number of facets [BFT01]. It holds that $\mathcal{P} \cup \mathcal{Q} \subseteq \text{env}(\mathcal{P}, \mathcal{Q})$ and that $\mathcal{P} \cup \mathcal{Q}$ is convex $\Leftrightarrow$ $\mathcal{P} \cup \mathcal{Q} = \text{env}(\mathcal{P}, \mathcal{Q})$. Note that the envelope of several polytopes can be a polyhedral set or even $\mathbb{R}^n$, e.g. the envelope of a star shaped object is $\mathbb{R}^n$.



(a) P-collection $\mathcal{R} = \bigcup_{i\in\{1,2\}} \mathcal{R}_i$. (b) Convex hull of $\mathcal{R}$.

Figure 3.4: Illustration of the convex hull operation.



(a) P-collection $\mathcal{R} = \bigcup_{i\in\{1,2\}} \mathcal{R}_i$. (b) Envelope env($\mathcal{R}$).

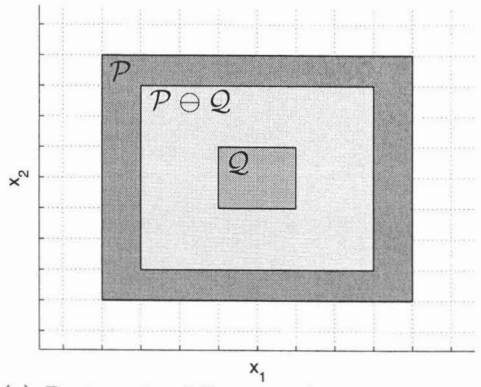
Figure 3.5: Illustration of the envelope operation.

**Vertex Enumeration:** The operation of extracting the vertices of a polytope $\mathcal{P}$ given in half-space representation is referred to as vertex enumeration [FP96, Fuk97]. This operation is the dual to the convex hull operation and the algorithmic implementation is identical to a convex hull computation, i.e. given a set of points $V = [v_1, \dots, v_P]$ it holds that $V = \text{vert}(\text{hull}(V))$, where the operator vert denotes the vertex enumeration.

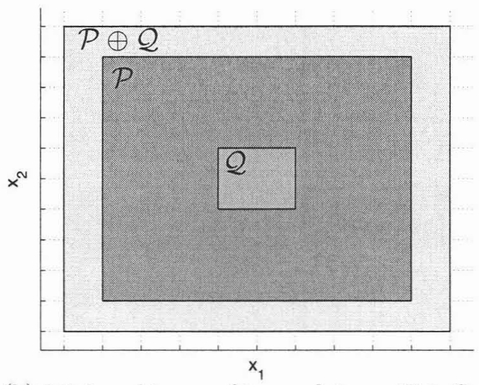
**Pontryagin Difference:** The Pontryagin difference (also known as Minkowski-Difference) of two polytopes $\mathcal{P}$ and $\mathcal{Q}$ is a polytope

$$\mathcal{P} \ominus \mathcal{Q} \triangleq \{x \in \mathbb{R}^n \mid x + q \in \mathcal{P},\ \forall q \in \mathcal{Q}\}. \tag{3.11}$$

The Pontryagin difference can be computed by solving one LP for each half-space defining $\mathcal{P}$ [KG98]. For special cases (e.g. when $\mathcal{Q}$ is a hypercube), even more efficient computational methods exist [KM03]. An illustration of the Pontryagin difference is given in Figure 3.6(a).



(a) Pontryagin difference of two polytopes $\mathcal{P}\ominus\mathcal{Q}$.



(b) Minkowski sum of two polytopes $\mathcal{P}\oplus\mathcal{Q}$.

Figure 3.6: Illustration of the Pontryagin difference and Minkowski sum operations.

**Minkowski Sum:** The Minkowski sum of two polytopes $\mathcal{P}$ and $\mathcal{Q}$ is a polytope

$$\mathcal{P}\oplus\mathcal{Q} \triangleq \{x+q \in \mathbb{R}^n \mid x\in\mathcal{P},\ q\in\mathcal{Q}\}. \tag{3.12}$$

If $\mathcal{P}$ and $\mathcal{Q}$ are given in vertex representation, the Minkowski sum can be computed in time bounded by a polynomial function of input and output size. If $\mathcal{P}$ and $\mathcal{Q}$ are given in half-space representation, the Minkowski sum is a computationally expensive operation which requires either vertex enumeration and convex hull computation in $n$-dimensions or a projection from $2n$ down to $n$ dimensions. The implementation of the Minkowsi sum via projection is described below.

$$P=\{y\in\mathbb{R}^n \mid P^y y\le P^o\},\qquad Q=\{z\in\mathbb{R}^n \mid Q^z z\le Q^o\},$$

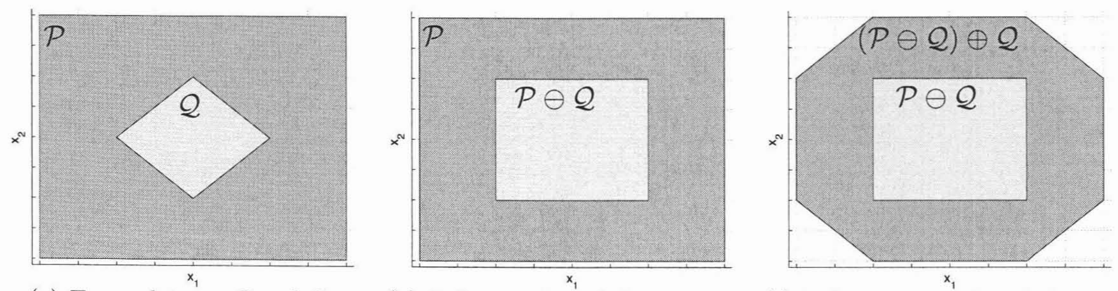
it holds that

$$\begin{aligned}
W &= P\oplus Q\\
&= \Big\{x\in\mathbb{R}^n \mid x=y+z,\ P^y y\le P^o,\ Q^z z\le Q^o,\ y,z\in\mathbb{R}^n\Big\}\\
&= \Big\{x\in\mathbb{R}^n \mid \exists y\in\mathbb{R}^n,\ \text{subj. to } P^y y\le P^o,\ Q^z(x-y)\le Q^o\Big\}\\
&= \left\{x\in\mathbb{R}^n \mid \exists y\in\mathbb{R}^n,\ \text{subj. to } \begin{bmatrix}0 & P^y\\ Q^z & -Q^z\end{bmatrix}\begin{bmatrix}x\\ y\end{bmatrix}\le\begin{bmatrix}P^o\\ Q^o\end{bmatrix}\right\}
\end{aligned}$$

$$= \text{proj}_x\left(\left\{x, y \in \mathbb{R}^n \mid \begin{bmatrix} 0 & P^y \\ Q^z & -Q^z \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \leq \begin{bmatrix} P^o \\ Q^o \end{bmatrix}\right\}\right).$$

Both the projection and vertex enumeration based methods are implemented in the MPT toolbox (see Part III or [KGB04]). An illustration of the Minkowski sum is given in Figure 3.6(b).

**Remark 3.2.2** *The Minkowski sum is* not *the complement of the Pontryagin difference. For two polytopes $\mathcal{P}$ and $\mathcal{Q}$, it holds that $(\mathcal{P} \ominus \mathcal{Q}) \oplus \mathcal{Q} \subseteq \mathcal{P}$. This is illustrated in Figure 3.7.*


$\mathcal{P}$
$\mathcal{Q}$
$x_2$
$x_1$
$\mathcal{P}$
$\mathcal{P} \ominus \mathcal{Q}$
$x_2$
$x_1$
$(\mathcal{P} \ominus \mathcal{Q}) \oplus \mathcal{Q}$
$\mathcal{P} \ominus \mathcal{Q}$
$x_2$
$x_1$

(a) Two polytopes $\mathcal{P}$ and $\mathcal{Q}$. (b) Polytope $\mathcal{P}$ and Pontryagin difference $\mathcal{P} \ominus \mathcal{Q}$. (c) Polytope $\mathcal{P} \ominus \mathcal{Q}$ and the set $(\mathcal{P} \ominus \mathcal{Q}) \oplus \mathcal{Q}$.

Figure 3.7: Illustration that $(\mathcal{P} \ominus \mathcal{Q}) \oplus \mathcal{Q} \subseteq \mathcal{P}$.

## 3.3 Operations on P-collections

This section covers some results and algorithms which are specific to operations with P-collections. P-collections are unions of polytopes and therefore the set of states contained in a P-collection can be represented in an infinite number of ways, i.e. the P-collection representation is not unique. For example, one can subdivide any polytope $\mathcal{P}$ into a number of smaller polytopes whose union is a P-collection which covers $\mathcal{P}$. Note that the complexity of all subsequent computations depends strongly on the number of polytopes representing a P-collection. The smaller the cardinality of a P-collection, the more efficient the computations.

The first two results given here show how the set difference of a P-collection and a P-collection (or polyhedron) may be computed:

**Lemma 3.3.1** *Let $\mathcal{C} \triangleq \bigcup_{j\in\{1,\ldots,J\}} \mathcal{C}_j$ be a P-collection, where all the $\mathcal{C}_j$, $j \in \{1,\ldots,J\}$, are non-empty polyhedra. If $\mathcal{S}$ is a non-empty polyhedron, then $\mathcal{C} \setminus \mathcal{S} = \bigcup_{j\in\{1,\ldots,J\}} (\mathcal{C}_j \setminus \mathcal{S})$ is a P-collection.*

**Lemma 3.3.2** *Let the sets $\mathcal{C} \triangleq \bigcup_{j\in\{1,\ldots,J\}} \mathcal{C}_j$ and $\mathcal{D} \triangleq \bigcup_{y=1,\ldots,Y} \mathcal{D}_y$ be P-collections, where all the $\mathcal{C}_j$, $j \in \{1,\ldots,J\}$, and $\mathcal{D}_y$, $y \in \{1,\ldots,Y\}$, are non-empty polyhedra. If $\mathcal{E}_0 \triangleq \mathcal{C}$ and $\mathcal{E}_y \triangleq \mathcal{E}_{y-1} \setminus \mathcal{D}_y$, $y \in \{1,\ldots,Y\}$ then $\mathcal{C} \setminus \mathcal{D} = \mathcal{E}_Y$ is a P-collection.*

The reader is referred to [RKM03] for proofs and comments on computational efficiency. That $\mathcal{C} \subseteq \mathcal{D}$ can be easily verified since $\mathcal{C} \subseteq \mathcal{D} \Leftrightarrow \mathcal{C} \setminus \mathcal{D} = \emptyset$, similarly $\mathcal{C} = \mathcal{D}$ is also easily verified since

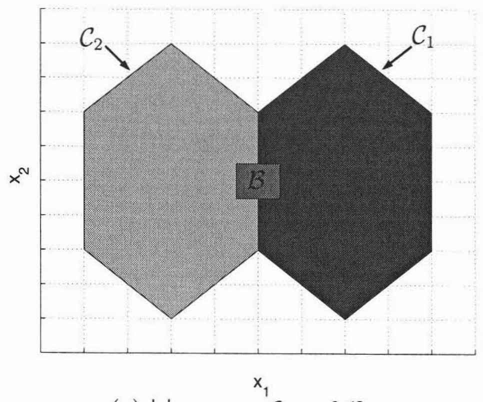
$$\mathcal{C} = \mathcal{D} \Leftrightarrow (\mathcal{C} \setminus \mathcal{D} = \emptyset \text{ and } \mathcal{D} \setminus \mathcal{C} = \emptyset)$$

Next, an efficient algorithm for computing the Pontryagin difference of a P-collection and a polytope is presented. If $\mathcal{S}$ and $\mathcal{B}$ are two subsets of $\mathbb{R}^n$ it is known that $\mathcal{S} \ominus \mathcal{B} = [\mathcal{S}^c \oplus (-\mathcal{B})]^c$ (see for instance [Ser88, Ker00]), where $(\cdot)^c$ denotes the set complement. The following algorithm taken from [RGK$^+$04] implements the computation of the Pontryagin difference of a P-collection $\mathcal{C} \triangleq \cup_{j\in\{1,\ldots,J\}} \mathcal{C}_j$, where $\mathcal{C}_j, j \in \{1,\ldots,J\}$ are polytopes in $\mathbb{R}^n$, and a polytope $\mathcal{B} \subset \mathbb{R}^n$.

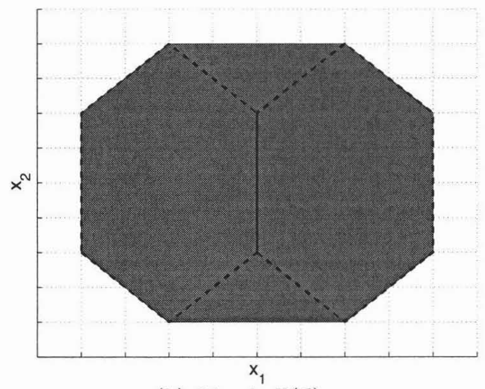
**Algorithm 3.3.3 (Pontryagin Difference for P-collections, $\mathcal{C} \ominus \mathcal{B}$)**

```
1. Input: P-collection $\mathcal{C}$, polytope $\mathcal{B}$;
2. $\mathcal{H} \triangleq \mathrm{env}(\mathcal{C})$ (or $\mathcal{H} \triangleq \mathrm{hull}(\mathcal{C})$);
3. $\mathcal{D} \triangleq \mathcal{H} \ominus \mathcal{B}$;
4. $\mathcal{E} \triangleq \mathcal{H} \setminus \mathcal{C}$;
5. $\mathcal{F} \triangleq \mathcal{E} \oplus (-\mathcal{B})$;
6. $\mathcal{G} \triangleq \mathcal{D} \setminus \mathcal{F}$;
7. Output: P-collection $\mathcal{G} \triangleq \mathcal{C} \ominus \mathcal{B}$.
```

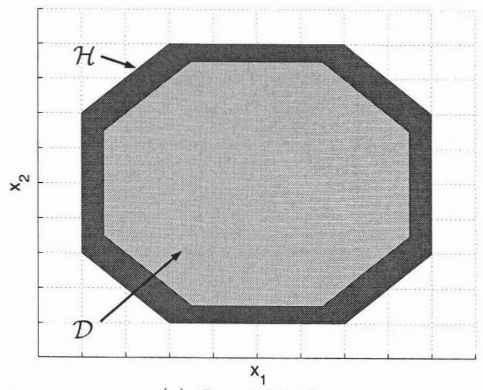
***Remark 3.3.4*** *Note that $\mathcal{H}$ in Step 2 of Algorithm 3.3.3 can be any convex set containing the P-collection $\mathcal{C}$. Furthermore, the computation of $\mathcal{H}$ is generally more efficient if the envelope operation is used instead of convex hull.*



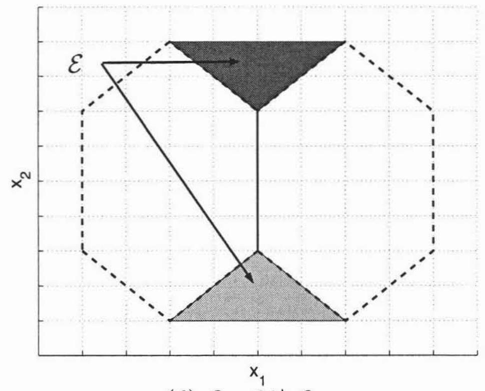
(a) $\bigcup_{j\in\{1,\dots,J\}}\mathcal{C}_j$ and $\mathcal{B}$.



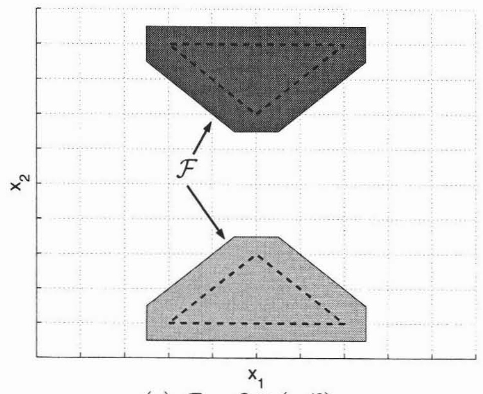
(b) $\mathcal{H} = \text{hull}(\mathcal{C})$.



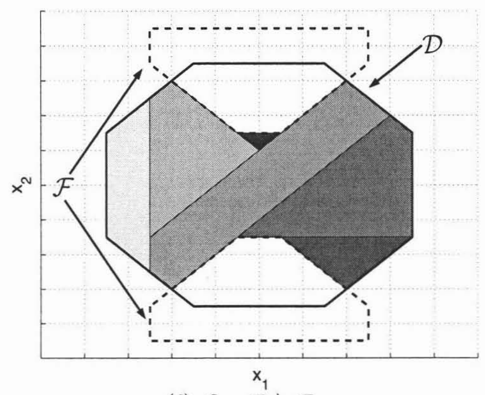
(c) $\mathcal{D} = \mathcal{H} \ominus \mathcal{B}$.



(d) $\mathcal{E} = \mathcal{H} \setminus \mathcal{C}$.



(e) $\mathcal{F} = \mathcal{E} \oplus (-\mathcal{B})$.



(f) $\mathcal{G} = \mathcal{D} \setminus \mathcal{F}$.

Figure 3.8: Graphical illustration of Algorithm 3.3.3.

***Remark 3.3.5*** *It is important to note that* $(\bigcup_{j\in\{1,\ldots,J\}} \mathcal{C}_j) \ominus \mathcal{B} \neq \bigcup_{j\in\{1,\ldots,J\}}(\mathcal{C}_j \ominus \mathcal{B})$*, where* $\mathcal{B}$ *and* $\mathcal{C}_j$ *are polyhedra; hence, the relatively high computational effort of computing the Pontryagin difference of a P-collection and a polytope.*

**Theorem 3.3.6 (Computation of Minkowski Difference, [RGK+04])** *For Algorithm 3.3.3,* $\mathcal{G} = \mathcal{C} \ominus \mathcal{B}$.

**Proof** It holds by definition that

$$\begin{aligned} \mathcal{D} &\triangleq \mathcal{H} \ominus \mathcal{B} = \{x \ rt \ x + w \in \mathcal{H}, \ \forall w \in \mathcal{B}\}, \\ \mathcal{E} &\triangleq \mathcal{H} \setminus \mathcal{C} = \{x \ rt \ x \in \mathcal{H} \text{ and } x \notin \mathcal{C}\}. \end{aligned}$$

By the definition of the Minkowski sum:

$$\begin{aligned} \mathcal{F} \triangleq \mathcal{E} \oplus (-\mathcal{B}) &= \{x \ rt \ x = z + w, \ z \in \mathcal{E}, w \in (-\mathcal{B})\} \\ &= \{x \ rt \ \exists w \in (-\mathcal{B}), \text{ s.t. } x + w \in \mathcal{E}\}. \end{aligned}$$

By definition of the set difference:

$$\begin{aligned} \mathcal{D} \setminus \mathcal{F} &\triangleq \{x \mid x \in \mathcal{D} \text{ and } x \notin \mathcal{F}\} \\ &= \{x \in \mathcal{D} \ rt \ \nexists \ w \in \mathcal{B} \text{ s.t. } x + w \in \mathcal{E}\} \\ &= \{x \in \mathcal{D} \ rt \ x + w \notin \mathcal{E}, \ \forall w \in \mathcal{B}\}. \end{aligned}$$

From the definition of the set $\mathcal{D}$:

$$\mathcal{D} \setminus \mathcal{F} = \{x \ rt \ x + w \in \mathcal{H} \text{ and } x + w \notin \mathcal{E}, \ \forall w \in \mathcal{B}\}$$

And from the definition of the set $\mathcal{E}$ and because $\mathcal{C} \subseteq \mathcal{H}$:

$$\begin{aligned} \mathcal{D} \setminus \mathcal{F} &= \{x \ rt \ x + w \in \mathcal{H} \text{ and } \ (x + w \notin \mathcal{H} \text{ or } x + w \in \mathcal{C}) \ \forall w \in \mathcal{B}\} \\ &= \{x \ rt \ x + w \in \mathcal{C}, \ \forall w \in \mathcal{B}\} \\ &= \mathcal{C} \ominus \mathcal{B}. \end{aligned}$$

□

Algorithm 3.3.3 is illustrated on a sample P-collection in Figures 3.8(a) to 3.8(f).

**Remark 3.3.7** *It should be noted that Algorithm 3.3.3 for computation of the Pontryagin difference is conceptually similar to the one proposed in [Ser88, Ker00, KM02]. However, the envelope [BFT01] operation employed in step 2 significantly reduces (in general) the number of sets obtained at step 4, which in turn results in fewer Minkowski set additions. Since the computation of a Minkowski set addition is expensive, a runtime improvement can be expected. The necessary computations can be efficiently implemented by using standard computational geometry software such as [Ver03, KGB04].*

# 4

# Multi-Parametric Programming

In this chapter, the basics of multi-parametric programming will be summarized. For a review of standard optimization techniques, we refer the reader to [BV04]. An in-depth discussion of multi-parametric programs is given in [Bor03] and [Tøn00].

## 4.1 Definitions

Consider the following optimization problem

$$\begin{aligned} J_N^*(x) &= \min_{U_N} V(x, U_N) && (4.1a) \\ \text{subj. to} &\quad GU_N \leq W + Ex, && (4.1b) \end{aligned}$$

where $U_N \in \mathbb{R}^N$ is the optimization variable and $x \in \mathbb{R}^n$ is the parameter with $G \in \mathbb{R}^{q \times N}$, $W \in \mathbb{R}^q$ and $E \in \mathbb{R}^{q \times n}$. In multi-parametric programming, the objective is to obtain the optimizer $U_N^*$ for a whole range of parameters $x$, i.e. to obtain $U_N^*(x)$ as an explicit function of the parameter $x$. The term *multi* is used to emphasize that the parameter $x$ is a vector and not a scalar. Depending on whether the objective function $V(x, U_N)$ is linear or quadratic in the optimization variable $U_N$, the terminology *multi-parametric Linear Program* (mp-LP) or *multi-parametric Quadratic Program* (mp-QP) is used. First, we give the basic definitions using the mp-QP nomenclature before re-stating the properties of both mp-QP and mp-LP solutions in Section 4.2.

Consider the following quadratic program

$$\begin{aligned} J_N^*(x) &= \min_{U_N} \left\{ U_N^T H U_N + x^T F U_N \right\} && (4.2a) \\ \text{subj. to} &\quad GU_N \leq W + Ex, && (4.2b) \\ &\quad H \succ 0, && (4.2c) \end{aligned}$$

where the column vector $U_N \in \mathbb{R}^N$ is the optimization vector. The number of constraints $q$ corresponds to the number of rows of $W$, i.e. $W \in \mathbb{R}^q$. Henceforth, $U_N^*(x)$ will be used to denote the optimizer of (4.2) for a given parameter $x$. For any given $x$, it is possible to obtain the optimizer by solving a standard quadratic programming problem[1]. Before going further, we will introduce the following definitions.

**Definition 4.1.1 (Feasible Set $\mathcal{X}_N$)** *We define the feasible set $\mathcal{X}_N \subseteq \mathbb{R}^n$ as the set of states $x$ for which the optimization problem (4.2) is feasible, i.e.*

$$\mathcal{X}_N = \{x \in \mathbb{R}^n | \exists U_N \in \mathbb{R}^N,\ GU_N \leq W + Ex\}. \tag{4.3}$$

The set $\mathcal{X}_\infty$ is defined accordingly by $\mathcal{X}_\infty \triangleq \lim_{N\to\infty} \mathcal{X}_N$. The set $\mathcal{X}_N$ can be computed via a projection operation as in (3.6).

**Definition 4.1.2 (Polytopic/Polyhedral Partition)** *A collection of polytopic (polyhedral) sets $\{\mathcal{P}_r\}_{r=1}^R = \{\mathcal{P}_1, \ldots, \mathcal{P}_R\}$ is a polytopic (polyhedral) partition of a polytopic (polyhedral) set $\Theta$ if* (i) $\bigcup_{r=1}^R \mathcal{P}_r = \Theta$, (ii) $(\mathcal{P}_r \backslash \partial\mathcal{P}_r) \cap (\mathcal{P}_q \backslash \partial\mathcal{P}_q) = \emptyset$, $\forall r \neq q$, *where $\partial$ denotes the boundary.*

**Definition 4.1.3 (PWA and PWQ)** *Consider the function $f$ over a polyhedral set $\mathcal{S}$.*
*$f : \mathcal{S} \to \mathbb{R}^d$ with $d \in \mathbb{N}_+$ is piecewise affine (PWA), if a partition $\{\mathcal{P}_r\}_{r=1}^R$ of set $\mathcal{S}$ exists, such that $f(x) = L_r x + C_r$ if $x \in \mathcal{P}_r$.*
*$f : \mathcal{S} \mapsto \mathbb{R}$ is piecewise quadratic (PWQ), if a partition $\{\mathcal{P}_r\}_{r=1}^R$ of set $\mathcal{S}$ exists, such that $f(x) = x^T Q_r x + L_r x + C_r$ if $x \in \mathcal{P}_r$.*

**Definition 4.1.4 (Active Constraints $\mathcal{A}^N(x)$)** *The set of active constraints $\mathcal{A}^N(x)$ at point $x$ of problem (4.2) is defined as*

$$\mathcal{A}^N(x) = \{i \in \mathcal{J} \mid G_{(i)}U_N^*(x) - W_{(i)} - E_{(i)}x = 0\},\ \mathcal{J} = \{1, 2, \ldots, q\},$$

*where $G_{(i)}$, $W_{(i)}$, and $E_{(i)}$ denote the $i$-th row of the matrices $G$, $W$, and $E$, respectively, and $q$ denotes the number of constraints, i.e. $W \in \mathbb{R}^q$.*

**Definition 4.1.5 (Linear Independence Constraint Qualification, [TJB03a])** *For an active set of constraints $\mathcal{A}^N$, we say that the linear independence constraint qualification (LICQ) holds if the set of active constraint gradients are linearly independent , i.e. $G_{\mathcal{A}^N}$ has full row rank.*

[1]The standing assumption here is that $H \succ 0$. The case $H \succeq 0$ is covered in [TJB03b].

## 4.2 Properties and Computation

As shown in [BMDP02, TJB01], we wish to solve problem (4.2) for all $x$ within the polyhedral set of values $\mathcal{X}_N$, by considering (4.2) as a multi-parametric Quadratic Program (mp-QP).

**Theorem 4.2.1 (Properties mp-QP, [BMDP02, Bor03])** *Consider the multi-parametric Quadratic Program (4.2). Then, the set of feasible parameters $\mathcal{X}_N$ is convex, the optimizer $U_N^* : \mathcal{X}_N \to \mathbb{R}^N$ is continuous and piecewise affine (PWA), i.e.*

$$U_N^*(x) = F_r x + G_r, \quad \text{if} \quad x \in \mathcal{P}_r = \{x \in \mathbb{R}^n | H_r x \le K_r\}, \; r = 1, \dots, R, \tag{4.4}$$

*and the optimal value function $J^* : \mathcal{X}_N \to \mathbb{R}$ is continuous, convex and piecewise quadratic.*

**Definition 4.2.2 (Region)** *Each polyhedron $\mathcal{P}_r$ of the polyhedral partition $\{\mathcal{P}_r\}_{r=1}^R$ is referred to as a region.*

For some mp-QP problem, the region partition $\{\mathcal{P}_r\}_{r=1}^R$ and PWQ value function $J^*(x)$ is depicted in Figures 4.1(a) and 4.2(a), respectively. Note that the evaluation of the PWA solution (4.4) of the mp-QP provides the same result as solving the quadratic program, i.e. for any given parameter $x$, the optimizer $U_N^*(x)$ in (4.4) is identical to the optimizer obtained by solving the quadratic program (4.2) for $x$.

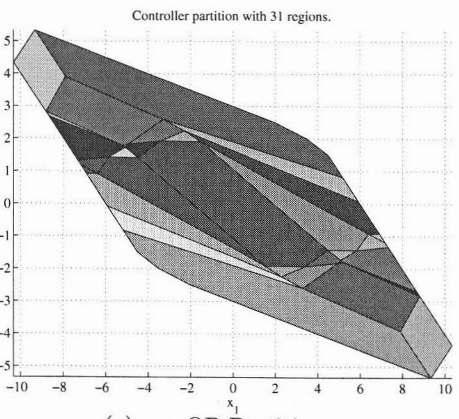
Problem (4.1) with an objective (4.1a) that is linear in the optimizer $U_N$ can be stated as an mp-LP [BBM00a]. The properties of mp-LP solutions are stated below.

**Theorem 4.2.3 (Properties mp-LP, [Bor03, Gal95])** *Consider the the optimization problem (4.1), with a linear objective $V(x, U_N) = x^T c_1^T U_N + c_2^T U_N$. Then, the set of feasible parameters $\mathcal{X}_N$ is convex, there exists an optimizer $U_N^* : \mathcal{X}_N \to \mathbb{R}^{Nm}$ which is continuous and piecewise affine (PWA), i.e.*

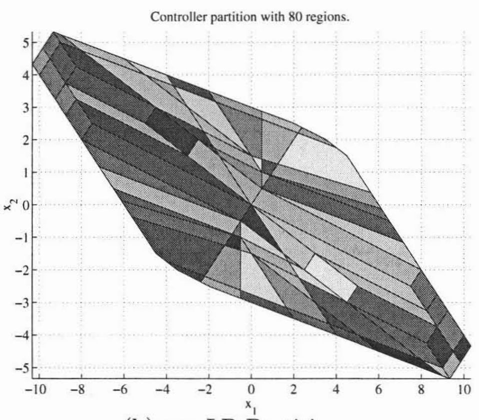
$$U_N^*(x) = F_r x + G_r, \quad \text{if} \quad x \in \mathcal{P}_r = \{x \in \mathbb{R}^n | H_r x \le K_r\}, \; r = 1, \dots, R,$$

*and the value function $J_N^* : \mathcal{X}_N \to \mathbb{R}$ is continuous, convex and piecewise affine.*

For some mp-LP problem, the region partition $\{\mathcal{P}_r\}_{r=1}^R$ and PWA value function $J^*(x)$ is depicted in Figures 4.1(b) and 4.2(b), respectively.



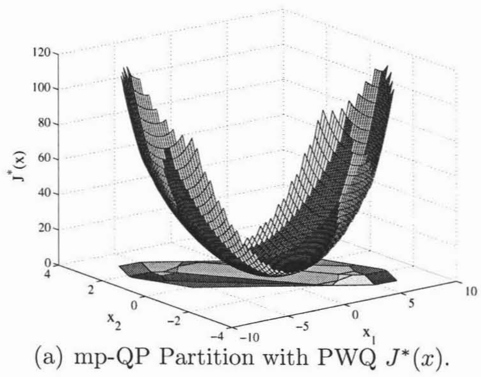
(a) mp-QP Partition.



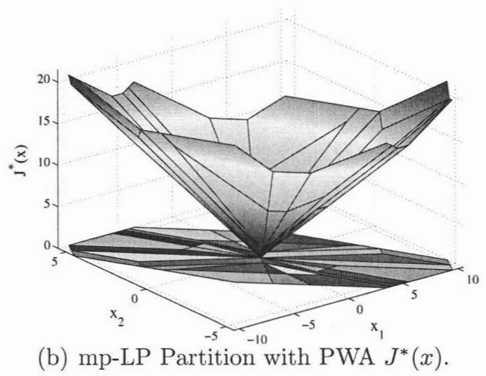
(b) mp-LP Partition.

Figure 4.1: Partition $\{\mathcal{P}_r\}_{r=1}^{R}$ for an mp-LP and an mp-QP problem. The constraints (4.1b) are identical for both problems. Therefore $\mathcal{X}_N$ is also identical for both problems.

**Remark 4.2.4** *Assume that the origin is contained in the interior of the constraint polytope* (4.1b) *in $x$-$U_N$ space. Because the value function for mp-QPs is PWQ, the origin is always contained in the interior of a single region. Specifically, the origin is always contained in the unconstrained region, i.e. the set of active constraints $\mathcal{A}^N(x) = \emptyset$ for $x = 0$. See Figure 4.1.*



(a) mp-QP Partition with PWQ $J^*(x)$.



(b) mp-LP Partition with PWA $J^*(x)$.

Figure 4.2: Partition $\{\mathcal{P}_r\}_{r=1}^{R}$ and value function $J^*(x)$ for an mp-LP and an mp-QP problem.

A brief outline of a generic mp-QP algorithm will be given next. For a detailed discussion of mp-QP algorithms we refer the reader to the literature [BMDP02,TJB03a, Bao02]. Before reviewing the algorithm, it is useful to define

$$z = U_N + H^{-1}F^Tx \tag{4.5}$$

and to transform the QP formulation (4.2) such that the state vector $x$ apears only in constraints, i.e.

$$J_N^*(x) = \min_z \left\{ z^T H z \right\} \tag{4.6a}$$

$$\text{subj. to } Gz \leq W + Sx \tag{4.6b}$$

where $S = E+GH^{-1}F^T$. An mp-QP computation scheme then consist of the following three steps:

1. **Active Constraint Identification:** A feasible parameter $\widehat{x}$ is determined and the associated QP (4.6) is solved. This will yield the optimiser $z$ and active constraints $\mathcal{A}(\widehat{x})$ defined as inequalities that are active at solution, i.e.

   $$\mathcal{A}(\widehat{x}) = \{i \in \mathcal{J} \mid G_{(i)}z = W_{(i)} + S_{(i)}\widehat{x}\},\ \mathcal{J} = \{1, 2, \ldots, q\}, \tag{4.7}$$

   where $G_{(i)}$, $W_{(i)}$, and $S_{(i)}$ denote the $i$-th row of the matrices $G$, $W$, and $S$, respectively, and $q$ denotes the number of constraints. The rows indexed by the active constraints $\mathcal{A}(\widehat{x})$ are extracted from the constraint matrices $G, W$ and $S$ in (4.6) to form the matrices $G_{\mathcal{A}}, W_{\mathcal{A}}$ and $S_{\mathcal{A}}$.

2. **Region Computation:** Next, it is possible to use the Karush-Kuhn-Tucker (KKT) conditions to obtain an explicit representation of the optimiser $U_N(x)$ which is valid in some neighborhood of $\widehat{x}$. These are for our problem defined as

   $$Hz + G^T\lambda = 0 \tag{4.8a}$$

   $$\lambda^T(Gz - W - S\widehat{x}) = 0 \tag{4.8b}$$

   $$\lambda \geq 0 \tag{4.8c}$$

   $$Gz \leq W + S\widehat{x} \tag{4.8d}$$

   Optimised variable $z$ can be solved from (4.8a)

   $$z = -H^{-1}G^T\lambda \tag{4.9}$$

Condition (4.8b) can be separated into active and inactive constraints. For inactive constraints holds $\lambda_{\mathcal{I}} = 0$. For active constraints are the corresponding Lagrange multipliers $\lambda_{\mathcal{A}}$ positive and inequality constraints are changed to equalities. Substituting for $z$ from (4.9) into equality constraints gives

$$-G_{\mathcal{A}}H^{-1}G_{\mathcal{A}}^T\lambda_{\mathcal{A}} + W_{\mathcal{A}} + S_{\mathcal{A}}\widehat{x} = 0 \tag{4.10}$$

and yields expressions for active Lagrange multipliers

$$\lambda_{\mathcal{A}} = -(G_{\mathcal{A}}H^{-1}G_{\mathcal{A}}^T)^{-1}(W_{\mathcal{A}} + S_{\mathcal{A}}\widehat{x}) \tag{4.11}$$

The optimal value of optimiser $z$ and optimal control trajectory $U_N$ are thus given as affine functions of $\widehat{x}$

$$\begin{aligned} z &= -H^{-1}G_{\mathcal{A}}^T(G_{\mathcal{A}}H^{-1}G_{\mathcal{A}}^T)^{-1}(W_{\mathcal{A}} + S_{\mathcal{A}}\widehat{x}) && (4.12)\\ U_N &= z - H^{-1}F^T\widehat{x} \\ &= -H^{-1}G_{\mathcal{A}}^T(G_{\mathcal{A}}H^{-1}G_{\mathcal{A}}^T)^{-1}(W_{\mathcal{A}} + S_{\mathcal{A}}\widehat{x}) - H^{-1}F^T\widehat{x} \\ &= F_r\widehat{x} + G_r && (4.13) \end{aligned}$$

where

$$\begin{aligned} F_r &= H^{-1}G_{\mathcal{A}}^T(G_{\mathcal{A}}H^{-1}G_{\mathcal{A}}^T)^{-1}S_{\mathcal{A}} - H^{-1}F^T && (4.14)\\ G_r &= H^{-1}G_{\mathcal{A}}^T(G_{\mathcal{A}}H^{-1}G_{\mathcal{A}}^T)^{-1}W_{\mathcal{A}} && (4.15) \end{aligned}$$

In a next step, the set of states is determined where the optimiser $U_N(x)$ satisfies the the same active constraints and is optimal. Such a region is characterised by two inequalities (4.8c), (4.8d) and is written compactly as $H_r x \leq K_r$ where

$$H_r = \begin{bmatrix} G(F_r + H^{-1}F^T) - S \\ (G_{\mathcal{A}}H^{-1}G_{\mathcal{A}}^T)^{-1}S_{\mathcal{A}} \end{bmatrix} \tag{4.16}$$

$$K_r = \begin{bmatrix} W - GG_r \\ -(G_{\mathcal{A}}H^{-1}G_{\mathcal{A}}^T)^{-1}W_{\mathcal{A}} \end{bmatrix} \tag{4.17}$$

3. **State Space Exploration:** Once the controller region is computed, the algorithm proceeds iteratively until the entire feasible state space $\mathcal{X}_N$ is covered with controller regions $\mathcal{P}_r$, i.e. $\mathcal{X}_N = \bigcup_{r=1,\dots,R} \mathcal{P}_r$.

**Remark 4.2.5** *The number of rows in $H_r$ is equal to the number of initial constraints* (4.1b), *i.e. $H_r$ consists of $q$ rows if $W \in \mathbb{R}^q$. Therefore, in order to obtain a non-redundant representation of $\mathcal{P}_r$, it is necessary to solve $q$ LPs (see Chapter 3) for each region $r \in \{1, \dots, R\}$. In most cases one can increase the computational efficiency of multi-parametric solvers by computing the non-redundant representation of the original constraint polytope* (4.1b) *before solving the multi-parametric program.*

## 4.3 Efficient Polytope Reduction in Multi-Parametric Programming

Computation of the minimal representations of the controller regions $\mathcal{P}_r = \{x \in \mathbb{R}^n | H_r x \leq K_r\}$ where $H_r$ and $K_r$ are given, respectively, by (4.16) and (4.17) can significantly reduce the computational load in most multi-parametric programming solvers [TJB01]. Therefore in this section we describe techniques which can be applied in order to speed up the process of obtaining minimal representations of convex polytopes.

The standard approach to detect if the $j$th constraint in the set

$$\begin{aligned} Hx &\leq K \\ H &= \begin{bmatrix} h_1 & h_2 & \dots & h_c \end{bmatrix}^T \\ K &= \begin{bmatrix} k_1 & k_2 & \dots & k_c \end{bmatrix}^T \end{aligned} \tag{4.18}$$

is redundant, is to define a new polyhedron with the $j$th constraint removed,

$$\begin{aligned} \widetilde{H} &= \begin{bmatrix} h_1 & h_{j-1} & h_{j+1} \dots & h_c \end{bmatrix}^T \\ \widetilde{K} &= \begin{bmatrix} k_1 & k_{j-1} & k_{j+1} \dots & k_c \end{bmatrix}^T \end{aligned}$$

and maximize $h_j^T x$ in the reduced polytope $\widetilde{H}x \leq \widetilde{K}$

$$\max_x \; h_j^T x \tag{4.19a}$$

$$s.t. \quad \widetilde{H}x \leq \widetilde{K} \tag{4.19b}$$

If the optimal objective value of this problem is less than or equal to $k_j$, the constraint is redundant and can be removed [Fuk00].

To detect and remove all redundant constraints, the algorithm requires the solution of $c$ LPs with, in the worst-case, $c-1$ constraints and $n$ variables. To improve the performance of this algorithm, we need to reduce the number of LPs to be solved, and preferably also their size. Our approach to do this is to perform an initial, computationally cheap, pre-solve analysis to detect a sub-set of the redundant and non-redundant constraints.

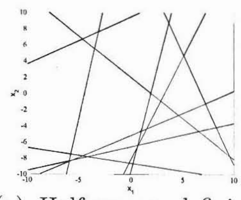
### 4.3.1 Detecting Non-Redundant Half-Spaces

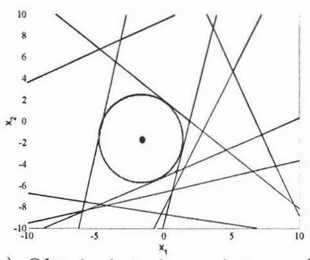
By detecting some of the non-redundant constraints, we can reduce the number of LPs that have to be solved to derive the minimal representation of a polytope. We first propose the application of a simple randomized ray-shooting approach [Bon83].

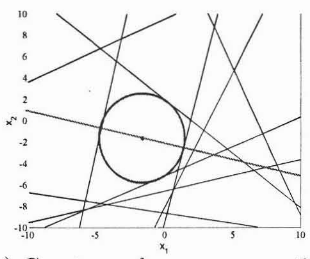
1. Initialize the set of non-redundant constraints $\mathcal{J}_N = \emptyset$
2. Calculate an interior point $x_{\text{int}}$, $Hx_{\text{int}} < K$
3. Generate a random direction $d \in \mathbb{R}^n$
4. Calculate intersections between the line $x_{\text{int}} + t_i d$ and the hyper-plane $h_i^T x = k_i$, giving $t_i = \frac{k_i - h_i^T x_{\text{int}}}{h_i^T d}$.
5. Find the closest intersecting hyper-planes along positive and negative direction $d$, corresponding to smallest positive and largest negative $t$ respectively. Let the corresponding indices to these hyper-planes be $i_p$ and $i_n$. These constraints are non-redundant such that $\mathcal{J}_N := \mathcal{J}_N \cup i_p \cup i_n$
6. Let the mid-point of the line between the two intersection points $x_{\text{int}} + t_p d$ and $x_{\text{int}} + t_n d$ serve as a new interior point, $x_{\text{int}} := x_{\text{int}} + \frac{t_{i_p} + t_{i_n}}{2} d$
7. Repeat from 3)

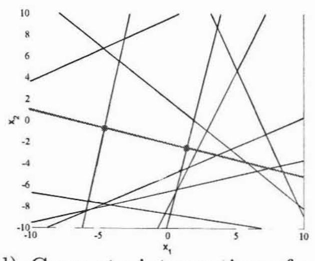
An illustration of this algorithm is given in Figure 4.3. The algorithm requires an interior point to begin with in step 2. To find one, we calculate the Chebychev center of the polytope, requiring the solution of one LP (see (3.5)).

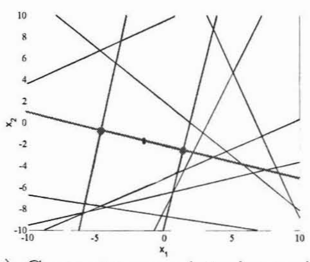
**Remark 4.3.1** *Note that the active constraints (see Definition 4.1.4) which are obtained when solving the Chebychev-Ball problem can also be used to initialize the set of non-redundant constraints $\mathcal{J}_N$. Obviously, all half-spaces which are 'touched' by the ball are non-redundant, provided all duplicate half-spaces have been removed.*

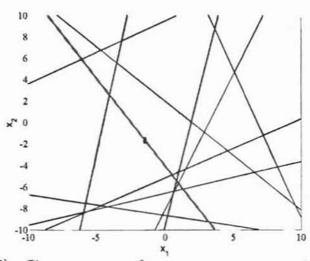

(a) Half-spaces defining the polytope.


(b) Obtain interior point $x_{\text{int}}$ by computing the Chebychev ball.


(c) Create random ray emanating from $x_{\text{int}}$.


(d) Compute intersection of ray with closest half-spaces (bold circles). These half-spaces are non-redundant.


(e) Compute new interior point $x_{\text{int}}$.


(f) Create random ray emanating from $x_{\text{int}}$.

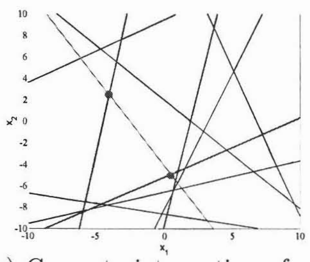

(g) Compute intersection of ray with closest-half spaces (bold circles).

Figure 4.3: Illustration of the scheme to detect non-redundant half-spaces. Here, all non-redundant constraints happen to be identified by computing the Chebychev ball (see Remark 4.3.1).

Of-course the number of ray-shooting iterations is an important parameter. In the current implementation, $\lceil c/2 \rceil$ iterations are performed. This value was heuristically determined by numerous simulation runs.

Although there is no guarantee that we find all, or even a significant part of the non-redundant half-spaces, the algorithm is simple enough to motivate its use. Note that the algorithm is most efficient when the fraction of redundant constraints is low.

### 4.3.2 Detecting Redundant Half-Spaces

By detecting redundant half-spaces, we not only reduce the number of LPs that have to be solved in (4.19), but we also reduce the size of these LPs, since the corresponding constraints can be removed.

Detecting redundant constraints in LPs is a standard problem, and is done in most LP solvers during a pre-solve analysis of the problem. The key idea in pre-solve algorithms is to exploit variable bounds $L \leq x \leq U$ to detect obviously redundant constraints [Gon97].

To detect if $h_i^T x \leq k_i$, $h_i = \begin{bmatrix} h_{i1} & h_{i2} & \dots & h_{in} \end{bmatrix}$ is redundant, each term in $h_i^T x$ is individually maximized to obtain an upper bound on $h_i^T x$

$$\sum_{i=1}^{n} h_i x_i \leq \sum_{j \in \{j : h_{ij} > 0\}} h_{ij} U_j + \sum_{j \in \{j : h_{ij} < 0\}} h_{ij} L_j \tag{4.20}$$

If the right-hand side of (4.20) is less than $k_i$, the constraint is redundant and can be removed. Hence, the set of redundant constraints detected in the pre-solve analysis is defined by

$$\mathcal{J}_R = \left\{ i \in \{1, \dots, c\} \mid \sum_{j \in \{j : h_{ij} > 0\}} h_{ij} U_j + \sum_{j \in \{j : h_{ij} < 0\}} h_{ij} L_j < k_i \right\} \tag{4.21}$$

Tight variable bounds $L$ and $U$ are crucial for this pre-solve algorithm to be efficient. In a pre-solver used in an LP solver, crude bounds are typically given by a priori knowledge, and by applying a more advanced pre-solve algorithm iteratively, the bounds can in some cases be improved upon by inferring more information from the constraints.

The standard pre-solve analysis that is applied before solving an LP is required to be cheap in order to actually yield runtime benefits, since the LP itself can be solved efficiently. In contrast, we are here solving a total of $c$ LPs for polytope reduction, where $c$ denotes the total number of constraints, i.e., the number of rows of $H$. Hence, we can spend a lot more effort on a pre-solve analysis since it benefits each of the $c$ LPs.

Since tight lower and upper bounds are crucial for the detection of redundant constraints using (4.20), we solve $2n$ LPs ($x \in \mathbb{R}^n$) to derive exact lower and upper bounds on $x$ in the polytope $Hx \leq K$. Specifically we solve the following LP for all $i \in \{1, \ldots, n\}$:

$$\min_{x} \ \pm x_{(i)} \tag{4.22a}$$
$$s.t. \quad Hx \leq K \tag{4.22b}$$

where $x_{(i)}$ denotes the $i$-th element of the vector $x \in \mathbb{R}^n$. Of-course, spending the effort of solving $2n$ LPs to find the bounding box of a polytope, to be used in the possibly inefficient algorithm (4.20), is only reasonable if the expected number of detected redundant constraints is large and $n$ is sufficiently small compared to $c$. This is generally the case if multi-parametric programming is used in the context of controller computation. An illustration of the bounding box computation is given in Figure 4.4.

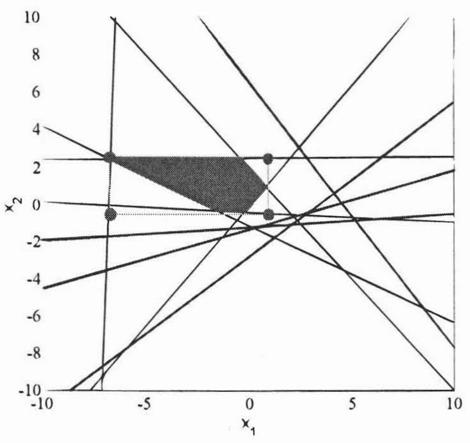


Figure 4.4: Illustration of a bounding box. If the bounding box does not intersect a hyper-plane, it is redundant.

## 4.3.3 Complete Algorithm

Putting the two parts together, we obtain the reduction algorithm.

**Algorithm 4.3.2 (Efficient Polytope Reduction)**

1. *Calculate upper and lower bounds using (4.22)*
2. *Apply (4.21) to remove redundant constraints $\mathcal{J}_R$.*
3. *Compute the Chebychev ball to find interior points and a subset $\mathcal{J}_C$ of non-redundant constraints*
4. *Find a subset $\mathcal{J}_N$ of the non-redundant constraints using ray-shooting on the constraints $\mathcal{J}\ /\mathcal{J}_R$.*
5. *Check redundancy of remaining unresolved constraints $h_i^T x \leq k_i$, $\forall i \in \mathcal{J}\ /(\mathcal{J}_R \cup \mathcal{J}_C \cup \mathcal{J}_N)$ by solving the LP (4.19).*

Observe that the ray-shooting algorithm is efficient for polytopes with few redundant constraints, while the bounding box method is most useful for polytopes with many easily detected redundant constraints. Hence, the two pre-analysis algorithms together cover many levels of redundancy.

The expected computational gains from the two pre-solve steps can be estimated if we take the computational complexity of solving an LP into account. A rough complexity analysis of a modern interior-point algorithm to solve an LP with $n$ variables and $s$ constraints would typically give $O((n^3 + n^2 s)\sqrt{s})$ operations[2] [dH94]. Hence, a polytope reduction algorithm, solving $s$ LPs, will have super-quadratic complexity with respect to the number of constraints in the original polytope. Consequently, the effect of removing redundant constraints by using the bounding box approach will be super-quadratic, i.e. removing half of the constraints will reduce the computational effort by more than a factor of four. The impact of the ray-shooting scheme on the total runtime will however only be linear, since the size of the remaining LPs is unaffected, only the number of LPs is reduced.

## 4.3.4 Other Usage of Bounding Boxes

The outer box approximations defined by (4.22) can be efficiently used in many problems arising in fields of reachability analysis for hybrid systems, approximate projec-

[2]The main computational burden in each interior-point iteration is the calculation of the Schur-matrix $H^T D H$ and factorization of this matrix ($D$ is a diagonal matrix which depends on the particular algorithm). Creating the Schur-matrix requires $O(n^2 s)$ operations and the factorization $O(n^3)$. Additional computations also have linear complexity in $s$. The number of iterations can be bounded by $O((n^3 + n^2 s)\sqrt{s})$, but is typically between 5 and 50.

tions and computation of explicit control laws for hybrid systems.

For instance in reach-set computation for hybrid systems [Tor03], bounding boxes can be used to decrease memory requirements by keeping only two extreme points of a bounding box instead of storing the complete half-space representation of a polytope $Hx \leq K$. This is mainly important because of the explosion of the number of polytopes at each step of the iterative exploration procedure.

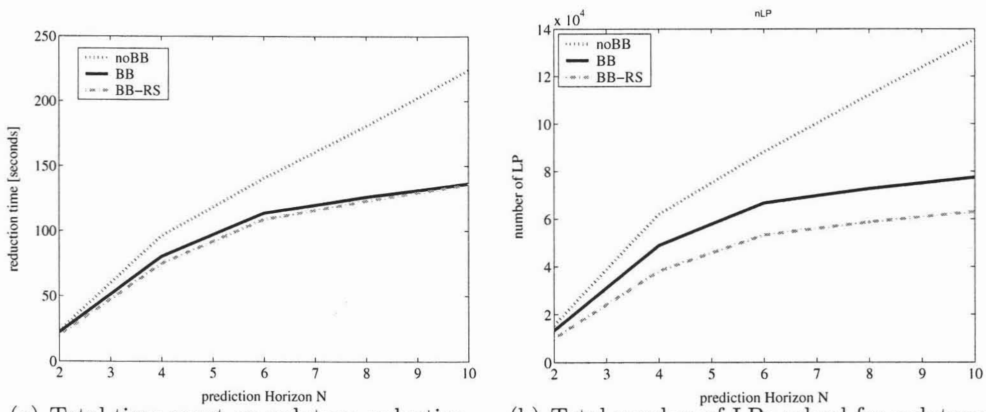
As already indicated, bounding boxes can be effectively used in the area of multi-parametric programming for PWA systems. Optimal control problems for PWA systems are generally solved in a dynamic programming fashion [BCM03a, BCM03b, KM02]. At each step of the dynamic program, the cost expression associated with a polytope over which the control law is defined needs to be compared to the cost of each other region which intersects the first one. To avoid unnecessary computation, it is useful to detect any possible intersections before further processing. This feature is also relevant in the context of stability analysis of PWA systems, since answering the question if two boxes intersect reduces to a simple set of IF-THEN statements. Despite the over-approximation nature of bounding boxes this method performs very well in practice.

Furthermore, search tree structures can be created more efficiently using the box approximations of polytopes [GTM04, TJB03a]. In Chapter 9 we will present a novel search algorithm which is entirely based on bounding boxes. Since the optimizer $U_N^*(x)$ is piecewise-affine over a polyhedral partition $\bigcup_{r\in\mathcal{R}} \mathcal{P}_r$, the procedure to obtain the control action for a given state $x$ reduces to a simple membership test. Without a search tree, one would need to check every region $\mathcal{P}_r$, $r \in \mathcal{R}$, which could be expensive when the number of regions becomes very large. In such search trees, each node of the tree consists of a hyperplane and a list of regions which satisfy this inequality and a list of regions which do not. Bounding boxes are a very effective tool in deciding to which list a region belongs to. Again, the speedup results from the fact that such an evaluation has to be performed only on two extreme points of the box, without the need to compute extreme points of the original polytope, which requires the solution to an LP. Hence, the construction of such search trees can be speeded-up significantly by the use of bounding boxes.

The computation of outer box approximations (4.22) in the algorithm described in the previous section is therefore not a one-purpose operation. The boxes can be stored along with the original polytope to significantly speed up subsequent operations, some of which were mentioned in this section.

### 4.3.5 Numerical Examples

The computational improvements of the proposed pre-solve approach depend strongly on the multi-parametric problem being solved. To find a general trend for problems typically solved using multi-parametric techniques, we have investigated the mp-QP solution to the Constrained Finite-Time Optimal Control problem (cf. Chapter 5) for 10 random stable linear systems with $n = 3$ states and $m = 2$ inputs. The respective control problems were solved for prediction horizons $N = 2, 4, 6, 8$ and 10, where the size of the horizon directly correlates with the size of the investigated multi-parametric quadratic program. Averaged results for the proposed polytope reduction algorithms are depicted in Figure 4.5. It should be noted that the polytope reduction contributed by roughly 60% to the overall runtime of the investigated mp-QP algorithm.



(a) Total time spent on polytope reduction. (b) Total number of LPs solved for polytope reduction.

Figure 4.5: Comparison of average time spent and average number of LPs solved for various polytope reduction schemes (noBB : standard polytope reduction, BB : polytope reduction using bounding boxes, BB-RS : polytope reduction using bounding boxes and ray-shooting)

The experiments indicate that the impact of efficient polytope reduction is increasing with the prediction horizon $N$. This was to be expected from the construction of the controller regions in 4.16-4.17, i.e. as $N$ increases, the number of initial half-spaces grows. On the other hand it has been observed that, in general, the number of half-spaces defining the controller regions grows sub-linearly. Therefore, the fraction of redundant half-spaces grows with increasing prediction horizon. As we described earlier, the computational efficiency of the bounding box approach grows quadratically with respect to the fraction of detected redundant constraints, so an improved performance for longer horizons is to be expected.

The impact of ray-shooting is less impressive. In the multi-parametric application, most half-spaces are redundant, hence few non-redundant half-spaces will be found. The number of solved LPs is decreased by the ray-shooting, but the cost to find the small number of non-redundant half-spaces is comparable to the cost of solving the additional LPs, at least with the current implementation of the ray-shooting algorithm.

# 5

# Optimal Control for Linear Time-Invariant Systems

Consider optimal control problems for discrete-time linear, time-invariant (LTI) systems

$$x(k+1) = Ax(k) + Bu(k), \tag{5.1}$$

with $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{m\times n}$. Let $x(k)$ denote the measured state at time $k$ and $x_k$ denote the predicted state at $k$ steps ahead, given the state $x(0)$. Let $u_k$ be the predicted input $k$ steps ahead, given $x(0)$. In this chapter we will give a brief overview of optimal control problems for LTI systems discussed in the literature.

## 5.1 Constrained Finite-Time Optimal Control (CFTOC)

Assume now that the states and the inputs of system (5.1) are subject to the following constraints

$$x(k) \in \mathbb{X} \subseteq \mathbb{R}^n, \qquad u(k) \in \mathbb{U} \subseteq \mathbb{R}^m, \qquad k \in \{0, \ldots, N\}, \tag{5.2}$$

where $\mathbb{X}$ and $\mathbb{U}$ are polyhedral sets containing the origin in their interior[1]. Now consider the constrained finite-time optimal control problem

$$J_N^*(x(0)) \quad = \quad \min_{u_0,\ldots,u_{N-1}} \left\{ \sum_{k=0}^{N-1} \left(u_k^T Q_u u_k + x_k^T Q_x x_k\right) + x_N^T Q_{x_N} x_N \right\} \tag{5.3a}$$

[1]The extension to mixed constraints $C^x x + C^u u \leq C^o$ is straightforward and omitted here.

$$\text{subj. to} \quad x_k \in \mathbb{X},\ u_{k1} \in \mathbb{U}, \qquad \forall k \in \{1,\dots,N-1\}, \tag{5.3b}$$

$$x_N \in \mathcal{T}_{\text{set}}, \tag{5.3c}$$

$$x_{k+1} = Ax_k + Bu_k, \quad x_0 = x(0), \tag{5.3d}$$

$$Q_u \succ 0, \quad Q_x \succeq 0, \quad Q_{x_N} \succeq 0. \tag{5.3e}$$

The terminal set constraint (5.3c) is an additional constraint which is often added to obtain certain properties (i.e. stability and constraint satisfaction. Henceforth, we will assume the terminal weight matrix $Q_{x_N}$ to be equal to the ARE matrix $P$ given by the solution of a corresponding Riccati equation. The solution to problem (5.3) has been studied in [BMDP02]. We will briefly summarize the main results. By substituting $x_k = A^k x(0) + \sum_{j=0}^{k-1} A^j B u_{k-1-j}$, problem (5.3) can be reformulated as a quadratic program (QP), i.e.

$$J_N^*(x(0)) = x(0)^T Y x(0) + \min_{U_N} \left\{ U_N^T H U_N + x(0)^T F U_N \right\} \tag{5.4a}$$

$$\text{subj. to} \quad GU_N \leq W + Ex(0), \tag{5.4b}$$

$$H \succ 0, \tag{5.4c}$$

where the column vector $U_N \triangleq [u_0^T, \dots, u_{N-1}^T]^T \in \mathbb{R}^{Nm}$ is the optimization vector and $H$, $F$, $Y$, $G$, $W$, $E$ are easily obtained from $Q_x$, $Q_u$, $Q_{x_N}$, the system (5.1) and the constraints (5.2) (see [Mac02] for details[2]).

**Remark 5.1.1** *The constraints $Q_u \succ 0$, $Q_x \succeq 0$ and $Q_{x_N} \succeq 0$ are imposed in* (5.3), *in order to guarantee that $H \succ 0$ in* (5.4).

The optimizer of (5.4) will henceforth be denoted by $U_N^*(x)$. It follows from Theorem 4.2.1 that $U_N^*(x)$ is a PWA function of the state $x$, which we can obtain by solving problem 5.4 as an mp-QP (see Chapter 4 for details).

If the objective function in (5.3) is linear, i.e.

$$J_N^*(x(0)) = \min_{u_0,\dots,u_{N-1}} \sum_{k=0}^{N-1} \left( ||Q_u u_k||_p + ||Q_x x_k||_p \right) + ||Q_{x_N} x_N||_p, \tag{5.5a}$$

$$\text{subj. to,} \quad x_k \in \mathbb{X},\ u_{k-1} \in \mathbb{U}, \qquad \forall k \in \{1,\dots,N\}, \tag{5.5b}$$

$$x_N \in \mathcal{T}_{\text{set}}, \tag{5.5c}$$

$$x_{k+1} = Ax_k + Bu_k, \quad x_0 = x(0), \tag{5.5d}$$

[2]For example, $Y = (A^N)^T Q_{x_N} A^N + \sum_{k=0}^{N-1} (A^k)^T Q_x A^k$.

where $||\cdot||_p$ denotes some linear vector norm (either the 1- or the $\infty$- norm), then the problem, with the optimizer $U_N = [u_0^T, \dots, u_{N-1}^T]^T$, can be recast as an LP [BBM00a, Bor03] by substituting $x_k = A^k x(0) + \sum_{j=0}^{k-1} A^k B u_{k-1-j}$. For instance, in the case of the $||\cdot||_\infty$ norm, we have

$$\begin{aligned}
J_N^*(x(0)) \quad &= \min_{\substack{U_N,\ \epsilon_0,\dots,\epsilon_{N-1},\\ \delta_0,\dots,\delta_{N-1},\gamma}} \sum_{k=0}^{N-1} \left(\epsilon_k + \delta_k\right) + \gamma && (5.6a)\\
\text{subj. to} \quad & GU_N \leq W + Ex(0), && (5.6b)\\
& Q_u u_k \leq \mathbf{1}\epsilon_k, \quad -Q_u u_k \leq \mathbf{1}\epsilon_k, \quad k = 0,\dots,N-1 && (5.6c)\\
& Q_x x_k \leq \mathbf{1}\delta_k, \quad -Q_x x_k \leq \mathbf{1}\delta_k, \quad k = 0,\dots,N-1 && (5.6d)\\
& Q_{x_N} x_N \leq \mathbf{1}\gamma, \quad -Q_{x_N} x_N \leq \mathbf{1}\gamma. && (5.6e)
\end{aligned}$$

Constraint (5.6b) corresponds to (5.2) and constraints (5.6c)-(5.6e) are used to describe the linear objective function. It follows from Theorem 4.2.3 that the optimizer $U_N^*(x)$ is a PWA function of the state $x$, which we can be obtained by solving problem (5.6) as an mp-LP (see Chapter 4 for details).

## 5.2 Receding Horizon Control

If the model predictive control problems (5.3) and (5.5), which are formulated over a finite prediction horizon $N$, are solved on-line for a particular initial condition $x_0 = x(0)$, one obtains the vector $U_N^*$ of optimal control moves which can be applied to the system in an open-loop fashion to move the system states from $x(0)$ to $x(N)$. If $x(N)$ is not equal to the desired set-point, the problem has to be re-solved for a new value of the initial condition $x_0 = x(N+1)$ in order to obtain a new sequence of control actions. However, there is no guarantee that the problem remains feasible at this point. To overcome this limitation, one can extend the prediction horizon $N$ to infinity, which leads to so-called infinite-time optimal control problems [GBTM04]. However, the infinite-time optimal control problems are often too complex to be computationally tractable. Therefore it has become a common practice to approximate the infinite-time solution by solving a sequence of finite time optimal control problems, a strategy which is commonly referred to as *Receding Horizon Control* (RHC). Another reason for applying the RHC strategy is the fact the predicted system behavior, represented by the prediction model based on which the optimal input trajectory $U_N^*$ is calculated, usually differs from the actual behavior of the plant. Due to this model-plant mismatch, just applying the whole open-loop sequence could lead either to sub-optimal performance or, in the worst case, even to violation of system constraints.

For a more detailed discussion of RHC, we refer the reader to the review paper [MRRS00]. For in-depth insights, we recommend the publications [CMT87, Mac02, LÖ3]. The RHC policy has become standard practice in modern control applications and besides numerous PhD theses [Mig02, Bor03, Ker00, Tøn00] and survey papers [QB97, MRRS00, BM99b, ABQ+99, GPM89, May01, ML99], several textbooks [Mac02, Ros03, CB99, KC01] have been published on this topic.

The RHC policy is based on solving finite-time optimal control problems at each time step to obtain the optimal input sequence $U_N^*$. Subsequently, only the first element of that sequence is applied to the system. At the next time step, the state is measured again and the procedure is repeated from the beginning for the updated value of the initial condition $x_0$. In the sequel by Receding Horizon Control we will denote any strategy which is based on finite horizon control problems implemented in a receding horizon fashion. Therefore, RHC applies to model predictive control, where the corresponding optimization problem is solved on-line (see Fig. 5.1), but also applies to cases where the solution to an optimal control problem was obtained by using techniques of parametric programming as described in Chapter 4, as depicted in Figure 5.2.

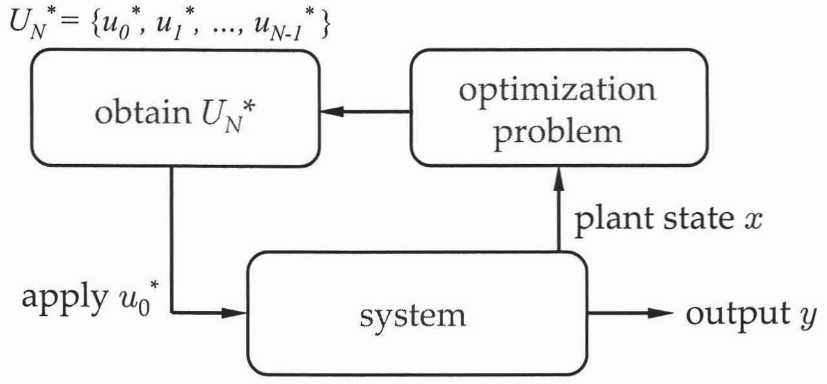


Figure 5.1: The MPC scheme solved on-line.

However, a mere repetition of the optimization at each step, on its own, is not enough to guarantee feasibility for all time. To attain such goal, the terminal set applied in (5.3c) has to be chosen appropriately. Therefore in Chapter 7 we present

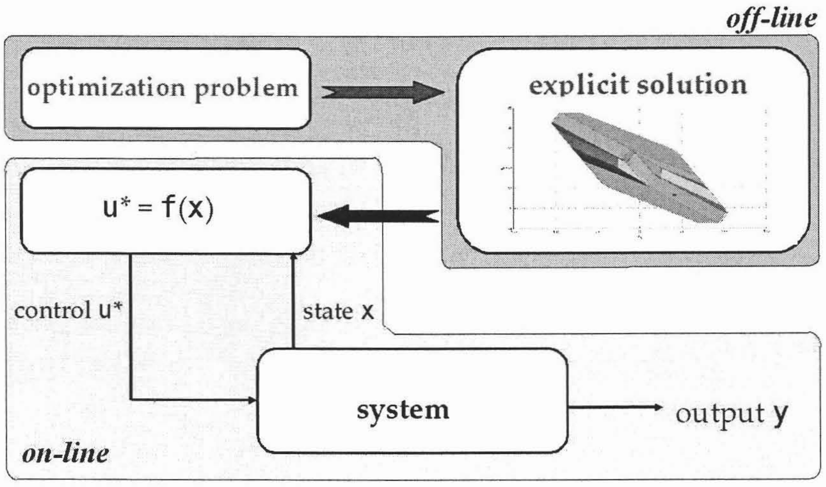


Figure 5.2: The RHC scheme based on a parametric solution.

a procedure which can be used to calculate target sets for generic PWA systems such that feasibility guarantees are maintained.

If the optimization problem is solved parametrically as an explicit function of the initial condition $x_0$, the optimal feedback law $u^* = f(x_0)$ takes a form of a look-up table. The on-line implementation of such table then reduces to a simple set-membership test, also known as the *point location* problem. Here, the table has to be searched through and the element which contains the current state measurement has to be found. Even though such search can be performed faster compared to solving the corresponding optimization problem on-line, the complexity of the table still limits the minimal admissible sampling time of the closed-loop system. Therefore in Chapter 8 we describe techniques which yield look-up tables with relatively small number of elements. In addition we also outline a novel algorithm which serves to speed up the search through the tables in Chapter 9.

# Part II

# EFFICIENT CONTROL OF PWA SYSTEMS

# 6

# Problem Description

## 6.1 Introduction

This part of the thesis will address the topic of feedback control of discrete-time, time-invariant, piecewise affine (PWA), systems subject to constraints. Optimal control of PWA systems has garnered increasing interest in the research community since this system type represents a powerful tool for approximating non-linear systems and because of its equivalence to many classes of hybrid systems [Tor03, HDB01, Son96, Son81]. The optimal control inputs for PWA systems may be obtained by solving mixed-integer optimization problems on-line [BM99a, MR03], or as was shown in [BCM03b, BBBM03, KM02, Bor03, DPar], by solving a number of multi-parametric programs off-line. Additional methods for controlling hybrid systems are reported in [LR03, MR03, KA02, MR02, BZ00, LTS99, TLS00].

In their pioneering work [BMDP02] the authors show how to formulate an optimal control problem for constrained linear discrete-time systems as a multi-parametric program (by treating the state vector as a parameter) and how to solve such a program (see Chapter 4). Basic ideas from [BMDP02] for linear systems were extended to PWA systems in [BCM03b,BBBM03,KM02,Bor03]. The associated solution (optimal control inputs) takes the form of a PWA state feedback law. If the control objective is linear, the state-space is partitioned into polyhedral sets and for each of these sets the optimal control law is given as an affine function of the state. For quadratic objectives the state space partition is not polyhedral, in general [BBBM03].

In the on-line implementation of these explicit controllers, input computation reduces to a simple set-membership test. Even though the approaches in [BCM03b, BBBM03, KM02, Bor03] rely on off-line computation of a feedback law, the computation quickly becomes prohibitive for larger problems. This is not only due to the high complexity of the multi-parametric programs involved [GM03, BMDP02], but mainly because of the large number of multi-parametric programs which need to be solved when a controller is computed in a dynamic programming fashion [BBBM03, KM02].

In addition, there are few results in the literature which explicitly address the issue of computing feedback controllers which provide stability guarantees. The few publications which address this issue (e.g., [MR03]) assume that the origin is contained in the interior of one unique dynamics or rely on end-point constraints (e.g., [BBM00b]). The only exception is the infinite horizon solution proposed in [BCM03a], which is computationally tractable for small problems only.

In the following chapters we therefore address the two main aspects of construction and implementation of control policies for PWA systems which guarantee closed-loop stability. Specifically, we first present a procedure to calculate a stabilizing terminal set along with the associated piecewise linear feedback law for PWA system in Chapter 7. The results allow one first to find the feedback law which guarantees exponential stability via solving a semidefinite program. Subsequently, an algorithm to calculate an associated maximal invariant set is presented. Such sets are then used in Chapter 8 to design feedback controllers which guarantee all-time feasibility and exponential stability. First two algorithms are based on the principle of steering all system states into a stabilizing target set in a minimum-time fashion. The schemes are shown to lead controllers of "lower" complexity compared to optimal controllers of e.g. [BCM03a, BBBM03] and features a "fast" construction of the control policy. These results are further extended in Section 8.3 where we show that if constraint satisfaction is dealt with independently of the stability analysis, controllers of even lower complexity can be obtained. Finally, in Chapter 9 we present an algorithm which serves to speed up the task of on-line implementation of parametric controllers. The procedure is based on utilizing bounding boxes already described in Section 4.3 to construct an interval search tree, which speeds up the task of region identification.

## 6.2 Background and Definitions

A detailed overview of multi-parametric programming principles is given in Chapter 4. Here we will give a basic introduction to RHC of PWA systems

Consider a discrete-time linear time-invariant system

$$x(k+1) \;=\; Ax(k) + Bu(k) \tag{6.1}$$

with $A \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{n \times m}$. Let $x(k)$ denote the measured state at time $k$ and $x_k$ ($u_k$) denote the predicted state (input) at time $k$, given $x(0)$. Assume now that the states and the inputs of the system in (6.1) are subject to the following constraints

$$x(k) \in \mathbb{X} \subset \mathbb{R}^n, \qquad u(k) \in \mathbb{U} \subset \mathbb{R}^m, \quad \forall k \geq 0, \tag{6.2}$$

where $\mathbb{X}$ and $\mathbb{U}$ are polytopic sets containing the origin in their interior.

**Remark 6.2.1** *For ease of notation, we restrict ourselves to separate constraints on state and input in* (6.2). *It is straightforward to modify all algorithms in this chapter to deal with systems subject to mixed constraints, i.e.* $C^x x(k) + C^u u(k) \leq C^c$, $\forall k \geq 0$.

Consider the constrained finite-time optimal control problem with a linear objective

$$J_N^*(x(0)) = \min_{u_0,\ldots,u_{N-1}} \sum_{k=0}^{N-1} \left( ||Q_u u_k||_{1,\infty} + ||Q_x x_k||_{1,\infty} \right) + ||Q_{x_N} x_N||_{1,\infty}, \tag{6.3a}$$

$$\text{subj. to,} \quad x_k \in \mathbb{X},\ u_k \in \mathbb{U}, \qquad \forall k \in \{0,\ldots,N-1\}, \tag{6.3b}$$

$$x_N \in \mathcal{T}_{\text{set}}, \tag{6.3c}$$

$$x_{k+1} \;=\; Ax_k + Bu_k, \qquad x_0 = x(0), \tag{6.3d}$$

where (6.3c) is a user defined set-constraint on the final state and $||\cdot||_{1,\infty}$ denotes the 1- or $\infty$-norm of a vector, respectively.

**Definition 6.2.2 (Feasible Set $\mathcal{X}_N$)** *We define the $N$-step feasible set $\mathcal{X}_N \subseteq \mathbb{R}^n$ as the set of initial states $x(0)$ for which the optimal control problem (6.3) is feasible, i.e.*

$$\begin{aligned}\mathcal{X}_N =&\{x(0) \in \mathbb{R}^n | \exists U_N = [u_0^T,\ldots,u_{N-1}^T]^T, \ \ x_{k+1} = Ax_k + Bu_k,\\ &x_k \in \mathbb{X},\ x_N \in \mathcal{T}_{set},\ u_k \in \mathbb{U},\ \forall k \in \{0,\ldots,N-1\}\}.\end{aligned}$$

where $U_N \in \mathbb{R}^{Nm}$ is the optimization vector. By considering $x(0)$ as a parameter, problem (6.3) can be stated as an mp-LP [BBM00a] which can be solved to obtain a feedback solution with the following properties (derived from [Bor03, Gal95]):

**Theorem 6.2.3** *Consider the finite time constrained regulation problem (6.3), with a linear objective in* (6.3a). *Then, the set of feasible parameters $\mathcal{X}_N$ is convex, there* *exists an optimizer $U_N^* : \mathcal{X}_N \to \mathbb{R}^{Nm}$ which is continuous and piecewise affine (PWA), i.e.*

$$U_N^*(x(0)) = F_r x(0) + G_r, \text{ if } x(0) \in \mathcal{P}_r = \{x \in \mathbb{R}^n | H_r x \leq K_r\}, \; r = 1, \ldots, R$$

*and the value function $J_N^* : \mathcal{X}_N \to \mathbb{R}$ is continuous, convex and piecewise affine.*

According to Theorem 6.2.3, the feasible state space $\mathcal{X}_N$ is partitioned into $R$ polytopic regions, i.e., $\mathcal{X}_N = \bigcup_{r=1,\ldots,R} \mathcal{P}_r$.

It was shown in [BBBM03, KM02, BBM00a] how to compute the optimal explicit feedback controller for PWA systems of the form

$$\begin{aligned} x(k+1) &= f_{\text{PWA}}(x(k), u(k)) = A_i x(k) + B_i u(k) + f_i, && (6.4a) \\ &\text{if } [x(k)^T \; u(k)^T]^T \in \mathcal{D}_i, \quad i \in \mathcal{I}, && (6.4b) \end{aligned}$$

where $x \in \mathbb{R}^n$ is the state vector, $u \in \mathbb{R}^m$ is the control vector and $\{\mathcal{D}_i\}_{i=1}^D$ is a bounded polyhedral partition of $(x, u) \subset \mathbb{R}^{n+m}$ space. For simplicity, the sets $\mathcal{D}_i$ here define both regions in which a particular state update equation is valid as well as the constraints on the state and input variables. The set $\mathcal{I}$ is defined as $\mathcal{I} \triangleq \{1, 2, \ldots, D\}$ where $D$ denotes the number of different dynamics. We will henceforth assume that the sets $\mathcal{D}_i$ are non-intersecting.
Henceforth, we will abbreviate (6.4a) and (6.4b) with $x(k+1) = f_{PWA}(x(k), u(k))$. The optimization problem considered here is thus given by

$$\begin{aligned} J_N^*(x) = \min_{u_0,\ldots,u_{N-1}} & \sum_{k=0}^{N-1} \left( ||Q_u u_k||_p + ||Q_x x_k||_p \right) + ||Q_{x_N} x_N||_p, && (6.5a) \\ \text{subj. to} \quad & x_N \in \mathcal{T}_{\text{set}}, && (6.5b) \\ & x_{k+1} = f_{PWA}(x_k, u_k), \; x_0 = x, && (6.5c) \end{aligned}$$

using either the standard squared Euclidean norm ($p = 2$) or linear norms ($p = 1$ and $p = \infty$).

In the case of linear norms, [BCM03b, KM02] suggest to solve multi-parametric Linear Programs (mp-LP) in a dynamic programming fashion to obtain the feedback solution to (6.5). It was shown that the resulting feedback law is piecewise affine over polyhedra. In [BBBM03], the feedback solution to (6.5) with a quadratic objective in (6.5a) was computed by solving a sequence of multi-parametric Quadratic Programs (mp-QP) in a dynamic programming fashion. It was shown that the resulting feedback law is piecewise affine over (possibly) non-convex sets bounded by quadratic surfaces. Various additional methods to obtain explicit feedback solutions to linear or quadratic optimization problems for PWA systems are given in [Bor03, MR03, BCM03a, KM02].

Consider now an autonomous PWA system given by

$$x(k+1) = f_a(x(k)) = \widetilde{A}_r x(k) + \widetilde{g}_r, \qquad \text{if } x(k) \in \mathcal{D}_r, \tag{6.6}$$

where the currently active dynamic $r$ is defined by the polytope $\mathcal{D}_r$. The system (6.6) can be obtained from (6.4) by assuming that the control variable $u(k)$ is driven by the expression $u(k) = F_r x(k) + G_r$ if the state $x(k)$ resides in the region $\mathcal{D}_r$. The remaining definition is derived from [Ker00].

**Definition 6.2.4 (Maximal Positively Invariant Set $\mathcal{O}_\infty^{\mathbf{PWA}}$)** *The maximal positively invariant set $\mathcal{O}_\infty^{PWA}$, for the discrete time system in* (6.6) *$(x(k+1) = f_a(x(k)))$ subject to the constraints in* (6.2) *$(x(k) \in \mathbb{X},\ \forall k \geq 0)$ is defined by*

$$\mathcal{O}_\infty^{PWA} \triangleq \{x(0) \in \mathbb{X} \mid \phi(k;x) \in \mathbb{X}, k \in \mathbb{N}_+\}, \tag{6.7}$$

*where $\phi(k;x_0)$ denotes the solution of $x(k+1) = f_a(x(k))$ at time $k$ if the initial state is $x(0)$.*

As already mentioned in Chapter 5.2, the concept of set invariance is important in order to derive MPC policy which guarantees infinite-time feasibility.

# 7

# Construction of Stabilizing Controllers for Piecewise Affine Systems

A large part of the literature has focused on end-point constraints to guarantee asymptotic stability of the closed-loop system (e.g., [Bor03, BBM00b, BM99a]). This type of constraint generally requires the use of large prediction horizons for the controller to cover the maximal positively invariant set, such that the computational complexity quickly becomes prohibitive. Other methods (e.g., [MR03]) only provide stability guarantees if the origin is contained in the interior of one of the dynamics $\mathcal{D}_i$. In this section, a method is presented for obtaining stabilizing controllers for generic PWA systems. The results in this section are derived from [GKBM05][1].

For any dynamical system, stability is guaranteed if an invariant set is imposed as a terminal state constraint in (6.5b) and the terminal cost in (6.5a) corresponds to a Lyapunov function for that set. In addition, the decay rate of the 'terminal Lyapunov function' must be bounded from above by the stage cost. Here we show how to compute a control invariant set $\mathcal{O}_{\infty}^{\text{PWA}}$ (6.7) with the associated Lyapunov function such that stability and constraint satisfaction of RHC is guaranteed. The scheme is based on the results in [MFTM00, RGK$^{+}$04] and was first published in [GKBM04].

**Remark 7.1.5** *We cover here the case where the origin is located on the boundary of multiple dynamics $\mathcal{D}_i$. The case where the equilibrium point is located on the boundary of multiple dynamics is by no means a pathologically rare case. Many physical systems exhibit a change in their dynamic behavior when certain states change their sign.*

[1]Note that identical results were simultaneously obtained by others in [LHWB04].

The computation scheme is based on the assumption that the origin is an equilibrium state of the PWA system and hence the closed loop dynamics $f_i = 0$, $\forall i \in \mathcal{I}_0$ (see (6.4)). If this assumption is not satisfied, the approach proposed here will fail.

We will now show how the terminal set $\mathcal{T}_{\text{set}}$ and cost $Q_{x_N}$ can be computed such that stabilizing RHC controllers can be constructed for generic PWA systems. In the first step, we stabilize each dynamics of the PWA system which contains the origin by a linear state feedback controller $F_i$ of the form $u = F_i x$ if $x \in \mathcal{D}_i$. We denote by $\mathcal{I}_0$ the set of indices of dynamics which contain the origin in their respective interiors, i.e.

$$\mathcal{I}_0 \triangleq \{i \in \mathcal{I} \mid 0 \in \mathcal{D}_i\}.$$

Then the search for stabilizing linear feedback controllers $F_i$ and an associated common quadratic Lyapunov function $V(x) = x^T P x$ can be posed as

$$x^T P x \geq 0, \qquad \forall x \in \mathbb{X},$$
$$x^T(A_i + B_i F_i)^T P (A_i + B_i F_i) x - x^T P x \leq -x^T Q_x x - x^T F_i^T Q_u F_i x, \quad \forall x \in \mathcal{D}_i, \forall i \in \mathcal{I}_0.$$

If we relax this condition by setting $\mathcal{D}_i = \mathbb{R}^n$, $\forall i \in \mathcal{I}_0$, the problem can be rewritten as an SDP by using Schur complements and introducing the new variables $Y_i = F_i Z$ and $Z = \frac{1}{\gamma} P^{-1}$ (see [BGFB94, KBM96, MFTM00] for details),

$$\min_{Y_i, Z, \gamma} \gamma, \quad \text{subj. to,} \tag{7.1a}$$

$$Z \succ 0, \tag{7.1b}$$

$$\begin{bmatrix} Z & (A_i Z + B_i Y_i) & (Q_x^{0.5} Z)^T & (Q_u^{0.5} Y_i)^T \\ (A_i Z + B_i Y_i)^T & Z & 0 & 0 \\ (Q_x^{0.5} Z) & 0 & \gamma I & 0 \\ (Q_u^{0.5} Y_i) & 0 & 0 & \gamma I \end{bmatrix} \succeq 0, \quad \forall i \in \mathcal{I}_0. \tag{7.1c}$$

where the scalar $\gamma$ is introduced to optimize for the worst case performance, whereby the 'worst case' corresponds to an arbitrary switching sequence. Note that it may not be possible for the worst case switching sequence considered in (7.1) to occur in practice due to the relaxation $\mathcal{D}_i = \mathbb{R}^n$, since in the original PWA model (6.4) not all dynamics $i$ are defined over the entire state space.

**Remark 7.1.6** *If (7.1) is posed for an LTI system (i.e. $\mathcal{I}_0 = 1$), the optimal LQR state feedback solution $K$ and the solution to the Algebraic Riccati Equation $P$ is recovered.*

In a second step, the maximal positively invariant set $\mathcal{O}_\infty^{\text{PWA}}$ (6.7) of the PWA system subject to the feedback controllers $F_i$ can be computed with the algorithm in [RGK+04], which is guaranteed to terminate in finite time for the problem at hand, since the closed loop system is asymptotically stable.

The proposed computation scheme is summarized in the following algorithm:

**Algorithm 7.1.7 (Computation of Maximal Positively Invariant Set $\mathcal{O}_\infty^{\text{PWA}}$)**

1. *Identify all dynamics $i$ which contain the origin, i.e. $i \in \mathcal{I}_0 \triangleq \{i \in \mathbb{N}^+ \mid 0 \in \mathcal{D}_i\}$.*
2. *Solve* (7.1) *for all $i \in \mathcal{I}_0$, to obtain $F_i$ and $P$. If* (7.1) *is infeasible, abort the algorithm.*
3. *Compute the maximal positively invariant set $\mathcal{O}_\infty^{PWA}$ corresponding to the closed loop system $x^+ = (A_i + B_iF_i)x$, if $x \in \mathcal{D}_i$ with the method in [RGK+04].*
4. *Return the calculated set $\mathcal{O}_\infty^{PWA}$, the feedback laws $F_i$ and the associated matrix $P$.*

**Theorem 7.1.8 (Exponential Stability of RHC for PWA Systems, [GKBM04, GKBM05])** *Assume the optimization problem* (6.5) *is given with a quadratic objective, i.e.* (6.5a) *corresponds to*

$$J_N^*(x(0)) = \min_{u_0,\ldots,u_{N-1}} \sum_{k=0}^{N-1} \left(u_k^T Q_u u_k + x_k^T Q_x x_k\right) + x_N^T Q_{x_N} x_N,$$
$$Q_x \succeq 0, \quad Q_{x_N} \succeq 0, \quad Q_u \succ 0,$$

*the terminal set is $\mathcal{T}_{set} = \mathcal{O}_\infty^{PWA}$ and the terminal cost is $\mathcal{Q}_{x_N} = P$ (obtained with Algorithm 7.1.7). If this problem is solved at each time step for the PWA system* (6.4) *and only the first input is applied (Receding Horizon Control as described in Chapter 5.2), then the closed loop system is exponentially stable.*

**Proof** Algorithm 7.1.7 trivially satisfies the conditions for exponential stability in [MRRS00, Section 3.3]. □

Note that we only need to consider a single convex terminal set for linear systems [GT91,MRRS00] whereas for PWA systems, the terminal set $\mathcal{O}_\infty^{\text{PWA}}$ is given as a union of several convex sets $\mathcal{O}_\infty^{\text{PWA}} = \bigcup \mathcal{X}_i$. If the union $\bigcup \mathcal{X}_i$ is convex, the regions can be merged with the method in [BFT01]. Convexity of the target set is a desirable property since simpler target sets $\mathcal{T}_{\text{set}}$ generally lead to reduced algorithm run-time and solution complexity for the type of optimization problem given in (6.5).

**Remark 7.1.9** *The procedure described in this section is merely sufficient for asymptotic stability. We cannot guarantee that the Lyapunov function and the associated state feedback laws will be found in the suggested manner. However, we have observed in an extensive case study that the approach works very well in practice. Short of the computationally very demanding construction of the infinite horizon solution proposed in [BCM03a], there is currently no alternative method for guaranteeing closed-loop stability for control of generic PWA systems.*

*Furthermore, the method we propose here can be easily used to calculate suitable target sets and terminal penalties which can be used in other controller computation schemes (e.g., [BBBM03, MR03, KM02, BCM03b]) to obtain controllers which guarantee closed-loop stability for PWA systems.*

# 8

# Low Complexity Feedback Control of Piecewise Affine Systems

This chapter will address the problem of deriving stabilizing control laws for the class of piecewise affine (PWA) systems. As outlined in Section 7, stability and feasibility guarantees can be obtained if a finite time optimal control problem is solved with a suitable choice of the terminal set and terminal cost. However, one of the key problems in solving such problems for PWA systems is the lack of convexity of the terminal sets. This fact, combined with the complexity of the objective function (6.3), gives rise to computational overhead when solving such optimal control problems either on-line, or using parametric programming techniques. To mitigate this problem, the so-called dynamic programming approach [BBBM03, KM02], based on iteratively solving the optimal control problem backwards in time, can be used. But even in this approach, the complexity of the cost function (affected mainly by the choice of the prediction horizon $N$) and the non-convex nature of terminal sets make it necessary, in the worst case, to explore an exponentially growing number of possible transitions during the iterations. The algorithms presented here avoid these issues to some extent by considering 'simpler' objective functions.

The first approach, presented in Section 8.1, is based on driving system states into a pre-specified target set in the minimal possible number of discrete time steps. In other words, instead of minimizing a weighted sum of state and input contributions in (6.3a) over a horizon $N$ (i.e. solving one problem of size $N$), we aim at minimizing the number of steps in which the system states enter a given terminal set. We will show that this goal can be achieved by solving a sequence of $N$ optimization problems with prediction horizon of 1. An advantage of this modification is that several sets generated at each iteration can be merged into larger parts, hence reducing the number of transitions which needs to be explored. This leads to a faster construction of the control law, but also reduces the complexity of the resulting feedback controller. Another advantage of this approach is that the set of states, for which the minimum-time problem is feasible, defines the maximal controllable set of states for a given PWA system.

The second approach, introduced in Section 8.2, is similar to the minimum-time approach described above in the sense that it also leads to controllers which drive the system states into a given target set in the least possible number of discrete steps. The addition here is that the control law is constructed in a way such that it tries to prevent the system states from changing switching to a different dynamics for as long as possible. A substantial advantage of this procedure, compared to the minimum-time approach, is that convexity of the terminal sets can be maintained for more than just for one step, which can lead to a substantial decrease of the runtime of the algorithm, as well as to a decrease in the complexity of the resulting control law.

The third approach, described in Section 8.3, is based on dealing with the issue of constraint satisfaction and asymptotic stability separately. In this scheme, in the sequel denoted as $M$-step control, the feedback law is obtained by solving the CFTOC problem (6.3) for a shorter prediction horizon $M$ (with $M < N$) with an additional target set constraint on the first predicted state (i.e. $x_1 \in \mathcal{T}_{\text{set}}$). We show that if $\mathcal{T}_{\text{set}}$ is equal to the set where the minimum-time problem was feasible, the $M$-step controller guarantees constraint satisfaction for all time. However, asymptotic stability of the closed-loop system is not enforced by construction. Instead, it has to be checked a-posteriori by looking for a suitable Lyapunov function. This function, if found, then certifies asymptotic stability. Because usually short prediction horizons are considered in this scheme (ideally $M = 1$), the main advantage of this approach is that the resulting feedback law is of low complexity. This is illustrated on an extensive comparison reported in Section 8.4. There we show that all introduced schemes yield controllers of much lower complexity compared to the traditional methods (e.g. [BCM03b, BBBM03, KM02]) that a whole new class of problems becomes tractable.

## 8.1 Minimum-Time Controller

A minimum-time controller aims at driving the system state $x(k)$ into a pre-specified target set in minimum number of time steps. Unlike the approaches in [BBBM03], the cost-to-go for the minimum-time controller assumes only integer values. Because of the 'simple' cost-to-go, the target sets which need to be considered at each iteration step are larger and fewer in number than those which would be obtained if an optimal controller with a different cost objective were to be computed [BBBM03, BCM03a]. Thus, both the complexity of the feedback law as well as the computation time are greatly reduced, in general.

A minimum-time controller computation scheme for PWA systems was first introduced in [KM02], using projection methods. Though giving general ideas about the computation concept and the character of the minimum-time solution, computational issues are not addressed in detail. A detailed algorithmic implementation of the minimum-time algorithm will be described in the following, using multi-parametric programming[1].

When the minimum-time algorithm terminates, the associated feedback controller will cover the $N$-step stabilizable set $\mathcal{K}_N^{\mathrm{PWA}}(\mathcal{O}_\infty^{\mathrm{PWA}})$.

**Definition 8.1.1 ($N$-step stabilizable set $\mathcal{K}_N^{\mathrm{PWA}}(\mathcal{O}_\infty^{\mathrm{PWA}})$)** *The set $\mathcal{K}_N^{PWA}(\mathcal{O}_\infty^{PWA})$ denotes the $N$-step stabilizable set for a PWA system (6.4), i.e., it contains all states which can be steered into $\mathcal{O}_\infty^{PWA}$ in $N$ steps. Specifically,*

$$\begin{aligned}\mathcal{K}_N^{PWA}(\mathcal{O}_\infty^{PWA}) = \{x(0) \in \mathbb{R}^n \mid \exists u(k) \in \mathbb{R}^m,\ s.t.\ x(N) \in \mathcal{O}_\infty^{PWA},\\ x(k+1) = f_{PWA}(x(k), u(k)),\ \forall k \in \{0, \dots, N-1\}\}.\end{aligned}$$

*Accordingly, the set $\mathcal{K}_\infty^{PWA}(\mathcal{O}_\infty^{PWA})$ denotes the maximal stabilizable set for $N \to \infty$.*

### 8.1.1 Minimum-Time Controller: Off-Line Computation

An algorithm for computing the minimum-time controller will be presented in this section. The computation scheme is based on solving a sequence of multi-parametric programs at each iteration step. The number of iterations corresponds to the number of time steps which are needed to reach the target set. At each iteration, a controller partition is computed which drives the state into the partition that was obtained in the previous iteration.

Before presenting the algorithm, some preliminaries will be introduced. Assume a P-collection $\mathcal{S}^0$ of $L^0$ polytopes $\mathcal{S}_l^0$, i.e. $\mathcal{S}^0 = \bigcup_{l\in\mathcal{L}^0} \mathcal{S}_l^0$, where $\mathcal{L}^0 \triangleq \{1, 2, \dots, L^0\}$. In

[1]Multi-parametric programming can be seen as a form of projection and thus the content of this section can be viewed as a special case of [KM02].

the following, the set $\mathcal{S}$ without subscript will be used to denote P-collections while the subscript is used to denote polytopes. All states which can be driven into the set $\mathcal{S}^0$ in one time step for the PWA system (6.4) are defined by:

$$\begin{aligned} Pre(\mathcal{S}^0) &= \{x \in \mathbb{R}^n \mid \exists u \in \mathbb{R}^m,\ f_{\text{PWA}}(x,u) \in \mathcal{S}^0\} \\ &= \bigcup_{i\in\mathcal{I}} \bigcup_{l\in\mathcal{L}^0} \left\{ x \in \mathbb{R}^n \mid \exists u \in \mathbb{R}^m,\ [x^T\ \ u^T]^T \in \mathcal{D}_i,\ A_i x + B_i u + f_i \in \mathcal{S}_l^0 \right\} \\ &= \bigcup_{j\in\mathcal{J}^0} \mathcal{X}_{1,j}. \end{aligned}$$

For the feasible set $\mathcal{X}_{1,j}$, the subindex 1 denotes that the set was obtained for a prediction horizon of 1 (see Definition 6.2.2). The second subindex, $j$, is used to access the different feasible sets which are obtained for various combinations of active dynamics and target sets. The index set $\mathcal{J}^0$ contains all valid values for $j$ in $\mathcal{X}_{1,j}$.

For a fixed $i$ and $l$, the target set $\mathcal{S}_l^0$ is convex and the dynamics affine, such that it is possible to apply standard multi-parametric programming techniques to compute the set of states which can be driven into $\mathcal{S}_l^0$ [BMDP02]. Therefore the set $Pre(\mathcal{S}^0)$ is a union of polytopes and can be computed by solving $J^0 = D \cdot L^0$ multi-parametric programs, where $D$ denotes the number of dynamics and $L^0$ is the number of polytopes which define $\mathcal{S}^0$. Each of these multi-parametric programs will yield a controller partition $\{\mathcal{P}_{j,r}^0\}_{r=1}^R$ consisting of $R$ controller regions whose union covers the feasible set $\mathcal{X}_{1,j} = \bigcup_{r=1,\dots,R} \mathcal{P}_{j,r}^0$ (see Definition 6.2.2). Since the set $Pre(\mathcal{S}^0)$ is computed via multi-parametric programming, we also obtain an associated feedback law $u(x)$ which provides feasible inputs as a function of the state (see Theorem 4.2.3). Note that the various controller partitions may overlap, but that each controller will drive the state into $\mathcal{S}^0$ in one time step, i.e. $f_{\text{PWA}}(x, u(x)) \in \mathcal{S}^0$. Henceforth, we will use the notation $\mathcal{S}^{iter+1} = Pre(\mathcal{S}^{iter}) = \bigcup_{j\in\mathcal{J}^{iter+1}} \mathcal{S}_j^{iter+1}$.

In the following, the algorithm for computing the minimum-time controller for PWA systems will be introduced.

### Algorithm 8.1.2 (Minimum-Time Controller Computation)

1. *Compute the invariant set $\mathcal{O}_\infty^{PWA}$ around the origin (see Figure 8.1(a)) as well as the associated Lyapunov function $V(x) = x^T P x$ and feedback laws $F_i$ as described by Algorithm 7.1.7.*

2. *Initialize the set list* $\mathcal{S}^0 = \mathcal{O}_\infty^{PWA}$ *and initialize the iteration counter* $iter = 0$.

3. *Compute* $\mathcal{S}^{iter+1} = Pre(\mathcal{S}^{iter}) = \bigcup_{j\in\mathcal{J}^{iter+1}} \mathcal{S}_j^{iter+1}$, *by solving a sequence of multi-parametric programs (see Figure 8.1(b)). Thus, a feedback controller partition* $\{\mathcal{P}_{j,r}^{iter+1}\}_{r=1}^R$ *is associated with each obtained set* $\mathcal{S}_j^{iter+1}$. *Obviously, the number of regions* $R$ *of each partition is a function of* $iter$ *and* $j$.

4. *For all* $j^* \in \mathcal{J}^{iter+1}$: *If* $\mathcal{S}_{j^*}^{iter+1} \subseteq \left\{\bigcup_{j\in\mathcal{J}^{iter+1}\setminus\{j^*\}} \mathcal{S}_j^{iter+1}\right\} \cup \left\{\bigcup_{i\in\{1,\dots,iter\}} \mathcal{S}^i\right\}$, *then discard* $\mathcal{S}_{j^*}^{iter+1}$ *from* $\mathcal{S}^{iter+1}$ *and set* $\mathcal{J}^{iter+1} = \mathcal{J}^{iter+1} \setminus \{j^*\}$ *(see Figures 8.1(c) and 8.1(d)).*

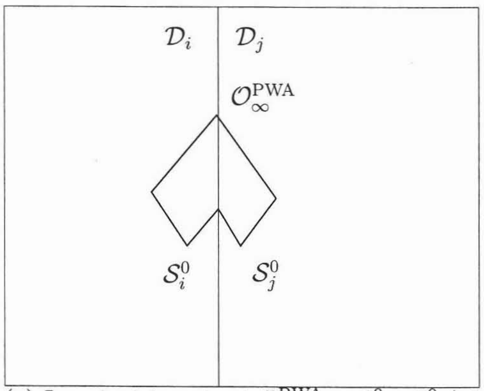
5. *If* $\mathcal{S}^{iter+1} \neq \varnothing$, *set* $iter = iter + 1$ *and goto Step 3.*

6. *For all* $k \in \{1,\dots,iter-1\}$ *and* $r \in \mathbb{N}^+$ *discard all controller regions* $\mathcal{P}_{j,r}^{k+1}$ *for which* $\mathcal{P}_{j,r}^{k+1} \subseteq \bigcup_{i\in\{1,\dots,k\}} \mathcal{S}^i$ *since the associated control laws are not time-optimal and will never be applied.*

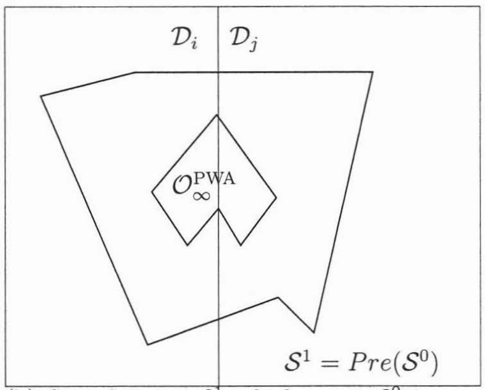
The index *iter* corresponds to the number of steps in which a state trajectory will enter the terminal set $\mathcal{O}_\infty^{\text{PWA}}$ if a receding horizon control policy is applied. If the algorithm terminates in finite time, then the set $\mathcal{S}^{iter}$ is the maximum controllable set $\mathcal{K}_\infty^{\text{PWA}}(\mathcal{O}_\infty^{\text{PWA}})$ as given in Definition 8.1.1.

**Remark 8.1.3** *Note that Algorithm 8.1.2 may not terminate in finite time (e.g. if no bounds are imposed on certain state variables while solving a corresponding optimization problem). This is a problem inherent property and not a result of the computation scheme. It is therefore advisable to specify a maximum step distance* $N$ *which can be used as a termination criterion in Step 5 of Algorithm 8.1.2. The resulting controller computation will then terminate in finite time and the feedback controller will cover* $\mathcal{K}_N^{PWA}(\mathcal{O}_\infty^{PWA})$.

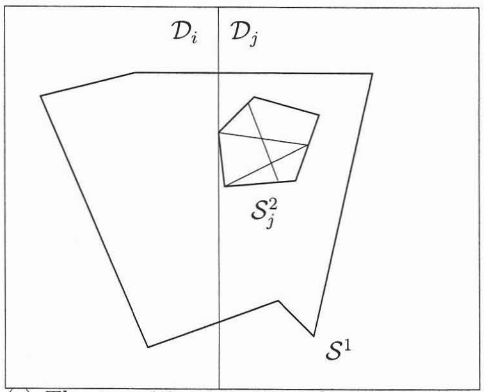
**Remark 8.1.4** *The implementation of Algorithm 8.1.2 requires a function that can detect if a convex polyhedron* $\mathcal{P}_0$ *is covered by a finite set of non-empty convex polyhedra* $\{\mathcal{P}_r\}_{r=1}^R$, *i.e. if* $\mathcal{P}_0 \subseteq \bigcup_{r\in\{1,\dots,R\}} \mathcal{P}_r$. *For instance, this operation is needed to check if two unions of polyhedra cover the same non-convex set [RGK⁺04] (e.g., Step 5 of Algorithm 8.1.2). We refer the reader to [BT03], where an efficient algorithm is given to perform this task.*



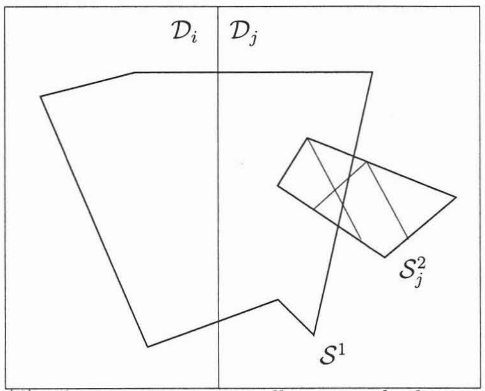
(a) Invariant target set $\mathcal{O}_\infty^{\mathrm{PWA}} = \mathcal{S}_i^0 \cup \mathcal{S}_j^0$ (cf. Step 1 of Alg. 8.1.2).


$\mathcal{D}_i$
$\mathcal{D}_j$
$\mathcal{O}_\infty^{\mathrm{PWA}}$
$\mathcal{S}^1 = Pre(\mathcal{S}^0)$

(b) Set of states $\mathcal{S}^1$ which enter $\mathcal{S}^0$ in one time step. The individual controller partitions defining $\mathcal{S}^1$ are not depicted (cf. Step 2 of Alg. 8.1.2).



(c) The transition partition does not expand the generated set of states. The individual controller partitions defining $\mathcal{S}^1$ are not depicted (cf. Step 4 of Alg. 8.1.2).

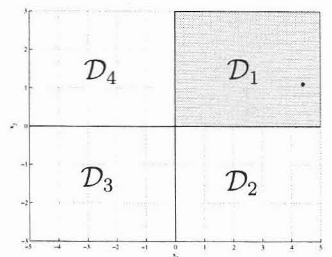


(d) The transition controller expands the controllable set of states. The individual controller partitions defining $\mathcal{S}^1$ are not depicted (cf. Step 4 of Alg. 8.1.2).

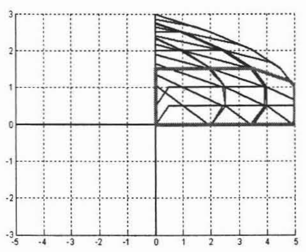
Figure 8.1: Description of Algorithm 8.1.2.

### 8.1.2 Minimum-Time Controller: On-Line Application

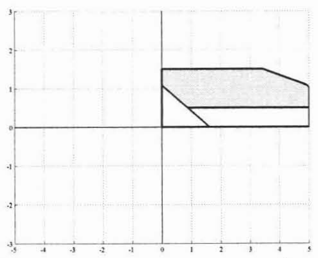
In the minimum-time algorithm presented in here, we can take advantage of some of the algorithm features to speed up the on-line region identification procedure. We propose a three-tiered search tree structure which serves to significantly speed up the region identification. Unlike the search tree proposed in [TJB03a], the tree structure proposed here is computed automatically by Algorithm 8.1.2, i.e. no post-processing



(a) Identify dynamics $\mathcal{D}_i$ containing the state measurement $x(0)$.



(b) Identify feasible controller set $\mathcal{X}_j^{iter}$ containing the state that has the smallest value for $iter$.



(c) Identify the region $\mathcal{P}_{j,r}^{iter}$ containing the current state measurement $x(0)$.

Figure 8.2: Illustration of Algorithm 8.1.5.

is necessary. The following algorithm illustrates how the controller obtained with Algorithm 8.1.2 can be applied, such that the resulting closed-loop trajectories are minimum-time optimal.

**Algorithm 8.1.5 (On-Line Application of Minimum-Time Controller)**

1. *Identify the active dynamics $i$, such that $x \in \mathcal{D}_i$, $i \in \mathcal{I}$ (see Figure 8.2(a))*[2].

2. *Identify controller set $\mathcal{S}_j^{iter}$ associated with dynamic $i$ which is 'closest' to the target set $\mathcal{S}^0$, i.e. $\min_{iter,j}$ $iter$, s.t. $x \in \mathcal{S}_j^{iter}$, $j \in \mathcal{J}^{iter}$ (see Figure 8.2(b)).*

3. *Extract the controller partition $\{\mathcal{P}_{j,r}^{iter}\}_{r=1}^{R}$ with the corresponding feedback laws $F_r, G_r$ and identify the region $r$ which contains the state $x \in \mathcal{P}_{j,r}^{iter}$ (see Figure 8.2(c)).*

4. *Apply the control input $u = F_r x + G_r$. Goto 1.*

Note that the association of controller partitions $\mathcal{S}_j^{iter}$ to active dynamics in Step 2 is trivially implemented by building an appropriate lookup-table during the off-line computation in Algorithm 8.1.2.

**Theorem 8.1.6 (Properties of Minimum-Time Control, [GKBM05])** *The controller obtained with Algorithm 8.1.2 and applied to a PWA system* (6.4) *in a receding horizon control fashion according to Algorithm 8.1.5, guarantees asymptotic stability and feasibility of the closed loop system, provided $x(0) \in \mathcal{K}_N^{PWA}(\mathcal{O}_\infty^{PWA})$.*

[2] Note that once the control law has been computed, a unique dynamic $i$ can be associated with each state, even if the original PWA system was defined in $x$-$u$-space

**Proof** Assume the initial state $x(0)$ is contained in the set $\mathcal{S}^{iter}$ with a step distance to $\mathcal{O}_\infty^{\text{PWA}}$ of *iter*. The control law at Step 4 of Algorithm 8.1.5 will drive the state into a set $\mathcal{S}^{iter-1}$ in one time step (see Step 3 of Algorithm 8.1.2). Therefore, the state will enter $\mathcal{O}_\infty^{\text{PWA}}$ in *iter* steps. Once the state enters $\mathcal{O}_\infty^{\text{PWA}}$ the feedback controllers associated with the common quadratic Lyapunov function ensure stability. $\square$

## 8.2 Reduced Switching Controller

In general, it is possible to obtain even simpler controllers and faster computation times by modifying Algorithm 8.1.2. Instead of computing a minimum-time controller, an alternative scheme which aims at reducing the number of switches can be applied. A change in the active system dynamic $\mathcal{D}_i \to \mathcal{D}_j$, $(i \neq j)$ is referred to as a switch. The proposed procedure does not guarantee the minimum number of switches, though straightforward modifications to the algorithm would yield such a solution. The "minimum number of switches" solution is not pursued here since computation time is the primary objective.

The proposed reduced switch controller will avoid switching the active dynamics for as long as possible. We will introduce the following operator for $i \in \mathcal{I}$:

$$Pre_i(\mathcal{S}_j^{iter}) = \{x \in \mathbb{R}^n \mid \exists u \in \mathbb{R}^m,\ [x^T \ \ u^T]^T \in \mathcal{D}_i,\ A_i x + B_i u + f_i \in \mathcal{S}_j^{iter}\}.$$

Once the $j-th$ controller set $\mathcal{S}_j^{iter}$ obtained at iteration *iter* is computed (see Algorithm 8.1.2, step 3) for dynamics $i$, the set is subsequently used as a target set for as long as the controllable set of states can be enlarged without switching the active dynamics $i$. With this scheme, the total number of convex sets needed to describe the controlled set $\mathcal{S}^{iter}$ remains constant while the size of $\mathcal{S}^{iter}$ increases. Therefore, this scheme generally results in significantly fewer sets during the iterations compared to Algorithm 8.1.2. Specifically, the algorithm works as follows:

**Algorithm 8.2.1 (Computation of Controller with Reduced Number of Switches)**

1. *Compute the invariant set* $\mathcal{O}_\infty^{PWA}$ *around the origin (see Figure 8.3(a)) as well*

*as the associated Lyapunov function $V(x) = x^T Px$ and linear feedback laws $F_r$ as described by Algorithm 7.1.7.*

2. *Initialize the set list $\mathcal{S}^0 = \mathcal{O}_\infty^{PWA} = \bigcup_{j\in\mathcal{J}^0} \mathcal{S}_j^0$ and initialize the iteration counter $iter = 0$.*

3. *Initialize $\mathcal{S}^{iter+1} = \emptyset$ and execute the following for all dynamics $i \in \mathcal{I}$ and set-indices $j \in \mathcal{J}^{iter}$:*

   a) *Initialize counter $c = 0$ and set $\mathcal{C}^0 = \mathcal{S}_j^{iter}$.*
   b) *Compute $\mathcal{C}^{c+1} = Pre_i(\mathcal{C}^c)$ (see Figure 8.3(b)) by using multi-parametric programming and store the associated controller partition. Thus, a feedback controller partition $\{\mathcal{P}_{j,r}^{c+1,iter}\}_{r=1}^R$ is obtained.*
   c) *If $\mathcal{C}^c \subset \mathcal{C}^{c+1}$ (see Figure 8.3(c)), set $c = c + 1$ and goto step 3b.*
   d) *Set $\mathcal{S}^{iter+1} = \mathcal{S}^{iter+1} \cup \mathcal{C}^c$ (see Figure 8.3(d)).*

4. *If $\mathcal{S}^{iter+1} \neq \mathcal{S}^{iter}$, set $iter = iter + 1$ and goto 3.*

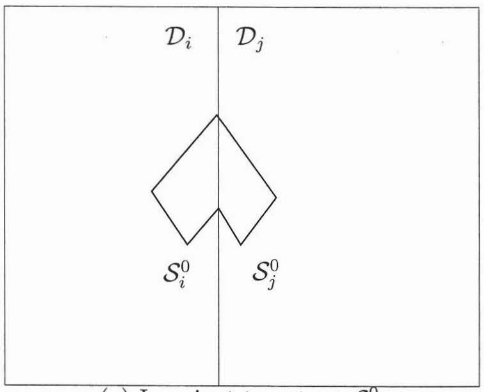
5. *For all $k \in \{1, \dots, iter - 1\}$, $c \in \mathbb{N}$ and $r \in \mathbb{N}^+$ discard all controller regions $\mathcal{P}_{j,r}^{c,k+1}$ for which $\mathcal{P}_{j,r}^{c,k+1} \subset \bigcup_{i\in\{1,\dots,k\}} \mathcal{S}^i$ since the associated control law has a non-minimum number of switches and will never be applied.*

The on-line computation is identical to the scheme described in Section 8.1.2 and the same finite time termination conditions as in Remark 8.1.3 apply.

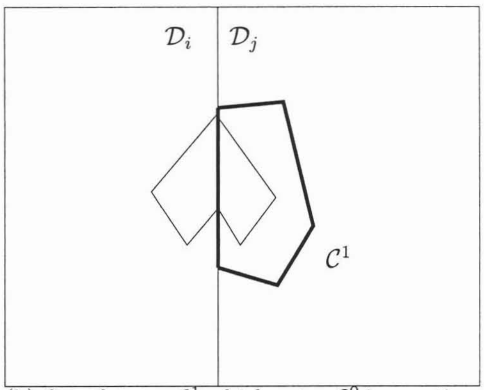
**Remark 8.2.2** *In Algorithm 8.2.1 the counter 'iter' associated with the control sets $\mathcal{S}^{iter}$ corresponds to the number of dynamic switches which will occur before the target set $\mathcal{O}_\infty^{PWA}$ is reached.*

**Remark 8.2.3** *If we always have $\mathcal{C}^c \not\subset \mathcal{C}^{c+1}$ in step 3c of Algorithm 8.2.1, then Algorithm 8.2.1 is identical to Algorithm 8.1.2. However if $\mathcal{C}^c \subset \mathcal{C}^{c+1}$, it is possible to perform a large part of the computations on convex sets, which makes Algorithm 8.2.1 more efficient than Algorithm 8.1.2, in general.*

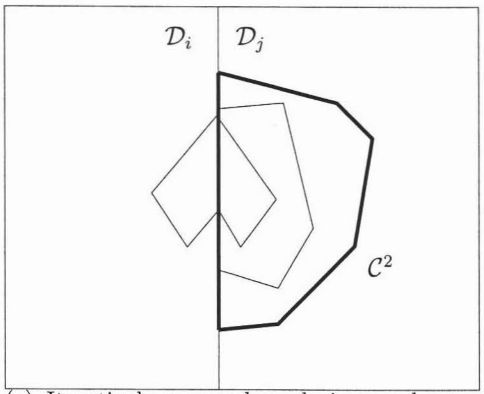
**Theorem 8.2.4 (Properties of Minimum-Switch Control, [GKBM05])** *A controller computed according to Algorithm 8.2.1 and applied to a PWA system (6.4) according to Algorithm 8.1.5, guarantees stability and feasibility of the closed loop system, provided $x(0) \in \mathcal{K}_N^{PWA}(\mathcal{O}_\infty^{PWA})$.*



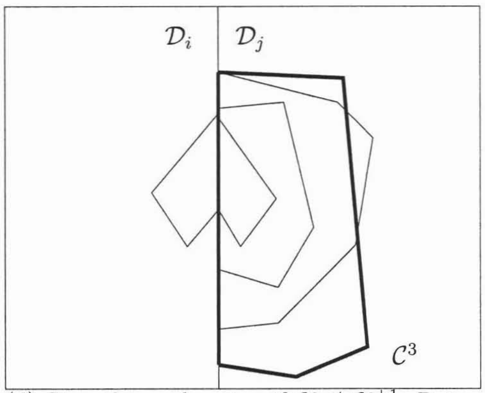
(a) Invariant target set $\mathcal{S}_j^0$.



(b) Set of states $\mathcal{C}^1$ which enter $\mathcal{S}_j^0$ in one time step.



(c) Iteratively proceed exploring as long as $\mathcal{C}^c \subset \mathcal{C}^{c+1}$.



(d) Stop the exploration if $\mathcal{C}^c \not\subset \mathcal{C}^{c+1}$. Return $\mathcal{C}^c$ in such case.

Figure 8.3: Description of Algorithm 8.2.1.

**Proof** Follows from Theorem 8.1.6. □

## 8.3 $M$-step Controller

In the previous section, stability was guaranteed by imposing an appropriate terminal set constraint $\mathcal{O}_\infty^{\mathrm{PWA}}$ and by driving all states towards this set in a minimum-time fashion. In order to cover large parts of the state space, this type of constraint generally entails the use of large prediction horizons which results in controllers with a large number of regions.

In this section, instead of enforcing asymptotic stability with an appropriate terminal set constraint (and the associated cost), we propose to enforce constraint satisfaction only. This can be easily achieved by imposing a set-constraint on the first predicted state in the formulation of the optimization problem. Hence, the terminal-set constraint (6.5b) becomes superfluous and we do not need to rely on large prediction horizons. Asymptotic stability is analyzed in a second step. This scheme is inspired by promising complexity reduction results for LTI systems in [GPM03, GM03].

### 8.3.1 Constraint Satisfaction

If the constrained finite time optimal control problem (6.5) is solved via multi-parametric programming for any prediction horizon $M < N$ with $x_M \in \mathcal{T}_{\text{set}} = \mathbb{R}^n$ in (6.5b) and the additional constraint $x(1) \in \mathcal{K}_N^{\text{PWA}}(\mathcal{O}_\infty^{\text{PWA}})$, the resulting RHC controller guarantees that the state remains within $\mathcal{K}_N^{\text{PWA}}(\mathcal{O}_\infty^{\text{PWA}})$ for all time. The set constraint on the first step guarantees that the resulting controller partition will be positive invariant, which directly implies feasibility for all time [Bla99, Ker00]. The set $\mathcal{O}_\infty^{\text{PWA}}$ can be computed as described by Algorithm 7.1.7 and $\mathcal{K}_N^{\text{PWA}}(\mathcal{O}_\infty^{\text{PWA}})$ can be obtained by applying Algorithm 8.1.2.

Since the target set $\mathcal{K}_N^{\text{PWA}}(\mathcal{O}_\infty^{\text{PWA}}) = \bigcup_{c \in \{1,\ldots,C\}} \mathcal{K}_N^c$ is non-convex in general (i.e. a union of $C$ polytopes $\mathcal{K}_N^c$) a controller partition can be obtained by solving a sequence of $C \cdot D$ multi-parametric programs, e.g. (5.5) or (5.3), where $D$ corresponds to the total number of different dynamics. Specifically, the $M$-step controller can be obtained by solving $C \cdot D$ multi-parametric programs (e.g., (5.5) or (5.3)) for an arbitrary $M$ with $\mathcal{T}_{\text{set}} = \mathcal{K}_N^c$ in (6.5b) ($C$ different sets) and for $D$ different dynamics in (6.4). The smaller $M$ the lower the controller complexity. However, the choice of $M$ has no impact on the size of the generated sets.

### 8.3.2 Stability Analysis

The controller partition obtained in the previous subsection will generally contain overlaps such that the closed-loop dynamics associated with a given state $x(0)$ may not be unique. It is therefore not possible to perform a non-conservative stability analysis of the closed-loop system. However, by using the PWA value function $J_N^*(x)$ in (6.5a) as a selection criterion it is possible to obtain a non-overlapping partition

( [GKBM05] or [Bor03], pg. 158-160) by solving a number of LPs, i.e. only the cost optimal controller is stored.

The resulting controller partition is invariant and a unique controller region $r$ ($x \in \mathcal{P}_r$) and unique dynamics $l$ ($x \in \mathcal{D}_l$) is associated with each state $x$, i.e. the closed loop system corresponds to an autonomous PWA system

$$x_{k+1} = (A_l + B_l F_r)x_k + B_l G_r + f_l, \qquad x_k \in \mathcal{P}_r \cap \mathcal{D}_l \tag{8.1a}$$
$$= \widetilde{A}_r x_k + \widetilde{f}_r, \qquad x_k \in \mathcal{P}_r. \tag{8.1b}$$

Since every controller region $\mathcal{P}_r$ is only contained in one unique dynamic $\mathcal{D}_l$, the update matrix $\widetilde{A}_r$ and vector $\widetilde{f}_r$ are uniquely defined. In the sequel we will show how formulate the search for a PWA Lyapunov function for autonomous PWA systems as a linear program.

It was shown how to use Semi-Definite Programming (SDP) to compute piecewise quadratic (PWQ) Lyapunov functions for continuous-time PWA systems in [JR98] and for discrete-time PWA systems in [FTCMM02, GLPM03]. The search for a PWQ Lyapunov function is conservative, since the SDP formulation is based on the $S$-procedure, which is not lossless for the cases considered [BGFB94]. Therefore, instead of searching for a PWQ Lyapunov function via SDP, we here show how to compute a PWA Lyapunov function via LP. The proposed scheme is based on results for continuous time systems which were published in [Joh03].

The computation scheme for the PWA Lyapunov function is non-conservative (i.e. if a PWA Lyapunov function exists for the given partition, it will be found) such that it may succeed when no PWQ Lyapunov function can be found with the schemes in [FTCMM02, GLPM03]. However, the converse is also true. Furthermore, the value function associated with a mp-LP controller partition is PWA, such that this function type is a natural candidate in the search for a Lyapunov function. The following Theorem is based on [Vid93, p. 267]:

**Theorem 8.3.1 (Asymptotic Stability)** *The origin* $x = 0$ *is* asymptotically stable *if there exists a function* $V(x)$ *and scalar coefficients* $\alpha > 0$, $\beta > 0$, $\rho > 0$ *such that:* $\beta\|x_k\| \geq V(x_k) \geq \alpha\|x_k\|$ *and* $V(x_{k+1}) - V(x_k) \leq -\rho\|x_k\|$, $\forall x_k \in \mathcal{X}$ *and* $V(x) = \infty$, $\forall x \notin \mathcal{X}$. *The successor state* $x_{k+1}$ *is defined in* (6.6), $\|\cdot\|$ *denotes a vector norm and* $\mathcal{X}$ *denotes the state space of interest.*

In order to pose the problem of finding a PWA Lyapunov function without introducing conservative relaxations, a region transition map is created. Specifically, a transition map $\mathcal{S}$ is created according to

$$\mathcal{S}(i,j) = \begin{cases} 1, & \text{if } \exists x_k \in \text{int}(\mathcal{P}_i), \quad \text{s.t. } x_{k+1} \in \mathcal{P}_j, \\ 0, & \text{otherwise.} \end{cases}$$

where $x_{k+1}$ is defined by (6.4) and Theorem 4.2.3 and $\text{int}(\cdot)$ denotes the strict interior of a set.

**Remark 8.3.2** *In principle, one LP needs to be solved for each element of the transition map $\mathcal{S}$, i.e. a total of $R^2$ LPs, where $R$ denotes the total number of system dynamics. However, instead of solving LPs directly, it is advisable to first compute bounding boxes (hyper-rectangles) for each region $\mathcal{P}_r$ ($r \in \mathcal{R}$). In addition, a bounding box of the affine map of the region $\mathcal{P}_r^+ = \{\widetilde{A}_r x + \widetilde{f}_r \in \mathbb{R}^n |\ x \in \mathcal{P}_r\}$ needs to be computed. The number of LPs which need to be solved in order to compute the bounding boxes is linear in the number of regions $R$ and state space dimension $n$. This computation is tractable even for very complex partitions. The bounding boxes can be efficiently checked for intersections, such that certain transitions $i \longrightarrow j$ can be ruled out. In our experience, the bounding box implementation is the most effective way to compute $\mathcal{T}$ for complex region partitions.*

In a second step, the polytopic transition sets $\mathcal{P}_{ij}$ for system (6.6) are explicitly computed as

$$\mathcal{P}_{ij} = \{x_k \in \mathbb{R}^n |\ x_k \in \mathcal{P}_i, \quad x_{k+1} \in \mathcal{P}_j\}, \quad \forall i,j \in \{1,\dots,R\}, \text{ s.t. } \mathcal{S}(i,j) = 1.$$

If $\mathcal{S}(i,j) = 0$, we can directly set $\mathcal{P}_{ij} = \emptyset$. Subsequently, the vertices of the transition sets ($vert(\mathcal{P}_{ij})$) and the controller sets ($vert(\mathcal{P}_i)$) are computed. The problem of finding a PWA Lyapunov function,

$$\text{PWA}_i(x) = L_i x + C_i, \qquad \text{if } x \in \mathcal{P}_i, \quad i = 1,\dots,R$$

for the autonomous PWA system (6.6) such that the conditions in Theorem 8.3.1 are satisfied can now be stated as

$$\beta\|x\|_1 \geq \text{PWA}_i(x) \geq \alpha\|x\|_1, \ \alpha,\beta > 0, \ \forall x \in vert(\mathcal{P}_i), \ \forall i \in \{1,\dots,R\}, \tag{8.2a}$$

$$\text{PWA}_j(x_{k+1}) - \text{PWA}_i(x_k) \leq \rho\|x_k\|_1, \ \rho < 0, \ \forall x_k \in vert(\mathcal{P}_{ij}), \ \forall i,j \in \{1,\dots,R\}. \tag{8.2b}$$

Since the vertices of all sets $\mathcal{P}_i$ and $\mathcal{P}_{ij}$ are known, the resulting problem is linear in $L_i, C_i, \alpha, \beta, \rho$ and can therefore be solved as an LP.

**Theorem 8.3.3** *If the LP* (8.2) *associated with the autonomous PWA system* (6.6) *is feasible, then this system is asymptotically stable.*

**Proof** Since the function $\text{PWA}_i(x)$ is piecewise affine, it follows that satisfaction of (8.2a) for all vertices of $\mathcal{P}_i$ implies that the inequalities in (8.2a) will also hold $\forall x \in \mathcal{P}_i$. Furthermore, if (8.2b) holds for all vertices of $\mathcal{P}_{ij}$, it follows from linearity of the system dynamics (6.6) that the inequality will hold for all states $x \in \mathcal{P}_{ij}$. Since the partition is invariant, it follows that $\bigcup_{i\in\{1,\ldots,R\}} \mathcal{P}_i = \bigcup_{i,j\in\{1,\ldots,R\}} \mathcal{P}_{ij}$. Therefore, the inequalities in (8.2a) and (8.2b) hold $\forall x \in \bigcup \mathcal{P}_i$ such that the conditions in Theorem 8.3.1 are satisfied, i.e. feasibility of (8.2) implies asymptotic stability of the autonomous PWA system (6.6) . □

It should be noted that the required computation time may become large because of the extensive reachability analysis, vertex enumeration and size of the final LP. Specifically, the LP (8.2) introduces one constraint for each vertex of each region $\mathcal{P}_r$, $\forall r \in \{1,\ldots,R\}$ (see (8.2a)) and one constraint for each vertex of each $\mathcal{P}_{ij}$, $\forall i,j \in \{1,\ldots,R\}$ (see (8.2b)). The number of variables is $(n+1)R$, where $R$ denotes the number of regions and $n$ the state space dimension.

However, in the authors experience, stability analysis problems for a couple of hundred regions in a state space dimension of less than five are tractable and the necessary computation effort is comparable to the approaches in [FTCMM02, GLPM03].

## 8.3.3 $M$-step Controller Computation

The $M$-step control scheme utilizes tools from invariant set computation and stability analysis in order to compute controllers with small prediction horizons which guarantee constraint satisfaction as well as asymptotic stability. The basic procedure consists of two main stages. In the first stage, a $M$-step optimal controller is computed which guarantees constraint satisfaction for all time. Since constraint satisfaction does not imply asymptotic stability, it is necessary to analyze the stability properties of the closed-loop system in a second stage. Specifically, the algorithm works as follows.

**Algorithm 8.3.4 ($M$-step Controller Computation)**

1. *Compute the invariant set $\mathcal{O}_\infty^{PWA}$ around the origin and an associated Lyapunov function as described by Algorithm 7.1.7.*

2. *Compute the set $\mathcal{K}_N^{PWA}(\mathcal{O}_\infty^{PWA}) = \bigcup_{c\in\{1,\dots,C\}} \mathcal{K}_N^c$ $(N \to \infty)$ by applying Algorithm 8.1.2.*

3. *Solve a sequence of $C \cdot D$ mp-LPs* (5.6) *for prediction horizon $M$ with $\mathcal{T}_{set} = \mathcal{K}_N^c,\ \forall c \in \{1,\dots,C\}$ in* (6.5b), *affine dynamics $i \in \mathcal{I} = \{1,\dots,D\}$ in* (6.4) *and $M \leq N$.*

4. *Remove the region overlaps by using the PWA value function $J_N^*(x)$ as a selection criterion (see [GKBM05] or [Bor03] for details).*

5. *Attempt to find a PWA Lyapunov function using the procedure described in Section 8.3.2, or a PWQ Lyapunov function as described in [Gri04, Chapter 8].*

There is no guarantee that Step 2 of Algorithm 8.3.4 will terminate in finite time to the set $\mathcal{K}_N^{\text{PWA}}(\mathcal{O}_\infty^{\text{PWA}})$ or that a Lyapunov function can be found in Step 5. If the former happens, following Remark 8.1.3 one can impose an artificial upper bound on the number of iterations in Step 5 of Algorithm 8.1.2, which leads to a set $\mathcal{K}_N^{\text{PWA}}(\mathcal{O}_\infty^{\text{PWA}}) \subset \mathcal{K}_\infty^{\text{PWA}}(\mathcal{O}_\infty^{\text{PWA}})$. Note that even if this case happens, the set $\mathcal{K}_N^{\text{PWA}}(\mathcal{O}_\infty^{\text{PWA}})$ is still positively invariant for all time in the sense of Definition 6.2.4. If no Lyapunov function is found, the resulting controller is still guaranteed to satisfy the system constraints for all time, but no proof of asymptotic stability can be given. However, the state is guaranteed to remain within a bounded invariant set.

**Remark 8.3.5** *If the stability analysis in Step 5 of Algorithm 8.3.4 fails, it is advisable to recompute the controller in Step 3 using different weights $Q_u, Q_x, Q_{x_N}$ and/or a different prediction horizon $M$ in* (6.5). *Slight modifications in these parameters may make the subsequent stability analysis in Step 5 feasible. Some hints on how to tune the respective parameters have been presented in [LÖ3], where it is advised that extending the prediction horizon and/or increasing the state weights $Q_x$ and $Q_{x_N}$ tends to lead controllers for which closed-loop stability can be shown. However as illustrated in [Löf03, Section 2.5] the domain of such parameters which provide closed-loop stability is non-convex even for linear systems, therefore the precise tuning is not straightforward.*

**Theorem 8.3.6 (Properties of $M$-step Control, [GKBM05])** *If the stability analysis in Step 5 of Algorithm 8.3.4 is successful and the feedback law obtained in*

*Step 4 is applied to system* (6.4) *in a RHC fashion, then the closed-loop system is exponentially stable on* $\mathcal{K}_N^{PWA}(\mathcal{O}_\infty^{PWA})$ *and the system constraints are satisfied for all time.*

**Proof** The partition computed in Step 4 is invariant by construction, hence constraint satisfaction is guaranteed. Exponential stability follows trivially from the successful stability analysis in Step 5. □

## 8.4 Numerical Examples

As was shown in [GM03,GPM03] and will also be illustrated in this section, algorithms of type 8.1.2, 8.2.1, and 8.3.4 generally yield controllers of significantly lower complexity than those which are obtained if a linear norm-objective is minimized as in (6.5) [BCM03a, BCM03b, KM02].

**Example 8.4.1** *Consider the 2-dimensional problem adopted from [MR03],*

$$x(k+1) = \begin{cases} \begin{bmatrix} 1 & 0.2 \\ 0 & 1 \end{bmatrix} x(k) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(k) + \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \text{if } x_{(1)}(k) \le 1 \\ \begin{bmatrix} 0.5 & 0.2 \\ 0 & 1 \end{bmatrix} x(k) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(k) + \begin{bmatrix} 0.5 \\ 0 \end{bmatrix} & \text{if } x_{(1)}(k) \ge 1 \end{cases}, \tag{8.3}$$

*subject to constraints* $-x_{(1)}(k) + x_{(2)}(k) \le 15$, $-3x_{(1)}(k) - x_{(2)}(k) \le 25$, $0.2x_{(1)}(k) + x_{(2)}(k) \le 9$, $x_{(1)}(k) \ge -6$, $x_{(1)}(k) \le 8$, *and* $-1 \le u(k) \le 1$. *Weight matrices in the cost function were chosen as* $Q_x = I$ *and* $Q_u = 0.1$ *in (6.5).*

**Example 8.4.2** *Consider the 3-dimensional PWA system introduced in [MR03],*

$$x(k+1) = \begin{cases} \begin{bmatrix} 1 & 0.5 & 0.3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} x(k) + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u(k) + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} & \text{if } x_{(2)}(k) \le 1 \\ \begin{bmatrix} 1 & 0.2 & 0.3 \\ 0 & 0.5 & 1 \\ 0 & 0 & 1 \end{bmatrix} x(k) + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u(k) + \begin{bmatrix} 0.3 \\ 0.5 \\ 0 \end{bmatrix} & \text{if } x_{(2)}(k) \ge 1 \end{cases}, \tag{8.4}$$

*subject to constraints* $-10 \le x_{(1)}(k) \le 10$, $-5 \le x_{(2)}(k) \le 5$, $-10 \le x_{(3)}(k) \le 10$, *and* $-1 \le u(k) \le 1$. *Again, weights in the cost function are* $Q_x = I$, $Q_u = 0.1$.

**Example 8.4.3** *Consider the 4-dimensional PWA system introduced in [MR03],*

$$x(k+1) = \begin{cases} \begin{bmatrix} 1 & 0.5 & 0.3 & 0.5 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} x(k) + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u(k) + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} & \text{if } x_{(2)}(k) \leq 1 \\ \begin{bmatrix} 1 & 0.2 & 0.3 & 0.5 \\ 0 & 0.5 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} x(k) + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u(k) + \begin{bmatrix} 0.3 \\ 0.5 \\ 0 \\ 0 \end{bmatrix} & \text{if } x_{(2)}(k) \geq 1 \end{cases}, \tag{8.5}$$

*subject to constraints* $-10 \leq x_{(1)}(k) \leq 10$, $-5 \leq x_{(2)}(k) \leq 5$, $-10 \leq x_{(3)}(k) \leq 10$, $-10 \leq x_{(4)}(k) \leq 10$, *and* $-1 \leq u(k) \leq 1$. *Weighting matrices in the cost function are* $Q_x = I$, $Q_u = 0.1$.

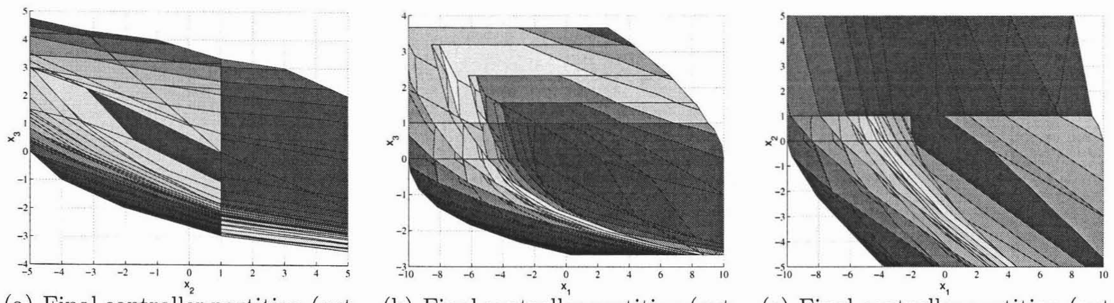
Once the set $\mathcal{O}_\infty^{\text{PWA}}$ is computed, Algorithms 8.1.2, 8.2.1, and 8.3.4 are applied to Examples 8.4.1 – 8.4.3. A runtime comparison of the computation procedures as well as complexity of the resulting solutions are reported in Table 8.1. It is worth noting that the stability analysis part of Algorithm 8.3.4 for Example 8.4.3 failed due to a prohibitive size of the LMI [Gri04, Inequality (8.6)] which needs to be solved in order to obtain a PWQ Lyapunov function. Specifically, for this particular example such LMI contains more than 236000 variables and more than one million constraints, which is not tractable by any of the available solvers. These numbers are directly related to the number of controller regions, in this case 4627. Adjusting the parameters of the mp-LPs (5.6) solved in Step 3 of Algorithm 8.3.4 according to the hints in Remark 8.3.5 didn't help to decrease the complexity to a tractable level. Controller regions for Example 8.4.2 are depicted in Figures 8.4(a)–8.4(c).

Example 8.4.3 nicely illustrates the benefits of the reduced-switching algorithm described in Section 8.2. In this case, the controller was calculated 40 times faster compared to the minimum-time algorithm. The reason for this substantial speedup is that in Algorithm 8.2.1 the target sets were convex for multiple successive steps, which simplified the controller construction.

In order to compare low complexity control strategies discussed in this chapter with the cost-optimal dynamic programming approach of [BCM03a], we generated 10 random PWA systems with 2 states, 1 input and 4 piecewise-affine dynamics. All elements in the state space matrices were assigned random values between $[-2, 2]$ (i.e.,

| | Alg. 8.1.2 | | Alg. 8.2.1 | | Alg. 8.3.4, $N = 1$ | | Alg. [BCM03a] | |
|---|---|---|---|---|---|---|---|---|
| | $t$ | $\#R$ | $t$ | $\#R$ | $t$ | $\#R$ | $t$ | $\#R$ |
| Ex. 8.4.1 | 61 sec. | 279 | 40 sec. | 186 | 53 sec. | 61 | 5.5 h | 1413 |
| Ex. 8.4.2 | 1153 sec. | 1519 | 755 sec. | 1044 | 286 sec. | 522 | $\star$ | $\star$ |
| Ex. 8.4.3 | 92 h | 7894 | 2.2 h | 2434 | 4.3 h$^\dagger$ | 4627$^\dagger$ | $\star$ | $\star$ |

Table 8.1: Off-line CPU-time $t$ and number of controller regions $\#R$ for different algorithms. The CPU-time for Algorithm 8.3.4 includes the stability analysis. The $\star$ denotes that the computations were not completed after 7 days. The $\dagger$ symbol denotes that the stability analysis procedure failed. The computation was run on a 2.8GHz Pentium IV CPU running the Windows version of MATLAB 6.5 along with the NAG foundation LP solver.



(a) Final controller partition (cut on $x_{(1)} = 0$).

(b) Final controller partition (cut on $x_{(2)} = 0$).

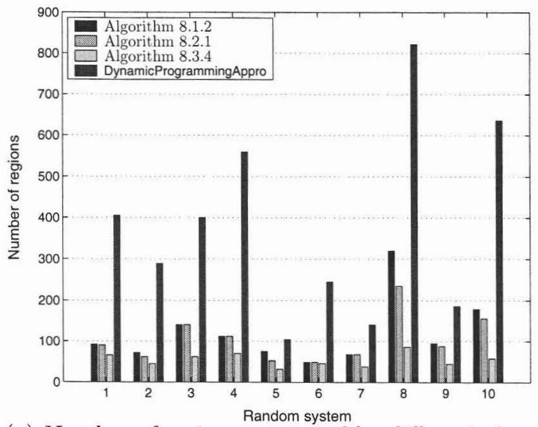
(c) Final controller partition (cut on $x_{(3)} = 0$).

Figure 8.4: The controller partition obtained by applying Algorithm 8.1.2 on Example 8.4.2. The actual partition is three dimensional (see (8.4)), but only the axis intersections are shown.

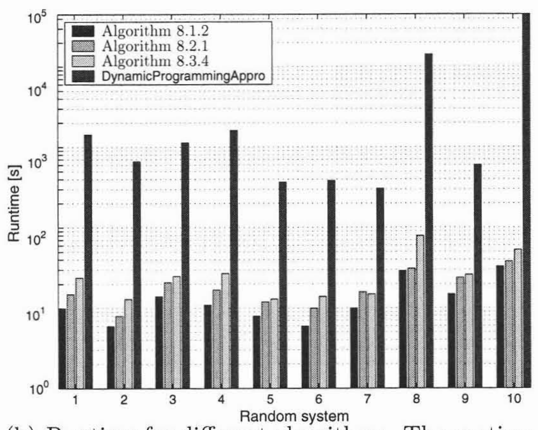
stable and unstable systems were considered). Each of the random PWA systems consists of 4 different affine dynamics which are defined over non-overlapping random sets whose union covers the square $\mathbb{X} = [-5, 5] \times [-5, 5]$. The origin was chosen to be on the boundary of multiple dynamics. All simulation runs as well as the random system generation were performed with the MPT toolbox [KGB04][3].

Algorithms 8.1.2, 8.2.1, and 8.3.4, as well as the cost-optimal strategy of [BCM03a] were applied to these systems. Complexity of the resulting solution and run time of each algorithm are depicted graphically in Figures 8.5(a) and 8.5(b).

[3]For random PWA systems `mpt_randPWAsys` of the MPT toolbox [KGB04] was called



(a) Number of regions generated by different algorithms.

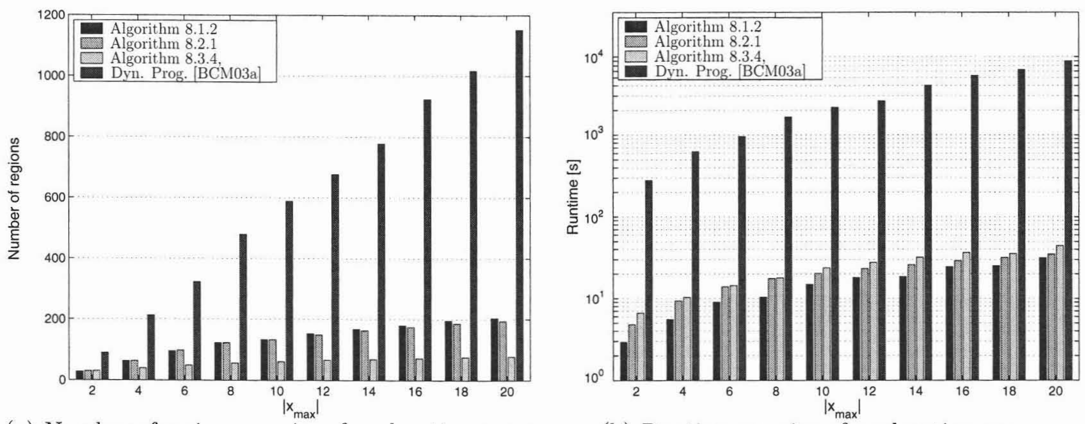


(b) Runtime for different algorithms. The runtime for Algorithm 8.3.4 includes the stability analysis.

Figure 8.5: Complexity and runtime for 10 random PWA systems.

To further investigate the behavior of different control strategies, another test on a set of 10 random PWA systems was performed to show how the complexity of Algorithms 8.1.2, 8.2.1, and 8.3.4 scales with increasing volume of the exploration space. A comparison with the approach in [BCM03a] is depicted in Figures 8.6(a) and 8.6(b). For the random systems considered here, the necessary runtime is reduced by two orders of magnitude and the solution complexity is reduced by one order of magnitude, on average. In addition, these differences become larger with increasing size of the state constraints. Although we have not come across any examples where the proposed schemes are inferior to the approaches in [BBBM03, KM02], such cases may exist.

None of the algorithms presented in this chapter guarantee optimal closed-loop performance in the sense of the cost-objective (6.5). In order to assess the degradation in performance, equidistantly spaced data points inside the set $\mathcal{K}_{\infty}^{\mathrm{PWA}}(\mathcal{O}_{\infty}^{\mathrm{PWA}})$ were generated as feasible initial states. Subsequently, the closed-loop trajectory cost for these initial states was computed according to the performance index (6.5a). The average decrease in performance with respect to the cost-optimal solution of [BCM03a] is summarized in Figures 8.7(a) and 8.7(b). It can be seen that closed-loop performance gets better with increasing size of the exploration space. The intuitive explanation of this observation is as follows. If the state is far away from the origin, going at "full throttle" will be the optimal strategy, since the contribution of the state penalty term



(a) Number of regions vs. size of exploration space (b) Runtime vs. size of exploration space.

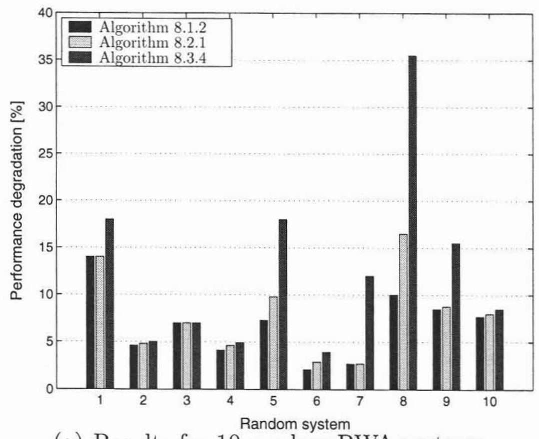
Figure 8.6: Complexity and runtime versus size of exploration space (average over 10 random PWA systems).

in (6.5a) is much higher than the term which penalizes the control action. Therefore almost the same performance is achieved with low complexity strategies as with cost-optimal algorithms for a majority of the controllable state-space, resulting in good average performance.

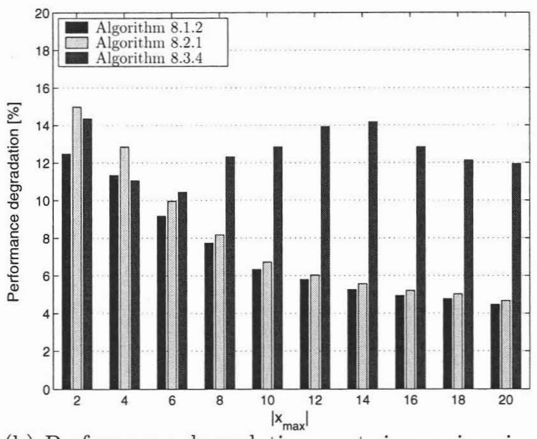
## 8.5 Conclusion

In this chapter, three novel algorithms to compute low complexity feedback controllers for constrained PWA systems have been presented. All three schemes lead to controllers which guarantee constraint satisfaction for all time as well as asymptotic stability. The first proposed computation scheme iteratively solves a series of optimal control problems with $N = 1$ such that a feedback controller is obtained which drives the state into a target set in a minimum time fashion. We have showed that by considering simpler objective function (which in this case takes a form of a piecewise constant function) one can decrease the number of possible transition which need to be checked during the controller construction phase. Furthermore, a search tree for efficient on-line identification of the resulting feedback law is automatically constructed by the minimum-time algorithm.

The second algorithm also uses the principle of driving system states into a corresponding terminal set in the least possible number of steps. But in addition the control



(a) Results for 10 random PWA systems.



(b) Performance degradation w.r.t. increasing size of exploration space (average over 10 random systems).

Figure 8.7: Performance degradation with respect to cost-optimal solution of [BCM03a]. The performance of Algorithm 8.3.4 can be improved by increasing $N$.

law is constructed in a way such that the number of switches between various dynamics of the controlled PWA systems is reduced. By doing this, convexity of the target sets used at each step of the design algorithm can be maintained for more than for one step, which leads to increased runtime performance and decrease of the complexity of solution.

The third computation scheme (referred to as $M$-step control) separately deals with the requirements of constraint satisfaction and asymptotic stability. We have illustrated that feasibility for all time can be achieved by imposing a suitable terminal set constraint on the first predicted state. In the $M$-step scheme, stability is not enforced but merely verified a posteriori. This is done by searching for a Lyapunov function, which then provides a certificate of closed-loop asymptotic stability. There is, however, no guarantee that such function will be found. In such case we have proposed how to adjust the parameters of the optimization problem to increase the chance of finding such function.

An extensive case study was also provided which clearly indicates that all presented algorithms reduce complexity versus optimal controllers [BBBM03, KM02] by several orders of magnitude, in general. The proposed procedures make problems tractable that were previously too complex to be tackled by standard methods. On the other hand, one usually has to face a performance drop of about 15% if the minimum-time and reduced-switching approaches are used to design a feedback controller, and up to 50% when employing the $M$-step scheme. It is therefore up to the control engineer to decide whether the significantly decreased controller complexity outperforms the decreased controller performance.

# 9

# Efficient Evaluation of Piecewise Control Laws defined over a Large Number of Polyhedra

In this chapter we consider the *point-location* or *set membership problem* [Sno97] for the class of discrete-time control problems with linear state and input constraints for which an explicit time-invariant piecewise state feedback control law over a set of possibly overlapping polyhedral regions is given. The point-location problem comes into play on-line when evaluating the control law. One must identify the state space region in which the measured state lies at the current sampling instance. As the number of defining regions grows, a purely *sequential search* (also known as *exhaustive search*) through the regions is too lengthy to achieve high sampling rates. Hence, it is important to find an efficient on-line search strategy in order to evaluate the control action 'in time' without the need of a heavy additional memory and preprocessing demand.

This work is motivated, but not limited, by the recent developments in the field of controller synthesis for hybrid systems [vS00, Hee99, Son81, BBM00b, Bor03, Joh03]. A significant amount of the research in this field has focused on solving constrained optimal control problems, both for continuous-time and discrete-time hybrid systems. We consider the class of constrained discrete-time *piecewise affine* (PWA) systems [Son81] that are obtained by partitioning the extended state-input space into polyhedral regions and associating with each region a different affine state update equation.

As shown in previous sections, for piecewise affine systems the constrained finite time optimal control problem can be solved by means of multi-parametric programming and the resulting solution is a time-varying piecewise affine state feedback control law. If the solution to the CFTOC problem is used in a *receding horizon control* [Mac02] strategy the time-varying PWA state feedback control law becomes time-invariant and can serve as a control 'look-up table' on-line, thus enabling receding horizon control to be used for fast sampled systems. However, due to the combinatorial nature of the problem the number of state space regions over which the control look-up table is defined grows in the worst case exponentially [Bor03, BMDP02] and therefore efficient on-line search strategies are required to achieve fast sampling rates.

In this section we present a novel, computationally efficient algorithm that performs the aforementioned point-location search for *general* closed-form piecewise (possibly nonlinear) state feedback control laws defined over a finite number of polyhedra or over a finite number of regions for which a bounding box [BFT04] computation is feasible. Moreover, control laws that do not form a polyhedral partition, but are composed of a collection of *overlapping* polytopic sets, are included naturally in the algorithm. The proposed point-location search algorithm offers a significant improvement in computation time at the cost of a low additional memory storage demand and *very low pre-computation* time for the construction of the search tree. This enables the algorithm to work for controller partitions with a large number of regions, which is demonstrated on numerical examples. In order to show its efficiency, the algorithm is compared with the procedure proposed in [TJB03a] where a binary search tree is pre-computed over the controller state space partition.

## 9.1 Point Location Problem

We consider arbitrary discrete-time control problems with a closed-form (possibly nonlinear) time-invariant piecewise state feedback control law of the form

$$\mu(x(t)) := \mu_i(x(t)), \qquad \text{if} \quad x(t) \in \mathcal{P}_i, \tag{9.1}$$

where $i = 1, \ldots, N_{\mathcal{P}}$. Here $x(t) \in \mathbb{R}^{n_x}$ denotes the state of the controlled system at time $t \geq 0$, $\mu_i(\bullet) \in \mathbb{R}^{n_u}$ are nonlinear control functions (or oracles), and the sets $\mathcal{P}_i$ are compact and possibly *overlapping*, i.e. there exist $\mathcal{P}_i$ and $\mathcal{P}_j$ with $i \neq j$ such that $\mathcal{P}_i \cap \mathcal{P}_j$ is full-dimensional. Moreover, $\mathcal{P} := \{\mathcal{P}_i\}_{i=1}^{N_{\mathcal{P}}}$ denotes the collection of sets $\mathcal{P}_i$.

In an on-line application the closed-form piecewise control law is $u(t) = \mu(x(t))$, where $u \in \mathbb{R}^{n_u}$ denotes the control input. In order to evaluate the control one needs to identify the state space region $\mathcal{P}_i$ in which the measured state $x(t)$ lies at the sampling instance $t$, i.e.

**Algorithm 9.1.1 (Control evaluation)**

```
1. measure the state $x(t)$ at time instance t
2. search for the index set of regions $\mathcal{I}$ such that $x(t) \in \mathcal{P}_i$
   for all $i \in \mathcal{I}$
       IF $\mathcal{I} = \emptyset$ THEN problem infeasible STOP
       IF $|\mathcal{I}| > 1$ THEN pick one element $i^\star \in \mathcal{I}$ according to certain strategy
3. apply the control input $u(t) = \mu_{i^\star}(x(t))$ to the system
```

The second step in Algorithm 9.1.1 is also known as the *point-location* or the *set membership problem* [Sno97]: in other words, given a point $x \in \mathbb{R}^{n_x}$ and a set of sets $\{\mathcal{P}_i\}_{i=1}^{N_\mathcal{P}}$, the goal is to list the set of indices $\mathcal{I}$ such that $x \in \mathcal{P}_i$ for all $i \in \mathcal{I}$. If the cardinality of the set $\mathcal{I}$ in Step 2 is bigger than one, the proper element $i^\star$ which corresponds to the optimal feedback $\mu_{i^\star}$ has to be decided based on the property of the solution[1].

## 9.2 Alternative Search Approaches

Due to the combinatorial nature of the CFTOC problem (5.5), the controller complexity, or the number $N_\mathcal{P}$ of state space regions $\mathcal{P}_i$, can grow exponentially with its parameters in the worst case [Bor03, BMDP02]. Hence, for general control problems, a purely sequential search through the regions is not sufficient in an on-line application. It is therefore important to utilize efficient on-line search strategies in order to evaluate the control action 'in time' without the need of a heavy additional memory demand.

Several authors addressed the point-location/memory storage issue but with moderate success for geometrically complex regions or controllers defined over a large number of regions. A few interesting ideas are mentioned in the following. For the solution to the particular CFTOC problem when, additionally, the system is constrained and linear, i.e.

$$f_{\text{PWA}}(x(t), u(t)) := Ax(t) + Bu(t), \text{ with } \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} \in \mathcal{D},$$

the authors in [BBBM01] propose a search algorithm based on the convexity of the piecewise affine value function. Even though this algorithm reduces the storage space

[1]For instance, if the partition was generated as a solution to a CFTOC problem (5.5), then $i^\star$ corresponds to the region in which the cost function for a corresponding state $x$ is minimal. If the partition represents a minimum-time controller described in Chapter 8 then $i^\star$ corresponds to the region which has the least step distance to the origin.

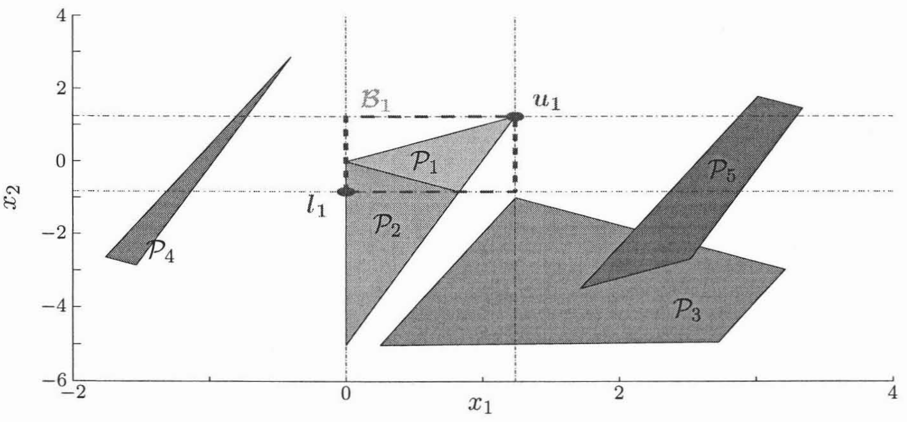


Figure 9.1: Overlapping collection of polytopic sets $\{\mathcal{P}_i\}_{i=1}^5$ with bounding box $\mathcal{B}_1$ of $\mathcal{P}_1$.

significantly, the storage demand as well as the search time are still linear in the number of regions. [JGR05] address this issue for the same CFTOC problem class by demonstrating a link between the piecewise affine value function of [BBBM01] and power diagrams (extended Voronoi diagrams). Utilizing standard Voronoi search methods the search time then reduces to $O(\log(N_{\mathcal{P}}))$.

To the authors knowledge only two other approaches tackle the more general problem, where the only restriction is that the domain of the control law is a non-overlapping polyhedral partition of the state space. (Note that this is more restrictive than the algorithm presented here, cf. 9.3.) [GTM04, GTM03] aim at pre-computing a minimal polyhedral representation of the original controller partition in order to reduce storage and search complexity. However, the computation is 'practically' limited to a small number of regions with a small number of facets[2] [Grü00], since the pre-computation time grows exponentially. Relaxations to a larger number of regions is possible at the cost of data storage and a higher search complexity.

An alternative approach, which will be used here for comparison, was proposed by Tøndel *et al.* in [TJB03a], where a binary search tree is constructed on the basis of

[2]A facet of a polyhedron $\mathcal{P}$ of dimension $n_x$ is any $(n_x - 1)$-dimensional intersection of $\mathcal{P}$ with a tangent hyperplane.

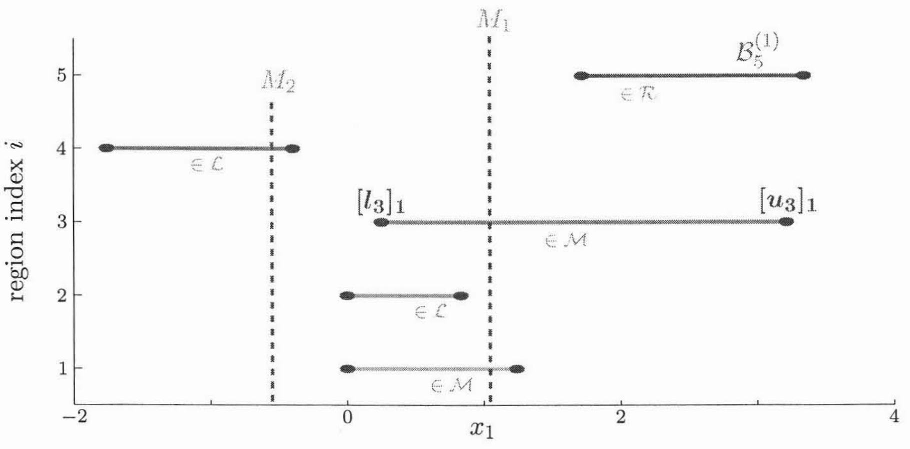


Figure 9.2: Projection $\mathcal{B}_i^{(1)}$ of the bounding boxes of the polytopic set-collection $\{\mathcal{P}_i\}_{i=1}^5$ of Figure 9.1 onto the $x_1$-space sorted by the region's index. Indicators for the construction of the first node level of the first dimension of the interval tree are represented in green.

the geometrical structure of the polyhedral partition[3] by utilizing the facets of the regions as separating hyperplanes to divide the polyhedral partition at each tree level. This however, can lead to a worst case combinatorial number of subdivisions of existing regions and therefore to an additional increase in the number of regions to be considered during the search procedure. The on-line point-location search time is in the best case logarithmic in the number of regions $N_\mathcal{P}$, but worst case linear in the total number of facets, which makes the procedure equivalent to sequential search in the worst case. Moreover, note that the total number of facets, $N_\mathrm{F}$, is typically larger than the original number of regions in the partition, i.e. $N_\mathrm{F} > N_\mathcal{P}$. Although the scheme works very well for polyhedral partitions that have a 'simple' geometric structure and/or have a small number of regions, it tends to be computationally prohibitive in the preprocessing time for more complex controller partitions. This is due to the fact that for each hyperplane that defines a facet of each region one has to determine on which side of the hyperplane every region lies. In the worst case $2N_\mathrm{F}N_\mathcal{P}$ linear programs need to

[3]Even though in the introduction of [TJB03a] it is mentioned that overlapping regions and 'holes' in the domain of the controller are handled by the proposed algorithm, these cases are not explicitly treated in the algorithm nor it is directly apparent how this will influence the complexity of the algorithm.

be solved, making this method untenable for moderate to large problems, i.e. greater than 10 000 regions, cf. Section 9.5.3. The memory storage requirement for the binary search tree is (in the worst case) in the order of $n_x N_{\mathrm{F}}$.

## 9.3 The Proposed Search Algorithm

The proposed search algorithm is based on minimal volume *bounding boxes* $\mathcal{B}_i$ for each region $\mathcal{P}_i$, which are defined as

$$\mathcal{B}_i := \{x \in \mathbb{R}^{n_x} \mid l_i \leq x \leq u_i\},$$

where the lower and upper bounds $l_i$ and $u_i$ are given by

$$\begin{aligned} (l_i, u_i) := \arg\min_{l,u} \quad & \mathrm{vol}\,(\mathcal{B}(l,u)) \\ \text{subj. to } & \mathcal{B}(l,u) = \{x \in \mathbb{R}^{n_x} \mid l \leq x \leq u\} \supseteq \mathcal{P}_i. \end{aligned}$$

In other words, $\mathcal{B}_i$ is the 'smallest' axis-aligned $n_x$-dimensional hyper-rectangle that contains $\mathcal{P}_i$. An example bounding box $\mathcal{B}_1$ can be seen in Figure 9.1.

**Remark 9.3.1** *Note that if the regions $\mathcal{P}_i$ are polytopes, then a minimal volume bounding box can be computed using $2n_x$ linear programs of dimension $n_x$ [BFT04].*

For a given a query point, or measured state $x(t)$, the proposed algorithm operates in two stages. First, a list $\mathcal{I}^{\mathcal{B}}$ of bounding boxes containing the point $x(t)$ is computed, i.e. $x(t) \in \mathcal{B}_i$ for all $i \in \mathcal{I}^{\mathcal{B}}$ (Section 9.3.1). Second, for each index $i \in \mathcal{I}^{\mathcal{B}}$, the region $\mathcal{P}_i$ is tested to determine if it contains $x(t)$ (Section 9.3.2). In the following $x(t)$ is simply denoted by $x$ for brevity.

The first stage of this procedure is extremely efficient and computationally 'inexpensive', since the containing bounding boxes can be reported in logarithmic time. This can be done by breaking the search down into one-dimensional range queries, which is possible due to the axis-aligned nature of the bounding boxes. The complexity of the second stage of the algorithm is a function of the overlap between the bounding boxes of adjacent regions. A significant advantage of the proposed search tree is a very simple and effective preprocessing step, which allows the method to be applied to controllers defined over a very large number of regions, i.e. several tens of thousands. As is shown in Section 9.5, there are several large problems of interest to control which have a structure that makes this procedure efficient.

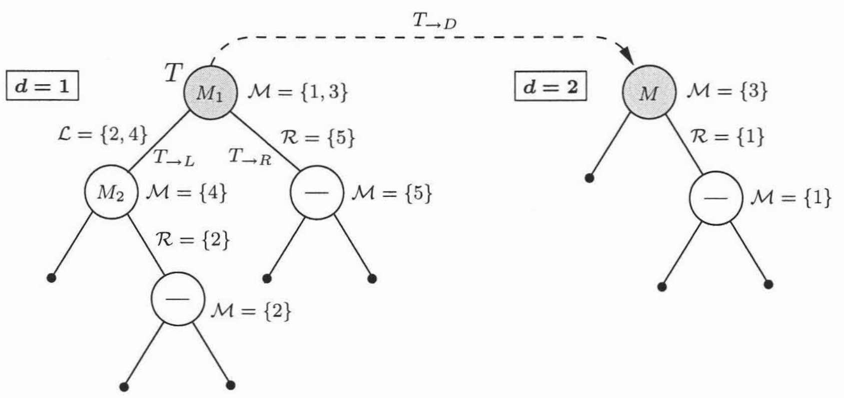


Figure 9.3: Two-dimensional interval tree for the collection of polytopes $\mathcal{P}$ in Figure 9.1. The gray indicated node in $d = 1$ is explored further in $d = 2$.

**Remark 9.3.2 (Overlapping regions)** *Note that overlapping regions are treated naturally and without any additional heuristics by the algorithm.*

### 9.3.1 Bounding Box Search Tree

In this section we will detail the procedure for reporting the set of indices $\mathcal{I}^{\mathcal{B}}$ of all bounding boxes that contain a given point $x$. The algorithm relies on the fact that one can decompose the search of a query point $x \in \mathbb{R}^{n_x}$ in a set of bounding boxes in an $n_x$-dimensional space into $n_x$ separate one-dimensional sequential or parallel searches, because the bounding boxes are all axis-aligned.

The basic steps for constructing the search tree are given in Algorithm 9.3.3.

**Algorithm 9.3.3 (Building the search tree)**

1. *compute the bounding box $\mathcal{B}_i$ for each $\mathcal{P}_i$*
2. *project each bounding box $\mathcal{B}_i$ onto each dimension $d = 1, \ldots, n_x$: define $\mathcal{B}_i^{(d)}$ as the resulting interval*
3. *build an $n_x$-dimensional interval tree*

Note that Step 2 of Algorithm 9.3.3 for axis-aligned bounding boxes is merely a coordinate extraction of the corner points $l_i$ and $u_i$.

The proposed search algorithm is an extension to the well known concept of *interval trees* [dSvO00, CLRS01]. Standard interval trees are efficiently ordered binary search trees for determining a set of possibly overlapping one-dimensional line segments that contain a given point or line segment. Consider Figure 9.2, in which the intervals of the bounding boxes in the first dimension for the example in Figure 9.1 are shown. The intervals are spread vertically, ordered by their respective index $i$, to make them easier to see.

Each node of the search tree, cf. Figure 9.3 and 9.2, is associated with a median point $M$. For example the root node $T$ in Figure 9.3 is associated with the point $M_1$ in Figure 9.2. The node splits the set of intervals into three sets: The set $\mathcal{L}$, consisting of those entirely on the left of the point $M$, $\mathcal{R}$ those entirely on the right and $\mathcal{M}$, those that intersect it. The set $\mathcal{M}$ is stored in the node and the left and right branches of the tree are formed by choosing points above and below $M$ and repeating this procedure on $\mathcal{L}$ and $\mathcal{R}$, respectively. By choosing the point $M$ to be the median

$$M := \frac{1}{2}\left(\min_{i\in\mathcal{J}}\{[l_i]_d\} + \max_{i\in\mathcal{J}}\{[u_i]_d\}\right)$$

of the considered intervals $\mathcal{J}$ at a given step, the number of intervals at each level of the tree drops logarithmically. This standard interval tree for the example in Figure 9.2 is shown in the left ($d = 1$) of Figure 9.3.

The tree can then be used on-line to determine the set $\mathcal{I}^{\mathcal{B}}$ of intervals containing a given point $[x]_1$, which is the first dimension of the query point $x$, as follows. Beginning at the root node $T$, the point $[x]_1$ is compared to the point $M_1$ associated with the root node. If we assume that the point $[x]_1$ is larger than $M_1$, then it is contained in all intervals in the set $\mathcal{M}$ whose right endpoint $[u_i]_1$ is larger than $[x]_1$, since $M_1$ is less than $[x]_1$ and is also contained in the interval. Note that this search over the set $\mathcal{M}$ can be done in logarithmic time by pre-sorting the endpoints of the intervals in $\mathcal{M}$. Finally, the tree is followed down the right branch, denoted $T_{\rightharpoondown R}$ in Figure 9.3, and this procedure is repeated recursively. If the point $[x]_1$ is less than $M_1$, then a similar procedure is carried out on the lower bounds and the left branch is followed, which is labeled $T_{\rightharpoondown L}$ in Figure 9.3.

We now extend this standard method to higher dimensions by building an interval tree over the sets $\mathcal{M}$ at each node using the next dimension $[x]_2$, i.e. $d = 2$. In Figure 9.3, the tree on the left resembles the interval tree for the first measured dimension $[x]_1$. The root node $T$ contains several elements $\mathcal{M} = \{1, 3\}$, i.e. $|\mathcal{M}| > 1$, and therefore an interval tree, labeled $T_{\rightharpoondown D}$ in Figure 9.3, over the second dimension ($d = 2$) is constructed for this node, in which only the elements $\{1, 3\}$ are considered and where the search is performed for the second dimension of the measured variable

$[x]_2$ only. This tree is shown on the right of the figure. By continuing in this manner, the approach is extended to arbitrary dimensions $n_x$.

### 9.3.2 Local Search

As already mentioned, the interval search tree only provides a list of candidates $\mathcal{I}^{\mathcal{B}}$ that are possible solutions to the point-location problem. In order to identify the particular index set $\mathcal{I} \subseteq \mathcal{I}^{\mathcal{B}}$ that actually contains the measured point $x(t)$, cf. Step 2 of Algorithm 9.1.1, a local search algorithm needs to be executed on the list of candidate regions by exhaustively testing a set membership $x \in \mathcal{P}_i$ for all $i \in \mathcal{I}^{\mathcal{B}}$.

If the cost function associated with a solution of a CFTOC Problem (5.5) is convex, one can use the approach of [BBBM01] in which the local search can be performed in $(2n_x + 2)|\mathcal{I}^{\mathcal{B}}|$ arithmetic operations.

## 9.4 Complexity

### 9.4.1 Preprocessing

The preprocessing phase for the proposed algorithm occurs in two steps. First, the bounding boxes for each region are computed, and then the $n_x$-dimensional interval tree is built. The calculation of a bounding box requires two linear programs per dimension per region. Therefore, if there are $N_{\mathcal{P}}$ regions, then the calculation of the bounding boxes requires exactly $2n_xN_{\mathcal{P}}$ linear programs of dimension $n_x$. The construction of the interval tree can be performed in $O(n_xN_{\mathcal{P}}\log(N_{\mathcal{P}}))$ [dSvO00] and as can be seen from the examples in Section 9.5, the required computation is insignificant compared to the computation of the bounding boxes.

Note that as the preprocessing for this algorithm requires two linear programs per region, it is guaranteed to take significantly less time than the initial computation of the controller. It follows that this approach can be applied to any system for which an explicit controller can be calculated. Note also that bounding boxes are computed in some parametric solvers as the solution is computed [KGB04], making the additional off-line computation negligible.

### 9.4.2 Storage

The algorithm requires the storage of a bounding box as well as a list of defining inequalities for each critical region. Furthermore the structure of the tree must be stored. A bounding box of dimension $n_x$ requires exactly $2n_x$ numbers to be stored and since each bounding box appears in exactly one node of the interval tree in each dimension, the tree has a worst case storage of $2n_x N_{\mathcal{P}}$ pointers, where the 2 is for the left and the right branch pointers of the tree.

### 9.4.3 On-line Complexity

The interval tree can be traversed in $O(\log(N_{\mathcal{P}}) + |\mathcal{I}^{\mathcal{B}}|)$ time, where $|\mathcal{I}^{\mathcal{B}}|$ is the number of intervals returned [dSvO00]. However, all current methods of doing the secondary search over the list of $|\mathcal{I}^{\mathcal{B}}|$ potential regions returned must be done in linear time. The worst-case complexity is therefore determined by the maximum number of regions that can be returned by the interval tree search, or equally the maximum number of bounding boxes that contain a single point. The efficiency of the proposed algorithm is therefore determined by the structure of the set collection $\mathcal{P}$. It is demonstrated by example in the following section that there exist control problems for which the proposed method offers a significant improvement over current approaches.

## 9.5 Numerical Examples

### 9.5.1 Constrained LTI System

The proposed algorithm of Section 9.3 was applied to the following linear system with three states and two inputs

$$x(t+1) = \begin{bmatrix} 7/10 & -1/10 & 0 \\ 1/5 & -1/2 & 1/10 \\ 0 & 1/10 & 1/10 \end{bmatrix} x(t) + \begin{bmatrix} 1/10 & 0 \\ 1/10 & 1 \\ 1/10 & 0 \end{bmatrix} u(t).$$

The system is subject to input constraints, $-5\mathbb{1}_2 \leq u(t) \leq 5\mathbb{1}_2$, and state constraints, $-20\mathbb{1}_3 \leq x(t) \leq 20\mathbb{1}_3$. The CFTOC Problem (5.5) is solved with $p = 1$, $N = 8$, $Q_x = I_3$, $Q_u = \frac{1}{10} I_2$, and $Q_{x_N} = 0_{3\times 3}$. The receding horizon state feedback control law (4.4) consists of 2 568 polyhedral regions in $\mathbb{R}^3$.

As can be seen from Table 9.1, the algorithm presented in Section 9.3 is required to solve $2 \cdot 3 \cdot 2\,568 = 15\,408$ linear programs in the preprocessing phase and needs to store 15 408 real numbers to represent the bounding boxes, as well as 4 424 pointers in order to represent the tree. Since the cost function for this example is piecewise affine and convex, it is possible to use the method in [BBBM01] for the local search, cf. Section 9.3.2, which requires an additional storage of 10 272 real numbers.

In comparison, the binary search tree of [TJB03a] for this case consists of 815 unique hyperplanes. For each such hyperplane $2N_{\mathcal{P}}$ LPs *must* be solved in the preprocessing phase to compute the index set which corresponds to 4 185 840 linear programs. An actual additional 1 184 782 linear programs are needed to construct the tree, which does not correspond to the worst case scenario.

In order to identify the control law on-line, one has to perform 707 floating point operations to traverse the interval tree in the worst case. Since the tree only provides a necessary condition for the point-location problem, one has to perform a local search on the regions identified by the tree as possible candidates (Section 9.3.2). To provide a worst case bound, an exhaustive check for all possible intersections of the intervals stored in the presented tree was performed. In the worst case 36 regions need to be checked using the method of [BBBM01], which corresponds to 216 flops. However, as can be seen from Figure 9.4, a unique control law is automatically reported by the here proposed search tree in 31 % of all cases without the requirement of doing a secondary local search. In addition, approximately 90 % of all search queries do not require an exhaustive check of more than 15 regions.

| | sequential search | Alg. in [TJB03a] | ***Alg. 9.3.3*** ( [BBBM01] locally) |
|---|---|---|---|
| number of LPs (off-line) | — | 5 370 622 | 15 408 |
| runtime (off-line) | — | 10 384 secs | 10 secs |
| on-line arithm. operations (worst case) | 106 295 | 110 | 923 |

Table 9.1: Computational complexity for the example in Section 9.5.1.

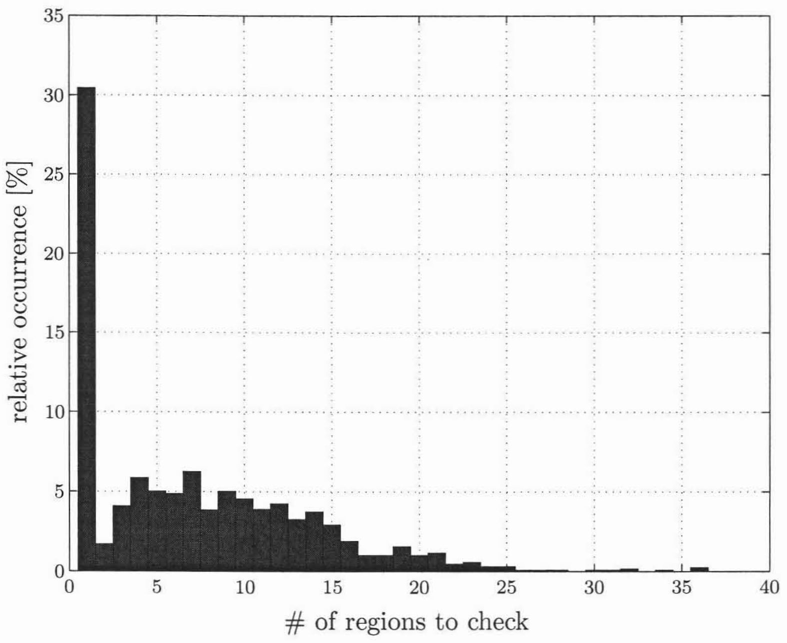


Figure 9.4: Histogram of the relative occurrence of the number of 'local' regions for the example in Section 9.5.1.

### 9.5.2 Constrained PWA System

Consider the following piecewise affine system from [MR03]

$$x(t+1) = \begin{cases} A_1 x(t) + Bu(t), & \text{if } [0,\,1,\,0]x(t) \leq 1, \\ A_2 x(t) + Bu(t) + a, & \text{otherwise,} \end{cases}$$

where

$$A_1 = \begin{bmatrix} 1 & 1/2 & 3/10 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}, A_2 = \begin{bmatrix} 1 & 1/5 & 3/10 \\ 0 & 1/2 & 1 \\ 0 & 0 & 1 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, a = \begin{bmatrix} 3/10 \\ 1/2 \\ 0 \end{bmatrix}.$$

The system is subject to input constraints, $-1 \leq u(t) \leq 1$, and state constraints, $[-10,\,-5,\,-10]' \leq x(t) \leq [10,\,5,\,10]'$. With $p = 1$, $N = 7$, $Q_x = I_{3\times3}$, $Q_u = 1/10$, and $Q_{x_N} = 0_{3\times3}$. The solution to the CFTOC Problem (5.5) resulted in a receding horizon state feedback control law (4.4) defined over 2 222 polyhedral regions in $\mathbb{R}^3$.

The off-line construction of the interval search tree for this example required 13 332 LPs to be solved, compared to $7.9 \cdot 10^6$ linear programs which are needed to construct

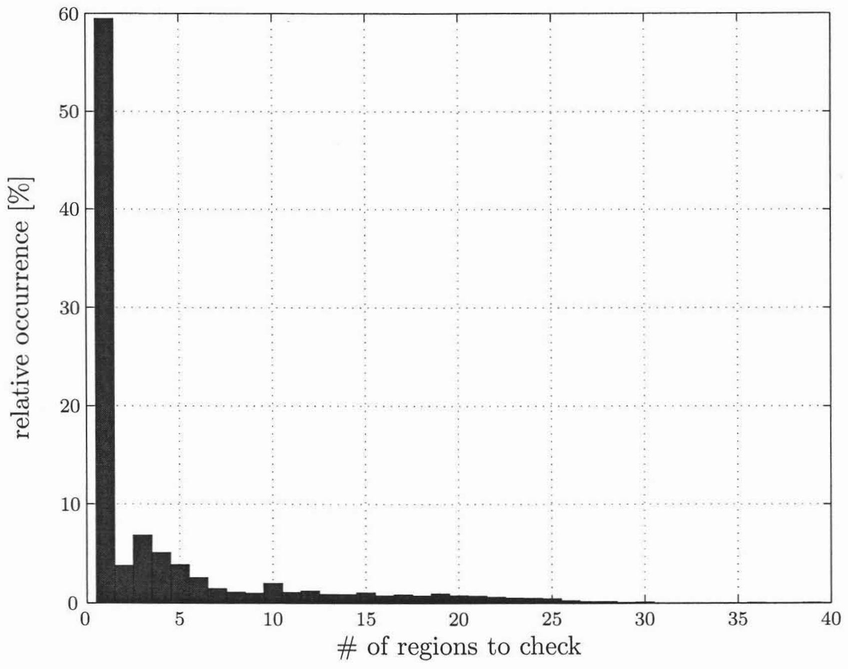


Figure 9.5: Histogram of the relative occurrence of the number of 'local' regions for the example in Section 9.5.2.

the binary search tree of [TJB03a]. (This does not correspond to the worst case scenario.) Since the cost function of a given CFTOC solution is not necessarily convex, one cannot use the method of [BBBM01], and therefore one must perform a sequential search as outlines in Section 9.3.2 on possible candidates. Using the same methodology as in the previous example, we have found that at most 39 regions have to be searched exhaustively. This however, takes at most 1 720 flops. The worst case number of floating point operations needed to traverse the interval tree is 882. Moreover, almost 60 % of all search queries result in a unique control law during the first phase of the algorithm (Section 9.3.1), cf. Figure 9.5. Therefore no additional sequential searches are necessary in these cases. Results on the computational complexity are summarized in Table 9.2.

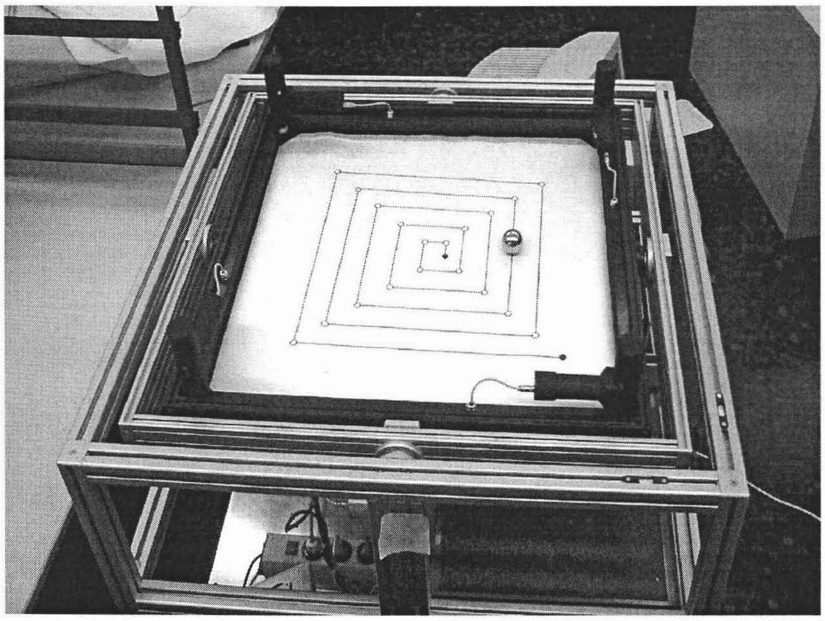


Figure 9.6: Ball & Plate laboratory setup. The ball follows a pre-specified trajectory.

| | sequential search | Alg. in [TJB03a] | ***Alg. 9.3.3*** |
|---|---|---|---|
| number of LPs (off-line) | — | 7 913 462 | 13 332 |
| runtime (off-line) | — | 12 810 secs | 4.8 secs |
| on-line arithm. operations (worst case) | 97 984 | 352 | 2 602 |

Table 9.2: Computational complexity for the example in Section 9.5.2.

### 9.5.3 Ball & Plate System

The mechanical 'Ball & Plate' system was introduced in [Bor03, Her96]. The experiment consists of a ball rolling over a gimbal-suspended plate actuated by two independent motors, cf. Figure 9.6. The control objective is to make the ball follow a prescribed trajectory, while minimizing the control effort. The dynamical model for the $y$-axis of such a device is given by

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 700 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -34.69 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 3.1119 \end{bmatrix} u(t), \tag{9.2}$$

where $x := [y,\ \dot{y},\ \alpha,\ \dot{\alpha}]'$ is the state. $-30 \leq y \leq 30$ and $-15 \leq \dot{y} \leq 15$ are the position and velocity of the ball with respect to the $y$-coordinate, $-0.26 \leq \alpha \leq 0.26$ and $-1 \leq \dot{\alpha} \leq 1$ denote the angular position and angular velocity of the plate, respectively. The input voltage to the motor is assumed to be constrained by $-10 \leq u \leq 10$. In order to take the tracking requirements into account, the state vector is extended with an additional element, which contains the reference signal, hence the augmented state vector is in $\mathbb{R}^5$. The model (9.2) was then discretized with sampling time $T_s = 0.03$ and a closed-form PWA feedback law (4.4) was derived for the CFTOC Problem (5.3), where the following parameters $N = 10$, $Q_x = \text{diag}([6, 0.1, 500, 100, 6])$, $Q_u = 1$, and $Q_{x_N} = Q$ were considered. The controller obtained using the Multi-Parametric Toolbox [KGB04] for MATLAB® is defined over 22 286 regions in $\mathbb{R}^5$.

The computational results for the respective search trees are summarized in Table 9.3. Due to the high number of regions, the binary search tree of [TJB03a] was not applicable to this example (denoted by $\star$ in Table 9.3), since it would require the solution of $3.5 \cdot 10^9$ LPs already in the preprocessing stage to determine the index set before building the binary search tree. In contrast, in the preprocessing stage, the here proposed algorithm has to solve 228 860 LPs to obtain the bounding boxes for all regions. The overall time needed to construct the complete search tree, including the computation of the bounding boxes, was just 80 seconds. 9 324 pointers are needed to represent the tree structure, and 222 860 floating point numbers are needed to describe the bounding boxes.

To estimate the average and the worst case number of arithmetic operations needed to identify the control law on-line, we have investigated 10 000 random initial conditions over the whole feasible state space. It can be seen from the histogram distribution depicted in Figure 9.7 that the search tree algorithm identifies at most 500 regions as

| | sequential search | Alg. in [TJB03a] | ***Alg. 9.3.3*** |
|---|---|---|---|
| number of LPs (off-line) | — | $3.5 \cdot 10^9$ | 228 860 |
| runtime (off-line) | — | $\star$ | 80 secs |
| on-line arithm. operations (worst case) | 1 400 178 | $\star$ | 208 849 |

Table 9.3: Computational complexity for the example in Section 9.5.3. $\star$ denotes that the algorithm in [TJB03a] is not computable for this example.

possible candidates in 86% of all tested initial conditions. The subsequent exhaustive check on 500 regions corresponds to 30 000 floating point operations. In 99% of all tested cases the algorithm identifies at most 1000 regions for subsequent local search, which corresponds to at most 60 000 flops. In the worst case, the search tree will identify as many as 2 544 regions as possible candidates for a sequential search. Notice that this number represents, in the worst case, only 11 % of the total number of regions. This amounts to a maximum of 152 640 flops, whereas traversal of the tree contributes another 56 209 flops. The sequential search on all regions, on the other hand, would require $1.4 \cdot 10^6$ operations and is currently the only method that can be applied to such a large system. The total number of flops which are needed to be performed on-line is thus reduced by one order of magnitude. To give a sensible feeling for this number of floating point operations, note that a 3 GHz Pentium 4 computer can execute approximately $800 \cdot 10^6$ flops/sec. Given this performance the controlled system can be run at a sampling rate of 4 kHz in the case of the presented search tree, whereas the sequential search has a limit of 500 Hz.

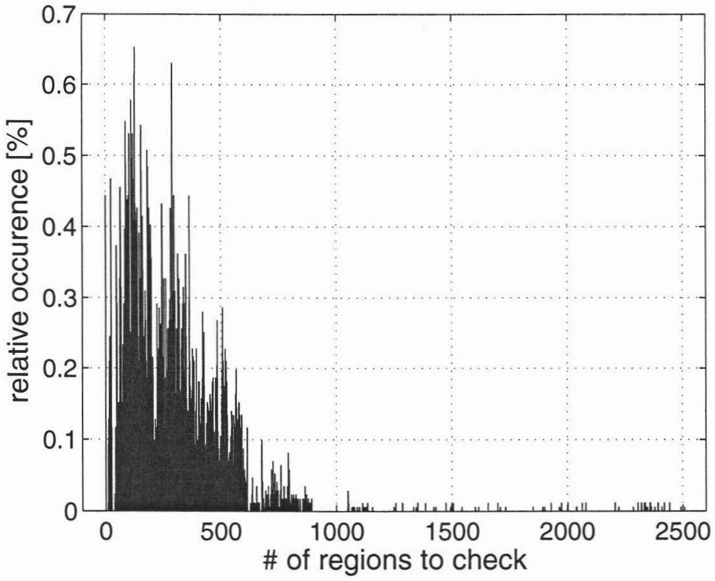


Figure 9.7: Histogram of the relative occurrence of the number of 'local' regions for the example in Section 9.5.3.

## 9.6 Conclusion

In this chapter we presented a search tree algorithm utilizing the concept of bounding boxes and interval trees for the fast on-line evaluation of piecewise closed-form control laws defined over compact regions. The power of the proposed approach lies in its ability to rapidly preprocess a large number of (possibly overlapping) regions into a tree that minimizes the on-line computational burden. The algorithm offers a significant improvement in the on-line controller evaluation and can handle much larger systems than current approaches. The procedure was compared with existing methods in the literature and its effectiveness was demonstrated for large examples

# Part III

# THE MULTI PARAMETRIC TOOLBOX

# 10

# Introduction

As already mentioned in the previous sections, optimal control of constrained linear and piecewise affine systems has garnered great interest in the research community due to the ease with which complex problems can be stated and solved. The aim of the *Multi-Parametric Toolbox* (MPT) is to provide efficient computational means to obtain feedback controllers for these types of constrained optimal control problems in a MATLAB [TM00] programming environment. As the name of the tool hints, it is mainly focused on calculation of feedback laws in the *parametric* fashion in which the feedback law takes a form of a PWA look-up table, as explained in Chapter 4. But the toolbox is also able to formulate and solve MPC problems on-line in the receding horizon fashion, i.e. by solving the optimization problem for a particular value of the initial condition at each time step.

In short, the Multi-Parametric Toolbox can be described as being a free Matlab toolbox for design, analysis and deployment of MPC-based control laws for constrained linear, nonlinear and hybrid systems. Efficiency of the code is guaranteed by the extensive library of algorithms from the field of computational geometry and multi-parametric optimization. The toolbox offers a broad spectrum of algorithms compiled in a user friendly and accessible format: starting from modeling systems which combine continuous dynamics with discrete logic (hybrid systems), through design of control laws based on different performance objectives (linear, quadratic, minimum time) to the handling of systems with persistent additive and polytopic uncertainties. Users can add custom constraints, such as polytopic, contraction, or collision avoidance constraints, or create custom objective functions. Resulting optimal control laws can either be embedded into control applications in the form of a C code, or deployed to target platforms using Real Time Workshop.

MPT can also be viewed as a unifying repository of hybrid systems design tools from international experts utilizing state-of-the-art optimization packages. The list of included software packages includes packages for linear programming (CDD [Fuk97], GLPK [Mak01]), quadratic programming (CLP), mixed-integer linear programming (GLPK), and semi-definite programming (SeDuMi [Stu99]). In addition, MPT ships with a dedicated solver for computing projections of convex polytopes, called ESP [Jon05], a boolean optimization package ESPRESSO, as well as with the HYSDEL modeling language [TBB+02].

The main factor which distinguishes this toolbox from other alternatives is the big emphasis on efficient formulation of the problems which are being solved. This means that the toolbox provides implementation of novel control design and analysis algorithms, but also offers the user an easy way to use them without the need to be an expert in the respective fields. MPT aims at providing tools which can be used in the whole chain of the process of successful control design. It allows users not only to design optimization-based controllers, but also to formally verify that they behave as desired, investigate the behavior of the closed-loop system, and to post-process the resulting feedback laws in order to simplify them without loosing prescribed design properties.

In the following chapters we present the toolbox from the user's perspective. In Chapter 11 we first describe how the user can define a dynamical model of plant to be controlled and how different types of constraints (e.g. constraints on plant states, inputs, outputs, as well as rate constraints) can be defined. In the subsequent chapter we present a novel tool, called *matrixHYSDEL*, which significantly expands the capabilities of the Hybrid Systems Description Language HYSDEL [TBB+02] used to easily define the behavior of hybrid systems in a user friendly way.

Chapter 13 then explains how different types of model predictive control problems can be formulated using the MPT framework. We illustrate that the toolbox can formulate and solve different types of control problems, ranging from finite-horizon setups, through infinite-time and minimum-time problems (cf. Section 8.1), up to low complexity strategies, represented by the $M$-step scheme presented in Section 8.3. In this chapter we also show that the user can freely modify the underlying optimization problem. This can be done either by adding custom constraints (such as constraints involving logic decisions, norms, move blocking type of constraints, etc.), or by modifying the objective function.

Chapter 14 is devoted to the analysis capabilities of the MPT toolbox. Specifically, we show that the toolbox is able to perform reachability analysis [Tor03] for the class of PWA systems. This module can be further used to verify certain properties of closed-loop control systems, including checking safety and liveness of such systems. In addition, the analysis library includes functions which can be used to calculate Lyapunov functions of different types (such as quadratic, piecewise linear, piecewise quadratic, or polynomial functions of higher order). These functions then serve as certificates of closed-loop stability. As part of this module MPT also provides functions which serve to decrease the complexity of parametric controllers, i.e. controllers composed of polyhedral regions. The reduction is achieved by performing the so-called "region merging" in which multiple controller regions are merged to larger convex parts.

In Chapter 15 we then describe ways how optimization-based controllers can be implemented on target devices in the form of a C code, and how the controllers can be used for simulations in the Simulink environment.

Visualization capabilities of the toolbox are then reported in Chapter 16, before presenting several examples in Chapter 17.

MPT on its own can also be used by users mainly concerned with computational geometry. Specifically, in Chapter 18 we show how wide range of operations on convex polytopes and nonconvex unions thereof can be performed using the toolbox. The functionality under consideration includes, but is not limited to, computation of convex hulls and convex unions of several polytopes, enumeration of extremal vertices, and calculation of Minkowski sums and Pontryagin differences of several convex polytopes. The theoretical background behind the presented methods is presented in Chapter 3.

In Chapter 19 we then summarize the review the main differences between MPT, the Hybrid Toolbox [Bem03] and the MPC toolbox, which are all tools devoted to design of MPC-based controllers. This chapter compares all three tools from the users perspective and highlights similarities and differences in the respective user interfaces and richness of offered features. In the same chapter we also review how AMPL [FGK93] (A Mathematical Programming Language) can be used to solve MPC-based problems on-line and how it compares to MPT.

Finally, in Chapter 20 we review possible goals for the future development of the Multi-Parametric Toolbox.

# 11 Modelling of LTI and PWA Systems

In this chapter we will show how to model linear and piecewise affine systems in the MPT framework. The behavior of a plant is in general driven by two major components: system dynamics and system constraints. Both these components have to be described in the system structure.

## 11.1 System Dynamics

MPT can deal with three types of discrete-time models of dynamical systems:

1. Linear Time-Invariant (LTI) models
2. Piecewise-Affine (PWA) models
3. Mixed Logical Dynamical (MLD)

This chapter only covers modeling of LTI and PWA systems, while modeling of systems in the MLD framework will be explained in Chapter 12.

### 11.1.1 LTI Dynamics

LTI dynamics can be captured by the following linear relations:

$$x(k+1) = Ax(k) + Bu(k) \tag{11.1}$$

$$y(k) = Cx(k) + Du(k) \tag{11.2}$$

where $x(k) \in \mathcal{R}^{n_x}$ is the state vector at time instance $k$, $x(k+1)$ denotes the state vector at time $k+1$, $u(k) \in \mathcal{R}^{n_u}$ and $y(k) \in \mathcal{R}^{n_y}$ are values of the control input and system output, respectively. $A$, $B$, $C$ and $D$ are matrices of appropriate dimensions, i.e. $A$ is a $n_x \times n_x$ matrix, dimension of $B$ is $n_x \times n_u$, $C$ is a $n_y \times n_x$ and $D$ a $n_y \times n_u$ matrix.

Dynamical matrices are stored in the following fields of the system structure:

```
sysStruct.A = A
sysStruct.B = B
sysStruct.C = C
sysStruct.D = D
```

**Example 11.1.1** *Assume a double integrator dynamics sampled at 1 second:*

$$x(k+1) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x(k) + \begin{bmatrix} 1 \\ 0.5 \end{bmatrix} u(k) \tag{11.3}$$

$$y(k) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x(k) + \begin{bmatrix} 0 \\ 0 \end{bmatrix} u(k) \tag{11.4}$$

*In MPT, the above described system can be defined as follows:*

```
sysStruct.A = [1 1; 0 1];
sysStruct.B = [1; 0.5];
sysStruct.C = [1 0; 0 1];
sysStruct.D = [0; 0]
```

### 11.1.2 PWA Dynamics

Piecewise-Affine systems are systems whose dynamics are affine and can be different in different parts of the state-input state. In particular, they are defined by

$$x(k+1) = A_i x(k) + B_i u(k) + f_i \tag{11.5}$$

$$y(k) = C_i x(k) + D_i u(k) + g_i \tag{11.6}$$

$$\text{if } \begin{bmatrix} x(k) \\ u(k) \end{bmatrix} \in \mathcal{D}_i \tag{11.7}$$

The subindex $i$ takes values $1 \dots N_{PWA}$, where $N_{PWA}$ is total number of PWA dynamics defined over a polyhedral partition $\mathcal{D}$. Dimensions of matrices in (11.5)–(11.7) are summarized in Table 11.1.2.

| Matrix | Dimension |
|---|---|
| A | $n_x \times n_x$ |
| B | $n_x \times n_u$ |
| f | $n_x \times 1$ |
| C | $n_y \times n_x$ |
| D | $n_y \times n_u$ |
| g | $n_y \times 1$ |

Table 11.1: Dimensions of matrices of a PWA system.

Matrices in equations (11.5) and (11.6) are stored in the following fields of the system structure:

Equation (11.7) defines a polyhedral partition of the state-input space over which the different dynamics are active. Different segments of the polyhedral partition $\mathcal{D}$ are defined using so-called *guard lines*, i.e. constraints on state and input variables. In general, the guard lines are described by the following constraints:

$$G_i^x x(k) + G_i^u u(k) \leq G_i^c \tag{11.8}$$

which means that dynamics $i$ represented by the tuple $[A_i, B_i, f_i, C_i, D_i, g_i]$ will be active in the part of state-input space which satisfies constraints (11.8). If at future time the state $x(k+T)$ or input $u(k+T)$ moves to a different sector of the polyhedral partition, say $G_j^x x(k+T) + G_j^u u(k+T) \leq G_j^c$, the dynamics will be driven by the tuple $[A_j, B_j, f_j, C_j, D_j, g_j]$, and so on.

In MPT, PWA systems are represented by the following fields of the system structure:

```
sysStruct.A = {A1, A2, ..., An}
sysStruct.B = {B1, B2, ..., Bn}
sysStruct.f = {f1, f2, ..., fn}
sysStruct.C = {C1, C2, ..., Dn}
sysStruct.D = {D1, D2, ..., Cn}
sysStruct.g = {g1, g2, ..., gn}
sysStruct.A = {A1, A2, ..., An}
sysStruct.guardX = {Gx1, Gx2, ..., Gxn}
sysStruct.guardU = {Gu1, Gu2, ..., Gun}
sysStruct.guardC = {Gc1, Gc2, ..., Gcn}
```

In PWA case, each field of the structure has to be a cell array of matrices of appropriate dimensions. Each index $i \in 1, 2, \ldots, n$ corresponds to one PWA dynamics, i.e. to one tuple $[A_i, B_i, f_i, C_i, D_i, g_i]$ and one set of constraints $G_i^x x(k) + G_i^u u(k) \leq G_i^c$

Unlike the LTI case, one can omit `sysStruct.f` and `sysStruct.g` if they are zero. All other matrices have to be defined in the structure.

## 11.2 Import of Models from External Sources

MPT can design control laws for discrete-time constrained linear, switched linear and hybrid systems. Hybrid systems can be described in Piecewise-Affine (PWA) or Mixed Logical Dynamical (MLD) representations and an efficient algorithm is provided to switch from one representation to the other form and vice-versa. To increase user's comfort, models of dynamical systems can be imported from various sources:

- Models of hybrid systems generated by the HYSDEL [TB02] and matrixHYSDEL languages,
- MLD structures generated by the function `mpt_pwa2mld`
- Nonlinear models defined by `mpt_nonlinfcn` template
- State-space and transfer function objects of the Control toolbox,
- System identification toolbox objects,
- MPC toolbox objects.

In order to import a dynamical system, one has to call

```
model=mpt_sys(object, Ts)
```

where `object` can be either a string (in which case the model is imported from a corresponding HYSDEL or matrixHYSDEL source files), or it can be a variable of one of the above mentioned object types. The second input parameter `Ts` denotes sampling time and can be omitted, in which case `Ts = 1` is assumed.

**Example 11.2.1** *The following code will first define a continuous-time state-space object which is then imported to MPT:*

```
% sampling time
Ts = 1;

% continuous-time model as state-space object
di = ss([1 1; 0 1], [1; 0.5], [1 0; 0 1], [0; 0]);

% import the model and discretize it
sysStruct = mpt_sys(di, Ts);
```

**Note:** *If the state-space object is already in discrete-time domain, it is not necessary to provide the sampling time parameter `Ts` to `mpt_sys`. After importing a model using `mpt_sys` it is still necessary to define system constraints as described previously.*

## 11.3 System Constraints

MPT allows to define following types of constraints:

- Min/Max constraints on system outputs
- Min/Max constraints on system states
- Min/Max constraints on manipulated variables
- Min/Max constraints on slew rate of manipulated variables

### 11.3.1 Constraints on System Outputs

Output equation is in general driven by the following relation for PWA systems

$$y(k) = C_i x(k) + D_i u(k) + g_i \tag{11.9}$$

and by

$$y(k) = Cx(k) + Du(k) \tag{11.10}$$

for LTI systems. It is therefore clear that by choice of $C = I$ one can use these constraints to restrict system states as well. Min/Max output constraints have to be given in the following fields of the system structure:

```
sysStruct.ymax = outmax
sysStruct.ymin = outmin
```

where `outmax` and `outmin` are $n_y \times 1$ vectors.

### 11.3.2 Constraints on System States

Constraints on system states are optional and can be defined by

```
sysStruct.xmax = xmax
sysStruct.xmin = xmin
```

where `xmax` and `xmin` are $n_x \times 1$ vectors.

### 11.3.3 Constraints on Manipulated Variables

Goal of each control technique is to design a controller which chooses a proper value of the manipulated variable in order to achieve the given goal (usually to guarantee stability, but other aspects like optimality may also be considered at this point). In most real plants values of manipulated variables are restricted and these constraints have to be taken into account in controller design procedure. These limitations are usually saturation constraints and can be captured by min / max bounds. In MPT, constraints on control input are given in:

```
sysStruct.umax = inpmax
sysStruct.umin = inpmin
```

where `inpmax` and `inpmin` are $n_u \times 1$ vectors.

### 11.3.4 Constraints on Slew Rate of Manipulated Variables

Another important type of constraints are rate constraints. These limitations restrict the variation of two consecutive control inputs ($\delta u = u(k) - u(k-1)$) to be within of prescribed bounds. One can use slew rate constraints when a "smooth" control action is required, e.g. when controlling a gas pedal in a car to prevent the car from jumping due to sudden changes of the controller action. Min/max bounds on slew rate can be given in:

```
sysStruct.dumax = slewmax
sysStruct.dumin = slewmin
```

where `slewmax` and `slewmin` are $n_u \times 1$ vectors.
**Note:** This is an optional argument and does not have to be defined. If it is not given, bounds are assumed to be $\pm\infty$.

## 11.4 Systems with Discrete Valued Inputs

MPT allows to define system with discrete-valued control inputs. This is especially important in a framework of hybrid systems where control inputs are often required to belong to certain set of values. We distinguish two cases:

1. All inputs are discrete
2. Some inputs are discrete, the rest are continuous

### 11.4.1 Purely Discrete Inputs

Typical application of discrete-valued inputs are various on/off switches, gears, selectors, etc. All these can be modelled in MPT and taken into account in controller design. Defining discrete inputs is fairly easy, all that needs to be done is to fill out

```
sysStruct.Uset = Uset
```

where `Uset` is a cell array which defines all possible values for every control input. If the system has, for instance, 2 control inputs and the first one is just an on/off switch (i.e. $u_1 = \{0, 1\}$) and the second one can take values from set $\{-5, 0, 5\}$, it can be defined as follows:

```
sysStruct.Uset{1} = [0, 1]
sysStruct.Uset{2} = [-5, 0, 5]
```

where the first line corresponds to $u_1$ and the second to $u_2$. If the system to be controlled has only one manipulated variable, the cell operator  can be omitted, i.e. one could write:

```
sysStruct.Uset = [0, 1]
```

The set of inputs doesn't have to be ordered.

### 11.4.2 Mixed Inputs

Mixed discrete and continuous inputs can be modelled by appropriate choice of `sysStruct.Uset`. For each continuous input it is necessary to set the corresponding entry to `[-Inf Inf]`, indicating to MPT that this particular input variable should be treated as a continuous input. For a system with two manipulated variables, where the first one takes values from a set $\{-2.5, 0, 3.5\}$ and the second one is continuous, one would set:

```
sysStruct.Uset{1} = [-2.5, 0, 3.5]
sysStruct.Uset{2} = [-Inf Inf]
```

## 11.5 System Structure sysStruct

System structure **sysStruct** is a structure which describes the system to be controlled. MPTcan deal with two types of systems:

1. Discrete-time linear time-invariant (LTI) systems
2. Discrete-time Piecewise Affine (PWA) Systems

Both system types can be subject to constraints imposed on control inputs and system outputs. In addition, constraints on slew rate of the control inputs can also be given.

### 11.5.1 LTI systems

In general, a constrained linear time-invariant system is defined by the following relations:

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) + Du(k) \\ \text{subt. to} & \\ y_{min} &\leq y(k) \leq y_{max} \\ u_{min} &\leq u(k) \leq u_{max} \end{aligned}$$

Such an LTI system is defined by the following mandatory fields:

```
sysStruct.A = A;
sysStruct.B = B;
sysStruct.C = C;
sysStruct.D = D;
sysStruct.ymax = ymax;
sysStruct.ymin = ymin;
sysStruct.umax = umax;
sysStruct.umin = umin;
```

Constraints on slew rate of the control input $u(k)$ can also be imposed by:

```
sysStruct.dumax = dumax;
sysStruct.dumin = dumin;
```

which enforces $\Delta u_{min} <= u(k) - u(k-1) <= \Delta u_{max}$.

**Note:** If no constraints are present on certain inputs/states, set the associated values to `Inf`.

LTI system which is subject to parametric uncertainty and/or additive disturbances is driven by the following set of relations:

$$\begin{aligned} x(k+1) &= A_{unc}x(k) + B_{unc}u(k) + w(k) \\ y(k) &= Cx(k) + Du(k) \end{aligned}$$

where $w(k)$ is an unknown, but bounded additive disturbance, i.e.

$$w(n) \in W \qquad \forall n \in (0...Inf)$$

To specify an additive disturbance, set

```
sysStruct.noise = W
```

where `W` is a polytope object bounding the disturbance. MPTalso supports lower-dimensional noise polytopes. If one wants to define noise only on a subset of system states, it can be done by defining `sysStruct.noise` as a set of vertices representing the noise. For instance, to impose a `+/- 0.1` noise on `x1`, but no noise should be used for `x2`, this can be done with:

```
sysStruct.noise = [-0.1 0.1; 0 0];
```

A polytopic uncertainty can be specified by a cell array of matrices `Aunc` and `Bunc` as follows:

```
sysStruct.Aunc = {A1, ..., An};
sysStruct.Bunc = {B1, ..., Bn};
```

### 11.5.2 PWA Systems

PWA systems are models for describing hybrid systems. Dynamical behavior of such systems is captured by relations of the following form:

$$\begin{aligned}
x(k+1) &= A_i x(k) + B_i u(k) + f_i \\
y(k) &= C_i x(k) + D_i u(k) + g_i \\
\text{subj. to} & \\
y_{min} &\leq y(k) \leq y_{max} \\
u_{min} &\leq u(k) \leq u_{max} \\
\Delta u_{min} &\leq u(k) - u(k-1) \leq \Delta u_{max}
\end{aligned}$$

Each dynamics $i$ is active in a polyhedral partition bounded by the so-called guard-lines:

$$guardX_i x(k) + guardU_i u(k) <= guardC_i$$

which means dynamics $i$ will be applied if the above inequality is satisfied.

Fields of `sysStruct` describing a PWA system are listed below:

```
sysStruct.A = {A1, ..., An}
sysStruct.B = {B1, ..., Bn}
sysStruct.C = {C1, ..., Cn}
sysStruct.D = {D1, ..., Dn}
sysStruct.f = {f1, ..., fn}
sysStruct.g = {g1, ..., gn}
sysStruct.guardX = {guardX1, ..., guardXn}
sysStruct.guardU = {guardU1, ..., guardUn}
sysStruct.guardC = {guardC1, ..., guardCn}
```

Note that all fields have to be cell arrays of matrices of compatible dimensions, $n$ stands for total number of different dynamics. If `sysStruct.guardU` is not provided, it is assumed to be zero.

System constraints are defined by:

```
sysStruct.ymax  = ymax;
sysStruct.ymin  = ymax;
sysStruct.umax  = umax;
sysStruct.umin  = umin;
sysStruct.dumax = dumax;
sysStruct.dumin = dumin;
```

Constraints on slew rate are optional and can be omitted.

MPTis able to deal also with PWA systems which are affected by bounded additive disturbances:

$$x(k+1) = A_i x(k) + B_i u(k) + f_i + w(k)$$

where the disturbance $w(k)$ is assumed to be bounded for all time instances by some polytope $W$. To indicate that the dynamical system is subject to such a disturbance, set

```
sysStruct.noise = W;
```

where `W` is a polytope object of appropriate dimension.

Mandatory and optional fields of the system structure are summarized in Tables 11.5.2 and 11.5.2, respectively.

| | |
|---|---|
| `A, B, C, D, f, g` | State-space dynamic martices in (5.1) and (11.5)–(11.7). Set elements to empty if they do not apply. |
| `umin, umax` | Bounds on inputs **umin** $\leq$ u(t) $\leq$ **umax**. |
| `ymin, ymax` | Constraints on the outputs **ymin** $\leq$ y(t) $\leq$ **ymax**. |
| `guardX, guardU, guardC` | Polytope cell array defining where the dynamics are active (for PWA systems). $\mathcal{D}_i = \{(x,u) \mid guardXi\ x + guardUi\ u \leq gu/ardCi\}$. |

Table 11.2: Mandatory fields of the system structure **sysStruct**.

| | |
|---|---|
| `Uset` | Declares discrete-valued inputs |
| `dumin, dumax` | Bounds on `dumin` $\leq$ u(t)-u(t-1) $\leq$ `dumax`. |
| `Pbnd` | Polytope limiting the feasible state-space of intersest. |

Table 11.3: Optional fields of the system structure `sysStruct`.

# 12

# Modelling of MLD Systems

## 12.1 Mixed Logical Dynamical Systems

Most of the control theory and tools have been developed for systems, whose evolution is described by smooth linear or nonlinear state transition functions. In many applications, however, the system to be controlled also comprises parts described by logic, such as for instance on/off switches or valves, gears or speed selectors, evolutions dependent on if-then-else rules. Often, the control of these systems is left to schemes based on heuristic rules inferred from practical plant operation. In the 1990s, researchers started dealing with hybrid systems, namely hierarchical systems comprising dynamical components at the lower level, governed by upper level logical/discrete components ( [GNAE93], [BBM98]). Hybrid systems arise in a large number of application areas, and have been attracting much attention in both academic theory-oriented circles as well as in industry.

In late 1990s, Tyler and Morari [TM99] and then Bemporad and Morari [BM99a] set out to establish a framework for modeling and controlling models of systems described by interacting physical laws, logical rules, and operating constraints. According to techniques described, for example, by Williams [Wil93], Cavalier et al. [CPS90] and Raman and Grossmann [RG92], propositional logic can be transformed into linear inequalities involving integer and continuous variables. Combining the logic with the continuous system we obtain mixed logical dynamical (MLD) systems described by linear dynamic equations subject to linear mixed-integer inequalities, i.e. inequalities involving both continuous and binary (or logical, or 0-1) variables:

$$x_{k+1} = Ax_k + B_1 u_k + B_2 \delta_k + B_3 z_k \tag{12.1a}$$

$$y_k = Cx_k + D_1 u_k + D_2 \delta_k + D_3 z_k \tag{12.1b}$$

$$E_2 \delta_k + E_3 z_k \leq E_1 u_k + E_4 x_k + E_5 \tag{12.1c}$$

where the state $x$, the output $y$ and the input $u$ can have continuous as well as binary components. The continuous variables $z$ are introduced when translating some propositional logic expressions into mixed-integer inequalities.

Though this may not be obvious at first sight, many practical discrete-time systems can be described in the MLD framework. MLD systems generalize a wide set of models, among which there are linear hybrid systems, finite state machines, some classes of discrete event systems, constrained linear systems, and nonlinear systems whose nonlinearities can be expressed (or, at least, suitably approximated) by piecewise linear functions. Indeed, when the described map is continuous, then Heemels et al. [HDB01] have shown that MLD systems are entirely equivalent in their expressiveness to a wide range of other system descriptions in discrete time, in particular, piecewise affine (PWA) systems, linear complementarity systems, max-plus systems, finite state machines, etc.

## 12.2 HYSDEL

In general, the derivation of an MLD model on the basis of an engineering description is a tedious task almost impossible to do by hand except for trivial example systems. Therefore we have developed a modeling language HYSDEL [TBB$^+$02] that makes the novel framework readily accessible to the engineering community.

The HYbrid System DEscription Language (HYSDEL) is a modeling language to describe hybrid systems in a textual fashion. In this subsection we will briefly introduce the language capabilities. Even though the HYSDEL description is only an abstract modeling step, the associated HYSDEL compiler then translates the description into several computational models, in particular into the MLD and PWA models.

A HYSDEL list is composed of two parts - INTERFACE and IMPLEMENTATION. The INTERFACE section contains the declarations, divided into subsections called STATE, INPUT, OUTPUT and PARAMETER, the order in which those sections are declared is not relevant. The first three declarations are referred to as variable declaration, while the last is a parameter declaration.

The second part, IMPLEMENTATION, is composed of specialized sections describing the relations among the variables. The IMPLEMENTATION section starts with an optional AUX section which contains the declarations of the internal signals of the DHA system, called also auxiliary variables. The declaration follows the general syntax of the variable declaration. The OUTPUT section allows specifying static linear and logic relations for the output vector. The HYSDEL section AD allows to define Boolean variables from continuous ones. The section LOGIC allows to specify arbitrary functions of Boolean variables. The HYSDEL section DA defines continuous variables according to if-then-else conditions on Boolean variables. The CONTINUOUS section describes the linear dynamics, expressed as difference equations. HYSDEL allows also to define a continuous variable as an affine function of continuous variables in the LINEAR section. The AUTOMATA section specifies the state transition equations of the finite state machine (FSM) as Boolean functions. Finally, the MUST section specifies constraints on continuous and Boolean variables, i.e., linear constraints and Boolean formulas. More generally, the MUST section allows also mixed constraints on states, inputs, and outputs).

Once the hybrid system is modeled properly, the HYSDEL compiler translates it into the MLD form (22.4) and optionally generates also a MATLAB simulator. Detailed description of how to obtain the MLD representation out of HYSDEL-represented model is given in [TBB$^+$02].

## 12.3 matrixHYSDEL

As already mentioned, the derivation of MLD models on the basis of an engineering description is difficult to do by hand, but can easily be automated using the modeling language HYSDEL. Although HYSDEL was successfully used in many case studies, it's modeling features are restricted to scalar variables, which can make modeling complex systems very time consuming. Because of that we developed an extension called matrixHYSDEL, which allows to use vector and matrix variables when modeling hybrid systems. In addition, repetitive tasks can be easily simplified by using matrixHYSDEL's support of nested `FOR` loops. We spend the rest of this section highlighting individual features of matrixHYSDEL which help to ease the effort of modeling of complex hybrid systems.

### 12.3.1 Vector and Matrix Variables

matrixHYSDEL extends syntax of HYSDEL models by adding the possibility to use vector and matrix variables and parameters. Dimensions of the variables can be defined in the standard MATLAB-like fashion, e.g.

```
REAL X(m, n)
```

will define a matrix variable with $m$ rows and $n$ columns whose elements are all real. It is also possible to define a vector variable by, e.g. `REAL X(m)`, which will be automatically interpreted as a $m \times 1$ vector. Bounds on variables can be defined if the variable is a vector, e.g.

```
REAL X(3) [-1, 1; -3, 0; 0, 4]
```

where each column in the matrix is separated by a comma (",") and each row is delimited by a semi-column (";").

## 12.3.2 Vector and Matrix Parameters

Vector and matrix parameters can be of two kinds. Either their value is explicitly given at compilation time, or their value is adjusted dynamically at the time when the generated MLD model is used inside of the MATLAB environment.

If symbolical parameters are used, it is necessary to define their dimensions, which will remain persistent at all further stages. Dimensions are defined in the same way as by vector and matrix variables, i.e.

```
REAL C(3, 3)
```

will define the parameter `C` as a $3 \times 3$ symbolic matrix. If a parameter is a vector, it is possible to omit the second dimension, e.g.

```
REAL D(3)
```

will define `D` as a $3 \times 1$ symbolic vector. Values of vector and matrix parameters can be specified by enclosing them in the `[,]` environment, e.g.

```
REAL A = [1, 1; 0, 1]
```

will define the parameter `A` as a $2 \times 2$ matrix. Note that it is not necessary to provide dimensions of parameters which have a defined value.

### 12.3.3 Indexing of Vectors and Matrices

Vector and matrix variables and parameters can be indexed using the MATLAB-like index operators. Indexes can be either scalar, e.g.

```
Z = X(2)
```

or defined by a range, e.g.

```
Z = X(1:3)
```

which corresponds to `Z = (X(1), X(2), X(3))`. In addition, increments can be specified as well. The increments can be both positive as well as negative, providing that they are all integer-valued. For instance

```
Z = X(1:2:4)
```

will result into `Z = (X(1), X(3))`. Variables can also be indexed by means of a non-symbolic parameter, e.g.

```
X(K)
```

where `K` is a pre-define real parameter. This case can be further extended to indexing by an indexed parameter, e.g.

```
K = [2, 3, 1]
Z = X(K(2))
```

will correspond to `Z = X(3)`. Matrix variables and parameters can also be indexed in the same fashion by providing a separate index for the rows and a separate index for the columns of the respective variables or parameters. For instance

```
Z = X(1:2, 3:4)
```

will return `Z` which contains first two rows of the 3rd and 4th columns of the matrix variable `X`.

**Example 12.3.1** *We show benefits of using vector variables and parameters on the following example. Assume we have a particle moving in a two-dimensional plane with forces in each dimensions as inputs. Furthermore we assume that movement in each direction is independent, i.e. there is no coupling. The model of such system in the* *discrete-time domain with sampling time of 1s can be thus approximated by two double integrators:*

$$\begin{pmatrix} p_x \\ s_x \\ p_y \\ s_y \end{pmatrix}^+ = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_x \\ s_x \\ p_y \\ s_y \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0.5 & 0 \\ 0 & 1 \\ 0 & 0.5 \end{pmatrix} U \tag{12.2}$$

*where $p_x$ and $p_y$ are positions of the particle in the $x$ and $y$ direction, respectively. $s_x$ and $s_y$ denote speed along each direction. Furthermore we assume there is an additional boolean input to our plant. If that input is true, forces applied to the system are amplified by a factor of 2. If the boolean input takes a false value, applied forces are not amplified, i.e.*

$$U = \begin{cases} \begin{pmatrix} F_x \\ F_y \end{pmatrix} & \textit{if } \; u_b = 0 \\ 2\begin{pmatrix} F_x \\ F_y \end{pmatrix} & \textit{if } \; u_b = 1. \end{cases} \tag{12.3}$$

*In addition, every time the force boost is applied, value of an associated counter is increased by one, i.e.*

$$b_c^+ = b_c + 1 \tag{12.4}$$

*Using a vector representations of the state vector $X = (p_x, s_x, p_y, s_y, b_c)^T$ and the input forces $F = (F_x, F_y)^T$ it is possible to model the aforementioned dynamical system by the following matrixHYSDEL code:*

```
SYSTEM xy {
  INTERFACE {
    STATE {
      REAL X(5);
    }
    INPUT {
      REAL F(2);
      BOOL u_b;
    }
    PARAMETER {
      REAL A = [1, 1, 0, 0; 0, 1, 0, 0; 0, 0, 1, 1; 0, 0, 0, 1];
      REAL B = [1, 0; 0.5, 0; 0, 1; 0, 0.5];
    }
  }
  IMPLEMENTATION {
    AUX {
      REAL Ft(2);
    }
    DA {
      Ft = {IF u_b THEN 2*F ELSE F};
    }
    CONTINUOUS {
      X(1:4) = A*X(1:4) + B*Ft;
      X(5)   = X(5) + (REAL u_b);
    }
  }
}
```

### 12.3.4 Loops

Repetitive tasks can easily be simplified using matrixHYSDEL's support of FOR-loops. A general syntax for loops is as follows:

```
<< iterator = bound1:[increment:]bound2 >>
```

where the special delimiters <<,>> enclose the cycle, **iterator** is an identifier of a variable which takes values from **bound1** to **bound2**. If the **increment** parameter is omitted, it is assumed to be equal to 1. Negative increments are also allowed, e.g.

```
<< i = 3:-1:1 >>
```

will result into i taking values 3, 2 and 1. Alternatively, values of the iterator can also be enumerated explicitly as a vector, e.g.

```
<< i = [2, 4, 5] >>
```

The identifier of the iterator must be explicitly provided in the AUX section by using the INDEX keyword. For instance

```
AUX { INDEX i; }
```

will define the variable `i` to act as an iterator in subsequent loops. The identifier must not conflict with any existing state, input, output, or other auxiliary variables. When the `FOR`-loop is used, it applies only to a statement which directly precedes the cycle itself, as will be illustrated by the following example.

**Example 12.3.2** *Consider the same system as in Example 12.3.1, but this time we assume that we want to model the movement of $N$ particles instead of just one. This goal can be achieved with matrixHYSDEL first by considering the state and input variables as matrices of appropriate size, e.g.*

```
SYSTEM xy_many {
  INTERFACE{
    PARAMETER {
      REAL N = 3;
    }
    STATE {
      REAL X(5, N);
    }
    INPUT {
      REAL F(2, N);
      BOOL u_b(N);
    }
  }
```

*where `N` denotes number of particles to consider. The state variables `X` are now defined as a $5 \times N$ matrix, with each column representing the state vector of one particle. The input forces `F` and the boosting trigger `u_b` are represented in a similar fashion.*

```
  IMPLEMENTATION {
    AUX {
      INDEX ip;
      REAL Ft(2);
    }
    DA {
      Ft(1:2, ip) = {IF u_b(ip) THEN 2*F(1:2, ip)
```

```
21.            ELSE F(1:2, ip)}                    << ip = 1:N >>;
22.       }
23.       CONTINUOUS {
24.         X(1:4, ip) = A*X(1:4, ip) + B*Ft(1:2, ip) << ip = 1:N >>;
25.         X(5, ip)   = X(5, ip) + (REAL u_b(ip))    << ip = 1:N >>;
26.       }
27.    }
28. }
```

*After the loop-counting variable* ***ip*** *was defined on line 16, we used it on lines 20, 23 and 24 to repeat given commands for all particles of our system (*$\boldsymbol{ip}= 1\ldots N$*).*

## 12.3.5 Nested Loops

`FOR`-loops can also be nested, which leads to multi-level cycles. General syntax of nested loops is as follows

```
<< i1 = L1:[inc1:]U1, i2 = L2:[inc2:]U2, ..., in = Ln:[incn:]Un >>
```

Again, it is possible to replace the range operator `Ln:[incn:]Un` with an explicit list of values. When such a nested cycle is evaluated, the inner-most iterators are incremented first. When the upper limit is reached, value of the inner-most operator is decreased back to it's original value, and value of the iterator directly preceding this one is incremented. To illustrate this, suppose we have the following `FOR`-cycle which consists of two iterators:

```
<< i = 1:2, j = 0:1 >>
```

When such a cycle is evaluated, following sequence of values of the iterator variables will be obtained:

```
i=1, j=0
i=1, j=1
i=2, j=0
i=2, j=1
```

**Example 12.3.3** *We extend Example 12.3.2 by adding further specifications that velocity of each particle along every axis must be restricted to be between* $\pm 10$ *units per* *time. We recall that the matrix of state variables as defined in Example 12.3.2 was defined as a* $5 \times N$ *matrix, where* $N$ *defines number of particles in our system. Hence the state vector corresponding to particle* $i$ *was* `X(1:5, i)`*. By further recalling system definition* (12.2) *we know that the velocities in respective directions are the 2nd and 4th components of the state vector. Since the objective is to add constraints on these elements of the state vector of each particle, we can do so by means of nested loops in the* `MUST` *section:*

```
AUX {
  INDEX ip, ix;
}
MUST {
  X(ix, ip) <=  10    << ip = 1:N, ix = [2, 4] >>;
  X(ix, ip) >= -10    << ip = 1:N, ix = [2, 4] >>;
}
```

Here the iterator `ip` loops through all particles and the iterator `ix` cycles through the states of interest.

### 12.3.6 Vector Functions

matrixHYSDEL defines three new internal functions which simplify operations with vector variables. The function `sum` sums up all elements of a given vector, i.e.

```
Z = sum(X);
```

is equivalent to writing `Z=X(1)+X(2)+...+X(n)`. Indexed vectors can also be provided as an input argument of the `sum` function, e.g.

```
Z = sum(X([1, 3, 4]))
```

will sum up only the 1st, 3rd and 4th element of a given vector `X`, i.e. it is identical to writing `Z=X(1)+X(3)+X(4)`.

The function `all` can be used to impose a logical AND condition on all elements of a given vector, e.g.

```
Z = all(B)
```

is identical to writing `Z = B(1) & B(2) & ... & B(n)` and the variable `Z` will take a true value if and only if all elements of the vector `B` are true. Again, indexed vectors can be used as input arguments of the function.

The function `any` is used to impose a logical OR condition on all elements of a given vector, e.g.

```
Z = any(B)
```

is identical to writing `Z = B(1) | B(2) | ... | B(n)`, i.e. `Z` will be true if any of the elements of the vector `B` are true. Again, indexed vectors can be used as input arguments of the function.

## 12.4 Import of Models Generated by HYSDEL and matrixHYSDEL into MPT

Once a model of a hybrid system is defined in the HYSDEL or matrixHYSDEL languages, it can be imported into MPT using the same approach as described in Chapter 11. Specifically, one can use the function `mpt_sys` to perform such import. The user has to provide the name of the file which contains description of the modeled system, e.g. for HYSDEL:

```
sysStruct = mpt_sys('hysdelfile.hys', Ts);
```

or, for matrixHYSDEL:

```
sysStruct = mpt_sys('matrixhysdelfile.mhys', Ts);
```

**Note:** Hybrid systems modeled in HYSDEL are already defined in the discrete-time domain, the additional sampling time parameter `Ts` is only used to set the sampling interval for simulations. If `Ts` is not provided, it is set to 1.

The description of the hybrid system defined in the file `hysdelfile.hys` is first transformed into an MLD model of the form 22.4 using the HYSDEL (matrixHYSDEL) compilers. Subsequently, an equivalent PWA representation of the same system is created automatically. It is possible to avoid the PWA transformation by calling

```
sysStruct = mpt_sys('hysdelfile.hys', Ts, 'nopwa');
```

or, for matrixHYSDEL:

```
sysStruct = mpt_sys('matrixhysdelfile.mhys', Ts, 'nopwa');
```

The import function also extracts whichever constraints have been defined in the original source code and automatically fills out the required fields of the system structure `sysStruct`.

# 13

# Control Design

## 13.1 Controller Computation

For constrained linear and hybrid systems, MPT can design optimal and sub-optimal control laws either in implicit form, where an optimization problem of finite size is solved on-line at every time step and is used in a Receding Horizon Control (RHC) manner or, alternatively, solve an optimal control problem in a multi-parametric fashion. If the latter approach is used, an explicit representation of the control law is obtained.

The solution to an optimal control problem can be obtained by a simple call of `mpt_control`. The general syntax to obtain an explicit representation of the control law is:

```
ctrl = mpt_control(sysStruct, probStruct)
```

On-line MPC controllers can be generated by

```
ctrl = mpt_control(sysStruct, probStruct, 'online')
```

Based on the system definition described by `sysStruct` (cf. Section 11.5) and problem description provided in `probStruct` (cf. Section 13.8), the main control routine automatically calls one of the functions reported in Table 13.1 to calculate the explicit solution to a given problem. `mpt_control` first verifies if all mandatory fields in `sysStruct` and `probStruct` structures are filled out. If not, the procedure will break with an appropriate error message. Note that the validation process sets the optional fields to default values if they are not present in the two respective structures. Again, an appropriate message is displayed.

Once the control law is calculated, the solution (here `ctrl`) is returned as an instance of the `mptctrl` object. Internal fields of this object, described in Section 13.2, can be accessed directly using the sub-referencing operator. For instance

| System | N | Suboptimality | Problem | Function | Reference |
|---|---|---|---|---|---|
| LTI | fixed | 0 | CFTOC | `mpt_optControl` | [Bao02, Bor03] |
| LTI | `Inf` | 0 | CITOC | `mpt_optInfControl` | [GBTM04] |
| LTI | `Inf` | 1 | CMTOC | `mpt_iterative` | [GM03, GPM03] |
| LTI | `Inf` | 2 | LowComp | `mpt_oneStepCtrl` | [GM03, GPM03] |
| PWA | fixed | 0 | CFTOC | `mpt_optControlPWA` | [BBBM03, KM02, Bor03] |
| PWA | `Inf` | 0 | CITOC | `mpt_optInfControlPWA` | [BCM03a] |
| PWA | `Inf` | 1 | CMTOC | `mpt_iterativePWA` | [GKBM04] |
| PWA | `Inf` | 2 | LowComp | `mpt_iterativePWA` | [GKBM04] |

Table 13.1: List of control strategies applied to different system and problem definitions.

```
Pn = ctrl.Pn;
```

will return the polyhedral partition of the explicit controller defined in the variable `ctrl`.

Control laws can further be analyzed and/or implemented by functions reported in Chapters 15 and 16.

MPT provides a variety of control routines which are being called from `mpt_control`. Solutions to the following problems can be obtained depending on the properties of the system model and the optimization problem. One of the following control problems can be solved:

A. Constrained Finite Time Optimal Control (CFTOC) Problem.

B. Constrained Infinite Time Optimal Control Problem (CITOC).

C. Constrained Minimum Time Optimal Control (CMTOC) Problem.

D. Low complexity setup.

The problem which will be solved depends on the parameters of the system and the problem structure, namely on the type of the system (LTI or PWA), prediction horizon (fixed or infinity) and the level of sub-optimality (optimal solution, minimum-time solution, low complexity). Different combinations of these three parameters lead to a different optimization procedure, as reported in Table 13.1.

## 13.2 Fields of the `mptctrl` object

The controller object includes all results obtained as a solution of a given optimal control problem. In general, it describes the obtained control law and can be used both for analysis of the solution, as well as for an implementation of the control law.

Fields of the object are summarized in Table 13.2. Every field can be accessed using the standard . (dot) sub-referencing operator, e.g.

```
Pn = ctrl.Pn;
Fi = ctrl.Fi;
runtime = ctrl.details.runtime;
```

| | |
|---|---|
| `Pn` | The polyhedral partition over which the control law is defined is returned in this field. It is, in general, a polytope array. |
| `Fi, Gi` | The PWA control law for a given state $x(k)$ is given by `u = Fi{r} x(k) + Gi{r}`. `Fi` and `Gi` are cell arrays. |
| `Ai, Bi, Ci` | Value function is returned in these three cell arrays and for a given state $x(k)$ can be evaluated as `J = x(k)' Ai{r} x(k) + Bi{r} x(k) + Ci{r}` where the prime denotes the transpose and $r$ is the index of the active region, i.e. the region of `Pn` containing the given state $x(k)$. |
| `Pfinal` | In this field, the maximum (achieved) feasible set is returned. In general, it corresponds to the union of all polytopes in `Pn`. |
| `dynamics` | A vector which denotes which dynamics is active in which region of `Pn`. (Only important for PWA systems.) |
| `details` | More details about the solution, e.g. total run time. |
| `overlaps` | Boolean variable denoting whether regions of the controller partition overlap. |
| `sysStruct` | System description in the `sysStruct` format. |
| `probStruct` | Problem description in the `probStruct` format. |

Table 13.2: Fields of MPT controller objects.

## 13.3 Design of Custom MPC Problems

MPTby means of the function `mpt_ownmpc` allows one to add (almost) arbitrary constraints to an MPC setup and to define a custom objective functions.

First we explain the general usage of the functionality. The design of custom MPC controllers is divided into three phases:

1. *Design phase.* In this part, general constraints and a corresponding cost function are designed
2. *Modification phase.* In this part, the user is allowed to add custom constraints and/or to modify the cost function
3. *Computation phase.* In this part, either an explicit or an on-line controller which respects user constraints is computed.

### 13.3.1 Design Phase

Aim of this step is to obtain constraints which define a given MPC setup, along with an associated cost function, and variables which represent system states, inputs and outputs at various prediction steps. In order to obtain said elements for the case of explicit MPC controllers, call:

```
>> [CON, OBJ, VARS] = mpt_ownmpc(sysStruct, probStruct)
```

or, for on-line MPC controllers, call:

```
>> [CON, OBJ, VARS] = mpt_ownmpc(sysStruct, probStruct, 'online')
```

Here the variable `CON` represents a set of constraints, `OBJ` denotes the optimization objective and `VARS` is a structure with the fields `VARS.x` (predicted states), `VARS.u` (predicted inputs) and `VARS.y` (predicted outputs). Each element is given as a cell array, where each element corresponds to one step of the prediction (i.e. `VARS.x1` denotes the initial state `x0`, `VARS.x2` is the first predicted state `x1`, etc.) If a particular variable is a vector, it can be indexed directly to refer to a particular element, e.g. `VARS.x3(1)` refers to the first element of the 2nd predicted state (i.e. `x2`).

### 13.3.2 Modification Phase

The MPC setup can be modified by adding own constraints and/or by modifying the objective function. The examples below given more information about this topic.

**Note:** It is strongly recommended to always add constraints on system states (`sysStruct.xmin`, `sysStruct.xmax`), inputs (`sysStruct.umin`, `sysStruct.umax`) and outputs (`sysStruct.ymin`, `sysStruct.ymax`) when designing a controller for PWA/MLD systems, or if logic constraints are going to be added. Not adding the constraints will cause the resulting optimization problem to be badly scaled, which can have bad impact on performance of numerical solvers.

### 13.3.3 Computation Phase

Once the constraints and/or the objective have been modified according to one's needs, either an explicit controller can be computed by

```
>> ctrl = mpt_ownmpc(sysStruct, probStruct, CON, OBJ, VARS)
```

or an on-line MPC controller can be constructed by calling

```
>> ctrl = mpt_ownmpc(sysStruct, probStruct, CON, OBJ, VARS, 'online')
```

**Example 13.3.1 (Polytopic constraints)** *Assume that we would like to introduce polytopic constraints of the form $Hx_k \leq K$ on each predicted state, including the initial state $x_0$. To do that, we simply add these constraints to our set* ***`CON`****:*

```
for k = 1:length(VARS.x)
  CON = CON + set(H * VARS.x{k} <= K);
end
```

*At this point one can continue with computation phase described above, which will return a controller which respects given constraints.*

**Example 13.3.2 (Polytopic constraints)** *We now extend the previous example and add the specification that polytopic constraints should only be applied on the 1st, 3rd and 4th predicted state, i.e. on $x_1$, $x_3$ and $x_4$. It is important to notice that the variables contained in the* ***`VARS`*** *structure are organized in cell arrays, where the first element of* ***`VARS.x`*** *corresponds to $x_0$, i.e. to the initial condition. Therefore to meet or specifications, we would write following code:*

```
for k = [1 3 4],
  % VARS.x{1} corresponds to x(0)
  % VARS.x{2} corresponds to x(1)
  % VARS.x{3} corresponds to x(2)
  % VARS.x{4} corresponds to x(3)
  % VARS.x{5} corresponds to x(4)
  % VARS.x{6} corresponds to x(5)
  CON = CON + set(H * VARS.x{k+1} <= K);
end
```

**Example 13.3.3 (Move blocking)** *Assume that we want to use more complicated move blocking with following properties:* $u_0 = u_1$, $(u_1 - u_2) = (u_2 - u_3)$, *and* $u_3 = Kx_2$. *These requirements can be implemented by*

```
% VARS.u{1} corresponds to u(0)
% VARS.u{2} corresponds to u(1), and so on

% u_0 == u_1
>> CON = CON + set(VARS.u{1} == VARS.u{2});

% (u_1-u_2) == (u_2-u_3)
>> CON = CON + set((VARS.u{2}-VARS.u{3}) == (VARS.u{3}-VARS.u{4}));

% u_3 == K*x_2
>> CON = CON + set(VARS.u{4} == K * VARS.x{3});
```

**Example 13.3.4 (Mixed constraints)** *As illustrated in the move blocking example above, one can easily create constraints which involve variables at various stages of the prediction. In addition, it is also possible to add constraints which involve different types of variables. For instance, we may want to add a constraint that the sum of control inputs and system outputs at each step must be between certain bounds. This specification can be expressed by:*

```
for k = 1:length(VARS.u)
  CON = CON + set(lowerbound < VARS.y{k} + VARS.u{k} < upperbound);
end
```

**Example 13.3.5 (Equality constraints)** *Assume that we want to add a constraint that the sum of all predicted control actions along the prediction horizon should be equal to zero. This can easily be done by*

```
>> CON = CON + set((VARS.u{1}+VARS.u{2}+VARS.u{3}+...+VARS.u{end}) == 0);
```

*or, in a vector notation:*

```
>> CON = CON + set(sum([VARS.u{:}]) == 0);
```

**Example 13.3.6 (Constraints involving norms)** *We can extend the previous example and add a specification that the sum of absolute values of all predicted control actions should be less than some given bound. To achieve this goal, we can make use of the 1-norm function, which exactly represents the sum of absolute values of each element:*

```
>> CON = CON + set(norm([V.u{:}], 1) <= bound);
```

*The same can of course be expressed in a more natural form:*

```
>> CON = CON + set(sum(abs([V.u{:}])) <= bound);
```

**Example 13.3.7 (Contraction constraints)** *The norm-type constraints can be used to define "contraction" constraints, i.e. constraints which force state $x_{k+1}$ to be closer to the origin (in the 1/Inf-norm sense) than the state $x_k$ has been:*

```
for k = 1:length(VARS.x)-1
  CON = CON + set(norm(VARS.x{k+1}, 1) <= norm(VARS.x{k}, 1));
end
```

*Note that these types of constraints are not convex and resulting problems will be difficult to solve (time-wise).*

**Example 13.3.8 (Obstacle avoidance)** *It is a typical requirement of control synthesis to guarantee that the system states will avoid some set of "unsafe" states (typically an obstacle or a set of dangerous operating conditions). These kinds of problems can be solved by MPTif one adds suitable constraints. If we want, for instance, the system to avoid a given polytopic set of states, we would proceed as follows:*

```
% first define set of unsafe states
>> Punsafe = polytope(H, K);

% now define the complement of the "usafe" set versus some large box,
% to obtain the set of states which are "safe":
>> Pbox = unitbox(dimension(Punsafe), 100);
>> Psafe =  Pbox \ Punsafe;

% now add constraints that each predicted state must be inside
% of the "safe" set of states
for k = 1:length(VARS.x)
  CON = CON + set(ismember(VARS.x{k}, Psafe));
end
```

*Here* ***set(ismember(VARS.xk, Psafe))*** *will impose a constraint which tells MPTthat it must guarantee that the state $x_k$ belongs to at least one polytope of the polytope array* ***Psafe****, and hence avoiding the "unsafe" set* ***Punsafe****. Notice that this type of constraints requires binary variables to be introduced, making the optimization problem difficult to solve.*

**Example 13.3.9 (Logic constraints)** *Logic constraints in the form of IF-THEN conditions can be added as well. For example, we may want to require that if the first predicted input $u_0$ is smaller or equal to zero, then the next input $u_1$ has to be bigger than 0.5:*

```
% if u(0) <= 0 then u(1) must be >= 0.5
>> CON = CON + set(implies(VARS.u{1} <= 0, VARS.u{2} >= 0.5));
```

*Notice that this constraint only acts in one direction, i.e. if $u_0 \leq 0$ then $u_1 \geq 0.5$, but it does enforce any particular value of $u_1$ if $u_0 > 0$.*

*To add an "if and only if" constraint, use the* ***iff()*** *operator:*

```
% if u(0) <= 0 then u(1) >= 0.5, and
% if u(0) > 0 then u(1) < 0.5
>> CON = CON + set(iff(VARS.u{1} <= 0, VARS.u{2} >= 0.5));
```

*which will guarantee that if $u_0 > 0$, then the value of $u_1$ will be smaller than 0.5.*

**Example 13.3.10 (Custom optimization objective)** *In the last example we have shown how to define custom objective functions. Depending on the value of* **`probStruct.norm`***, the objective can either be quadratic, or linear.*

*To write a custom cost function, we simply sum up the terms we want to penalize. For instance, the standard quadratic cost function can be defined by hand as follows:*

```
OBJ = 0;
for k = 1:length(VARS.u),
  % cost for each step is given by x'*Q*x + u'*R*u
  OBJ = OBJ + VARS.x{k}' * Q * VARS.x{k};
  OBJ = OBJ + VARS.u{k}' * R * VARS.u{k};
end

% cost for the last predicted state x_N'*P_N*x_N
OBJ = OBJ + VARS.x{end}' * P_N * VARS.x{end};
```

*For 1/Inf-norm cost functions, one can use the overloaded* **`norm()`** *operator, e.g.*

```
OBJ = 0;
for k = 1:length(VARS.u),
  % cost for each step is given by ||Q*x|| + ||R*u||
  OBJ = OBJ + norm(Q * VARS.x{k}, Inf);
  OBJ = OBJ + norm(R * VARS.u{k}, Inf;
end

% cost for the last predicted state ||P_N*x_N||
OBJ = OBJ + norm(P_N * VARS.x{end}, Inf);
```

*If we now want to penalize deviations of predicted outputs and inputs from a given time-varying trajectories, we can do so by defining a cost function as follows:*

```
yref = [4 3 2 1];
uref = [0 0.5 0.1 -0.2]
OBJ = 0;
for k = 1:length(yref)
  OBJ = OBJ + (VARS.y{k} - yref(k))' * Qy * (VARS.y{k} - yref(k));
  OBJ = OBJ + (VARS.u{k} - uref(k))' * R * (VARS.u{k} - uref(k));
end
```

## 13.4 Soft Constraints

Since MPT 2.6 it is possible to denote certain constraints as soft. This means that the respective constraint can be violated, but such a violation is penalized. To soften certain constraints, it is necessary to define the penalty on violation of such constraints:

- `probStruct.Sx` - if given as a "nx" x "nx" matrix, all state constraints will be treated as soft constraints, and violation will be penalized by the value of this field.
- `probStruct.Su` - if given as a "nu" x "nu" matrix, all input constraints will be treated as soft constraints, and violation will be penalized by the value of this field.
- `probStruct.Sy` - if given as a "ny" x "ny" matrix, all output constraints will be treated as soft constraints, and violation will be penalized by the value of this field.

In addition, one can also specify the maximum value by which a given constraint can be exceeded:

- `probStruct.sxmax` - must be given as a "nx" x 1 vector, where each element defines the maximum admissible violation of each state constraints.
- `probStruct.sumax` - must be given as a "nu" x 1 vector, where each element defines the maximum admissible violation of each input constraints.
- `probStruct.symax` - must be given as a "ny" x 1 vector, where each element defines the maximum admissible violation of each output constraints.

The aforementioned fields also allow to specify that only a subset of state, input, or output constraint should be treated as soft constraints, while the rest of them remain hard. Say, for instance, that we have a system with 2 states and we want to soften only the second state constraint. Then we would write:

```
>> probStruct.Sx = diag([1 1000])
>> probStruct.sxmax = [0; 10]
```

Here `probStruct.sxmax(1)=0` tells MPT that the first constraint should be treated as a hard constraint, while we are allowed to exceed the second constraints by at most 10 and every such violation will be penalized by the factor of 1000.

## 13.5 Control of Time-Varying Systems

MPT can use models of time-varying systems for the synthesis of optimal controllers. This approach can be used either to design control laws for systems whose parameters vary with respect to time, or for systems whose constraints are time-dependent. To tell MPT that it should consider a time-varying system, one system structure for each step of the prediction has to be defined, e.g.

```
>> Double_Integrator
>> S1 = sysStruct;
>> S2 = sysStruct; S2.C = 0.9*S1.C;
>> S3 = sysStruct; S3.C = 0.8*S1.C;
```

Here we have three different models which differ in the `C` element. Now we can define the time-varying model as a cell array of system structures by

```
>> model = {S1, S2, S3};
>> probStruct.N = 3;
```

Note that order of systems in the `model` variable determines that the system `S1` will be used to make predictions of states `x(1)`, while the predicted value of `x(2)` will be determined based on model `S2`, and so on. Once the model is defined, one can now compute either the explicit, or an on-line MPC controller using the standard syntax:

```
>> explicitcontroller = mpt_control(model, probStruct)
>> onlinecontroller   = mpt_control(model, probStruct, 'online')
```

Systems with time-varying constraints can be defined in a similar fashion, e.g.

```
>> Double_Integrator
>> S1 = sysStruct; S1.ymax = [5; 5]; S1.ymin = [-5; -5];
>> S2 = sysStruct; S2.ymax = [4; 4]; S2.ymin = [-4; -4];
>> S3 = sysStruct; S3.ymax = [3; 3]; S3.ymin = [-3; -3];
>> S4 = sysStruct; S4.ymax = [2; 2]; S4.ymin = [-2; -2];
>> probStruct.N = 4;
>> ctrl = mpt_control({S1, S2, S3, S4}, probStruct);
```

One can go as far as combining different classes of dynamical systems at various stages of the predictions, for instance one can arbitrarily combine linear, Piecewise-Affine (PWA) and Mixed Logical Dynamical (MLD) systems. For instance, one can use a detailed PWA model for the first prediction, while having a simple LTI model for the rest:

```
>> pwa_DI; pwa = sysStruct;                 % PWA model with 4 dynamics
>> Double_Integrator; lti = sysStruct;      % simple LTI model
>> probStruct.N = 5;
>> model = {pwa, pwa, lti, lti, lti};
>> ctrl = mpt_control(model, probStruct);
```

## 13.6 On-line MPC for Nonlinear Systems

MPTallows to solve on-line MPC problems based on nonlinear or piecewise nonlinear systems. In order to define models of such systems, one has to create a special function based on the `mpt_nonlinfcn.m` template. Once the describing function is defined, one can use the `mpt_sys` function to convert it into format suitable for further computation:

```
>> sysStruct = mpt_sys(@function_name)
```

where `function_name` is the name of the function which contains description of the system. An on-line MPC controller can then be constructed using the standard syntax:

```
>> ctrl = mpt_control(sysStruct, probStruct, 'online');
```

or

```
>> [C, O, V] = mpt_ownmpc(sysStruct, probStruct, 'online');
% modify constraints and objective as needed
>> ctrl = mpt_ownmpc((sysStruct, probStruct, C, O, V, 'online');
```

The resulting controller can be used either in Simulink, or in Matlab-based simulations invoked either by

```
>> u = ctrl(x0);
```

or by

```
>> [X, U, Y] = sim(ctrl, x0, number_of_simulation_steps)
>> simplot(ctrl, x0, number_of_simulation_steps)
```

**Note:** It should be no surprise for the reader that generic nonlinear problems are *very* difficult to solve. MPT, at this stage, only formulates the nonlinear problem and passes it to an external nonlinear solver. Also note that currently only polynomial type of nonlinearities is supported, i.e. no `1/x` terms or `log/exp` functions are allowed.

## 13.7 Move Blocking

Move blocking is a popular technique used to decrease complexity of MPC problems. In this strategy the number of free control moves is usually kept low, while some of the control moves are assumed to be fixed. To enable move blocking in MPT, define the control horizon in

```
>> probStruct.Nc = Nc;
```

where `Nc` specifies the number of free control moves, and this value should be less than the prediction horizon `probStruct.N`. Control moves $u_0$ up to $u_{N_c-1}$ will be then treated as free control moves, while $u_{N_c}, \dots, u_{N-1}$ will be kept identical to $u_{N_c-1}$, i.e.

```
u_(Nc-1) == u_Nc == u_(Nc+1) == ... == u_(N-1)
```

## 13.8 Problem Structure `probStruct`

Problem structure `probStruct` is a structure which states an optimization problem to be solved by MPT.

### 13.8.1 One and Infinity Norm Problems

The optimal control problem with a linear performance index is given by:

$$\begin{aligned} \min_{u(0),\dots,u(N-1)} \quad & ||P_N x(N)||_p + \sum_{k=0}^{N-1} ||Ru(k)||_p + ||Qx(k)||_p \\ \text{subj. to} \quad & \\ & x(k+1) = f_{dyn}(x(k), u(k), w(k)) \\ & u_{min} \leq u(k) \leq u_{max} \end{aligned}$$

$$
\begin{aligned}
&\Delta u_{min} \leq u(k) - u(k-1) \leq \Delta u_{max}\\
&y_{min} \leq g_{dyn}(x(k), u(k)) \leq y_{max}\\
&x(N) \in T_{set}
\end{aligned}
$$

where:

| | |
|---|---|
| $u$ | vector of manipulated variables over which the optimization is performed |
| $N$ | prediction horizon |
| $p$ | linear norm, can be 1 or `Inf` for 1- and Infinity-norm, respectively |
| $Q$ | weighting matrix on the states |
| $R$ | weighting matrix on the manipulated variables |
| $P_N$ | weight imposed on the terminal state |
| $u_{min}, u_{max}$ | constraints on the manipulated variable(s) |
| $\Delta u_{min}, du_{max}$ | constraints on slew rate of the manipulated variable(s) |
| $y_{min}, y_{max}$ | constraints on the system outputs |
| $T_{set}$ | terminal set |

the function $f_{dyn}(x(k), u(k), w(k))$ is the state-update function and is different for LTI and for PWA systems (see Section 11.5 for more details).

## 13.8.2 Quadratic Cost Problems

In case of a performance index based on quadratic forms, the optimal control problem takes the following form:

$$
\begin{aligned}
\min_{u(0),\dots,u(N-1)} \quad & x(N)^T P_N x(N) + \sum_{k=0}^{N-1} u(k)^T R u(k) + x(k)^T Q x(k)\\
\text{subj. to} \quad & \\
& x(k+1) = f_{dyn}(x(k), u(k), w(k))\\
& u_{min} \leq u(k) \leq u_{max}\\
& \Delta u_{min} \leq u(k) - u(k-1) \leq \Delta u_{max}\\
& y_{min} \leq g_{dyn}(x(k), u(k)) \leq y_{max}\\
& x(N) \in T_{set}
\end{aligned}
$$

If the problem is formulated for a fixed prediction horizon $N$, we refer to it as to Constrained Finite Time Optimal Control (CFTOC) problem. If $N$ is infinity, the Constrained Infinite Time Optimal Control (CITOC) problem is formulated. Objective of the optimization is to choose the manipulated variables such that the performance index is minimized.

### 13.8.3 Mandatory Fields

In order to specify which problem the user wants to solve, mandatory fields of the problem structure `probStruct` are listed in Table 13.3.

| | |
|---|---|
| `probStruct.N` | prediction horizon |
| `probStruct.Q` | weights on the states |
| `probStruct.R` | weights on the inputs |
| `probStruct.norm` | either 1 or `Inf` for linear cost, or 2 for quadratic cost objective |
| `probStruct.subopt_lev` | level of optimality |

Table 13.3: Mandatory fields of the problem structure `probStruct`.

### 13.8.4 Level of Optimality

MPT can handle different setups of control problems. Specifically:

1. The cost-optimal solution that leads a control law which minimizes a given performance index. This strategy is enforced by

   ```
   probStruct.subopt_lev = 0
   ```

   The cost optimal solution for PWA systems is currently supported only for linear performance index, i.e. `probStruct.norm = 1` or `probStruct.norm = Inf`.

2. Another possibility is to use the time-optimal solution, i.e. the control law will push a given state to an invariant set around the origin as fast as possible. This strategy usually leads to simpler control laws, i.e. fewer controller regions are generated. This approach is enforced by

   ```
   probStruct.subopt_lev = 1
   ```

3. The last option is to use a low-complexity control scheme. This approach aims at constructing a one-step solution and subsequently a PWQ or PWA Lyapunov function computation is performed to verify stability properties. The approach generally results in a small number of regions and asymptotic stability as well as closed-loop constraint satisfaction is guaranteed. If one wants to use this kind of solution, he/she should set:

   ```
   probStruct.subopt_lev = 2
   ```

### 13.8.5 Optional Fields

Optional fields are summarized in Table 13.4.

| | |
|---|---|
| `probStruct.Qy` | used for output regulation. If provided the additional term $\|Q(y - y_{ref})\|_p$ is introduced in the cost function and the controller will regulate the output(s) to the given references (usually zero, or provided by `probStruct.yref`. |
| `probStruct.tracking` | 0/1/2 flag<br>**0** – no tracking, resulting controller is a state regulator which drives all system states (or outputs, if probStruct.Qy is given) towards origin<br>**1** – tracking with $\Delta u$-formulation. The controller will drive the system states (or outputs, if `probStruct.Qy` is given) to a given reference. The optimization is performed over the difference of manipulated variables $(u(k)-u(k-1))$, which involves an extension of the state vector by $nu$ additional states where $nu$ is the number of system inputs.<br>**2** – tracking without $\Delta u$-formulation. The same as `probStruct.tracking=1` with the exception that the optimization is performed over $u(k)$, i.e. no $\Delta u$-formulation is used and no state vector extension is needed. Note, however, that offset-free tracking cannot be guaranteed with this setting.<br>Default setting is `probStruct.tracking = 0`. |
| `probStruct.yref` | instead of driving a state to zero, it is possible to reformulate the control problem and rather force the output to zero. To ensure this task, define `probStruct.Qy` which penalizes the difference of the actual output and the given reference. |
| `probStruct.P_N` | weight on the terminal state. If not specified, it is assumed to be zero for quadratic cost objectives, or $P_N = Q$ for linear cost. |
| `probStruct.Nc` | control horizon. Specifies the number of free control moves in the optimization problem. |
| `probStruct.Tset` | a polytope object describing the terminal set. If not provided and `probStruct.norm = 2`, the invariant LQR set around the origin will be computed automatically to guarantee stability properties. |

Table 13.4: Optional field of the `probStruct` structure.

# 14

# Analysis and Post-Processing

The toolbox offers broad functionality for analysis of hybrid systems and verification of safety and liveliness properties of explicit control laws. In addition, stability of closed-loop systems can be verified using different types of Lyapunov functions.

## 14.1 Reachability Computation

MPT can compute forward $N$-steps reachable sets for linear and hybrid systems assuming the system input either belongs to some bounded set of inputs, or when the input is driven by some given explicit control law.

To compute the set of states which are reachable from a given set of initial conditions `X0` in `N` steps assuming system input $u(k) \in \mathcal{U}_0$, one has to call:

```
R = mpt_reachSets(sysStruct, X0, U0, N);
```

where `sysStruct` is the system structure, `X0` is a polytope which defines the set of initial conditions ($x(0) \in \mathcal{X}_0$), `U0` is a polytope which defines the set of admissible inputs and `N` is an integer which specifies for how many steps should the reachable set be computed. The resulting reachable sets `R` are returned as a polytope array. We illustrate the computation on the following example:

**Example 14.1.1** *First we define the dynamical system for which we want to compute reachable sets*

```
% define matrices of the state-space object
A = [-1 -4; 4 -1]; B = [1; 1]; C = [1 0]; D = 0;
syst = ss(A, B, C, D);
Ts = 0.02;
```

```
% create a system structure by discretizing the continous-time model
sysStruct = mpt_sys(syst, Ts);

% define system constraints
sysStruct.ymax = 10; sysStruct.ymin = -10;
sysStruct.umax = 1; sysStruct.umin = -1;
```

*Now we can define a set of initial conditions* `X0` *and a set of admissible inputs* `U0` *as polytope objects.*

```
% set of initial states
X0 = polytope([0.9 0.1; 0.9 -0.1; 1.1 0.1; 1.1 -0.1]);

% set of admissible inputs
U0 = unitbox(1,0.1);  % inputs should be such that |u| <= 0.1
```

*Finally we can compute the reachable sets.*

```
N = 50;
R = mpt_reachSets(sysStruct, X0, U0, N);

% plot the results
plot(X0, 'r', R, 'g');
```

*The reachable sets (green) as well as the set of initial conditions (red) are depicted in Figure 14.1.*

To compute reachable sets for linear or hybrid systems whose inputs are driven by an explicit control law, the following syntax can be used:

```
R = mpt_reachSets(ctrl, X0, N);
```

where `ctrl` is the controller object as generated by `mpt_control`, `X0` is a polytope which defines a set of initial conditions ($x(0) \in \mathcal{X}_0$), and `N` is an integer which specifies for how many steps should the reachable set be computed. The resulting reachable sets `R` are again returned as polytope array.

**Example 14.1.2** *In this example we illustrate the reachability computation on the* double integrator *example*

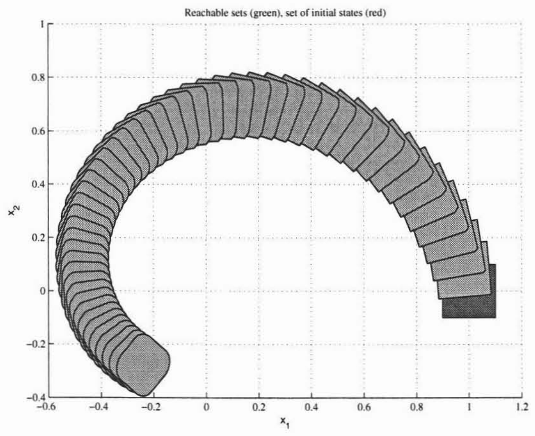


Figure 14.1: Reachable sets for Example 14.1.1.

```
% load system and problem parameters
Double_Integrator

% compute explicit controller
ctrl = mpt_control(sysStruct, probStruct);

% define the set of initial conditions
X0 = unitbox(2,1) + [3;0];

% compute the 5-Steps reachable set
N = 5;
R = mpt_reachSets(ctrl, X0, N);

% plot results
plot(ctrl.Pn, 'y', X0, 'r', R, 'g');
```

*The reachable sets (green) as well as the set of initial conditions (red) are depicted on top of the controller regions (yellow) in Figure 14.2.*

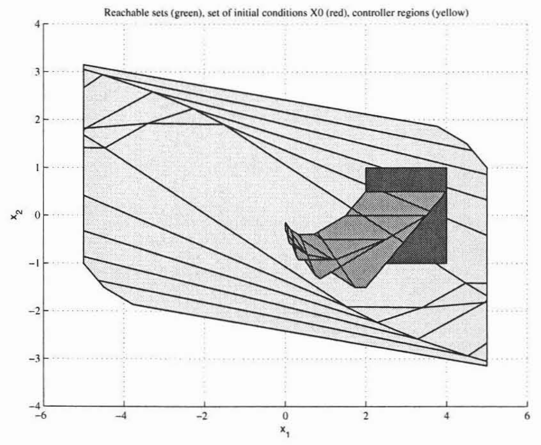


Figure 14.2: Reachable sets for Example 14.1.2.

## 14.2 Verification

Reachability computation can be directly extended to answer the following question: *Do the states of a dynamical system (whose inputs either belong to some set of admissible inputs, or whose inputs are driven by an explicit control law) enter some set of "unsafe" states in a given number of steps?*

**Example 14.2.1** *In this example we show how to answer the verification question for the first case, i.e. system inputs belong to some set of admissible inputs ($u(k) \in \mathcal{U}_0$). Although we use a linear system here, exactly the same procedure applies to hybrid systems in PWA representation as well.*

```
% define matrices of the state-space object
A = [-1 -4; 4 -1]; B = [1; 1]; C = [1 0]; D = 0;
syst = ss(A, B, C, D);
Ts = 0.02;

% create a system structure by discretizing the continous-time model
sysStruct = mpt_sys(syst, Ts);

% define system constraints
sysStruct.ymax = 10; sysStruct.ymin = -10;
sysStruct.umax = 1; sysStruct.umin = -1;
```

```
% define the set of initial condintions as a polytope object
X0 = polytope([0.9 0.1; 0.9 -0.1; 1.1 0.1; 1.1 -0.1]);

% set of admissible inputs as a polytope object
U0 = unitbox(1,0.1);  % inputs should be such that |u| <= 0.1

% set of final states (the ''unsafe'' states)
Xf = unitbox(2,0.1) + [-0.2; -0.2];

% number of steps
N = 50;

% perform verification
[canreach, Nf] = mpt_verify(sysStruct, X0, Xf, N, U0);
```

*If the system states can reach the set `Xf`, `canreach` will be* true, *otherwise the function will return* false. *In case `Xf` can be reached, the optional second output argument `Nf` will return the number of steps in which `Xf` can be reached from `X0`.*

**Example 14.2.2** *It is also possible to answer the verification question if the system inputs are driven by an explicit control law:*

```
% load dynamical system
Double_Integrator

% compute explicit controller
expc = mpt_control(sysStruct, probStruct);

% define set of initial condintions as a polytope object
X0 = unitbox(2,1) + [3;0];

% set of final states (the ''unsafe'' states)
Xf = unitbox(2,0.1) + [-0.2; -0.2];

% number of steps
```

| lyaptype | Type of Lyapunov function |
|---|---|
| 'quad' | Common quadratic Lyapunov function |
| 'sos' | Common sum-of-squares Lyapunov function |
| 'pwa' | Piecewise affine Lyapunov function |
| 'pwq' | Piecewise quadratic Lyapunov function |
| 'pwp' | Piecewise polynomial Lyapunov function |

Table 14.1: Allowed values of the `functiontype` parameter in `mpt_lyapunov`.

```
N = 10;

% perform verification
[canreach, Nf] = mpt_verify(expc, X0, Xf1, N);
```

## 14.3 Lyapunov-type Stability Analysis

In terms of stability analysis, MPT offers functions which aim at identifying quadratic, sum-of-squares, piecewise quadratic, piecewise affine or piecewise polynomial Lyapunov functions. If such a function is found, it can be used to show stability of the closed-loop systems even in cases where no such guarantee can be given a priori based on the design procedure. To compute a Lyapunov function, one has to call

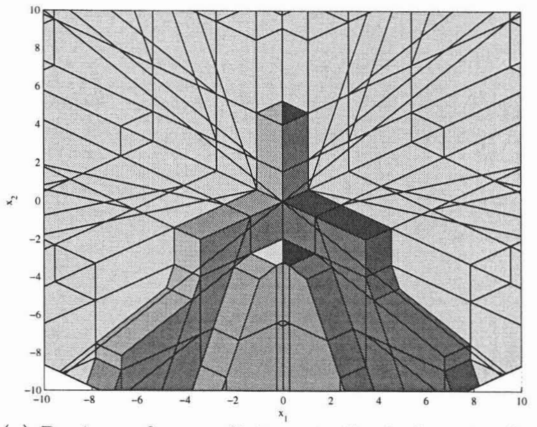
```
ctrl_lyap = mpt_lyapunov(ctrl, lyaptype}
```

where `ctrl` is an explicit controller and `lyaptype` is a string parameter which defines the type of a Lyapunov function to compute. Allowed values of the second parameter are summarized in Table 14.1. Parameters of the Lyapunov function, if one exists, will be stored in

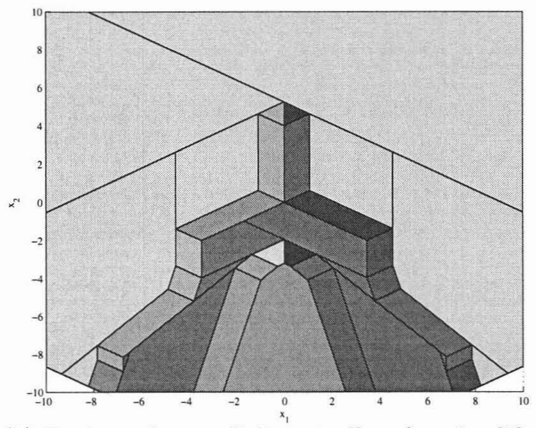
```
lyapfunction = ctrl_lyap.details.lyapunov
```

## 14.4 Complexity Reduction

MPT also addresses the issue of complexity reduction of the resulting explicit control laws. As explained in more detail in Chapter 9, the on-line evaluation of explicit control laws involves checking which region of the controller contains a given measured state. Although such an effort is usually small, it can become prohibitive for complex controllers with several thousands or even more regions. Therefore MPT allows to reduce this complexity by simplifying the controller partitions over which the control law is defined. This simplification is performed by merging regions which contain the same expression of the control law. By doing so, the number of regions may be greatly reduced, while maintaining the same performance as the original controller. The results of the merging procedure for a sample explicit controller of a hybrid system is depicted in Figure 14.3.



(a) Regions of an explicit controller before simplification (252 regions).



(b) Regions of an explicit controller after simplification (39 regions).

Figure 14.3: Region merging results.

To simplify the representation of a given explicit controller by merging regions which contain the same control law, one has to call:

```
ctrl_simple = mpt_simplify(ctrl)
```

If the function is called as indicated above, a heuristic merging will be used. It is also possible to use optimal merging based on boolean minimization:

```
ctrl_simple = mpt_simplify(ctrl, 'optimal')
```

Note, however, that the optimal merging can be prohibitive for dimensions above 2 due to an exponential complexity of the merging procedure [Gey05].

# 15

# Implementation of Control Laws

## 15.1 Evaluation of Explicit Control Laws

The control law obtained as a result of `mpt_control` is stored in a respective controller object `mptctrl` (see Section 13.2 for more details). The explicit controller takes the form a of Piecewise Affine control law where the actual control action is given by

$$U(k) = F_i^r x(k) + G_i^r \tag{15.1}$$

where the superindex $r$ denotes the active region, i.e. the region which contains the given state $x(k)$.

In the controller structure, the matrices $F_i$ and $G_i$ are stored as cell arrays, i.e.

```
ctrl.Fi = { Fi{1} Fi{2} ...  Fi{n} }
ctrl.Gi = { Gi{1} Gi{2} ...  Gi{n} }
```

Regions of the state-space, where each affine control (15.1) is active, are stored as a polytope array in the following field:

```
ctrl.Pn = [ Pn(1) Pn(2) ... Pn(n)]
```

Moreover, the expression of the value function is stored in

```
ctrl.Ai = { Ai{1} Ai{2} ... Ai{n} }
ctrl.Bi = { Bi{1} Bi{2} ... Bi{n} }
ctrl.Ci = { Ci{1} Ci{2} ... Ci{n} }
```

Therefore the cost associated with a given state $x(k)$ can easily be obtained by simply evaluating the cost expression, which is defined by

$$J = x(k)^T A_i^r x(k) + B_i^r x(k) + C_i^r \tag{15.2}$$

Therefore the procedure to obtain the control action for a given state $x(k)$ reduces to a simple membership-test. First, the index of the active region $r$ has to be identified. Since the polyhedral partition is a polytope object, the function `isinside` will return indices of regions which contain the given state $x(k)$. Since certain types of optimization problems naturally generate overlapping regions, the active region corresponds to the region in which the cost expression (15.2) is minimal. Once the active region is identified, the control action is calculated according to (15.1) and can be applied to the system.

If the optimal control problem was solved for a fixed prediction horizon $N$, the evaluation (15.1) gives a vector of control moves which minimize the given performance criterion, i.e.

$$U \triangleq [u(0)^T u(1)^T ... u(N)^T]^T \tag{15.3}$$

When applying the obtained control law in closed-loop, only the first input $u(0)$ is extracted from the sequence $U$ and is applied to the system. This policy is refereed to as the Receding Horizon Policy.

The algorithm to identify the active control law is summarized below:

**Algorithm 15.1.1** ***getInput***$(x_0, P_n, F_i, G_i, A_i, B_i, C_i)$

***Input:*** *Polyhedral partition $P_n$, PWA control law $F_i$, $G_i$, matrices of the cost expression $A_i, B_i, C_i$.*

***Output:*** *Optimal control action $U$ associated to a given state, Index of the active region $r$*

```
1. Identify regions of $P_n$ which contain the point $x_0$. Denote array of of the associated regions by $R$.
2. IF $R = \emptyset$, return ERROR - No associated control law found.
3. Set $\mathcal{J} = \emptyset$
4. FOR each element of $R$ DO
    a) $J = x_0^T A_i^r x_0 + B_i^r x_0 + C_i$
    b) Add the ordered pair $\{J, r\}$ to $\mathcal{J}$
5. END FOR
6. Identify the minimal cost from the set of ordered pairs $\mathcal{J}$.
```

| | |
|---|---|
| `U` | Feedback control law obtained by (15.1) |
| `feasible` | Boolean variable (0/1) denoting whether there is at least one region which contains the point `x0` in it's interior |
| `region` | Index of the active region in `ctrl.Pn` |
| `cost` | Cost associated with the given state `x0` |
| `ctrl` | Controller structure |
| `x0` | State vector |
| `Options` | Additional optional arguments |

Table 15.1: Input and output arguments of `mpt_getInput`.

7. *Extract from $\mathcal{J}$ the region $r$ associated to the minimal cost*
8. *Compute the optimal input sequence $U = Kx_0 + F_i^T x_0 + G_i$ ($K$ will be zero unless feedback pre-stabilization enabled.*
9. *Return $U$, $r$*

## 15.2 Implementation

The Algorithm 15.1.1 is implemented by the function `mpt_getInput`. The syntax of the function is the following

```
[U, feasible, region, cost] = mpt_getInput(ctrl, x0)
[U, feasible, region, cost] = mpt_getInput(ctrl, x0, Options)
```

where the input and arguments are described in Table 15.1

The function returns the optimizer $U$ associated with a region in which the cost expression (15.2) is minimal. If there is no region associated with a given state $x_0$, the variable `feasible` will be set to **0** (zero).

Unless specified otherwise, the function `mpt_getInput` returns only the first element of the sequence $U$ (15.3), i.e. `U = u(0)`, which can be directly applied to the system to obtain the successor state $x(k+1)$. If the user wants to return the full sequence $U$, `Options.openloop` has to be set to `1`.

The above described function (`mpt_getInput`) processes the controller structure as an input argument. If, for any reason, the solution to a given multi-parametric program was obtained by a direct call to `mpt_mpLP` or `mpt_mpQP`, the function

```
[U, feasible, region]=mpt_getOptimizer(Pn, Fi, Gi, x0, Options)
```

can be used to extract the sequence of arguments which minimize the given performance criterion. Note that unlike Algorithm 15.1.1, `mpt_getOptimizer` does not take into account overlaps. This is because overlapping regions are (usually) not generated by mpLP and mpQP algorithms which are implemented in MPT.

The function `sim` calculates the open-loop or closed-loop state evolution from a given initial state $x_0$. In each time step, the optimal control action is calculated according to Algorithm 15.1.1 by calling `mpt_getInput`. Subsequently, the obtained control move is applied to the system to obtain the successor state $x(k+1)$. The evolution is terminated once the state trajectory reaches the origin. Because of numerical issues, a small box centered at origin is constructed and the evolution is stopped as soon as all states enter this small box. The size of the box can be specified by the user. For tracking problems, the evolution is terminated when all states reach their respective reference signals. The validation of input and output constraints is performed automatically and the user is provided with a textual output if the bounds are exceeded.

General syntax is the following:

```
[X,U,Y]=sim(ctrl,x0)
[X,U,Y]=sim(ctrl,x0,N)
[X,U,Y]=sim(ctrl,x0,N,Options)
[X,U,Y,cost,feasible]=sim(ctrl,x0,N)
[X,U,Y,cost,feasible]=sim(ctrl,x0,N,Options)
```

where the input and output arguments are summarized in Table 15.2. **Note:** If the third argument is an empty matrix (`N = []`), the evolution will be automatically stopped when system states (or system outputs) reach a given reference point with a pre-defined tolerance.

The trajectories can be visualized using the `simplot` function:

```
simplot(ctrl)
simplot(ctrl, x0)
simplot(ctrl, x0, N)
```

If `x0` is not provided and the controller partition is in $\mathbb{R}^2$, the user will be able to specify the initial state just by clicking on the controller partition.

| | |
|---|---|
| `X` | Matrix containing evolution of the system states, i.e. $X = [x(0)x(1)...x(n+1)]^T$ |
| `U` | Matrix containing the control actions applied at each time step, i.e. $U = [u(0)u(1)...u(n)]^T$ |
| `Y` | Matrix containing the evolution of system outputs, i.e. $Y = [y(0)y(1)...y(n)]^T$ |
| `cost` | Overall cost obtained as a sum of (15.2). |
| `feasible` | Boolean variable (0/1) denoting whether there is at least one region which contains the point `x0` in it's interior |
| `ctrl` | Controller object |
| `x0` | Initial conditions |
| `N` | Number of steps for which the evolution should be computed. |
| `Options` | Additional optional arguments |

Table 15.2: Input and output arguments of the `sim` function.

## 15.2.1 Using different dynamical system in `sim` and `simplot`

It is possible to specify different dynamical systems to be used in simulations. In such a case the control actions obtained by a given controller can be applied to a different system than that which was used for computing the controller:

```
sim(ctrl, system, x0, N, Options)
simplot(ctrl, system, x0, N, Options)
```

Note that the `N` and `Options` arguments are optional. The user can specify his/hers own dynamics in two ways:

1. By setting the `system` parameter to a system structure, i.e.

   ```
   sim(ctrl, sysStruct, x0, N, Options)
   ```

2. By setting the `system` parameter to a handle of a function which will provide updates of system states in a discrete-time fashion:

   ```
   sim(ctrl, @sim_function, x0, N, Options)
   ```

   Take a look at `help di_sim_fun` on how to write simulation functions compatible with this function.

## 15.3 Simulink Library

The MPT Simulink library can be accessed by starting

```
>> mpt_sim
```

on Matlab command prompt. At this time the library offers 3 blocks:

The `MPT Controller` block supplies the control action as a function of the measured state. Auxiliary state/output references can be provided for tracking controllers (`probStruct.tracking = 1|2`). If the controller is an explicit one, it is possible to directly compile a Simulink model which includes one or more of the `MPT Controller` blocks using the Real Time Workshop.

The `Dynamical System` block serves for simulations of constrained linear and hybrid systems described by means of the MPT `sysStruct` structures. The user must specify initial values of the state vector in a dialog box.

The `In polytope` block returns *true* if a input point lies inside of a given polytope, *false* otherwise. If the polytope variable denotes a polytope array, the output of this block will be the index of a region which contains a given point. If no such region exists, 0 (zero) will be returned.

## 15.4 Export of Controllers to C-code

It is possible to export explicit controllers to standalone code using

```
  mpt_exportc(ctrl)
  mpt_exportc(ctrl, filename)
```

If the function is called with only one input argument, a file called `mpt_getInput.h` will be created in the working directory. It is possible to change the filename by providing a second input argument to `mpt_exportc`. The header file is then compiled along with `mpt_getInput.c` and the target application:

```
% generate an explicit controller using 'mpt_control'
>> Double_Integrator
>> controller = mpt_control(sysStruct, probStruct);

% export the explicit controller to C-code
```

```
>> mpt_exportc(controller);

% compile the example
>> !gcc mpt_example.c -o mpt_example
```

## 15.5 Export of Search Trees to C-code

If a binary search tree was calculated for a given controller by calling `mpt_searchTree`, it is possible to export such tree into a standalone C-file by calling

```
>> mpt_exportST(ctrl, filename)
```

where the `filename` argument specifies the name of the file which should be created. The controller `ctrl` used in this example must have the search tree stored inside. If it does not, use the `mpt_searchTree` function to calculate it first:

```
>> ctrl = mpt_searchTree(ctrl);
```

# 16

# Visualization

MPT provides broad range of functionality for visualization of polytopes, polyhedral partitions, control laws, value functions, general PWA and PWQ functions defined over polyhedral partitions. Part of the functions operate directly on the resulting controller object `ctrl` obtained by `mpt_control`, while the other functions accept more general input arguments.

## 16.1 Plotting of Polyhedral Partitions

The explicit solution to a optimal control problem results in a PWA control law which is defined over regions of a polyhedral partition. If the solution was obtained by a call to `mpt_control`, it is returned in the form of the controller object `ctrl`, which encompasses the polyhedral partition over which the control law is defined (see Section 13.2 for more details). The polyhedral partition `ctrl.Pn` is a polytope object and can therefore be plotted using the overloaded `plot` function. However, MPT provides also a more sophisticated plotting method, where, depending on the type of the solution, regions are depicted in approapriate colors which helps to understand behavior of the controller. This kind of plot is obtained by a call to

```
plot(ctrl)
```

i.e. the `plot` function is overloaded to accept `mptctrl` objects directly.

If `ctrl` contains a solution to Constrained Infinite Time Optimal Control Problem, or to Constrained Time Optimal Control Problem, the regions are depicted in a red-green shading. Generally speaking, red regions are close to the origin, while the more green color the region contains, the more steps will be needed to reach the origin.

## 16.2 Visualization of Closed-loop and Open-loop Trajectories

Once the explicit solution to a given optimal control problem is obtained, the resulting control law can be applied to the original dynamical system. MPT provides several ways of visualizing the closed-loop and open-loop evolution of state trajectories. As mentioned in Chapter 15, the PWA feedback law which corresponds to a given state $x(k)$ has to be identified and evaluated in order to obtain the successor state $x(k+1)$. Moreover, when applying the RHC strategy, this procedure has to be repeated at each time instance. The function `sim` described in Section 15.2 can be used to perform this repeated evaluation, and subsequently returns evolution of state, input and output trajectories assuming the initial state $x(0)$ was provided. To visualize the computed trajectories, the following command can be used:

```
simplot(ctrl)
```

which allows to pick up the initial state $x(0)$ by a mouse click, providing the controller object represents an explicit controller and dimension of the associated polyhedral partition is equal to 2. Subsequently, the state trajectory is calculated and plotted on top of the polyhedral partition over which the control law is defined. If the solution was obtained for a tracking problem, the user is first prompted to choose the reference point, again by a mouse click. Afterwards, the initial state $x(0)$ has to be selected. Finally, the evolution of states is again plotted on top of the polyhedral partition.

If the same command is used with additional input arguments, e.g.

```
simplot(ctrl, x0, horizon)
```

then the computed trajectories are visualized with respect to time. The system is not limited in dimension or number of manipulated variables. Unlike the point-and-click interface, the initial point $x(0)$ has to be provided by the user. In addition, the maximal number of steps can be specified in `horizon`. If this variable is missing, or set to an empty matrix, the evolution will continue until the origin (or the reference point for tracking problems) is reached. Additional optional argument `Options` can be provided to specify additional requirements. Similarly, as described by Section 15.2.1, also the `simplot` function allows the user to use different system dynamics when calculating the system evolution.

## 16.3 Visualization of General PWA and PWQ Functions

A piecewise affine function is defined by

$$f(x) = L^r x + C^r \qquad \text{if } x \in P_n^r \tag{16.1}$$

where the superindex $r$ indicates that the expression for the function is different in every region $r$ of a polyhedral partition $P_n$.

Piecewise Quadratic functions can be described as follows

$$f(x) = x^T M^r x + L^r x + C^r \qquad \text{if } x \in P_n^r \tag{16.2}$$

Again, expression for the cost varies in different regions of the polyhedral set $P_n$. MPT allows one to visualize both aforementioned types of functions. For instance, the command

```
mpt_plotPWA(Pn, L, C)
```

plots the PWA function (16.1) defined over the polyhedral partition `Pn`. Typical application of this function is to visualize the control law and value function obtained as a solution to a given optimal control problem. For the first case (visualization of control action), one would type:

```
mpt_plotPWA(ctrl.Pn, ctrl.Fi, ctrl.Gi)
```

since the control law is affine over each polytope of `ctrl.Pn`.

**Note:** The function supports 2-D partitions only.

To visualize the value function, one simply calls

```
mpt_plotPWA(ctrl.Pn, ctrl.Bi, ctrl.Ci)
```

to get the desired result. The same limitation applies also in this case.

Piecewise quadratic functions defined by (16.2) can be plotted by function

```
mpt_plotPWQ(Pn, Q, L, C, meshgridpoints)
```

Inputs are the polytope array `Pn`, cell arays `Q`, `L` and `C`. When plotting a PWQ function, the space covered by `Pn` has to be divided into a mesh grid. The fourth input argument (`meshgridpoints`) states into how many points should each axis of the space of interest be divided. Default value for this parameter is 30. Note that dimension of Pn has to be at most 2.

MPT provides a "shortcut" function to plot value of the control action with respect to the polyhedral partition directly, without the need to pass each input (Pn, L, C) separately:

```
mpt_plotU(ctrl)
```

If the function is called with a valid controller object, value of the control action in each region will be depicted. If the polyhedral partition Pn contains overlapping regions, the user will be prompted to use the appropriate reduction scheme (mpt_removeOverlaps) first to get a proper result.

Similarly, values of the cost function associated to a given explicit controller can be plotted by

```
mpt_plotJ(ctrl)
```

Also in this case the partition is assumed to contain no overlaps.

# 17

# Examples

In order to obtain a feedback controller, it is necessary to specify both a system as well as the problem.

**Example 17.0.1**
*We demonstrate the procedure on a simple second-order double integrator, with bounded input $|u| \leq 1$ and output $||y(k)||_\infty \leq 5$.*

```
>> sysStruct.A=[1 1; 0 1];              % x(k+1)=Ax(k)+Bu(k)
>> sysStruct.B=[0 1];                   % x(k+1)=Ax(k)+Bu(k)
>> sysStruct.C=[1 0; 0 1];              % y(k)=Cx(k)+Du(k)
>> sysStruct.D=[0;0];                   % y(k)=Cx(k)+Du(k)

>> sysStruct.umin=-1;                   % Input constraints umin<=u(k)
>> sysStruct.umax=1;                    % Input constraints u(k)<=umax
>> sysStruct.ymin=[-5 -5]';             % Output constraints ymin<=y(k)
>> sysStruct.ymax=[5 5]';               % Output constraints y(k)<=ymax
>> sysStruct.xmin = [-5; -5];           % State constraints x(k)>=xmin
>> sysStruct.xmax = [5; 5];             % State constraints x(k)<=xmax
```

For this system we will now formulate the problem with quadratic cost objective in (5.3) and a prediction horizon of $N = 5$:

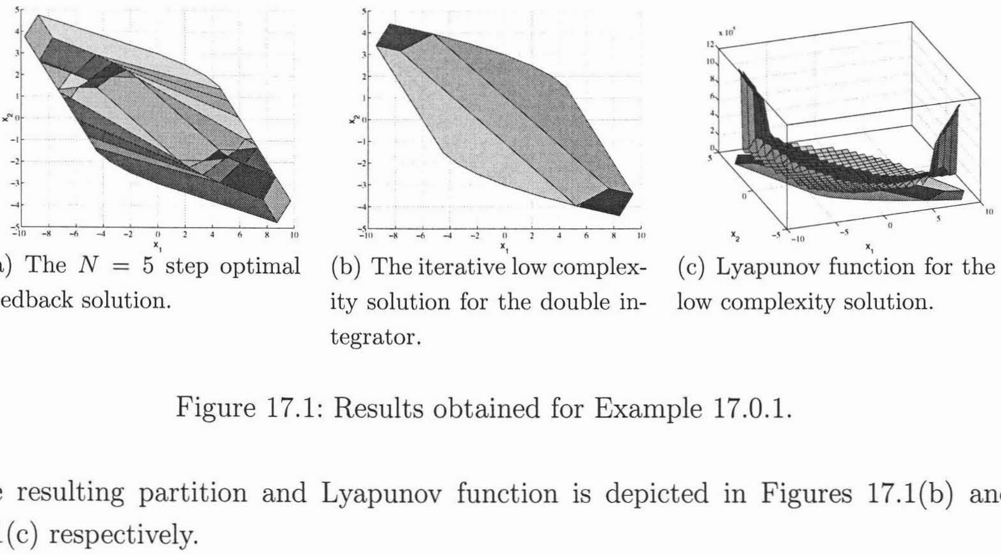
```
>> probStruct.norm=2;               %Quadratic Objective
>> probStruct.Q=eye(2);             %Objective: min_U J=sum x'Qx + u'Ru...
>> probStruct.R=1;                  %Objective: min_U J=sum x'Qx + u'Ru...
>> probStruct.N=5;                  %...over the prediction horizon 5
>> probStruct.subopt_lev=0;         %Compute optimal solution, not low complexity.
```

If we now call

```
>> ctrl=mpt_control(sysStruct,probStruct);   %Compute feedback controller
>> plot(ctrl)
```

the controller for the given problem is returned and plotted (see Figure 17.1(a)), i.e., if the state $x \in PA(i)$, then the optimal input for prediction horizon $N = 5$ is given by $u = F_i ix + G_i i$. If we wish to compute a low complexity solution, we can run the following:

```
>> probStruct.subopt_lev=2;         % Compute low complexity solution.
>> probStruct.N = 1;                % Use short prediction horizon
>> ctrl = mpt_control(sysStruct,probStruct);
>> plot(ctrl)                       % Plot the controller partition
>> Q = ctrl.details.lyapunov.Q;
>> L = ctrl.details.lyapunov.L;
>> C = ctrl.details.lyapunov.C;
>> mpt_plotPWQ(ctrl.finalPn,Q,L,C); % Plot the Lyapunov Function
```



(a) The $N = 5$ step optimal feedback solution.

(b) The iterative low complexity solution for the double integrator.

(c) Lyapunov function for the low complexity solution.

Figure 17.1: Results obtained for Example 17.0.1.

The resulting partition and Lyapunov function is depicted in Figures 17.1(b) and 17.1(c) respectively.

## Example 17.0.2

*Now we will solve the PWA problem introduced in [MR03] by defining two different dynamics which are defined in the left- and right half-plane of the state space respectively.*

```
>> H=[-1 1; -3 -1; 0.2 1; -1 0; 1 0; 0 -1];  %Polytopic state constraints Hx(k)<=K
>> K=[  15;    25;     9;    6;    8;   10];  %Polytopic state constraints Hx(k)<=K

>> sysStruct.C{1} = [1 0];                    %System Dynamics 1: y(k)=Cx(k)+Du(k)+g
>> sysStruct.D{1} = 0;                        %System Dynamics 1: y(k)=Cx(k)+Du(k)+g
>> sysStruct.g{1} = [0];                     %System Dynamics 1: y(k)=Cx(k)+Du(k)+g
>> sysStruct.A{1} = [0.5 0.2; 0 1];          %System Dynamics 1: x(k+1)=Ax(k)+Bu(k)+f
>> sysStruct.B{1} = [0; 1];                  %System Dynamics 1: x(k+1)=Ax(k)+Bu(k)+f
>> sysStruct.f{1} = [0.5; 0];                %System Dynamics 1: x(k+1)=Ax(k)+Bu(k)+f
>> sysStruct.guardX{1} = [1 0; H];           %Dynamics 1 defined in guardX*x <= guardC
>> sysStruct.guardC{1} = [  1; K];           %Dynamics 1 defined in guardX*x <= guardC

>> sysStruct.C{2} = [1 0];                   %System Dynamics 2: y(k)=Cx(k)+Du(k)+g
>> sysStruct.D{2} = 0;                       %System Dynamics 2: y(k)=Cx(k)+Du(k)+g
>> sysStruct.g{2} = [0];                     %System Dynamics 2: y(k)=Cx(k)+Du(k)+g
>> sysStruct.A{2} = [0.5 0.2; 0 1];          %System Dynamics 2: x(k+1)=Ax(k)+Bu(k)+f
>> sysStruct.B{2} = [0; 1];                  %System Dynamics 2: x(k+1)=Ax(k)+Bu(k)+f
>> sysStruct.f{2} = [0.5; 0];                %System Dynamics 2: x(k+1)=Ax(k)+Bu(k)+f
>> sysStruct.guardX{2} = [-1 0; H];          %Dynamics 2 defined in guardX*x <= guardC
>> sysStruct.guardC{2} = [  -1; K];          %Dynamics 2 defined in guardX*x <= guardC

>> sysStruct.ymin = -10;                     %Output constraints for dynamic 1 and 2
>> sysStruct.ymax = 10;                      %Output constraints for dynamic 1 and 2
>> sysStruct.umin = -1;                      %Input constraints for dynamic 1 and 2
>> sysStruct.umax = 1;                       %Input constraints for dynamic 1 and 2
```

we can now compute the low complexity feedback controller by defining the problem

```
>> probStruct.norm=2;               %Quadratic Objective
>> probStruct.Q=eye(2);             %Objective: min_U J=sum x'Qx + u'Ru...
>> probStruct.R=0.1;                %Objective: min_U J=sum x'Qx + u'Ru...
>> probStruct.subopt_lev=1;         %Compute low complexity controller.
```

and calling the control function,

```
>> ctrl=mpt_control(sysStruct,probStruct);
>> plot(ctrl)
```

The result is depicted in Figure 17.2.

For more examples we recommend to look at the demos which can be found in respective subdirectories of the `mpt/examples` directory of your MPTinstallation.

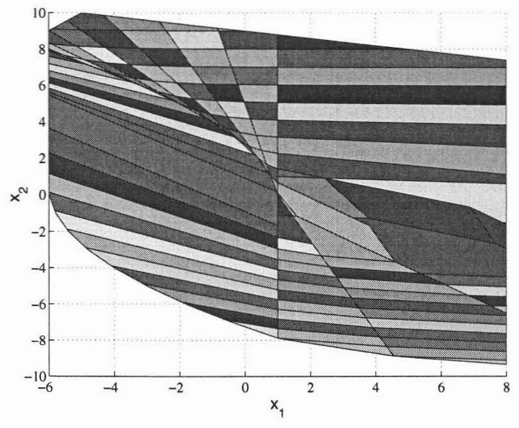


Figure 17.2: Controller partition obtained for Example 17.0.2.

# 18

# Polytope Library

As mentioned in Section 3, a polytope is a convex bounded set which can be represented either as an intersection of a finite number of half-spaces ($\mathcal{H}$-representation) or as a convex hull of vertices ($\mathcal{V}$-representation). Both ways of defining a polytope are allowed in MPT and one can switch from one representation to the other one. However, by default all polytopes are generated in $\mathcal{H}$-representation only to avoid unnecessary computations.

## 18.1 Creating a Polytope

A polytope in MPT is created by a call to the polytope constructor as follows:

```
P = polytope(H,K)
```

creates a polytope by providing it's $\mathcal{H}$-representation (3.2), i.e. the matrices `H` and `K` which form the polytope

$$\mathcal{P} = \{x \in \mathbb{R}^n \mid Hx \leq K\}. \tag{18.1}$$

If input matrices define some redundant constraints, these will be automatically removed to form a minimal representation of the polytope. In addition, center and diameter of the largest ball which can be inscribed into the polytope are computed as well and the $\mathcal{H}$-representation is normalized to avoid numerical problems. The constructor then returns a polytope object.

A polytope can also be defined by it's vertices (3.4) as follows:

```
P = polytope(V)
```

where V is a matrix which contains vertices of the polytope in the following format:

$$\mathtt{V} = \begin{bmatrix} v_{1,1} & \dots & v_{1,n} \\ \vdots & \vdots & \vdots \\ v_{k,1} & \dots & v_{k,n} \end{bmatrix} \tag{18.2}$$

where $k$ is the total number of vertices and $n$ is the dimension. Hence vertices are stored row-wise. Before the polytope object is created, the $\mathcal{V}$-representation is first converted to the half-space description by eliminating all points from V which are not extreme points. The convex hull (3.9) of the remaining points is then computed to obtain the corresponding $\mathcal{H}$-representation. Extreme points will be stored in the polytope object and can be returned upon request without additional computational effort.

## 18.2 Accessing Data Stored in a Polytope Object

Each `polytope` object is internally represented as a structure, but because of the Object-Oriented approach, this information cannot be directly obtained by using structure deferencing through the . (dot) operator. Special functions have to be called in order to retrieve individual fields.

Fields of the polytope structure are summarized it Table 18.2.

| | |
|---|---|
| `H, K` | $\mathcal{H}$-representation of the polytope |
| `xCheb, RCheb` | Center and radius of Chebyshev's ball |
| `normal` | Flag whether the $\mathcal{H}$-representation is normalized (1/0) |
| `minrep` | Flag whether the $\mathcal{H}$-representation is reduced (1/0) |
| `Vertices` | Extreme points of the $\mathcal{V}$-representation) of the polytope |

Table 18.1: Data stored in the`polytope` object.

In order to access the $\mathcal{H}$-representation (matrices H and K), one has to use the command `double` as follows:

```
[H,K] = double(P)
```

to store matrices H and K individually, or alternatively:

```
HK = double(P)
```

which returns a matrix $HK = [H\ K]$.

Center and radius of Chebyshev's ball can be obtined by:

```
[xCheb, RCheb] = chebyball(P)
```

If the polytope is in normalized representation, call to

```
flag = isnormal(P)
```

will return 1, 0 otherwise.

The command

```
flag = isminrep(P)
```

return 1 if polytope `P` is in minimal representation (i.e. the $\mathcal{H}$-representation contains no redundant hyperplanes), 0 otherwise.

The polytope is bounded if

```
flag = isbounded(P)
```

returns 1 as the output.

The dimension of a polytope can be obtained by

```
d = dimension(P)
```

and

```
nc = nconstr(P)
```

will return the number of constraints (i.e. number of half-spaces) defining the given polytope `P`.

The vertex representation of a polytope can be obtained by:

```
V = extreme(P)
```

which returns the vertices stored row-vise in the matrix `V`. As enumeration of extreme vertices is an expensive operation, the computed vertices can be stored in the polytope object. To do it, we always recommend to call the function as follows:

```
[V,R,P] = extreme(P)
```

which returns extreme points `V`, extreme rays `R` and the update polytope object with vertices stored inside (`P`).

To check if a given point `x` lies in a polytope `P`, use the following call:

```
flag = isinside(P,x)
```

The function returns 1 if $x \in P$, 0 otherwise. If `P` is a polyarray (see Section 18.3 for more details about polyarrays), the function call can be extended to provide additional information:

```
[flag, inwhich, closest] = isinside(P,x)
```

which returns a 1/0 flag which denotes if the given point `x` belongs to any polytope of a polyarray `P`. If the given point lies in more than one polytope, `inwhich` contains indexes of the regions which contain `x`. If there is no such region, index of a region which is closest to the given point `x` is returned in `closest`.

The functions mentioned in this chapter are summarized in Table 18.2.

| | |
|---|---|
| `P=polytope(H,K)` | Constructor for creating the polytope $P = \{x \in \mathbb{R}^n \mid Hx \leq K\}$. |
| `P=polytope(V)` | Constructor for creating the polytope out of extreme points |
| `double(P)` | Access internal data of the polytope, e.g. `[H,K]=double(P)`. |
| `display(P)` | Displays details about the polytope $P$. |
| `nx=dimension(P)` | Returns dimension of a given polytope P |
| `nc=nconstr(P)` | For a polytope $P = \{x \in \mathbb{R}^n \mid Hx \leq K\}$ returns number of constraints of the $H$ matrix (i.e. number of rows). |
| `[ , ]` | Horizontal concatenation of polytopes into an array, e.g. `PA=[P1,P2,P3]`. |
| `( )` | Subscripting operator for polytope arrays, e.g. `PA(i)` returns the i-th polytope in `PA`. |
| `length(PA)` | Returns number of elements in a polytope array $P_A$. |
| `end` | In indexing functions returns the final element of an array. |
| `[c,r]=chebyball(P)` | Returns center $c$ and radius $r$ of the Chebychev ball inside $P$. |
| `V=extreme(P)` | Computes extreme points (vertices) of a polytope $P$. |
| `bool=isfulldim(P)` | Checks if polytope $P$ is full dimensional. |
| `bool=isinside(P,x)` | Checks if $x \in P$. Works also for polytope arrays. |

Table 18.2: Functions defined for class `polytope`.

## 18.3 Polytope Arrays

Instances of the `polytope` object can be concatenated into arrays. Currently, only one-dimensional arrays are supported by MPT and it does not matter if the elements are stored row-wise or column-wise. An array of polytopes is created using standard Matlab concatenation operators `[,]`, e.g. `A = [B C D]`.

It does not matter whether the concatenated elements are single polytopes or polyarrays. To illustrate this, assume that we have defined polytopes `P1, P2, P3, P4, P5` and polyarrays `A = [P1 P2]` and `B = [P3 P4 P5]`. Then the following polyarrays `M` and `N` are equivalent:

```
M = [A B]
N = [P1 P2 P3 P4 P5]
```

Individual elements of a polyarray can be obtained using the standard referencing `(i)` operator, i.e.

```
P = M(2)
```

will return the second element of the polyarray `M` which is equal to `P2` in this case. More complicated expressions can be used for referencing:

```
Q = M([1,3:5])
```

will return a polyarray `Q` which contains first, third, fourth and fifth element of polyarray `M`.

If the user wants to remove some element from a polyarray, he/she can use the referencing command as follows:

```
M(2) = []
```

which will remove the second element from the polyarray `M`. Again, multiple indices can be specified, e.g.

```
M([1 3]) = []
```

will erase first and third element of the given polyarray. If some element of a polyarray is deleted, the remaining elements are shifted towards the start of the polyarray. This means that, assuming `N = [P1 P2 P3 P4 P5]`, after

```
N([1 3]) = []
```

the polyarray `N = [P2 P4 P5]` and the length of the array is 3. No empty positions in a polyarray are allowed. Similarly, empty polytopes are not being added to a polyarray.

A **`polyarray`** is still a polytope object, hence all functions which work on polytopes support also polyarrays. This is an important feature mainly in the geometric functions.

Length of a given `polyarray` is obtained by

```
l = length(N)
```

A polyarray can be flipped by the following command:

```
Nf = fliplr(N)
```

i.e. if `N = [P1 P2 P3 P4 P5]` then `Nf = [P5 P4 P3 P2 P1]`.

## 18.4 Geometric Operations on Polytopes

The polytope library of MPT can efficiently perform many geometric manipulations on polytopes and polyarrays (non-convex unions of polytopes). A theoretical description of some basic operations has been already been given in Chapter 3. A list of computational geometry functions is provided in Table 18.3.

Except of `bounding_box`, all other functions are implemented to take polytopes and/or polyarrays as input arguments. We recommend to consult help files for respective functions for more details about extended function calls and other details.

The following examples show how to use some of the functionality described in Table 18.3:

### Example 18.4.1

```
>> P=polytope([eye(2);-eye(2)],[1 1 1 1]');         %Create Polytope P
>> [r,c]=chebyball(P)                               %Chebychev ball inside P
    r=[0 0]'
    c=1
>> W=polytope([eye(2);-eye(2)],0.1*[1 1 1 1]'); %Create Polytope W
>> DIF=P-W;                                         %Pontryagin difference P-W
>> ADD=P+W;                                         %Minkowski addition P+W
>> plot(ADD, P, DIF, W);                            %Plot polytope array
```

| | |
|---|---|
| `P == Q` | Check if two polytopes are equal ($P = Q$). |
| `P ~= Q` | Check if two polytopes are not-equal ($P \neq Q$). |
| `P >= Q` | Check if $P \supseteq Q$. |
| `P <= Q` | Check if $P \subseteq Q$. |
| `P > Q` | Check if $P \supset Q$. |
| `P < Q` | Check if $P \subset Q$. |
| `P & Q` | Intersection of two polytopes, $P \cap Q$. |
| `P \| Q` | Union of two polytopes, $P \cup Q$. If the union is convex, the polytope $P \cup Q$ is returned, otherwise the polyarray $[P\ Q]$ is returned. |
| `P + Q` | Minkowski sum, $P \oplus Q$ (cf. (3.12)). |
| `P - Q` | Pontryagin difference, $P \ominus Q$ (cf. (3.11)). |
| `P \ Q` | Set difference operator (cf. (3.7)). |
| `B=bounding_box(P)` | Computes minimal hyper-rectangle containing a polytope $P$. |
| `E=envelope(P,Q)` | Computes envelope $E$ of two polytopes $P$ and $Q$ according to (3.10). |
| `P=range(Q,A,f)` | Affine transformation of a polytope. $P = \{Ax + f \in \mathbb{R}^n \mid x \in Q\}$. |
| `P=domain(Q,A,f)` | Compute polytope that is mapped to Q. $P = \{x \in \mathbb{R}^n \mid Ax + f \in Q\}$. |
| `R=projection(P,dim)` | Orthogonal projection of `P` onto coordinates given in `dim` (cf. (3.6)) |

Table 18.3: Computational geometry functions

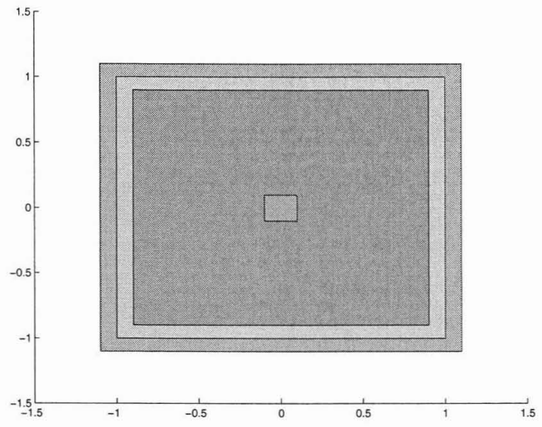
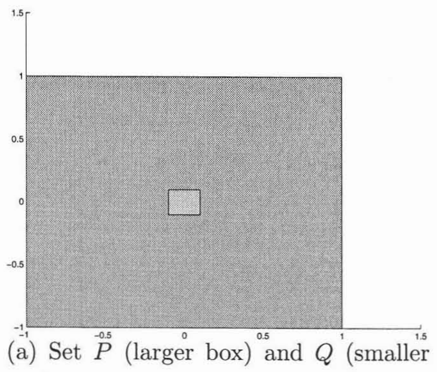


Figure 18.1: The result of the `plot` call in Example 18.4.1

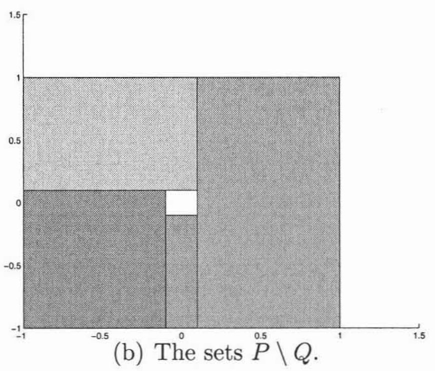
The resulting plot is depicted in Figure 18.1. When a `polytope` object is created, the constructor automatically normalizes its representation and removes all redundant constraints. Note that all elements of the polytope class are private and can only be accessed as described in the tables. Furthermore, all information on a polytope is stored in the internal polytope structure. In this way unnecessary repetitions of the computations during polytopic manipulations in the future can be avoided.

**Example 18.4.2**

```
>> P=unitbox(2, 1);     %Create Polytope P as a box in 2D with sides of size 1
>> Q=unitbox(2, 0.1);   %Create Polytope Q as a box in 2D with sides of size 0.1
>> D=P\Q;               %Compute set difference between P and Q
>> length(D)            %D is a polytope array with 4 elements
    ans=4
>> U=D|Q;               %Compute union of D and Q
>> length(U)            %Union is again a polytope
    ans=1
>> U==P                 %Check if two polytopes are equal
    ans=1
```



(a) Set $P$ (larger box) and $Q$ (smaller box).



(b) The sets $P \setminus Q$.

Figure 18.2: Visualization of the sets $P$, $Q$, and $D = P \setminus Q$ in Example 18.4.2.

The polytopes $P$ and $Q$, as well as the set difference $D$ are depicted in Figure 18.2. The next example will illustrate the use of the `hull` and `extreme` functions.

**Example 18.4.3**

```
>> P=polytope([eye(2);-eye(2)],[0 1 1 1]');      %Create Polytope P
>> Q=polytope([eye(2);-eye(2)],[1 1 0 1]');      %Create Polytope Q
>> VP=extreme(P);                        %Compute extreme vertices of P
>> VQ=extreme(Q);                        %Compute extreme vertices of P
>> D1=hull([P Q]);                       %Create convex Hull of P and Q
>> D2=hull([VP;VQ]);                     %Create convex Hull of vertices VP and VQ
>> D1==D2                                %Check if hulls are equal
    ans=1
```

The purpose of the `extreme` function is to convert the polytope given by the $\mathcal{H}$-representation (18.1) into an equivalent $\mathcal{V}$-representation (18.2), i.e. to enumerate the extremal vertices $v_i$. The `hull` function then performs then reverse operation, i.e. given a set of vertices, calculate the corresponding $\mathcal{H}$-representation of the convex hull of these points. As can be seen from the example, the `hull` function is overloaded in a way such that it takes both elements of the `polytope` class as well as matrices of points as input arguments.

# 19
# Overview of Other Available Software Packages

This chapter provides an overview of available software packages and identifies strong and weak points of each such package. We focus on the main players on the field of MPC-based control, namely the Hybrid Toolbox and the MPC Toolbox and compare them to the Multi-Parametric Toolbox presented in this thesis. The general comparison is focused on user interface, amount of provided functionality and integration with Simulink. In Section 19.4 we also give a comparison between MPT and AMPL (A Mathematical Programming Language), which is a modeling language for rapid prototyping of optimization problems.

## 19.1 Hybrid Toolbox

The Hybrid Toolbox [Bem03] is a Matlab toolbox for design of MPC control laws for linear and hybrid systems. The most notable features of the toolbox include:

- Uses HYSDEL to obtain models of hybrid systems
- Is capable of computing off-line solutions to MPC problems for linear and hybrid systems based on the Constrained Finite Time Optimal Control policy
- Features a Simulink interface which allows to simulate on-line and off-line controllers in Simulink
- Is capable of exporting off-line controllers into a standalone C-code
- Contains solvers for multi-parametric linear and quadratic programming and for mixed-integer multi-parametric linear programming
- Is shipped with the CDD, GLPK and QPACT solvers

## 19.2 Model Predictive Control Toolbox

The Model Predictive Control Toolbox is a commercial Matlab toolbox focused at design and simulation of on-line MPC control laws for linear systems with constraints. Only quadratic cost functions are supported by the toolbox. Controllers designed by the toolbox always use the $\Delta u$ formulation and support tracking of free output references. For synthesis, the toolbox can use models affected by measured or unmeasured disturbances. State estimators can be designed such that output feedback can be used. More details will given in Section 19.3.

## 19.3 Comparison of MPT, Hybrid Toolbox and the MPC Toolbox

This section is devoted to a more detailed comparison of the three software tools which can be used for control design – MPT, the Hybrid Toolbox and the MPC Toolbox. The comparison is reported with respect to three categories: the user interface, amount of functionality provided by each package, and, finally, the integration of each tool with Simulink.

### 19.3.1 User Interface

As outlined in the introduction section, all toolboxes use different input data when designing MPC controllers.

The user interacts with the MPT toolbox by means of three objects:

- Description of the plant model (the `sysStruct` structure)
- Description of the control problem (the `probStruct` structure)
- A controller object

The system structure `sysStruct` contains description of the plant model, along with plant constraints. It can be obtained automatically from following sources:

- HYSDEL source code
- Matrices of a Mixed Logical Dynamical (MLD) model

- Matrices of a Piecewise Affine (PWA) model
- Control toolbox objects (both state-space and transfer function representation)
- MPC toolbox objects
- System identification toolbox objects

Each conversion is fully automatic, with as much information extracted from the source object as possible. MPT supports LTI, PWA and MLD models with constraints as target models. Several functions are available in MPT which allow open-loop simulations of various models either in Matlab or in Simulink. Similarly, conversion functions between PWA and MLD models exist.

The problem structure `probStruct` describes parameters of a given control problem. It is used by users to define prediction horizon, penalty matrices, target set constraints and/or stability properties of the resulting control laws. Although easy to define for users, the problem structure will never be capable to capture all possible desired problem description. To solve this usability problem MPT introduced the "Design your own MPC" function, which gives experienced users the option to alter the problem setup by means of adding constraints involving the optimization variables directly. This new approach received very positive critiques from the user community.

Finally, the controller object encapsulates a given control law and provides users with accessible methods to perform common tasks. These include, but are not limited, to:

- Simulation of closed-loop systems
- Simplification of off-line control laws by means of merging
- Generation of binary search trees which facilitate faster application of off-line control laws on-line
- Export of off-line control laws to C-code
- Stability analysis of closed-loop systems
- Verification of safety and liveness properties of controllers

Important to notice is that the controller object viewed from outside is a unique compact object. This means that regardless whether the controller represents an on-line or an off-line (parametric) controller, users interact with such controllers in the same way, by calling overloaded methods and functions.

The Hybrid Toolbox accepts the plant model as an object representing either LTI, PWA or MLD dynamics. Only HYSDEL can be used to derive hybrid models (PWA or MLD). Parameters of the optimization problem, such as the prediction horizon, penalty matrices or constraints have to be provided as a comma separated list to the controller synthesis function. Depending on whether one aims at an on-line controller for hybrid systems, or an on-line controller for linear systems, or an off-line controller, the toolbox offers three different functions which produce three different types of controller objects. It is then possibly to "convert" an on-line controller into its equivalent off-line form while preserving the original controller untouched. It is not, however, possible to modify the controller object directly. If a parameter changes, a new controller has to be built. The toolbox also provides a unified set of overloaded functions which then operate on all three types of the controller objects.

The Model Predictive Control toolbox uses the Control Toolbox objects (state-space and transfer function objects) as a basis for controller design. Once the plant dynamics is given, the toolbox constructs a default controller object where it presets certain parameters of the optimization problem to their default values (e.g. the prediction horizon or penalties). The user is then given the possibility to change these values according to his needs. It is therefore the controller object itself acting as a single "storage" for all user-tunable parameters.

### 19.3.2 Provided Functionality

MPT is the most feature-packed toolbox of the three available choices. The MPC Toolbox focuses entirely on on-line MPC for linear plants based on quadratic performance index. The Hybrid Toolbox is comparable to MPT with respect to functionality for Constrained Finite Time Optimal Control of linear and hybrid plants (both on-line and off-line strategies) and also provides a function to perform reachability analysis (based on solution to a feasibility MILP). With respect to functionality MPT at this point represents almost a superset of what is contained elsewhere[1]. Moreover it fea-

[1]Currently missing is only support for non-symmetrical soft constraints and dealing with measured disturbances as present in the MPC Toolbox

tures advanced control strategies such as minimum-time or low-complexity algorithms or stability analysis.

A major advantage of MPT, however, is its ability to expose MPC formulations to users and to let them add custom constraints or modify the performance index. This allows one to easily deal with topics like polytopic or collision avoidance constraints, without cluttering the user interface with custom options for each particular type of constraint.

### 19.3.3 Simulink Integration

All three toolboxes offer specialized blocks which allow one to use controllers in Simulink schemes. MPT provides a single block which automatically recognizes whether a given controller can be exported to C-code (which is a case of off-line controllers). If it can, the block will automatically switch to a C-code implementation. The code was intentionally written in a way such that it can not only be used to run within Simulink, but can also be directly compiled by the Real-Time Workshop and automatically downloaded to a target machine. It should be noted that as of MPT 2.6.1 we are not yet able to export all types of off-line controllers into C-code. Specifically, controllers which contain overlapping polyhedra where the value of the cost function needs to be checked (CFTOC solutions for quadratic cost for PWA systems) are not yet supported. If the controller represents an on-line MPC controller, a Matlab-coded function is used to evaluate the controller for a particular value of the state vector during the simulation.

The Hybrid Toolbox provides three different Simulink blocks which are responsible for evaluating controllers on-line. The first such block accepts off-line controllers and supports also checking of cost functions for overlapping partitions. Unlike MPT, such blocks need to be compiled every time the Simulink model is started (MPT comes with a pre-compiled version of such block). The other two blocks are each responsible for either on-line controllers for LTI systems or on-line controllers for hybrid systems. This implies that the user has to manually exchange the blocks if he changes the type of the controller (MPT auto-detects such change and switches automatically). The block which evaluates off-line controllers for linear systems includes state estimation. On-line controllers are evaluated using Matlab-based functions.

The MPC Toolbox only provides one block which allows to use on-line MPC controllers in Simulink. The block links description of the quadratic programming problem with a QP solver (qpdantz) on a C-code level. No other QP solvers are supported. A state estimator is included in the controller block.

## 19.4 Comparison between MPT and AMPL

AMPL (A Mathematical Programming Language) is a comprehensive and powerful algebraic modeling language for linear and nonlinear optimization problems, including discrete or continuous variables. AMPL lets users use a common notation and familiar concepts to formulate optimization models and examine solutions, while the computer manages communication with an appropriate solver. AMPL's flexibility and convenience render it ideal for rapid prototyping and model development, while its speed and control options make it an especially efficient choice for repeated production runs.

Due to AMPL's support for discrete as well as continuous optimization variables, it allows to formulate MPC problems involving models of hybrid systems. It was, however, observed on numerous test studies that the underlying optimization problem can be solved more efficiently if HYSDEL[2] is used to obtain numerical description of the MLD model of the system to be controlled. As will be explained in more details in Section 19.4.2, this is mainly due to the fact that HYSDEL automatically derives numerically better suited models. The same quality of models can also be achieved in AMPL, however the model has to be tuned manually, which is a tedious and not straightforward job.

### 19.4.1 Testing Methodology and Numerical Results

We have investigated the effect of each modeling approach on the time needed to solve the MPC problem on-line for a given initial state $x(0)$. Here we present two examples - the "car on a PWA hill" [KGBC06] reported in Appendix 24, and the "three tanks benchmark" [MM01] reported in Appendix 25.

Results in terms of time needed to solve the optimal control problem for each modeling approach are summarized in Tables 19.1 and 19.2. As can be seen from both tables, the optimization problem formulated by MPT based on HYSDEL-generated models is, on average, ten times faster compared to AMPL-formulated problems. In both cases the resulting mixed-integer problem has been solved with CPLEX 9.0 on a

[2]In the rest of this section by HYSDEL we refer to both the HYSDEL language and compiler, as well as to the matrixHYSDEL extension.

2.8 GHz Pentium 4 machine. AMPL 8.1 was used to model the optimization problems.

| Horizon | HYSDEL+MPT | AMPL |
|---|---|---|
| 30 | 7 secs | 51 secs |
| 32 | 10 secs | 107 secs |
| 34 | 27 secs | 182 secs |
| 36 | 27 secs | 591 secs |
| 38 | 99 secs | 1750 secs |
| 40 | 190 secs | 3244 secs |

Table 19.1: Time needed to solve the MPC problem for the "car on a PWA hill" benchmark for different values of the prediction horizon.

| Horizon | HYSDEL+MPT | AMPL |
|---|---|---|
| 7 | 0.1 secs | 1 secs |
| 8 | 0.6 secs | 3 secs |
| 9 | 8.5 secs | 31 secs |
| 10 | 50 secs | 252 secs |

Table 19.2: Time needed to solve the MPC problem for the three tanks benchmark for different values of the prediction horizon.

### 19.4.2 Discussion

Even though the formulation of an MPC problem done by MPT, based on models generated by HYSDEL, and by AMPL is, in principle, identical, obtained results summarized in Tables 19.1 and 19.2 show superiority of our approach. In the rest of this section we explain what we believe is the reason for the superior performance of the approach reported here versus AMPL. First, we recall standard results from propositional calculus [BM99a]:

$$[f(x) \leq 0] \leftrightarrow [\delta = 1] \ \text{ is true iff} \begin{cases} f(x) \leq M(1-\delta) \\ f(x) \geq \epsilon + (m-\epsilon)\delta. \end{cases} \tag{19.1}$$

where $f : \mathcal{R}^n \mapsto \mathcal{R}$ is linear and $M$ and $m$ are defined by

$$M = \max_{x \in \mathcal{X}} f(x), \tag{19.2}$$

and

$$m = \min_{x \in \mathcal{X}} f(x), \tag{19.3}$$

where $\mathcal{X}$ is a given bounded set and $\epsilon$ is a small positive number (typically the machine precision). Equation (19.1) tells that the statement that a binary variable $\delta$ will take a true value if and only if $f(x) \leq 0$ can be expressed as a set of two inequalities involving parameters $M$, $m$, and $\epsilon$. Ideally, $M$ should be chosen as the maximum of the function $f(x)$ on a certain domain, but any sufficiently big number will do. If the function $f(x)$ is very complex (which happens often in modeling of real plants), it is not always possible by hand to make a good estimate of what the value of $M$ should be. The same argument is also true for finding a good estimate of $m$.

Expressions such as (19.1) are important for us because they are directly related to the way how hybrid systems described in the HYSDEL language are converted into a corresponding MLD form. The following code of the AD section can be directly translated into inequalities of the form (19.1):

```
AD {
  d = x1 + 2*x2 - 3*x3 <= 0
}
```

by assuming $f(x) = x_1 + 2x_2 - 3x_3$. If bounds on the variables x1, x2 and x3 are known, HYSDEL can automatically compute the upper and lower bound according to equations (19.2) and (19.3), respectively. Since HYSDEL only allows linear or affine expressions to be defined, it is always possible to find the exact maximum (minimum) of the function $f(x)$ by a simple enumeration. AMPL, on the other hand, is a much more general language, not specifically tailored to modeling of hybrid systems. Therefore, it is the user's responsibility to provide the bounds $M$ and $m$. As will be illustrated later by Example 19.4.3, strictness of these bounds has a significant effect on time needed to solve problems which involve propositional logic of this type. In order to proceed with the discussion, we need an insight in how optimization problems involving continuous and discrete variables are solved. For now consider that the objective function of such an optimization problem is linear, in which case the problem can be written as a mixed-integer linear program (MILP) of the following form:

$$\min f(x,d) \tag{19.4a}$$
$$\text{subj. to} \quad x \in \mathcal{R}^n, \tag{19.4b}$$
$$d \in \{0,1\}, \tag{19.4c}$$
$$g(x,d) \leq 0, \tag{19.4d}$$

where $x$ is an $n$-dimensional real vector and $d$ is a vector of boolean variables. The objective $f(\cdot)$ and the constraints $g(\cdot)$ are assumed to be linear in the respective variables. In order to find the global optimum to the problem (19.4) using the Branch & Bound technique, the following algorithm can be applied [NW88]:

**Algorithm 19.4.1 (Branch & Bound)**

```
1. Relax binary constraints $d_i \in \{0,1\} \rightarrow 0 \leq d_i \leq 1$.
2. Solve the relaxed problem (bounding phase).
3. Choose an index i and (branching phase):
    a) Fix $d_i = 0$; go to Step 2.
    b) Fix $d_i = 1$; go to Step 2.
```

The branching phase in Step 3 of Algorihtm 19.4.1 traverses through a tree of possible choices of the binary variables $d$, whereas the bounding phase in Step 2 is used to prune the tree. Specifically, one can stop exploring a given branch if one of the following conditions is satisfied:

1. Relaxation is infeasible.
2. Relaxation gives a binary result.
3. Relaxation has worse cost than best binary so far.

Usually it is the last case which is most important for good performance of Algorithm 19.4.1. In most practical problems, the tighter the relaxation is, the higher is the chance that exploration of a given branch in the tree will be terminated based on the objective value of the relaxed problem. The example below illustrates how relaxation of mixed-integer problems depends on the choice of parameters $M$ and $m$ used to convert logic statements into mixed integer inequalities (cf. relation (19.1)).

**Example 19.4.2** *We have analyzed the impact of parameters $M$ and $m$ on the relaxation of mixed-integer problems for the following example:*

$$y = \begin{cases} 2x+1 & \text{if } x \leq 0, \\ 0.5x+1 & \text{otherwise}, \end{cases} \tag{19.5}$$

*with* $\|x\| \leq 3$ *and* $\|y\| \leq 3$.

Using the propositional calculus [BM99a] system (19.5) can be rewritten into the following set of mixed-integer inequalities:

$$\begin{aligned}
x &\leq M(1-\delta_1) && \text{(19.6a)}\\
y-2x-1 &\leq -m(1-\delta_1) && \text{(19.6b)}\\
-(y-2x-1) &\leq M(1-\delta_1) && \text{(19.6c)}\\
-x &\leq M(1-\delta_2) && \text{(19.6d)}\\
y-0.5x-1 &\leq -m(1-\delta_2) && \text{(19.6e)}\\
-(y-0.5x-1) &\leq M(1-\delta_2) && \text{(19.6f)}\\
\delta_1+\delta_2 &= 1 && \text{(19.6g)}\\
-3 \leq\ & x \leq 3 && \text{(19.6h)}\\
-3 \leq\ & y \leq 3 && \text{(19.6i)}
\end{aligned}$$

where $\delta_1$ and $\delta_2$ are binary variables. Obviously, any feasible solution to (19.6) must lie on the lines given by $y = 2x+1$ (for $x \leq 0$) and $y = 0.5x+1$ (for $x > 0$), respectively. We have investigated how the choice of parameters $M$ and $m$ influences the tightness of the relaxed problem in Step 2 of Algorithm 19.4.1. Knowing the bounds on $x$ and $y$ in (19.6), one can compute precise values of parameters $M$ and $m$ in (19.6a)-(19.6f) by solving (19.2) and (19.3), respectively. By relaxing the binary variables and by assuming $0 \leq \delta_i \leq 1$, the feasible set of the problem (19.6) with $M$ and $m$ as in Table 19.3 is denoted by $\mathcal{F}_{tight}$ in Figure 19.1. If values of $M$ and $m$ were to be fixed to a single value in (19.6a)-(19.6f), e.g. $M = 10$ and $m = -10$, it would result in a much larger feasible set denoted by $\mathcal{F}_{loose}$ in Figure 19.1. Having large feasible sets of the relaxed problem is undesirable, since they negatively impact the bounding properties in Step 2 of Algorithm 19.4.1. Tight relaxations, on the other hand, often lead to significant performance improvement of Branch & Bound algorithms.

| Constraint | LHS expression | $m$ | $M$ |
|---|---|---|---|
| (19.6a) | $x$ | $-3$ | 3 |
| (19.6b) | $y - 2x - 1$ | $-10$ | 8 |
| (19.6c) | $-(y - 2x - 1)$ | $-8$ | 10 |
| (19.6d) | $-x$ | $-3$ | 3 |
| (19.6e) | $y - 0.5x - 1$ | $-5.5$ | 3.5 |
| (19.6f) | $-(y - 0.5x - 1)$ | $-3.5$ | 5.5 |

Table 19.3: Values of $M$ and $m$ for constraints in (19.6) computed by (19.2) and (19.3), respectively.

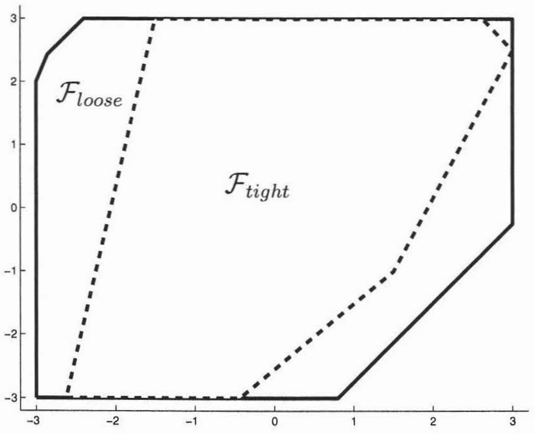


Figure 19.1: Feasible sets of the relaxed mixed-integer problem of Example 19.4.2.

**Example 19.4.3** *To show the impact of the relaxation parameters $M$ and $m$ on the performance of mixed-integer programs, we have examined again the three tanks benchmark and compared how the choice of $M$ ($m$) influences the total time needed to solve the particular mixed-integer program. As can be seen from Table 19.4, the impact is very big and a good choice of $M$ can change the solution time by almost one order of magnitude. However, it is very difficult to tighten the bounds as much as possible just by hand. In our experiments we have used one fixed value of $M$ ($m$) for all logic rules in the model of the system, which introduces some conservatism. Fine-tuning of the two parameters is difficult, though, since a wrong choice can even lead to infeasibility of the mixed integer program.*

*On the other hand, HYSDEL can automatically compute a specific set of $M$ and*

*$m$ parameters for every logic statement separately, leading to even better performance (cf. Table 19.2).*

| Horizon | $M = 50$ | $M = 25$ | $M = 10$ |
|---|---|---|---|
| 7 | 1 secs | 1 secs | 1 secs |
| 8 | 6 secs | 5 secs | 3 secs |
| 9 | 265 secs | 52 secs | 31 secs |

Table 19.4: Time needed to solve the MPC problem for the three tanks benchmark with AMPL for different values of the prediction horizon and different values of $M$. Value of $m$ was chosen as $m = -M$.

# 20

# Future Development

In this chapter we outline what we think are the important points the future development of the Multi-Parametric Toolbox should focus on. As a main goal, MPT's development should concentrate on being an MPC-based control toolbox, with high emphasis on providing novel, efficient algorithms. Simultaneously, it is desired to keep the tool attractive for users working in other areas, such as the computational geometry community. While design, analysis and implementation of off-line MPC control laws was the main driving factor in the initial period of MPT's development, on-line MPC should be considered equally important. Specifically, users should be encouraged to use on-line MPC for performance tuning and for initial verification of the design. Subsequently, it should be possible, in a user friendly way, to derive the off-line form of the control law, either for the purpose of implementation on target platforms or for tasks like stability analysis.

More effort should be invested into providing easy ways for the implementation of resulting MPC controllers. Specifically, the area of state estimation will need to be covered along with better export of off-line controllers into executable code. Better support for industrial target platforms, such as dSPACE, is also needed.

A major challenge will be to keep MPT a "research tool inside of an engineering tool". This implies that new advances made in the field of off-line MPC should be easily added to the current framework, without affecting the underlying engine and without changing the user interface. This task will involve a re-design of data exchange structures and turning more functionality into relatively separated library-like modules.

The possibility of turning MPT into a domain-tailored tool should also be investigated. For instance, the Multi-Parametric Toolbox was successfully applied to control of systems with frictions. A Simulink library containing specialized blocks, such as different friction models, on/off switches or standardized instruments could be created. Users would then use such blocks to model their plants. An automatic analyzer would then parse such Simulink scheme and automatically generate an appropriate hybrid model which can then be used for controller synthesis. Completing this task will involve creating a methodology for dealing with compositional hybrid systems. The easiest way of achieving this goal seems to be to write a YALMIP-based parser of HYSDEL models, because it is very easy to "connect" different models together and obtain the overall model once individual subsystems are defined by means of YALMIP variables.

Future versions of MPT should therefore aim at achieving the following global goals:

- Encourage a better workflow
- Complete the cycle of implementation of MPC controllers by adding support for state estimation and improve export of controllers to C-code
- Provide better interface for both internal and external tools
- Allow new algorithms and techniques to be added easily

While attacking given goals, one should stick to the following design principles:

- Keep more functionality in self-contained libraries, similar to the polytope library. This mainly involves more explicit support for functions defined over polyhedral domains. If this principle is strictly followed, further enhancements will boil down to adding a couple of new functions, instead of rewriting a substantial amount of code.
- Correctness, consistency and coherence of the individual modules contained in MPT should be maintained at all costs. It should be possible to easily exchange results generated at various stages by various modules. This includes higher-level abstractions provided to tools like YALMIP. Extensive testing is the key to eliminate human mistakes.
- Speed and efficiency are important, but they should not be achieved at the cost of lowering the level of maintainability of the code. While it is possible to rewrite critical parts of the code into C to gain better runtime, such effort should only concentrate on a limited amount of functions. Matlab code will always be easier to read and extend than C programs.

- If a particular problem setup cannot be handled by the tool, the user should be clearly warned and should be given understandable hints of what can be done in such cases. Error messages should be helpful.

Individual tasks needed to achieve these goals are described in subsequent sections.

## 20.1 Improved Export of Controllers to C-code

Off-line controllers in MPT can be exported to C-code in three different ways:

1. As a standalone C-code implementing the binary search tree algorithm.

2. As a standalone C-code implementing the exhaustive search algorithm.

3. As a C-code Simulink S-function implementing the exhaustive search algorithm.

These three approaches are represented by three different code templates. Even though there are substantial similarities on the code level between methods 2 and 3, there is currently no code sharing between the actual implementations. The exhaustive search implementation is also currently not capable of searching through partitions with overlapping regions based on checking the minimal value of the cost function. Another missing piece is a Simulink version of a block which would implement the binary search tree algorithm.

The following improvements are therefore planned:

- Extend the Simulink MPT controller block to support binary search trees. This will allow one to export such trees using Real-Time Workshop to target platforms.

- Extend the exhaustive search C-code implementation to support overlapping partitions.

- Move the search algorithm into a single library-like code and reuse it at all places.

The changes will, however, need to be coordinated with the progress achieved while completing tasks as described in section 20.3.

## 20.2 State Estimation

The last missing piece in the process of implementation of MPC controllers in the MPT framework is state estimation. Design and correct application of estimators is currently left to the users. It is desirable to come up with a unified scheme of design and Simulink implementation of state estimators, at least for linear systems. The whole procedure must be general enough to be able to cope with moving horizon estimators in the future as well.

Individual coding tasks are outlined as follows:

- Analyze possibility of (semi)automatic generation of state estimators for linear and hybrid systems.
- Create a new object to express state estimators.
- Create a Simulink block to evaluate the estimator.
- Make sure the block is compatible with Real-Time Workshop.

## 20.3 Re-design of Controller Objects

As outlined in the introduction, one of the main future goals is to encourage better workflow when designing MPC controllers. Specifically, the idea is to allow tuning of controllers by directly changing the controller objects. Currently, such tasks can only be achieved by first modifying the problem structure with subsequent call to a function which constructs a new controller object.

The new approach should be very similar to what the MPC Toolbox offers. Namely, change the MPT controller structure and access methods such that parameters like the prediction horizon, penalties or constraints can be changed on-the-fly in the object itself. This also implies another major change – controller objects, by default, should represent on-line MPC controllers. Only upon a user request they should be converted into an off-line form. This will allow a much more intuitive approach to performance tuning and will save users the time otherwise spent in constant re-computation of off-line solutions. Such a change will also allow to clearly separate model and design constraints and will allow a more consistent definition of time-varying elements, such as penalties.

To achieve this task, the following steps will be needed:

- Review the current structure of controller objects and identify elements which can be directly adjusted by the user. The structure of MPC Toolbox controllers could be used as a template.
- Consider changing the default behavior to always generate on-line controllers instead of off-line ones. Will need to clearly communicate such change to users.
- Consider using default values for certain optimization parameters. The MPC Toolbox, for instance, sets defaults for the prediction horizon and for penalty matrices, according to pre-defined rules. This, on one hand, allows faster jump-start for novice users, on the other hand it is difficult to come up with good default setting especially in view of complexity of off-line solutions.
- Make description of time-varying elements, such as constraints or penalties, more intuitive and more consistent.
- Analyze how the concept of the "Design your own MPC" function fits into this new framework. Major obstacle currently being that YALMIP objects representing constraints cannot be saved to disk and re-loaded later.
- Adjust the way an off-line solution is stored in a controller object. Use the new objects as described in Section 20.4. Make such storage general enough in view of perspective extension in the future (e.g. support for off-line MPC for polynomial systems).

## 20.4 New Object Representing Functions Defined over Polytopes

Increasing usage of YALMIP in and with MPT showed a shortcoming of the MPT design in the area of data exchange structures. On one hand MPT provides many useful algorithms, such as optimal complexity reduction, reachability analysis, or visualization functions. On the other hand, the way how external tools, such as YALMIP, interact with said functions is not ideal. Specifically, each such functionality was originally designed to cope with MPT's internal controller objects, which encompass elements unrelated to the scope of external tools (e.g. model of the plant or specific internal information). This implies that each external tool first needs to create a "dummy" controller object, call one of the MPT's native functions and then remove all unnecessary fields from the result.

The type of exchanged data we are discussing here are functions defined over a specific domain. The aim is to make the formulation as general as possible. We start by implementing objects representing functions defined over polyhedral regions, such as Piecewise Affine or Piecewise Quadratic functions. Later, objects describing functions defined over regions defined by polynomial boundaries will be added. There will be a well defined set of functions operating on these objects, such that the external tool or user doesn't have to care what type of object he deals with. This is important mainly in view of dealing with outputs of multi-parametric solvers. Regardless of whether the output is a PWQ function (e.g. a value function produced as a solution to mpQP) or a PWA function (e.g. the control law), the user should be able to use the result as one compact object and use identical methods while processing it. Currently such processing is not straightforward, since the user has to deal with the polytopic domain and the quadratic, affine and constant terms individually. This is error-prone and imposes unnecessary load on the user.

The main goal in this area is to come up with a library of functions dealing with functions defined over a polyhedral or polynomial domains. The polytope library present in MPT clearly showed that such approach is superior to using individual, self-contained functions.

The plan is to separate the "core" part of stored information into a new object, which will be used both by external tools as well as by MPT's native controller object. The main motivation being to stabilize the data interchange object and make its structure independent of possible modifications of the controller object in the future. This would give external tools a very stable interface which will, ideally, endure for the whole remaining lifetime.

The implementation plan is as follows:

- Identify a set of overloaded functions which should be supported.
- Implement objects representing functions defined over polytopes.
- Refactor current code and generate the new objects where appropriate.

Once finished, the overall code structure will be much simpler, since it will use a bottom-to-up approach, using smaller building blocks and combining them together.

This is in contrast to the current implementation, where almost each function is self-contained. It will also make further adjustments and enhancements much easier and more transparent.

## 20.5 Polytope Library

One major missing piece of the polytope library contained in MPT, as of now, is the support for lower-dimensional polytopes and polyhedra. This is viewed as a deficiency since flat polytopes occur frequently e.g. in reachability analysis or when dealing with lower-dimensional regions which can arise while modelling certain PWA systems. Therefore it is of imminent importance to include such functionality into the Multi-Parametric Toolbox. However, the implementation has to be done carefully as to take all the theoretical issues into account.

## 20.6 Main Goals for HYSDEL

Following goals are envisioned as future possible extensions of HYSDEL:

- Rewrite the language parser into Matlab. The currently available C++ implementation is poorly extendible and exhibits certain memory leaks when compiling models with a very large number of symbolic parameters. Morover, a pure Matlab-based compiler would allow to use advanced techniques, such as removal of redundant constraints or access to linear programming solvers. YALMIP looks to be an ideal candidate of an underlying engine for a Matlab-based implementation. It would much simplify handling of vector and matrix elements, along with generation of better models in the sense of tighter big-M relaxations.

- Extend the Matlab-based parser to handle compositional hybrid systems, where each subsystem is described by its own HYSDEL code and set of interconnecting conditions is specified either on a textual level, or by means of a Simulink scheme.

- Extend the HYSDEL syntax. This includes one-way implication statements in AD sections, or improved syntax of IF-THEN-ELSE statements. Specifically, it should be possible to use continuous conditions in the IF part of the statement. Doing so would simplify the language and will not require users to define auxiliary boolean variables to denote fulfillment conditions involving continuous variables.

- Investigate the possibility and benefit of an automatic conversion of HYSDEL models into AMPL. The apparatus for doing such conversion was already developed as part of the matrixHYSDEL project and further work will only involve writing a simple XML converter.

# Part IV

# CASE STUDIES

# 21

# Introduction

In this part of the thesis we illustrate how the Multi-Parametric Toolbox, described in the previous part, can be used to synthesize model predictive controllers. We consider two case studies.

First, in Chapter 22 we investigate the problem of optimal infusion policy for injecting morphine and ketamine intravenously. In this project the aim is to optimally schedule time instances on which the drugs are injected into humans veins such that certain level of anaestesia is achieved and pain during a surgery is suppressed. In order to design an optimal injection policy, first the model of the drug concentration in the body (referred to as the pharmacokinetic model) is derived. The parameters of the model can be estimated from real measurement data collected during a surgery process. In addition, it is necessary to derive a model which, simply speaking, represents the patient's reaction to the drugs. This so-called "well-being" model, once derived, is then used to identify an optimal range of drugs which has to be reached by properly dosing the drug injection. Once both models are available, the control problem, which takes the discrete nature of the drug boluses into account, can be formulated. The optimal control problem involves both discrete variables (time instances at which a bolus is applied) and continuous variables (concentration of drugs in the blood and their effect on the patient's shape). The objective of the control problem is then to decide time instances on which to inject a bolus, such that a given concentration range of the drugs in the blood is reached. This is reached by optimizing the time instances such that the distance between the predicted drug concentration and a certain operating point is minimized over a finite prediction horizon. We show that control problems like this can be easily formulated using MPT and solved on-line in a receding horizon fashion.

The second case study concerns the real-time optimal control of a mechanical device with backlash. This kind of systems is very common in practice. Therefore, the task of optimally controlling such devices is of imminent importance. First, we show that backlash-like behavior can be captured with enough precision by the PWA modelling framework. The parameters of the model are derived using basic mass conservation relations and equations of motion. Accuracy of the model is subsequently verified with experimentally collected data. The controller design procedure is again based on a constrained finite time optimal control setup. Because the sampling time of the plant under consideration is very small ($T_s = 0.04$ seconds), in this case study we solve the control problem off-line using the techniques of multi-parametric programming outlined in Chapter 4. Specifically, we have investigated two different optimization-based control strategies. The first approach is based on formulating a CFTOC problem for a linear model of the system. This approach is later extended to cope with the PWA model of the device. Both problems have been formulated and solved using the Multi-Parametric Toolbox to obtain the optimal feedback laws in a look-up table form. Finally, on simulations we have compared the performance of both controllers against a simple LQR controller. As will be shown in Section 23.6, the MPC-based controllers provide satisfactory performance and guarantee constraint satisfaction.

# 22

# Optimal Infusion Policy of Intravenous Morphine and Ketamine

In anaesthesia, different drugs are injected into patients' bodies in order to eliminate the pain during a surgery and to keep patients asleep. Since different drugs have different effects on the human body, combinations of different drugs are usually used in order to amplify positive effects of the drugs and to mitigate the side effects. Therefore it is of imminent importance for the anaestheologist to choose a suitable ratio of multiple drugs to be used, as well as to properly inject such a mixture into the patients' blood. Although the process of anaesthesia is almost exclusively governed by experienced professionals, in this chapter we try to design an automated infusion policy which could, eventually, free the anaestheologist from routine.

Specifically, we aim at designing an optimal infusion strategy which takes the patients condition into account and administers the drugs following certain rules. Based on the known model of drug interactions and drug concentrations in humans blood we formulate an optimal control problem which takes this dynamical behavior into account, maximizes the patients comfort during the surgery, minimizes the side effects of the injected drugs, and also minimizes the drug consumption.

The general aim of this case study is to validate the application of the model on the clinical relevant question of the combination of intravenous morphine and ketamine and to show how this theoretical approach can be used to determine optimal drugs dosing.

To do so, we first derive a model of the drug concentration in humans body, as well as a model for drug interactions [MSS$^+$00]. We show that the parameters of the model can be estimated from real measured data. Then we consider the *w*ell-being

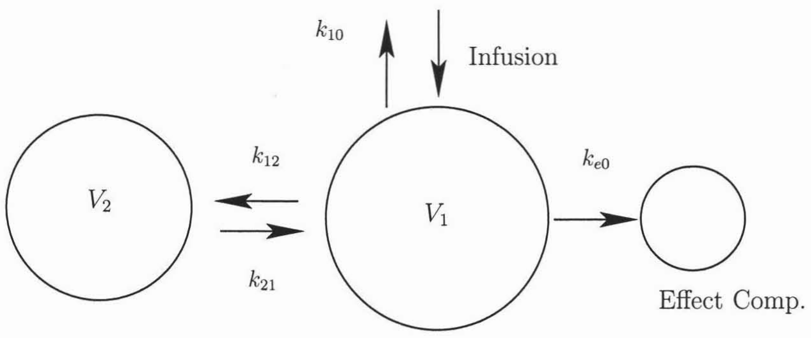


Figure 22.1: Representation of PK and link model.

parameter as the global outcome of the patient after drugs administration. The goal will be to keep the value of this parameter near a desired range.

## 22.1 Methods

### 22.1.1 The model

After administration, any drug distributes in the body reaching the site where the action takes place ("effect compartment"). The process can be described in a simplified way representing the body as composed of a limited set of compartments exchanging drug mass, as illustrated in Fig. 22.1, where the $k_{ij}$ represent the mass transport coefficients. The model of the distribution from infusion to plasma is referred to as pharmacokinetic (PK) model, while the link-model accounts for the distribution from the plasma to the effect compartment. The evolution of the effect can be described as a sigmoidal function of the drug concentration in the effect compartment (PD model). We consider each combination of drugs $A$ and $B$ as having a positive ($E_P$) and a side effect ($E_S$). According to the model [ZSS+05] these effects are described by the following equations:

$$E_i = E_{max_i}(\theta_i)\frac{(\frac{U_{A_i}+U_{B_i}}{U_{50_i}(\theta_i)})^{\gamma_i(\theta)}}{1+(\frac{U_{A_i}+U_{B_i}}{U_{50_i}(\theta_i)})^{\gamma_i(\theta_i)}}, \quad i = P, S \tag{22.1}$$

where $U_{A_i}$ and $U_{B_i}$ are the normalized drug concentrations:

$$U_{A_i} = C_A/EC_{50,A_i}$$

$$U_{B_i} = C_B / EC_{50,B_i}$$
$$\theta_i = \frac{U_{B_i}}{U_{A_i} + U_{B_i}}$$

and

$$\begin{array}{rcl} E_{max_i}(\theta_i) & = & E_{maxA_i} + (E_{maxB_i} - E_{maxA_i} - \beta_{E_i}) \cdot \theta_i + \beta_{E_i} \cdot \theta_i^2 \\ \gamma_i(\theta_i) & = & \gamma_{A_i} + (\gamma_{B_i} - \gamma_{A_i} - \beta_{\gamma_i}) \cdot \theta_i + \beta_{\gamma_i} \cdot \theta_i^2 \\ U_{50_i}(\theta_i) & = & 1 - \beta_{U_{50\,i}} \cdot \theta_i + \beta_{U_{50\,i}} \cdot \theta_i^2 \end{array}$$

where $E_{maxj_i}$, $\gamma_{j_i}$, $EC_{50_{j\,i}}$ are the parameters of the PD model for drug $j$ when administered alone. According to the model [ZSS$^+$05], the well-being ($W$) of the patient can be defined as the algebraic sum of positive and side effects:

$$W = E_P - \omega \cdot E_S$$

where $\omega$ is the relative weight of the two effects. The well-being can be represented as a 3-D function of the drugs effect compartment concentrations.
In order to draw the well-being surface of the combination of morphine and ketamine, all the parameters mentioned above are needed. According to [ZSS$^+$05] the following model simplifications can be made:

- $E_{max,A} = E_{max,B} = E_{max,AS} = E_{max,BS} = 1$
- $\beta_{E_{max}} = \beta_{Emax_S} = 0$
- $\beta_\gamma = \beta_{\gamma_S} = 0$

Thus, only the two interaction parameters $\beta_{U_{50}}$ and $\beta_{U_{50_S}}$ need to be known, in the following referred to as $\beta$ and $\beta_S$. The PD parameters for both drugs and the two interaction parameters $\beta$ and $\beta_S$ were either found in the literature [MPL91] or estimated from published experimental data.
Further, in order to relate the infused drug amount to the obtained effect computed through the well-being surface, the parameters of the PK and the link models are needed. All the PK model parameters for both drugs were available in the literature [PLC$^+$87], [HSB$^+$90], while the parameters of the link model for both drugs were estimated from published experimental data [MATC00], [MPL91]. In particular, for morphine and ketamine, we considered analgesia as positive effect and dizziness as negative, since this last was the most commonly experienced by patients in the administered dose-range. For details of the estimation procedures, we refer the reader to [SZS$^+$05].

## 22.1.2 The Optimal Control Problem

In this section we show how drug infusion policy can be optimized based on a known model of drug concentrations and their interactions, while respecting certain constraints. Following [SGE+03], the objectives for the optimal control problem can be formulated assuming that:

- The patients receive a solution containing a fixed mass ratio $\phi$ of the two drugs.
- The solution is administered always in bolus doses (impulses) of 1 ml.
- The minimal time interval between two boluses is 400 seconds.

The aim of the optimization is to find the optimal drug ratio $\phi$ in the solution and the optimal time sequence of boluses to bring and maintain the patients at the target with minimal drug consumption.
First, we state the optimal control problem for a class of linear systems in discrete-time domain of the following form:

$$x(k+1) = Ax(k) + Bu(k), \tag{22.2a}$$

$$y(k) = Cx(k), \tag{22.2b}$$

where $x(k) \in \mathcal{R}^n$, $u(k) \in \mathcal{R}^m$, and $y(k) \in \mathcal{R}^p$ are the state, input and output vectors respectively. In this specific case, the matrices A, B and C are given by:

$$A = \begin{bmatrix} A_{morph} & 0 \\ 0 & A_{ket} \end{bmatrix} \quad B = \begin{bmatrix} B_{morph} & 0 \\ 0 & B_{ket} \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Note that the mass ratio $\phi$ enters the equations as $u_1/u_2$. According to their PK and link-models, the distribution of both drugs can be described by two compartments plus the effect compartment. Thus, the matrices $A_{ket}$ and $A_{morph}$ are:

$$A_j = \begin{bmatrix} -k_{10_j} - k_{12_j} & k_{21_j} V_{2_j}/V_{1_j} & 0 \\ k_{12_j} V_{1_j}/V_{2_j} & -k_{21_j} & 0 \\ k_{e0_j} & 0 & -k_{e0_j} \end{bmatrix}$$

$$B_j = \begin{bmatrix} 1/(V_{1_j} \cdot t_{bol}) \\ 0 \\ 0 \end{bmatrix},$$

where $j$=ket, morph., $t_{bol} = 20sec$ is the time interval within which a bolus dose of 1 ml is administered and $u(k)$ is the vector of drug doses of morphine and ketamine in $[mg]$.

The optimal sequence of inputs $U_N = [u(0), \dots, u(N-1)]$ can be obtained by solving the following optimization problem:

$$J_N^*(U_N, x_0) = \min_{u_0,\dots,u_{N-1}} \sum_{k=0}^{N-1} ||Ru_k||_1 + ||Q(y_k - r)||_1 \tag{22.3a}$$

$$\text{subj. to} \quad x_k \in \mathcal{X}, \qquad \forall k \in \{0, \dots, N-1\}, \tag{22.3b}$$

$$u_k \in \mathcal{U}, \qquad \forall k \in \{0, \dots, N-1\}, \tag{22.3c}$$

$$y_k \in \mathcal{Y}, \qquad \forall k \in \{0, \dots, N-1\}, \tag{22.3d}$$

$$x_{k+1} = Ax_k + Bu_k, \ \forall k \in \{0, \dots, N-1\}, \tag{22.3e}$$

$$y_k = Cx_k, \qquad \forall k \in \{0, \dots, N-1\}, \tag{22.3f}$$

where $x_k$ ($y_k$) denotes the state (output) vector at time $k$ obtained from $x_0$ by applying the sequence of inputs $U_k = [u_0, \dots, u_{k-1}]$ to the system (22.2). $Q$ and $R$ are user defined penalty matrices and $N$ is the prediction horizon. The variable $r$ denotes a reference point, $|| \cdot ||_1$ is a vector 1-norm, and $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{U}$ are given polyhedral sets which define constraints on system states, system outputs and system inputs, respectively. For a given initial state $x_0 = x(0)$, and a finite value of the prediction horizon $N$, problem (22.3) can be solved as a Linear Program (LP). The solution is a sequence of control moves $U_N$ which minimizes a given objective function (22.3a). As stated previously, it is possible to use the concept of optimal control to obtain the optimal drug mass ratio $\phi$ as well as the time instances at which the boluses should be delivered. The behavior of the system can be captured by the following set of `IF-THEN-ELSE` conditions:

**Algorithm 22.1.1**

```
1. IF $t = 0$
2.     choose $\phi_0$
3.     assume $x(k+1) = Ax(k) + B\phi_0,\ y(k) = Cx(k)$
4. ELSEIF $(k - t_B) \geq \tau$ AND apply = true
```

```
5.     $t_B = t$
6.     assume $x(k+1) = Ax(k) + B\phi_0,\ y(k) = Cx(k)$
7. ELSE
8.     assume $x(k+1) = Ax(k),\ y(k) = Cx(k)$
9. END
10. $t = t + 1$
11. Goto 1.
```

The first condition (lines 1-3) specifies that we are only allowed to select the mass ratio of the two drugs ($\phi_0$) at the very beginning, i.e. at time $t = 0$. After $\phi_0$ is chosen, it is assumed to stay fixed for all subsequent time instances. The condition on line 4 dictates that we are only allowed to deliver the bolus if $\tau$ time instances have elapsed since the last bolus. These assumptions were chosen to exactly reproduce the conditions available during a patients controlled analgesia [SGE+03]. The variable *apply* denotes a logical decision which evaluates to *true* if we decide to administer the bolus, and to *false* otherwise. We remark here that the value of the variable *apply* is to be decided by the optimization procedure. If the two conditions both evaluate to a true statement, we first remember the time when we decided to apply the bolus (line 5) and further assume that the evolution of the states of the system is driven by a given state update equation, assuming the value of $\phi_0$ was already decided on line 2. If neither the condition on line 1 nor the expression on line 4 are satisfied, we assume that we do not deliver any bolus and the system evolves autonomously according to line 8. Finally, the time variable $t$ is incremented and we return to line 1. Using the tool HYSDEL [Tor02], the above mentioned logic rules can be translated into a mathematical model which is suitable for optimization. The model in question is the so-called Mixed Logical Dynamical (MLD) [BM99a] model which is described by the following set of equalities and constraints:

$$x(k+1) = Ax(k) + B_1u(k) + B_2\delta(k) + B_3z(k) \tag{22.4a}$$

$$y(k) = Cx(k) + D_1u(k) + D_2\delta(k) + D_3z(k) \tag{22.4b}$$

$$E_2\delta(k) + E_3z(k) \leq E_1u(k) + E_4x(k) + E_5 \tag{22.4c}$$

Table 22.1: Estimated $k_{e0}$ and PD parameters of morphine and Ketamine

| | Morphine | Ketamine |
|---|---|---|
| $k_{e0}$ $[1/min]$ | 0.046 [MPL91] | 0.02 [MATC00] |
| $EC_{50}$ $[mg/l]$ | 0.063 [MPL91] | 0.032 [MATC00] |
| $\gamma$ $[-]$ | 1.4 [MPL91] | 1.54 [MATC00] |
| $EC_{50,S}$ $[mg/l]$ | 0.11 [ZLF+94] | 1.46 [MATC00] |
| $\gamma_S$ $[-]$ | 0.6 [ZLF+94] | 0.15 [MATC00] |

where $x \in \mathcal{R}^{n_c} \times \{0,1\}^{n_l}$ is a vector of continuous and binary states, $u \in \mathcal{R}^{m_c} \times \{0,1\}^{m_l}$ denotes the input vector, and $y \in \mathcal{R}^{p_c} \times \{0,1\}^{p_l}$ the output vector. $\delta \in \{0,1\}^{r_l}$, $z \in \mathcal{R}^{r_c}$ represent auxiliary binary and continuous variables introduced when transforming logic rules into mixed-integer linear inequalities (22.4c). Assuming that the evolution of the system states and outputs in (22.3e)- (22.3f) is driven by the MLD model (22.4), the optimal control problem can be formulated as a Mixed-Integer Linear Program (MILP), which can be solved efficiently using state-of-the-art software [ILO04].

## 22.2 Results

### 22.2.1 The model

The estimated mean values of the link model and the PD parameters of morphine and ketamine are reported in Table 22.1. The interaction parameters have been estimated as $\beta = 2.5$ and $\beta_S = -3$ [SGE+03]. For the details of the estimation procedures, please refer to [SZS+05]. The resulting well-being surface is shown in Fig. 22.2 and the corresponding contour lines in Fig. 22.4.

### 22.2.2 The Control Problem

We have solved (22.3) with the following parameters: prediction horizon $N = 54$, sampling time $t_s = 20$ seconds, minimum time between two boluses $\tau = 400$ seconds, penalty on manipulated variables $R = 10^{-4}I$, and penalty on variation of system

Table 22.2: Optimal time instances (in seconds) on which to apply a bolus.

| Index of bolus | Time of bolus | Index of bolus | Time of bolus |
|---|---|---|---|
| 1 | 0 (initial bolus) | 8 | 5000 |
| 2 | 400 | 9 | 6200 |
| 3 | 800 | 10 | 7400 |
| 4 | 1400 | 11 | 8800 |
| 5 | 2200 | 12 | 10200 |
| 6 | 3000 | 13 | 11600 |
| 7 | 4000 | 14 | 13000 |

outputs from a given references $Q = 10^3 I$. The reference point $r$ was chosen as

$$r = [0.03, 0.05]^T$$

Assuming that the patient has not received any drug before time zero (i.e. $x(0) = [0 \ldots 0]^T$), the given optimal control problem (22.3) with model of the system in the form of (22.4) has been formulated using the Multi-Parametric Toolbox and was solved in 5.5 seconds on a Pentium 4 2.8 GHz computer, using CPLEX 9.0 [ILO04]. The obtained optimal mass ratio of the two drugs was $\phi_{opt} = 0.55$.
To determine the time instances at which the boluses should be administered, such that the concentration of the two drugs in the effect compartment reaches the prescribed reference point (22.2.2), the optimal control problem (22.3) has to be solved at every time step for the updated measurements of the state $x_0 = x(k)$. The average time needed to solve the optimal control problem (22.3) on a 2.8 GHz Pentium 4 processor using Matlab and the CPLEX solver was 3.7 seconds, which is far below the selected sampling time of 20 seconds. We have carried out the calculations for the time interval of 4 hours and obtained the results reported in Table 22.2. The time evolutions of the concentrations of morphine and ketamine in the effect compartment are depicted in Figure 22.3. As can be seen from the picture, both concentrations are kept close to their respective reference points. Figure 22.4 depicts the concentrations of ketamine and morphine concentration during the simulation.

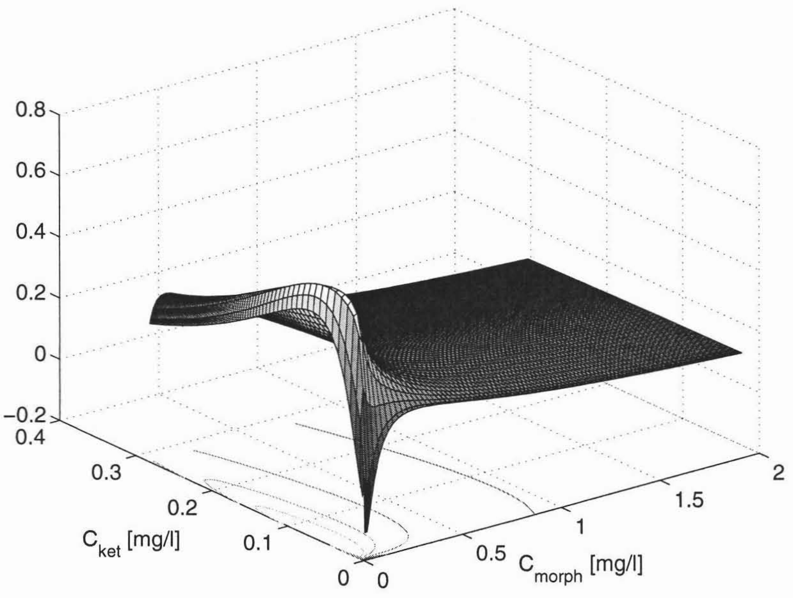


Figure 22.2: Well-being surface as a function of the effect compartment concentrations of morphine and ketamine, according to the estimated parameters.

## 22.3 Discussion

In the previously proposed model [ZSS$^+$05] the limitations on the administrable amount of drugs imposed by the occurrence of side effects are directly taken into account in the global expression of the well-being function. This allows for the identification of the effect compartment concentrations range maximizing the well-being of the patient. This approach provides a tool to investigate the behavior of drug combinations and, in particular, a tool to address the critical issue of optimal drug dosing, by stating the identification of the optimal dosing as a control problem. The complexity of the control of biological systems is often complicated by the non-linearity describing the underlying processes. The application of the new interaction model allows for overcoming this complication by reformulating the problem of controlling a defined effect into the control of the corresponding effect compartment concentrations. The formulation of the problem in terms of a Mixed-Integer Linear Programs is able to capture the intrinsic logic behind common drug administration in clinical practice. The results depicted in Fig. 22.3 and 22.4 show that the computed administration policy succeeded in bringing and maintaining the drug concentrations inside the optimal

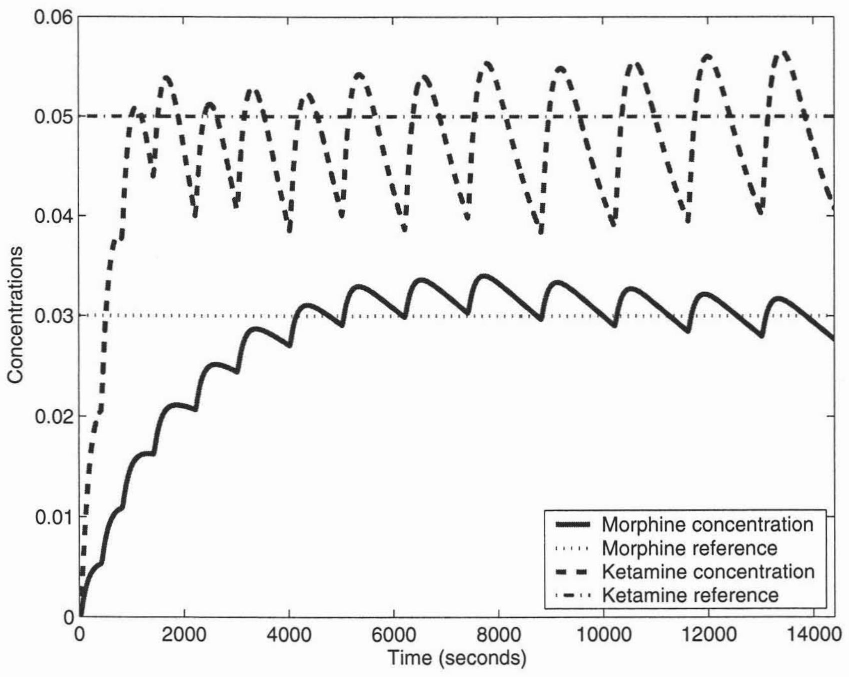


Figure 22.3: Evolution of concentrations of morphine and ketamine with respect to time according to the optimal infusion policy.

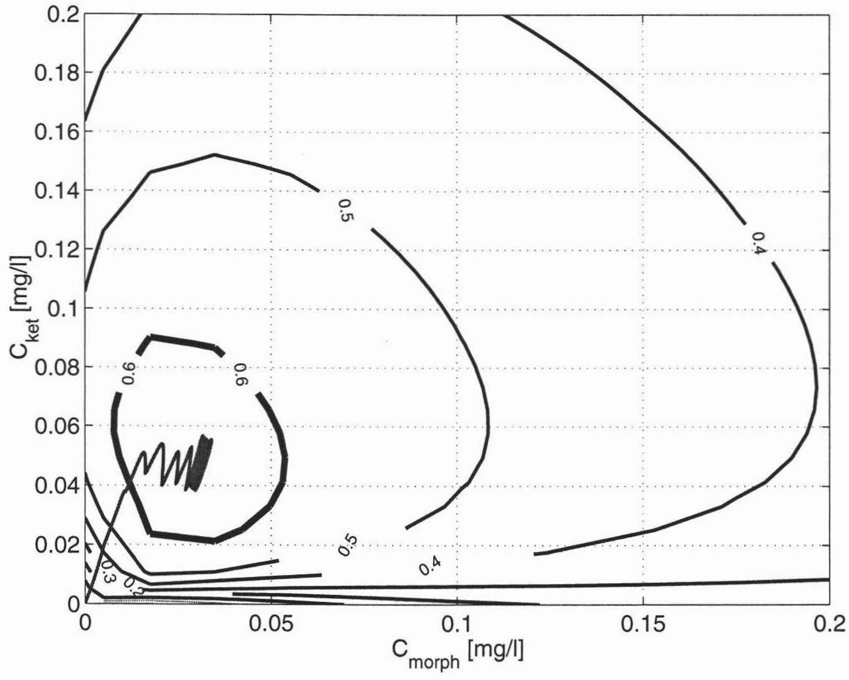


Figure 22.4: Well-being contour lines and the concentrations of morphine and ketamine obtained during a simulated period of 4 hours, according to the optimal infusion policy.

range. The concentration of morphine rapidly reaches the reference value and oscillates around this point. The concentration of morphine shows a slower response, therefore the reference value is reached slightly later. An oscillatory behavior is present in both cases, because of the discrete nature of the input. A decrease in the minimal time span between two consecutive boluses would lead to a decrement of the oscillations and in a smoother reference tracking. The result is satisfactory, since after having quickly pushed the concentrations into the optimal range, the controller is able to keep the system in this condition and thus to provide the maximal well-being for the patient.

## 22.4 Conclusion

In this chapter we have shown that the new proposed model for drug interactions can successfully be applied to a clinically relevant combination and builds the framework for an optimal approach to drug administration. The methodology is based on an optimal control framework, assuming that the model of the system can be described with a set of `IF-THEN-ELSE` conditions. The resulting optimization problem can be formulated as a Mixed-Integer Linear Program which can be solved in order to obtain the optimal mass ratio of morphine and ketamine in any bolus, as well as the time instances on which the boluses should be administered. The results show that the optimal control framework is well suited for the given task and provides a systematic approach to deal with the logic involved in drug administration.

# 23

# Control of Mechanical Systems with Backlash

## 23.1 Introduction

Backlash is a common problem in mechanical systems occurring whenever there is a gap in the transmission link, e.g. in the differential gearbox of a power train. This transmission gap causes problems when the system input changes from acceleration to braking or vice versa. During a short time interval the driving torque will not be transmitted to the load. When the backlash gap is traversed, sudden contact will cause a large change in the torque exercised on the load, causing undesirable bumps and possible damages of the mechanical elements in contact. Furthermore, improperly designed control for mechanical systems with backlash may cause undesired vibrations, thus limiting performance and causing additional wear of the mechanical parts.

The problem of controlling mechanical systems with backlash has been considered for a long time by the control community. A recent survey of this topic includes [Lag01] and [NG02]. A detailed treatment of different approaches towards control and estimation of mechanical systems with backlash is given in [Lag04]. New developments have followed the developments in the theory of *hybrid systems*, [MBB03], and (relatively) recent achievements in the area of *model predictive control* (MPC), [BBBM05].

A natural way to model a mechanical system with backlash is to distinguish between two operating modes, namely the "backlash mode", when the two mechanical parts are not in contact, and the "contact mode", when the contact between the two mechanical parts is established and the transmission of the momentum takes place. The inherent switching between these two modes makes this system a prime example of a hybrid system and motivates the "hybrid approach" to modeling and control of mechanical systems with backlash.

A Model predictive control strategy is particularly convenient when the system to be controlled is subject to *constraints*. The *Explicit solution* to a MPC problem, which is obtained off-line, makes the application of this control scheme possible also for systems requiring a fast control sampling rate. The approach is successful for constrained linear system, as well as for a class of hybrid systems, namely *piecewise affine* (PWA) systems. The application of MPC to mechanical systems containing backlash is particularly attractive since it enables a control design incorporating constraints which could increase the safety and reduce the wear of mechanical parts, while preserving satisfactory control performance. An application of MPC to automotive power trains with backlash has been reported in [LE05], where the authors deploy a linear acceleration controller for the system in backlash mode and MPC to traverse the backlash gap.

In this chapter we perform a comparative study of three different control strategies on a mechanical system with elastic shaft and backlash in the transmission. We compare the performance of classical LQR to MPC designed for a linear model (and applied to the actual system with backlash) and MPC designed using a *hybrid model* of the system. Unlike [LE05], we present a more holistic approach by modeling the mechanical system as a hybrid system and computing the explicit MPC for such a system. The emphasis of this chapter is on potential benefits that could be obtained by applying MPC to such systems, in particular, related to the satisfaction of different constraints. Furthermore, our aim is also to demonstrate a systematic hybrid control design procedure based on recently developed, freely available MPC controller design tools, [KGBM03a] and [Löf04]. In the comparison both performance and complexity of the controllers are taken into account.

## 23.2 Mechanical System with Backlash

To be able to evaluate control and estimation results not only in theory but in a real application, a laboratory physical model of a mechanical system with backlash has been constructed. This laboratory setup, which models an automotive power train system, can be seen in Figure 23.1. The main elements of the experimental system are two rotating masses, the backlash element, a spring, two motors and two encoders that provide position measurements. The system is driven by a DC motor while another motor of the same type is used on the load side. The rotating masses represent the

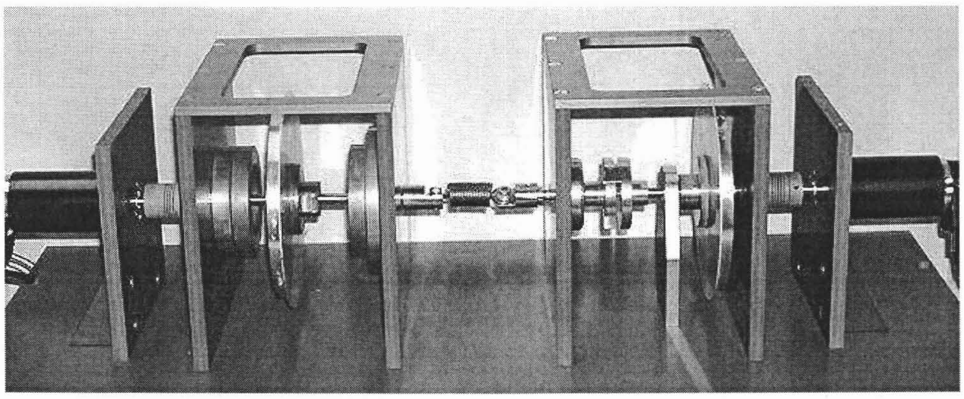


Figure 23.1: The experimental setup of a mechanical system with backlash.

inertia of the motor and the load. The spring connecting both sides (see Fig. 23.1) has been included to model the flexibility of the shaft. In contrast to automotive power trains, where shafts are usually rather stiff, we have chosen an elastic shaft, to make the laboratory-scale experiment more demanding. The backlash gap size of the backlash element can be changed to four different values, either 2°, 4°, 6° or 10°. The measured signals are the angles of the motor shaft $\theta_m$ and the load shaft $\theta_l$, which are obtained from two incremental encoders with a resolution of 2000 counts per revolution, i.e. approximately 0.0031 rad. This resolution is sufficient to measure the position of the system in backlash mode for the smallest backlash gap used ($2° \approx 0.035$ rad). However, if only sensors with a lower resolution are available, Kalman filters as proposed by [LE03a] can be used.

## 23.3 Modeling and Parameter Identification

The system described in Section 23.2 has been modeled using a first principle model. The modeling and parameter identification procedure is illustrated in this section.

### 23.3.1 Modeling of Hybrid Systems

Recognizing the hybrid nature of the system makes the process of modeling, control and state estimation more accurate and systematic. The models presented will be

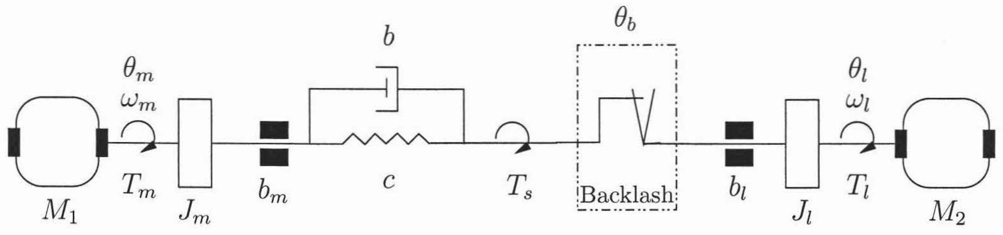


Figure 23.2: Schematic representation of a rotating mechanical system with backlash.

given in the form of piecewise-affine (PWA) systems

$$\begin{aligned} x_{k+1} &= f_{\text{PWA}}(x_k, u_k) = A^{\{i\}}x_k + B^{\{i\}}u_k + f^{\{i\}}, \\ &\text{if } \begin{bmatrix} x_k \\ u_k \end{bmatrix} \in \mathcal{D}^{\{i\}}, \\ \mathcal{D}^{\{i\}} &:= \left\{ \begin{bmatrix} x \\ u \end{bmatrix} \mid \begin{bmatrix} (P^x)^{\{i\}} & (P^u)^{\{i\}} \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} \le (P^0)^{\{i\}} \right\}, \end{aligned} \tag{23.1}$$

where $k \ge 0$, $x \in \mathbb{R}^n$ is the state vector, $u \in \mathbb{R}^m$ is the control vector and $\{\mathcal{D}^{\{i\}}\}_{i=1}^{D}$ is a bounded polyhedral partition of $(x, u) \subset \mathbb{R}^{n+m}$ space. The constraints $(P_x)^{\{i\}}x + (P_u)^{\{i\}}u \le (P^0)^{\{i\}}$ define both regions in which a particular state update equation is valid as well as constraints on the state and input variables. Under some technical assumptions, PWA system representation is *equivalent* to several other models of hybrid systems and one can convert one into the other, [HDB01]. For our purpose, i.e. the computation of the explicit solution to MPC for the hybrid systems, the PWA model (23.1) is the most suitable one.

### 23.3.2 Backlash Model

A schematic representation of the rotating mechanical system with backlash is shown in Figure 23.2. The motor $M_1$ is the driving motor. The inertia $J_m$ represents the motor flywheel, the inertia $J_l$ represents the load. The spring-damper combination models a flexible shaft with damping. The dampers $b_m$ and $b_l$ represent viscous friction. The second motor, $M_2$, can be used to model disturbance torques caused e.g. by different road friction. An important parameter in a rotating mechanical system with backlash is the size of the backlash gap, which shall be denoted as $2\alpha$.

The configuration of Figure 23.2 can be described approximately by the following differential equations, using balance of moments:

$$J_m \dot{\omega}_m = -b_m \omega_m + T_m - T_s, \quad (23.2)$$

$$J_l \dot{\omega}_l = -b_l \omega_l + T_s, \quad (23.3)$$

where $T_m$ is the motor torque and $T_s$ is the shaft torque. When shaft damping is taken into account, the shaft torque, $T_s$, is given by

$$T_s = c(\Delta\theta - \theta_b) + b(\omega_m - \omega_l)\,. \quad (23.4)$$

Here, the angles $\Delta\theta = \theta_m - \theta_l$ and $\theta_b$ are the total shaft displacement and the position in backlash, respectively. The backlash position angle $\theta_b$ can be described by the following nonlinear differential equation, [Ne97],

$$\dot{\theta}_b = \begin{cases} \max[0, \dot{\theta}_d + \frac{c}{b}(\theta_d - \theta_b)] & \text{if } \theta_b = -\alpha \\ \dot{\theta}_d + \frac{c}{b}(\theta_d - \theta_b) & \text{if } |\theta_b| < \alpha \\ \min[0, \dot{\theta}_d + \frac{c}{b}(\theta_d - \theta_b)] & \text{if } \theta_b = \alpha \end{cases}, \quad (23.5)$$

where $\alpha$ is half the backlash gap size. From this differential equation for $\theta_b$ conditions can be derived which define when the system is in backlash mode. The system is in backlash mode if one of the following three conditions holds

$$|\theta_b| < \alpha \quad (23.7a)$$

$$\theta_b = -\alpha \ \wedge \ \dot{\theta}_d + \tfrac{c}{b}(\theta_d - \theta_b) > 0 \quad (23.7b)$$

$$\theta_b = \alpha \ \wedge \ \dot{\theta}_d + \tfrac{c}{b}(\theta_d - \theta_b) < 0 \quad (23.7c)$$

Conditions (23.7*b*) or (23.7*c*) become true, if the system is in positive or negative contact mode and the backlash elements starts to move away from the driving shaft. The distribution of the backlash and the contact mode over the backlash angle $\theta_b$ and its derivative $\dot{\theta}_b$ is visualized in Figure 23.3. While the contact mode is marked by the two thick lines, the backlash mode consists of the remaining reachable space.

This leads to the state update equation:

$$\dot{x}(t) = \begin{cases} A_{co}x(t) + Bu(t) & \text{(Positive contact)} \\ A_{bl}x(t) + Bu(t) & \text{(Backlash)} \\ A_{co}x(t) + Bu(t) & \text{(Negative contact)} \end{cases} \quad (23.8)$$

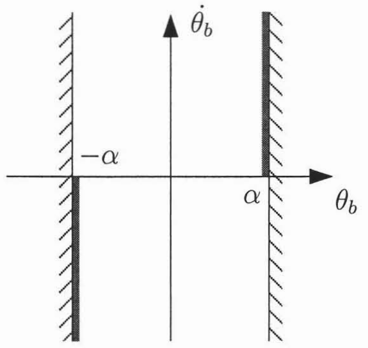


Figure 23.3: Distribution of backlash and contact mode. Dark line: contact mode, white space: backlash mode

where the state $x(t)$ and the matrices $A_{co}$, $A_{bl}$ and $B$ are given by

$$x = \begin{bmatrix} \omega_m & \omega_l & \theta_m & \theta_l & \theta_b \end{bmatrix}^T, \tag{23.9}$$

$$A_{co} = \begin{bmatrix} -\frac{b_m+b}{J_m} & -\frac{b}{J_m} & -\frac{c}{J_m} & \frac{c}{J_m} & \frac{c}{J_m} \\ \frac{b}{J_l} & -\frac{b_l+b}{J_l} & \frac{c}{J_l} & -\frac{c}{J_l} & -\frac{c}{J_l} \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{23.10}$$

$$A_{bl} = \begin{bmatrix} -\frac{b_m}{J_m} & 0 & 0 & 0 & 0 \\ 0 & -\frac{b_l}{J_l} & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & \frac{c}{b} & -\frac{c}{b} & -\frac{c}{b} \end{bmatrix}, \tag{23.11}$$

$$B = \begin{bmatrix} \frac{K}{J_m} & 0 & 0 & 0 & 0 \end{bmatrix}^T. \tag{23.12}$$

Note that, since $\Delta\dot{\theta} = \omega_m - \omega_l$ can be expressed in terms of state variables, Eq. (23.5) together with Eq. (23.5) describes a continuous-time PWA system defined over a polyhedral partition.

For using discrete-time model predictive control, the continuous model from Equation (23.8) still has to be discretized. This can be done for each linear subsystem separately either straight forward via the Euler discretization or with a more sophisticated discretization algorithm. Finally, a discrete hybrid model

$$x[k+1] = \begin{cases} \widetilde{A}_{co}x[k] + \widetilde{B}u[k] & \text{(Positive contact)} \\ \widetilde{A}_{bl}x[k] + \widetilde{B}u[k] & \text{(Backlash)} \\ \widetilde{A}_{co}x[k] + \widetilde{B}u[k] & \text{(Negative contact)} \end{cases} \tag{23.13}$$

can be obtained where the $\widetilde{}$ denotes matrices of a discrete-time model. The models were implemented with special tools alongside Matlab to deal with hybrid systems.For simulation of PWA systems and design and simulation of controllers for PWA systems the Multi-Parametric Toolbox (MPT) was used.

### 23.3.3 Parameter Identification

In order to identify the parameters for the model, the backlash element was removed from the experimental system and the shafts were connected. This leads to a system with almost linear behavior governed by the following differential equations:

$$\begin{aligned} J_m\dot{\omega}_m &= -b_m\omega_m + T_m - c(\theta_m - \theta_l) \\ J_l\dot{\omega}_l &= -b_l\omega_l + c(\theta_m - \theta_l) + b(\omega_m - \omega_l) \end{aligned} \tag{23.14}$$

The numerical values for the parameter can now be obtained using linear subspace identification methods, e.g. with the Matlab Identification Toolbox, [Lju06]. Since we used a first principle modeling approach, the parameters of the linear system are the same as the corresponding contact mode parameters in the hybrid model. Therefore, the identified parameters of the linear system can be used in the hybrid model that includes backlash. Afterwards backlash mode parameters can be extracted using physical relations.

For the identification of the system in contact mode, a pseudo-random binary signal has been used with an amplitude of $\pm 5$ [V]. Figure 23.4 shows the first three seconds of the identification signal and the corresponding system response, i.e. the evolution of the motor shaft angle $\theta_m$ and the angle of the load shaft $\theta_l$.

The derived linear model is verified by comparing the model simulations to measurements as shown in Figure 23.5. It can be concluded that the model is quite accurate although some nonlinearities are not considered, in particular the strong influence of Coulomb friction.

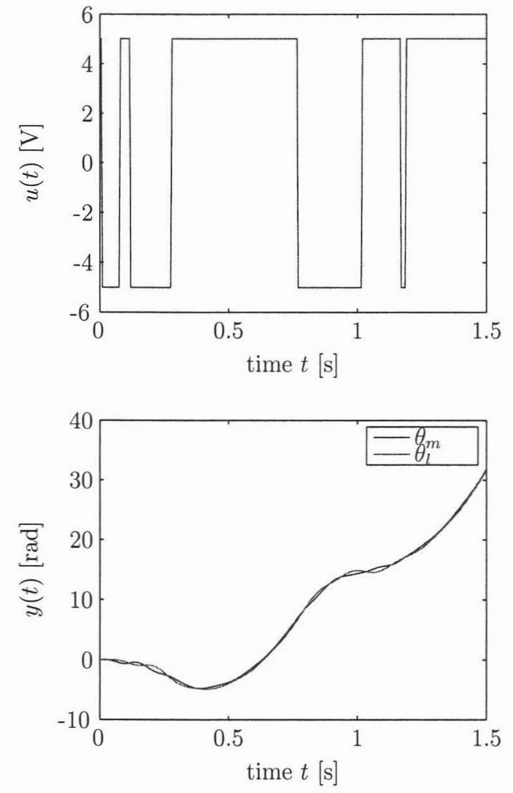


Figure 23.4: First three seconds of input and output signals during identification.

## 23.4 State Estimation

In the experimental setup considered here only the motor and load position are directly available through measurement. For the purpose of state-feedback control the remaining of the states, i.e. position in backlash $\theta_b$ and rotational speed of motor and load $\omega_m$ and $\omega_l$ need to be estimated. This motivates the design of a state estimator described in this section.

In applications where the switching signal is assumed to be available or can be derived easily by observing sign-changes in the motor input signal, switching between linear observers can be used to recover the states of the system, [AC01]. Due to the high elasticity of the spring in the experimental system at hand, a considerable time delay between sign-changes in the input signal and the actual switching can occur. In order to cope with this, a switching observer is designed which does not rely on an external switching sequence, [LE03b].

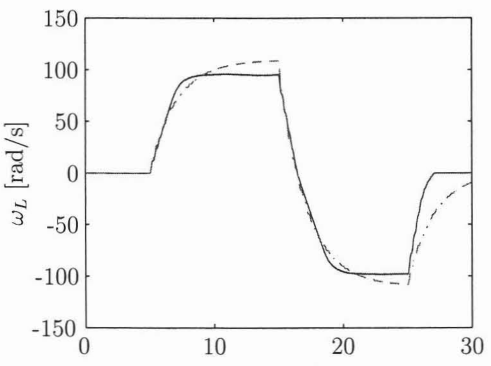


Figure 23.5: Comparison between measurements (—) and simulation results (– –).

The size of the backlash gap $2\alpha$ is considered to be known. In general however this gap size may be unknown. In these cases this parameter can either be identified in the model identification process or estimated during the system operation. In [LE03b] it is argued that in most cases on-line estimation is the only feasible option.

Since the piecewise-affine model (23.8) consists only of two distinct linear dynamics, the design of an extended Kalman filter (EKF) is similar to the design of two Kalman filters for each linear dynamic. A linearisation of the nonlinear dynamic done at each time-step in a classical EKF is nothing more but using the linear dynamic valid in each region. The advantage of this decoupled design, proposed in [LE03b], is that linear steady state gains can be calculated as stationary solutions to the Riccati equation. In general this is not possible in EKF design.

For the switched observer we choose the following structure

$$\dot{\widehat{x}}(t) = \begin{cases} A_{co}\widehat{x}(t) + Bu(t) + L_1\Delta y(t) & \text{(Contact)} \\ A_{bl}\widehat{x}(t) + Bu(t) + L_2\Delta y(t) & \text{(Backlash)} \end{cases} \tag{23.15$a$}$$

$$\widehat{y}(t) = C\widehat{x}(t), \tag{23.15$b$}$$

and switching condition:

$$\text{(Contact)}: \begin{cases} & \widehat{\theta}_b & = -\alpha & \wedge\ \widehat{\omega}_m - \widehat{\omega}_l + \frac{c}{b}(\widehat{\theta}_d - \widehat{\theta}_b) < 0 \\ \vee & \widehat{\theta}_b & = \alpha & \wedge\ \widehat{\omega}_m - \widehat{\omega}_l + \frac{c}{b}(\widehat{\theta}_d - \widehat{\theta}_b) > 0 \end{cases} \tag{23.16$a$}$$

$$\text{(Backlash)}: \begin{cases} & \widehat{\theta}_b & = -\alpha & \wedge\ \widehat{\omega}_m - \widehat{\omega}_l + \frac{c}{b}(\widehat{\theta}_d - \widehat{\theta}_b) \geq 0 \\ \vee & |\widehat{\theta}_b| & < \alpha & \\ \vee & \widehat{\theta}_b & = \alpha & \wedge\ \widehat{\omega}_m - \widehat{\omega}_l + \frac{c}{b}(\widehat{\theta}_d - \widehat{\theta}_b) \leq 0 \end{cases} \tag{23.16$b$}$$

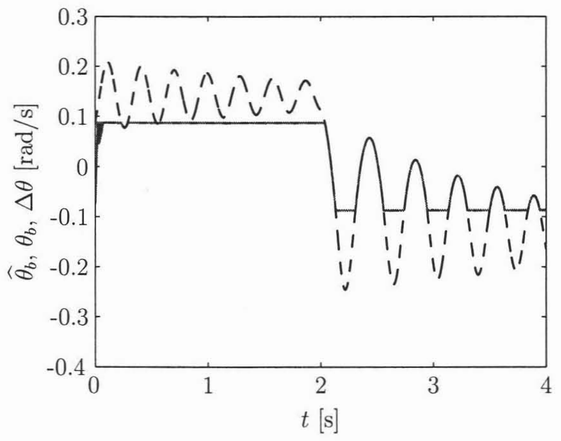


Figure 23.6: Simulation of the state estimation scheme, Position in Backlash $\widehat{\theta}_b$ (—) and $\Delta\theta$ (- -)

where $\widehat{x}(t) = [\widehat{\omega}_m, \widehat{\omega}_l, \widehat{\theta}_m, \widehat{\theta}_l, \widehat{\theta}_b] \in \mathbb{R}^5$ is the estimated state vector at time $t$, $\Delta y(t) = y(t) - \widehat{y}(t)$ is the output estimation error at time $t$ and $L_1, L_2 \in \mathbb{R}^{5\times 2}$ are Kalman filter gains, computed for each linear subsystem separately.

## 23.4.1 Validation

The validation of the proposed filter has been done in two steps. First a continuous time model of the system including measurement quantization has been simulated together with the designed observer. A comparison between estimated backlash position and simulated position within backlash is shown in Figure 23.6. The validation is similar for the other states: the employed estimation scheme shows a fast convergence to the actual states.

Finally the observer was verified experimentally. Since only position measurements $\theta_m$ and $\theta_l$ are available, the comparison (Fig. 23.7) is done using the measurement of the difference between both positions, $\Delta\theta$, and the estimated position within backlash $\theta_b$. The position in backlash can be estimated sufficiently well as shown in simulation and experimental verification.

It could be observed during experimental verification that the switching signal shows undesirably high frequency variations in contact mode for high weights on the position $\theta_b$. Low values however lead to a steady state position error in $\theta_m$ and $\theta_l$. A proper trade off has been found with the values stated above.

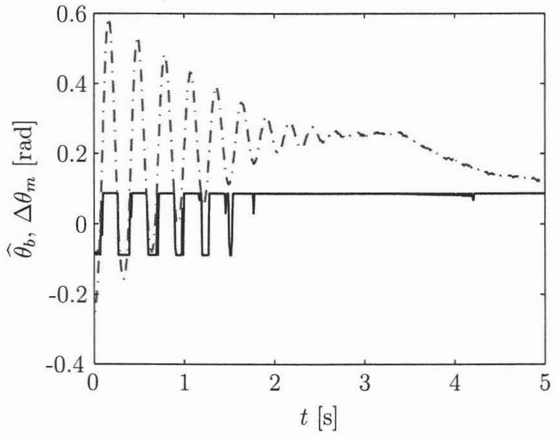


Figure 23.7: Experimental verification of the state estimation scheme, Position in Backlash $\widehat{\theta}_b$ (—) and $\Delta\theta$ (- -)

*Remark:* Stability of the designed estimation scheme has not yet been verified. This would be possible with a Lyapunov analysis of the state estimation error, see [JHW02].

## 23.5 Controller Design

In this section we will briefly describe design of explicit MPC based on the PWA model presented in section 23.3.2. The model includes the position in backlash, $\theta_b$, in contrast to the approach in [LE05], were a simplified model is used. This allows us to use all available information from the state observer presented in the previous section and may lead to better control performance. The main aim of the control will be to keep the impact velocity between the motor and the load part within certain bounds.

### 23.5.1 Model Predictive Control for constrained linear and PWA systems

In this section we will give a brief overview of the results related to the explicit solution of the MPC problem for constrained linear and PWA systems.

We consider constrained discrete-time linear systems defined by the following state update equation:

$$x_{k+1} = Ax_k + Bu_k, \qquad x \in \mathbb{X}, u \in \mathbb{U} \tag{23.17}$$

where $\mathbb{X}$ and $\mathbb{U}$ are polyhedral sets defining constraints on states $x \in \mathbb{R}^n$ and inputs $u \in \mathbb{R}^m$.

Given a model of the system, we denote the model based $k-$step prediction of state $x$ at time instance $t$ as $x_{t+k|t}$ for a given sequence of inputs $u_{t+k|t}$. Consider the following cost function of states and inputs:

$$J(U_t^{N_c-1}, x_0) := \|(x_{t+N|t} - x_r)\|_{P_N}^2 + \sum_{k=0}^{N-1} \|(x_{t+k|t} - x_r)\|_Q^2 + \sum_{k=0}^{N_c-1} \|u_{t+k|t} - u_r\|_R^2, \quad (23.18)$$

where $N$ and $N_c \leq N$ stand for the prediction and control horizon, respectively. We use the following shortened notation: $\|x\|_M^2 = x^T M x$. Weight matrices $P_N$, $Q$ and $R$ in (23.18) are assumed to be positive semi-definite. Vector $U_t^{N_c-1} = \left[u_{t|t}^T, \ldots, u_{t+N_c-1|t}^T\right]^T \in \mathbb{R}^{mN_c}$ contains all control inputs over the horizon and $x_r$ and $u_r$ denote reference (desired) values for the state and control vectors, respectively.

Consider the following *constrained finite time optimal control* (CFTOC) problem:

$$J_N^*(x_{t|t}) := \min_{U_t^{N-1}} J_N(U_t^{N-1}, x_{t|t}), \quad (23.19a)$$

$$\text{subj. to} \quad \begin{cases} x_{t+k+1|t} = f_{\text{DYN}}(x_{t+k|t}, u_{t+k|t}), \\ x_{t+N|t} \in \mathbb{T}, \end{cases} \quad (23.19b)$$

where $f_{DYN}(x_{t+k|t}, u_{t+k|t})$ represents a state update equation ((23.1) or (23.17)). Set $\mathbb{T}$ is a compact polyhedral *terminal set*, i.e. the set of admissible states at the final (finite) time instance $N$.

Classical MPC implementation assumes solving the problem (23.19a)-(23.19b) *on-line*, i.e. at each time instance $t$ for a given (measured) current state $x_{t|t}$ and using the optimal $u_{t|t}^*$, if exists, as the control input at time $t$. This amounts to solving a *mixed-integer quadratic program* (MIQP) at each time instance $t$. Due to computational requirements, this approach is intractable for control systems requiring high sampling rate.

As shown in [BBBM05], the problem (23.19a)-(23.19b) for constrained LTI and PWA systems can be solved explicitly *off-line* using dynamic programming and multi-parametric quadratic program. The generated explicit solution is a look-up table whose structure is defined by the properties of the solution to the problem (23.19a)- (23.19b)). Those are summarized by the following theorems [Bor03]:

**Theorem 23.5.1 (CFTOC solution for constrained linear systems)** *For an optimal control problem (23.19a)-(23.19b) with the state update equation* (23.17) *and* *cost function* $J(U_t^{N_c-1}, x_{t|t})$ *defined by* (23.18), *the optimizer function* $u^*(x_{t+k|t})$ *is* continuous and piecewise-affine over polyhedra, *i.e. of the form:*

$$u^*(x_{t+k|t}) = F_k^{\{i\}} x_{t|t} + G_k^{\{i\}} \quad \textit{if} \ x_{t+k|t} \in \mathcal{R}_k^{\{i\}}, \tag{23.20}$$

*where* $\mathcal{R}_k^{\{i\}}$, $i = 1, \ldots, N_k$ *are polyhedra defining a polyhedral partition of the set* $\mathbb{X}_k$ *of states* $x_{t+k|t}$ *for which the problem (23.19a)-(23.19b) has a feasible solution.*

**Theorem 23.5.2 (CFTOC solution for PWA systems)** *The solution to the optimal control problem (23.19a)-(23.19b) with the quadratic cost function* (23.18) *is a PWA state feedback control law:*

$$u^*(x_{t+k|t}) = F_k^{\{i\}} x_{t|t} + G_k^{\{i\}} \quad \textit{if} \ x_{t+k|t} \in \mathcal{R}_k^{\{i\}}, \tag{23.21}$$

*where the regions* $\mathcal{R}_k^{\{i\}}$, $\quad i = 1, \ldots, N_k$ *whose closure is given by:*

$$\bar{\mathcal{R}}_k^{\{i\}} = \{x \mid x'^T H_k^i x + q_k^i x \le r_k^i\} \tag{23.22}$$

*define a partition of the set of feasible states* $\mathbb{X}_k$ *in the* $k$*-th step .*

The on-line operation of the controller consists of identification of pre-computed affine feedback control law for a measured (estimated) state $x_{t|t}$ by searching for a region $\mathcal{R}_0^{\{i\}}$ containing the state vector $x_{t|t}$. This way the computational overhead is moved off-line and the real-time deployment of the MPC becomes possible.

### 23.5.2 MPC design for mechanical system with backlash

The goal we wish to achieve with MPC is tracking of a speed reference for the load, with satisfaction of certain constraints. For the comparison we design MPC based on a constrained linear model describing the system in *contact mode* (i.e. ignoring the backlash mode) and MPC using the PWA model of the system.

We introduce constraints on the difference between $\omega_m$ and $\omega_l$ in order to reduce impact forces between mechanical parts when traversing from backlash to contact mode. Additionally, in order to avoid feasibility problems that might occur from this constraint we define this constraint on speed difference to be a *soft constraint.*

Since we are interested only in the reference tracking of a particular state (speed of the load), we modify the general formulation by augmenting the state vector in the following way:

$$\widetilde{x}_{t+k|t} = \begin{bmatrix} x_{t+k|t} \\ u_{t+k-1|t} \\ \omega_{lR} \end{bmatrix}, \tag{23.23}$$

where $x_{t+k|t} = \begin{bmatrix} \omega_m & \omega_l & \Delta\theta & \theta_b \end{bmatrix}$ is the original state vector, $u_{t+k-1|t}$ is the control input in the previous sampling interval and $\omega_{lR}$ is a referent value for the rotational speed of the load ($\omega_l$). Instead of estimating the reference values for $u_r$, we use $\Delta u_{t+k|t} \triangleq u_{t+k|t} - u_{t+k-1|t}$ as the optimization variable. Formulation of the MPC control problem is the following:

$$\min_{\Delta U_t^{N_c-1},s} q_s s^2 + \sum_{k=0}^{N-1} \|\omega_l - \omega_{lR}\|_Q^2 + \sum_{k=0}^{N_c-1} \|\Delta u_{t+k|t}\|_R^2, \tag{23.24a}$$

$$\text{subj. to} \begin{cases} x_{t+k+1|t} = f_{\mathrm{DYN}}(x_{t+k|t}, u_{t+k|t}), \\ |\omega_l - \omega_m| \leq \Delta\omega_{\max} + s, \\ s \geq 0, \end{cases} \tag{23.24b}$$

where $s$ is an additional optimization variable introduced to enforce the soft constraints and $\Delta\omega_{\max}$ is the maximal difference between the rotation speeds between the load and the motor.

Augmentation of the state vector, needed for the tracking of arbitrary references, increases the dimension of the explicit solution. In order to reduce the complexity a move-blocking strategy is used, i.e. the optimization problem (23.19a)-(23.19b) is simplified by considering only $u_{t|t}$ and $u_{t+1|t}$ as optimization variables and set the value $u_{t+k|t} = u_{t+1|t}$ for $k > 1$.

## 23.6 Comparison of different control strategies

The control aim of the simulation experiment is to have the load shaft follow a certain speed reference trajectory. The simulation experiment involves one braking procedure from 25 [rad/s] to −10 [rad/s] and one accelerating procedure back to 15 [rad/s]. The initial speeds of the motor and the load shaft have been set to be above the reference trajectory and to differ slightly (29 [rad/s] and 30 [rad/s], respectively). The following constraints need to be fulfilled:

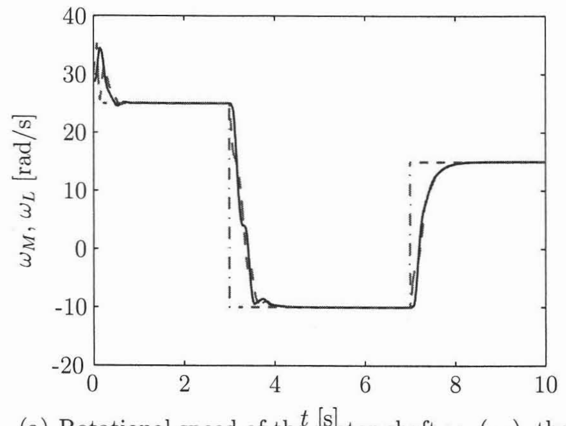
$$\begin{aligned} |u| &\leq 5\,\mathrm{V}, \\ |\theta_b| &\leq 5^\circ, \\ |\Delta\omega| &\leq \tfrac{\pi}{2}\,\mathrm{rad/s}, \end{aligned} \tag{23.25}$$

where the last one ($\Delta\omega$) is added as a soft constraint in the MPC design. As mentioned in the introduction, we compared the performance of the standard integral action LQR controller with MPC based on the constrained linear model and MPC using a hybrid PWA model of the system, as described in the Section 23.5.

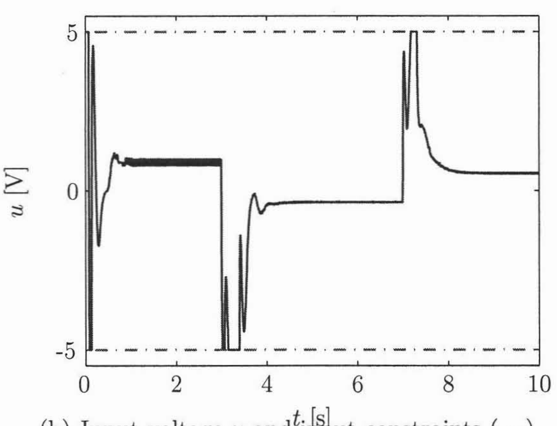
For the LQR controller design, the state vector is augmented by an integral of the tracking error to eliminate steady-state error. In the design a large weight was put on the state $\omega_l$ to achieve fast reference tracking. While it is physically impossible to exceed the input and the backlash angle constraints, the safety constraint can be violated.

Similar as in LQR, in both MPC a large weight is put on the term penalizing the tracking error in the cost function. For MPC we used discrete-time models with sampling period of $T_s = 0.04$ [s] and prediction horizon $N = 5$. As stated in Section 23.5.2, only two values of the control input are considered as optimization variables, thus achieving significant reduction in complexity of the explicit controllers. With this setup, we obtained an explicit MPC based on a constrained LTI model comprising 115 polyhedral regions in 6 dimensions and hybrid MPC containing 4890 polyhedral regions in 6 dimensions.

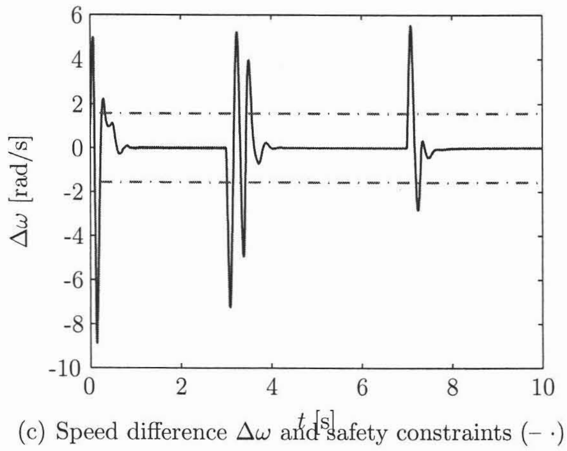
Simulation results are shown on Figures 23.8–23.10. It can be seen that all three controllers have approximately same performance in tracking the reference signal. Therefore, evaluation of these three control strategies has to be done by considering the complexity issue and the satisfaction of prescribed constraints. LQR admits very simple control law and provides satisfactory tracking performance. However, the constraint on the speed difference between the load and the motor cannot be enforced without sacrificing performance. MPC based on the linear model (Fig. 23.9) is relatively simple in structure and gives satisfactory performance with proper constraint satisfaction for the control input $u$. Again, MPC based on linear model comes quite close to fulfilling the constraints on $\Delta\omega$ (Fig. 23.9 (c)). Finally, MPC using the hybrid model of the system offers very tight constraint satisfaction (Fig. 23.10). At the same time, the complexity of this controller (storage and computational requirements), though still reasonable for performing laboratory tests, represents an obstacle for its practical implementation.



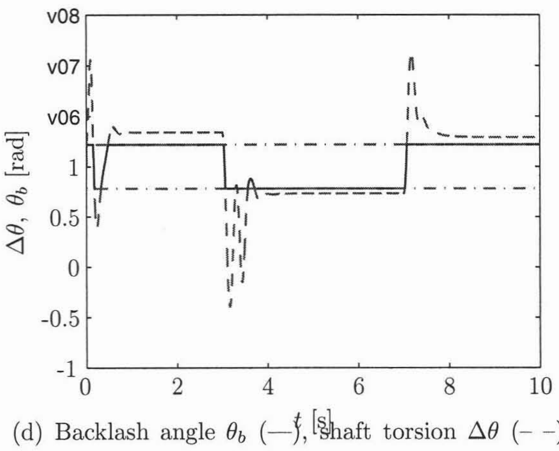
(a) Rotational speed of the motor shaft $\omega_m$(—), the load shaft $\omega_l$ (– –) and reference speed $\omega_{ref}$ (– ·)



(b) Input voltage $u$ and input constraints (– ·)

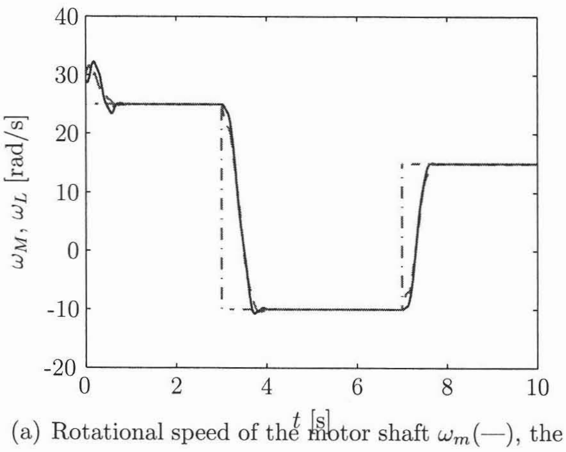


(c) Speed difference $\Delta\omega$ and safety constraints (– ·)

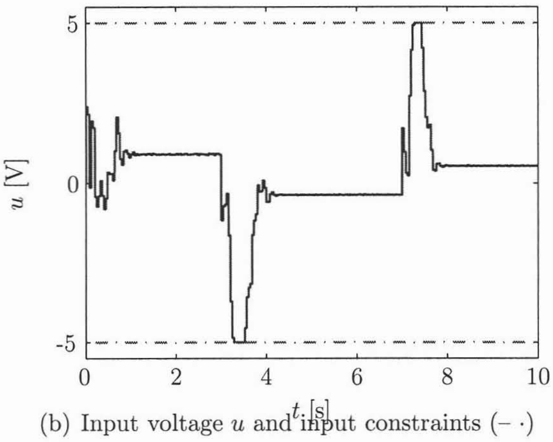


(d) Backlash angle $\theta_b$ (—), shaft torsion $\Delta\theta$ (– –) and backlash gap $\alpha$ (– ·)

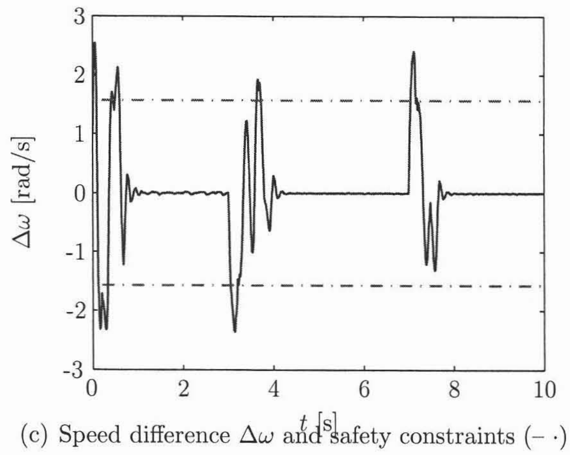
Figure 23.8: State evolution of the backlash system under LQR control.



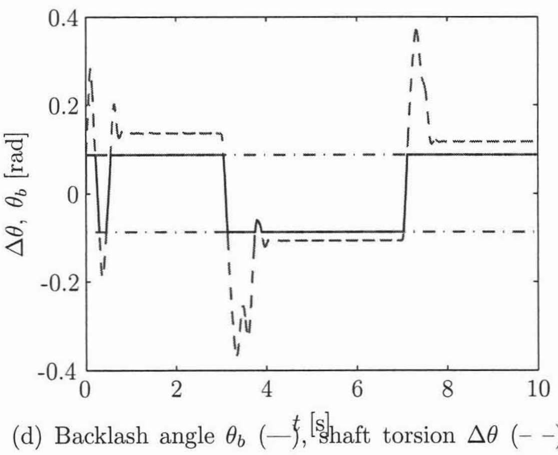
(a) Rotational speed of the motor shaft $\omega_m$(—), the load shaft $\omega_l$ (– –) and reference speed $\omega_{ref}$ (– ·)



(b) Input voltage $u$ and input constraints (– ·)

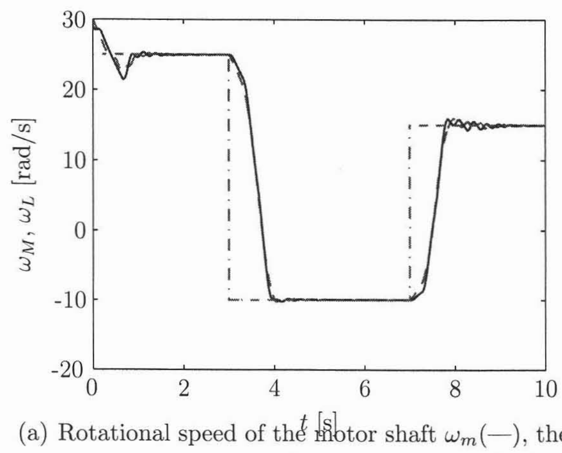


(c) Speed difference $\Delta\omega$ and safety constraints (– ·)

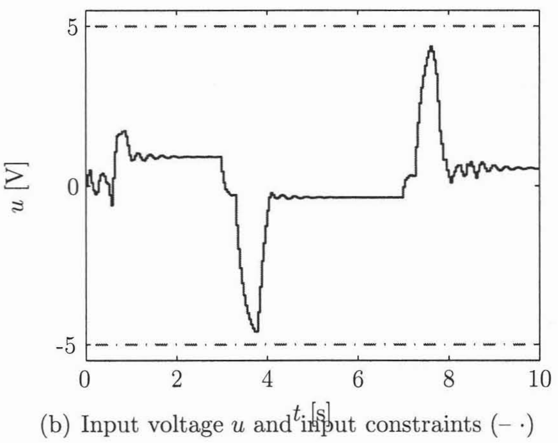


(d) Backlash angle $\theta_b$ (—), shaft torsion $\Delta\theta$ (– –) and backlash gap $\alpha$ (– ·)

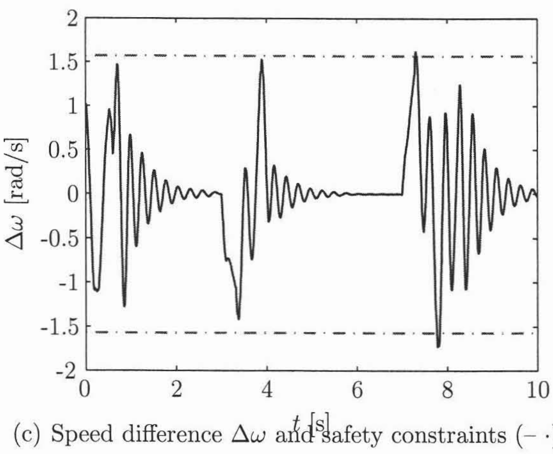
Figure 23.9: State evolution of the backlash system under linear MPC.



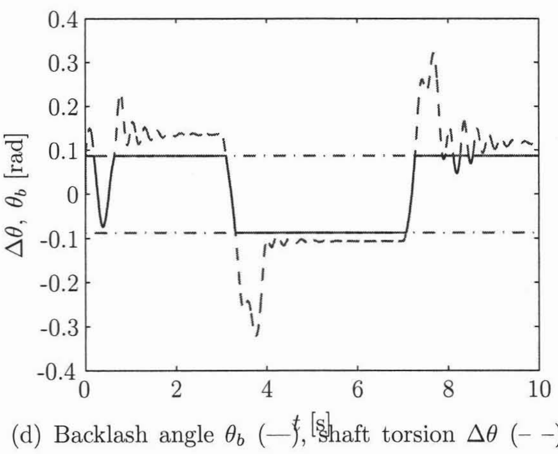
(a) Rotational speed of the motor shaft $\omega_m$(—), the load shaft $\omega_l$ (– –) and reference speed $\omega_{ref}$ (– ·)



(b) Input voltage $u$ and input constraints (– ·)



(c) Speed difference $\Delta\omega$ and safety constraints (– ·)



(d) Backlash angle $\theta_b$ (—), shaft torsion $\Delta\theta$ (– –) and backlash gap $\alpha$ (– ·)

Figure 23.10: State evolution of the backlash system under hybrid MPC.

## 23.7 Conclusion

A mechanical system with backlash has been specified and built, providing a benchmark system for hybrid control and identification strategies. Modern control strategies for hybrid systems can be verified and their realisability on a real system can be investigated. A first step towards a "hybrid" benchmark problem was obtained.

The full design cycle containing modeling, estimation and control design for this rotational system with backlash has been performed. An observer, based on [LE03b] was implemented and tested in simulations and experiments with the laboratory setup. In both situations the observer works satisfactorily. The goal was to design a controller that tracks the rotational speed of the load shaft while minimizing bumps or damages that can occur when the system is operated and backlash is not taken into account. This was realized by constraining the difference in speed between the drive and the load $\omega_m - \omega_l$.

Three different controllers were designed and investigated in a simulated scenario. A standard LQR controller, where constraints cannot be considered directly, was compared to model predictive control for a linear and a hybrid model. To ensure the implementability in real backlash systems like power trains of automobiles, explicit solutions have been computed in the latter cases. The different handling of the constraints by the investigated controllers have been shown during these simulations. While the constraints are violated significantly with LQR control, MPC with a linear model shows only small violations of the constraints. With a hybrid model, which considers the backlash in the system, violations of the constraints can be avoided.

The explicit solution for the hybrid model comprises thousands of regions, such that additional effort has to be spent in future research to reduce the complexity. Explicit solutions based on the minimum time control concept seem to be more suitable for a real time implementation. Analyzing these questions is part of the ongoing work to find an efficient realization of the MPC control scheme for the experimental system.

## Acknowledgments

This work was supported by the European Commission research project FP6-IST-511368 *Hybrid Control (HYCON).*

# Part V

# APPENDIX

# 24

# Car on a PWA Hill

## 24.1 System dynamics

Assume a frictionless car moving horizontally on a hill with different slopes, as illustrated in Figure 24.1. Dynamics of the car is driven by Newton's laws of motion:

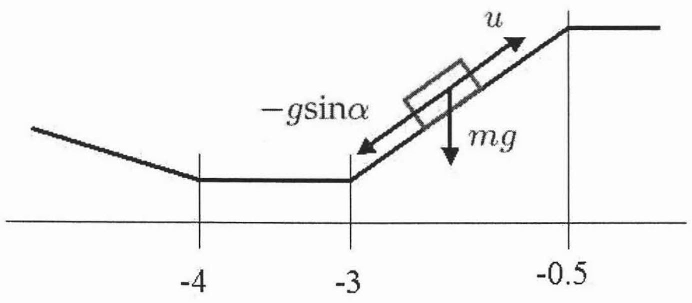


Figure 24.1: Car moving on a PWA hill.

$$\frac{\mathrm{d}p}{\mathrm{d}t} = v \tag{24.1}$$

$$m\frac{\mathrm{d}v}{\mathrm{d}t} = u - mg\sin\alpha \tag{24.2}$$

where $p$ denotes horizontal position and $v$ stands for velocity of the object. If we now define $x = [p, \ \ v]^T$, assume the mass $m = 1$ and discretize the system with sampling time of 0.1 seconds, we obtain the following affine system:

$$x(k+1) = \begin{bmatrix} 1 & 0.1 \\ 0 & 1 \end{bmatrix} x(k) + \begin{bmatrix} 0.005 \\ 0.1 \end{bmatrix} u(k) + \begin{bmatrix} c \\ -g\sin\alpha \end{bmatrix} \tag{24.3}$$

It can be seen that speed of the car depends only on the force applied to the car (manipulated variable $u$) and slope of the road $\alpha$. Slope is different in different sectors of the road. In particular we have:

$$\begin{array}{lcl} \text{Sector 1:} & p \geq -0.5 & \Rightarrow \alpha = 0^o \\ \text{Sector 2:} & -3 \leq p \leq -0.5 & \Rightarrow \alpha = 10^o \\ \text{Sector 3:} & -4 \leq p \leq -3 & \Rightarrow \alpha = 0^o \\ \text{Sector 4:} & p \leq -4 & \Rightarrow \alpha = -5^o \end{array} \tag{24.4}$$

In addition, there are constraints defined on position of the car ($-40 \leq p \leq 40$) and speed of the car ($-40 \leq v \leq 40$). Input force is restricted to satisfy $-5 \leq u \leq 5$.

## 24.2 Control Objectives

The control objective is to drive the car from a given initial conditions ($p = -3.5$, $v = 0$) to the origin ($p = 0$, $v = 0$), while minimizing the fuel consumption. To take both these objectives into account, following performance index was specified:

$$J_N^*(x(0)) = \min_{u_0,\dots,u_{N-1}} \sum_{k=0}^{N-1} ||Ru_k||_1 \tag{24.5}$$

$$\text{subj. to} \quad x_N = 0 \tag{24.6}$$

and the predictions $x_k$ are made based on equation 24.3 by taking state and input constraints into account. The penalty on fuel consumption has been chosen as $R = 0.1$.

## 24.3 matrixHYSDEL model

```
SYSTEM pwa_car {
  INTERFACE {
    STATE {
      REAL x(2) [-40, 40];
    }
    INPUT {
      REAL u(N) [-5, 5];
    }
    PARAMETER {
```

```
    REAL cf1 = [0.01541; 0.854998];         /* slope 1 */
    REAL cf2 = [-0.02568; -1.703489];       /* slope 2 */
    REAL A = [1, 0.1; 0, 1];
    REAL B = [0.005; 0.1];
    REAL N = 30;                            /* prediction horizon */
  }
}
IMPLEMENTATION {
  AUX {
    REAL f1(2, N), f2(2, N), z(2, N+1);
    BOOL d1(N), d2(N), d3(N);
    INDEX k;
  }
  LINEAR {
    z(1:2, 1) = x;
  }
  AD {
    d1(k) = z(1, k) <= -0.1     << k = 1:N >>;
    d2(k) = z(1, k) <= -3       << k = 1:N >>;
    d3(k) = z(1, k) <= -4       << k = 1:N >>;
  }
  DA {
    f1(1:2, k) = { IF d3(k) THEN cf1 }             << k = 1:N >>;
    f2(1:2, k) = { IF (~d2(k) & d1(k)) THEN cf2 } << k = 1:N >>;
  }
  LINEAR {
    z(1:2, k+1) = A*z(1:2, k) + B*u(k) +
        f1(1:2, k) + f2(1:2, k)                    << k = 1:N >>;
  }
  CONTINUOUS {
    x = z(1:2, N+1);
  }
  MUST {
    /* state constraints */
    f1 <= 40;
```

```
      f1 >= -40;
      f2 <= 40;
      f2 >= -40;
      z <= 40;
      z >= -40;
    }
  }
}
```

## 24.4 AMPL model

```
param T integer >=1;    # horizon length
param nx integer >=1;   # number of states
param nu integer >=1;   # number of inputs
param M >=0;            # big number for the logical
                        # constraints transformation
param m <= 0;           # small number
param epsilon >=0;      # small tolerance


set TIME ordered := 0..T ;
set INPUT ordered := 1..nu ;
set STATE ordered := 1..nx ;

param A{STATE,STATE};         #system matrix
param B{STATE,INPUT};         #system matrix
param State0{STATE};          #initial state
param StateF{STATE};          #final state
param maxstate{STATE} >=0;    #maximum allowed velocity
param minstate{STATE} <=0;    #maximum negative allowed velocity
param maxinp{INPUT} >=0;      #maximum allowed input
param mininp{INPUT} <=0;      # minimum allowed input
param cf1{STATE};
param cf2{STATE};
```

```
var Input {t in TIME,u in INPUT: t < last(TIME)} >= mininp[u],<=maxinp[u];
var State {t in TIME,s in STATE} >=minstate[s],<=maxstate[s];
var d1 {t in TIME: t < last(TIME)} binary;
var d2 {t in TIME: t < last(TIME)} binary;
var d3 {t in TIME: t < last(TIME)} binary;
var mode {t in TIME: t < last(TIME)} binary;
var f1 {t in TIME,s in STATE: t < last(TIME)} <=maxstate[s], >=minstate[s];
var f2 {t in TIME,s in STATE: t < last(TIME)} >=minstate[s],<=maxstate[s];

minimize energy:
   sum {t in TIME, i in INPUT: t < last(TIME)} 0.1*Input[t,i]*Input[t,i];

subject to initial {s in STATE}:
    State[0,s]=State0[s];

subject to final {s in STATE}:
    State[last(TIME),s]=StateF[s];

subject to system {t in TIME,s in STATE: t < last(TIME)}:
    State[t+1,s]= sum{pr in STATE} A[s,pr]*State[t,pr]
                + sum{i in INPUT} B[s,i]*Input[t,i]
                    + f1[t, s] + f2[t, s];

#====================================================
subject to guard1a {t in TIME: t < last(TIME)}:
    # d1(t) = State(t, 1) >= -0.1
    -0.1 -State[t,1] <= M*(1-d1[t]);

subject to guard1b {t in TIME: t < last(TIME)}:
    # d1(t) = State(t, 1) >= -0.1
    -0.1 -State[t,1] >= epsilon + (m-epsilon)*d1[t];
#====================================================
#==================================================
subject to guard2a {t in TIME: t < last(TIME)}:
    # d2(t) = State(t, 1) >= -3
    -3 -State[t,1] <= M*(1-d2[t]);

subject to guard2b {t in TIME: t < last(TIME)}:
    # d2(t) = State(t, 1) >= -3
    -3 -State[t,1] >= epsilon + (m-epsilon)*d2[t];
#==================================================


#==================================================
subject to guard3a {t in TIME: t < last(TIME)}:
    # d3(t) = State(t, 1) >= -4
    -4 -State[t,1] <= M*(1-d3[t]);

subject to guard3b {t in TIME: t < last(TIME)}:
    # d3(t) = State(t, 1) >= -4
    -4 -State[t,1] >= epsilon + (m-epsilon)*d3[t];
#==================================================


#==================================================
subject to mode_a {t in TIME: t < last(TIME)}:
    # mode = ~d1[t] & d2[t]
    -(1-d1[t]) + mode[t] <= 0;

subject to mode_b {t in TIME: t < last(TIME)}:
    # mode = ~d1[t] & d2[t]
    -d2[t] + mode[t] <= 0;

subject to mode_c {t in TIME: t < last(TIME)}:
    # mode = ~d1[t] & d2[t]
    (1-d1[t]) + d2[t] - mode[t] <= 1;
#==================================================
#====================================================
subject to slope1_a {t in TIME, s in STATE: t < last(TIME)}:
    # ~d3 -> f1 = cf1
    f1[t, s] <= M * (1-d3[t]);

subject to slope1_b {t in TIME, s in STATE: t < last(TIME)}:
    # ~d3 -> f1 = cf1
    f1[t, s] >= m * (1-d3[t]);

subject to slope1_c {t in TIME, s in STATE: t < last(TIME)}:
    # ~d3 -> f1 = cf1
    f1[t, s] <= cf1[s] - m*d3[t];

subject to slope1_d {t in TIME, s in STATE: t < last(TIME)}:
    # ~d3 -> f1 = cf1
    f1[t, s] >= cf1[s] - M*d3[t];
#====================================================



#====================================================
subject to slope2_a {t in TIME, s in STATE: t < last(TIME)}:
    # mode -> f2 = cf2
    f2[t, s] <= M * mode[t];

subject to slope2_b {t in TIME, s in STATE: t < last(TIME)}:
    # mode -> f2 = cf2
    f2[t, s] >= m * mode[t];

subject to slope2_c {t in TIME, s in STATE: t < last(TIME)}:
    # mode -> f2 = cf2
    f2[t, s] <= cf2[s] - m*(1 - mode[t]);
```

```
subject to slope2_d {t in TIME, s in STATE: t < last(TIME)}:
    # mode -> f2 = cf2
    f2[t, s] >= cf2[s] - M*(1 - mode[t]);
#===================================================
```

# 25

# The Three Tanks System

## 25.1 System dynamics

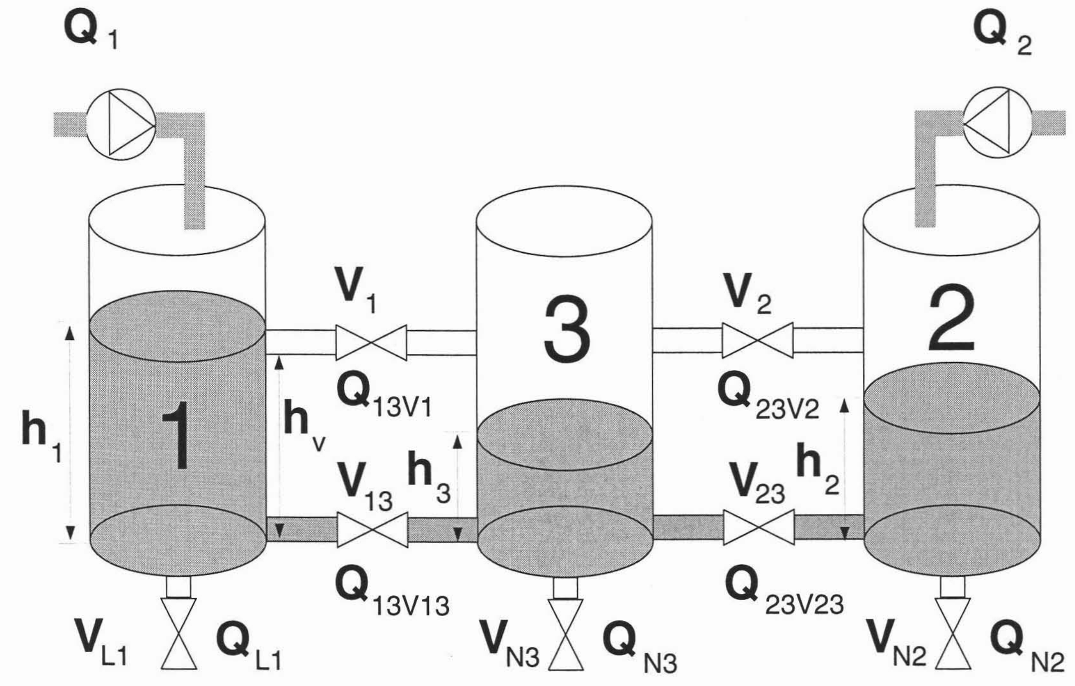


Figure 25.1: COSY Three Tank Benchmark System

The three tank system in Figure 25.1, has been proposed as benchmark system for fault detection and reconfigurable control. This benchmark system has been developed within the COSY (Control of Complex Systems) project of the European Science Foundation.

The system consists of three liquid tanks, that can be filled with two independent pumps acting on the outer tanks 1 and 2. The pumps deliver the liquid flows $Q_1$ and $Q_2$ and they can be continuously manipulated from a flow of 0 to a maximum flow $Q_{max}$. The tanks are interconnected to each other through upper and lower pipes. The flow through these pipes can be interrupted with switching valves $V_1, V_2, V_{13}, V_{23}$, that can assume either the completely open or the completely closed position. The liquid levels $h_1, h_2, h_3$ in each tank can be measured with continuous valued level sensors. The nominal outflow from the system is located at the middle tank. Physical parameters of the plant can be found in [MM01].

## 25.2 Control Objectives

The control objective is to steer heights of liquid levels in each tank to a desired reference value by manipulating the liquid flows $Q_1$ and $Q_2$ and by opening or closing of valves $V_1, V_2, V_{13}, V_{23}$. To achieve this purpose, following optimization problem can be formulated and solved:

$$\begin{aligned} J_N^*(x(0)) &= \min_{u_0,\dots,u_{N-1}} \sum_{k=0}^{N-1} ||Ru_k||_1 && (25.1) \\ \text{subj. to} \quad & x_N = r && (25.2) \end{aligned}$$

and by taking system dynamics and constraints into account. The value $r$ specifies a reference value which should be reached by the final predicted state.

## 25.3 matrixHYSDEL model

```
SYSTEM tank03nf {
  INTERFACE {
    STATE {
      REAL h1 [0, 0.62], h2 [0, 0.62], h3 [0, 0.62];
    }
    INPUT {
      REAL Q(2,N) [0, 1];
      BOOL V(2,N);
    }
  PARAMETER {
    REAL N = 7;
    REAL Area = 0.0143; /* Cross-Area of tank */
    REAL g = 9.81; /* Gravity Constant */
    REAL s13 = 10.9e-6;
    REAL s23 = 8.89e-6;
    REAL s2 = 5.54e-6;
    REAL s1 = 9.36e-6; /* Cross Section of valves */
    REAL dT = 10; /* sampling time */
    REAL hv = 0.3; /* m */
    REAL hmax = 0.62; /* m */
    REAL k1 = 7.3291e-5, k2 = 4.3379e-5;
    REAL k13 = 6.1317e-5, k23 = 5.001e-5;
    REAL TdA = 699.3;
  }
} /* end interface */
IMPLEMENTATION {
AUX {
  REAL xx(N+1,3) [0, 0.62];
  REAL z0(N,3) [-0.32,0.32], z1(N) [-0.32, 0.32];
  REAL z2(N) [-0.32, 0.32], z13(N), z23(N);
  BOOL d0(N,3);
  INDEX k,s;
}
LINEAR {
  /* initial state */
  xx(1,1) = h1;
  xx(1,2) = h2;
  xx(1,3) = h3;
  xx(k+1,1) = xx(k,1)+TdA*(Q(1,k)/1e4-k1*z1(k)-k13*z13(k))
      <<k=1:N>>;
  xx(k+1,2) = xx(k,2)+TdA*(Q(2,k)/1e4-k2*z2(k)-k23*z23(k))
      <<k=1:N>>;
  xx(k+1,3) = xx(k,3)+TdA*(k1*z1(k)+k2*z2(k)+k13*z13(k)+k23*z23(k))
      <<k=1:N>>;
  }
  AD {
    d0(k,s) = -xx(k,s) + hv <= 0.0 <<k=1:N,s=1:3>>;
  }
  DA {
    z0(k,s) = {IF d0(k,s) THEN (xx(k,s) - hv) }   <<k=1:N,s=1:3>>;
    z1(k)   = {IF V(1,k) THEN z0(k,1) - z0(k,3) } <<k=1:N>>;
    z2(k)   = {IF V(2,k) THEN z0(k,2) - z0(k,3) } <<k=1:N>>;
    z13(k)  = {IF V(1,k) THEN xx(k,1) - xx(k,3) } <<k=1:N>>;
    z23(k)  = {IF V(2,k) THEN xx(k,2) - xx(k,3) } <<k=1:N>>;
  }
  CONTINUOUS {
    h1 = xx(N+1,1);
    h2 = xx(N+1,2);
    h3 = xx(N+1,3);
  }
  MUST {
    z0 <= 0.62;
    z0 >= 0;
    z1 <= 0.32;
    z1 >= -0.32;
    z2 <= 0.32;
    z2 >= -0.32;
    z13 <= 0.62;
    z13 >= -0.62;
    z23 <= 0.62;
    z23 >= -0.62;
    xx <= 0.62;
    xx >= 0;
  }
 } /* end implementation */
} /* end system */
```

## 25.4 AMPL model

```
param T integer >=1;    #horizon length
param nx := 3;
param nur := 2;
param nub := 2;
param M >=0;            #big number for the logical constraints transformation
param m <= 0;            # small number
param epsilon >=0;  # small tolerance

param hmax := 0.62;
param Q1max := 1;
param Q2max := 1;
param Area := 0.0143;
param g := 9.81;
param s13 := 10.9e-6;
param s23 := 8.89e-6;
param s2 := 5.54e-6;
param s1 := 9.36e-6;
param dT := 10;
param hv := 0.3;
param TdA = dT / Area;
param k1 := s1 * sqrt(2 * g / (hmax - hv));
param k2 := s2 * sqrt(2 * g / (hmax - hv));
param k13 := s13 * sqrt(2 * g / hmax);
param k23 := s23 * sqrt(2 * g / hmax);

set TIME ordered := 0..T ;
set INPUTREAL ordered := 1..nur;
set INPUTBOOL ordered := 1..nub;
set STATE ordered := 1..nx ;

param State0{STATE};         #initial state
param StateF{STATE};         #final state
param maxstate{STATE} >=0;  #maximum allowed velocity
param minstate{STATE} <=0;  #maximum negative allowed velocity
param maxinp{INPUTREAL} >=0;          #maximum allowed input
param mininp{INPUTREAL} <=0;          # minimum allowed input

var InputReal {t in TIME,u in INPUTREAL: t < last(TIME)} >= mininp[u],<=maxinp[u]
var InputBool {t in TIME, u in INPUTBOOL: t < last(TIME)} binary;
var State {t in TIME,s in STATE} >=minstate[s],<=maxstate[s];

var d {t in TIME, s in STATE: t < last(TIME)} binary;
var z0 {t in TIME, s in STATE: t < last(TIME)} <= 1, >= -1;
var z1 {t in TIME: t < last(TIME)} <= 1, >= -1;
var z2 {t in TIME: t < last(TIME)} <= 1, >= -1;
var z13 {t in TIME: t < last(TIME)} <= 1, >= -1;
var z23 {t in TIME: t < last(TIME)} <= 1, >= -1;
var EpsReal {t in TIME, u in INPUTREAL: t < last(TIME)} >=mininp[u], <= maxinp[u]

minimize energy:
   sum {t in TIME, u in INPUTREAL: t < last(TIME)} EpsReal[t, u] +
   sum {t in TIME, ub in INPUTBOOL: t < last(TIME)} InputBool[t, ub];


subject to slacks_real1 {t in TIME, i in INPUTREAL: t < last(TIME)}:
    -EpsReal[t,i] <= InputReal[t,i];

subject to slacks_real2 {t in TIME, i in INPUTREAL: t < last(TIME)}:
    -EpsReal[t,i] <= -InputReal[t,i];

subject to initial {s in STATE}:
    State[0,s]=State0[s];

subject to final {s in STATE}:
    State[last(TIME),s]=StateF[s];

subject to system_a {t in TIME: t < last(TIME)}:
    State[t+1, 1] = State[t, 1] + TdA *
```

```
        (-k1 * z1[t] - k13 * z13[t] + InputReal[t, 1] / 1e4);

subject to system_b {t in TIME: t < last(TIME)}:
    State[t+1, 2] = State[t, 2] + TdA *
        (-k2 * z2[t] - k23 * z23[t] + InputReal[t, 2] / 1e4);

subject to system_c {t in TIME: t < last(TIME)}:
    State[t+1, 3] = State[t, 3] + TdA *
        (k1 * z1[t] + k2 * z2[t] + k13 * z13[t] + k23 * z23[t]);


################ AD section ######################
#==================================================
subject to guard_a {t in TIME, s in STATE: t < last(TIME)}:
    # d_i(t) = -State(t, i) + hv <= 0
    -State[t,s] + hv <= M*(1-d[t, s]);

subject to guard_b {t in TIME, s in STATE: t < last(TIME)}:
    # d_i(t) = -State(t, i) + hv <= 0
    -State[t,s] + hv >= epsilon + (m-epsilon)*d[t,s];
#==================================================



################ DA section ######################
#==================================================
subject to z0_a {t in TIME, s in STATE: t < last(TIME)}:
    # d_i -> z0_i = h_i - hv
    z0[t,s] <= M * d[t,s];

subject to z0_b {t in TIME, s in STATE: t < last(TIME)}:
    # d_i -> z0_i = h_i - hv
    z0[t,s] >= m * d[t,s];

subject to z0_c {t in TIME, s in STATE: t < last(TIME)}:
    # d_i -> z0_i = h_i - hv
    z0[t,s] <= (State[t,s] - hv) - m*(1-d[t,s]);

subject to z0_d {t in TIME, s in STATE: t < last(TIME)}:
    # d_i -> z0_i = h_i - hv
    z0[t,s] >= (State[t,s] - hv) - M*(1 - d[t,s]);
#==================================================



#==================================================
# z1 = {IF V1 THEN z01 - z03};
subject to z1_a {t in TIME: t < last(TIME)}:
    # V1 -> z1 = z01 - z03
    z1[t] <= M * InputBool[t,1];

subject to z1_b {t in TIME: t < last(TIME)}:
    # V1 -> z1 = z01 - z03
    z1[t] >= m * InputBool[t,1];

subject to z1_c {t in TIME: t < last(TIME)}:
    # V1 -> z1 = z01 - z03
    z1[t] <= (z0[t, 1] - z0[t, 3]) - m*(1-InputBool[t,1]);

subject to z1_d {t in TIME: t < last(TIME)}:
    # V1 -> z1 = z01 - z03
    z1[t] >= (z0[t, 1] - z0[t, 3]) - M*(1 - InputBool[t,1]);
#==================================================



#==================================================
# z2 = {IF V2 THEN z02 - z03};
subject to z2_a {t in TIME: t < last(TIME)}:
    # V2 -> z2 = z02 - z03
    z2[t] <= M * InputBool[t,2];
```

```
subject to z2_b {t in TIME: t < last(TIME)}:
    # V2 -> z2 = z02 - z03
    z2[t] >= m * InputBool[t,2];

subject to z2_c {t in TIME: t < last(TIME)}:
    # V2 -> z2 = z02 - z03
    z2[t] <= (z0[t, 2] - z0[t, 3]) - m*(1-InputBool[t,2]);

subject to z2_d {t in TIME: t < last(TIME)}:
    # V2 -> z2 = z02 - z03
    z2[t] >= (z0[t, 2] - z0[t, 3]) - M*(1 - InputBool[t,2]);
#==================================================


#==================================================
# z13 = {IF V1 THEN h1 - h3 };
subject to z13_a {t in TIME: t < last(TIME)}:
    # V1 -> z13 = h1 - h3
    z13[t] <= M * InputBool[t,1];

subject to z13_b {t in TIME: t < last(TIME)}:
    # V1 -> z13 = h1 - h3
    z13[t] >= m * InputBool[t,1];

subject to z13_c {t in TIME: t < last(TIME)}:
    # V1 -> z13 = h1 - h3
    z13[t] <= (State[t, 1] - State[t, 3]) - m*(1-InputBool[t,1]);

subject to z13_d {t in TIME: t < last(TIME)}:
    # V1 -> z13 = h1 - h3
    z13[t] >= (State[t, 1] - State[t, 3]) - M*(1 - InputBool[t,1]);
#==================================================


#==================================================
# z23 = {IF V2 THEN h2 - h3 };
subject to z23_a {t in TIME: t < last(TIME)}:
    # V2 -> z23 = h2 - h3
    z23[t] <= M * InputBool[t,2];

subject to z23_b {t in TIME: t < last(TIME)}:
    # V2 -> z23 = h2 - h3
    z23[t] >= m * InputBool[t,2];

subject to z23_c {t in TIME: t < last(TIME)}:
    # V2 -> z23 = h2 - h3
    z23[t] <= (State[t, 2] - State[t, 3]) - m*(1-InputBool[t,2]);

subject to z23_d {t in TIME: t < last(TIME)}:
    # V2 -> z23 = h2 - h3
    z23[t] >= (State[t, 2] - State[t, 3]) - M*(1 - InputBool[t,2]);
#==================================================
```

# 26

# Curriculum Vitae

Michal Kvasnica

Born on June 20, 1977 in Povazska Bystrica, Slovak Republic

| | |
|---|---|
| 09/2002 – 02/2008 | Doctorate at Automatic Control Laboratory, ETH Zürich (Dr. sc. ETH Zürich) |
| 07/2000 – 09/2001 | Researcher at Institute of Automation, Slovak University of Technology in Bratislava, Slovakia |
| 09/1995 – 06/2000 | Graduate studies, Faculty of Chemical and Foot Technology Slovak University of Technology in Bratislava, Slovakia (Ing.) |
| 09/1991 – 06/1995 | High School Puchov, Slovakia |

# 27

# Publication List

Here, all my publications and technical reports are listed in chronological order:

- *Low Complexity Control of Piecewise Affine Systems with Stability Guarantee*; P. Grieder, M. Kvasnica, M. Baotic, and M. Morari; Technical Report AUT03-13, 2003; Automatic Control Lab, ETH, Switzerland. [GKBM03].

- *Multi-Parametric Toolbox (MPT)*; M. Kvasnica, P. Grieder, M. Baotic, and M. Morari; Technical Report AUT03-17; Automatic Control Lab, ETH, Switzerland, 2003. [KGBM03b].

- *Multi-Parametric Toolbox (MPT)*; M. Kvasnica, P. Grieder, M. Baotic, and M. Morari; HSCC (Hybrid Systems: Computation and Control ), vol. 2993, Lecture Notes in Computer Science, 2004. [KGBM03a].

- *Low Complexity Control of Piecewise Affine Systems with Stability Guarantee*; P. Grieder, M. Kvasnica, M. Baotic, and M. Morari; American Control Conference, Boston, Massachusetts, 2004. [GKBM04].

- *Computation of Invariant Sets for Piecewise Affine Discrete Time Systems subject to Bounded Disturbances*; S. Rakovic, P. Grieder, M. Kvasnica, D. Mayne, and M. Morari; IEEE Conference on Decision and Control, Bahamas, 2004. [RGK$^+$04].

- *Efficient Computation of Controller Partitions in Multi-Parametric Programming*; R. Suard, J. Lfberg, P. Grieder, M. Kvasnica, and M. Morari; IEEE Conference on Decision and Control, Bahamas, 2004. [SLG$^+$04].

- *Multi-Parametric Toolbox (MPT): User's Manual*; M. Kvasnica, P. Grieder, M. Baotic, and F.J. Christophersen; Technical Report; Automatic Control Lab, ETH, Switzerland, 2004. [KGBC06].

- *Optimal Infusion of intravenous morphine and ketamine - A Mixed-Integer Linear Programming application*; V. Sartori, E. Zanderigo, M. Kvasnica, and M. Morari; IEEE Engineering in Medicine and Biology Society, Shanghai, China, 2005. [SZKM05].
- *Stabilizing low complexity feedback control of constrained piecewise affine systems*; P. Grieder, M. Kvasnica, M. Baotic, and M. Morari; Automatica, vol. 41, issue 10, 2005. [GKBM05].
- *Efficient Evaluation of Piecewise Control Laws defined over a Large Number of Polyhedra*; F.J. Christophersen, M. Kvasnica, C.N. Jones, and M. Morari; European Control Conference, Kos, Greece, 2007. [CKJM07].

# Bibliography

[ABQ+99] Allgöwer, G., T.A. Badgwell, S.J. Qin, J.B. Rawlings and S.J. Wright: *Nonlinear Predictive Control and Moving Horizon Estimation - An Introduction Overview.* Advance in Control: Highlights of ECC'99, pages 391–449, 1999.

[AC01] Alessandri, A. and P. Colleta: *Design of Luenberger observers for a class of hybrid systems.* In *Proc. of Hybrid Systems and Control*, Rome, Italy, 2001.

[Bal98] Balas, E.: *Projection with a minimum system of inequalities.* Computational Optimization and Applications, 10:189–193, 1998.

[Bao02] Baotić, M.: *An Efficient Algorithm for Multi–Parametric Quadratic Programming.* Technical Report AUT02–05, Automatic Control Laboratory, ETH Zurich, Switzerland, April 2002. Available from http://control.ee.ethz.ch/research/publications/publications.msql?

[BBBM01] Borrelli, F., M. Baotić, A. Bemporad and M. Morari: *Efficient On-Line Computation of Constrained Optimal Control.* In *Proc. of the Conf. on Decision & Control*, Orlando, Florida, USA, December 2001.

[BBBM03] Borrelli, F., M. Baotić, A. Bemporad and M. Morari: *An Efficient Algorithm for Computing the State Feedback Optimal Control Law for Discrete Time Hybrid Systems.* In *Proc. 2003 American Control Conference*, Denver, Colorado, USA, June 2003.

[BBBM05] Borrelli, F., M. Baotic, A. Bemporad and M. Morari: *Dynamic programming for constrained optimal control of discrete-time linear hybrid systems.* Automatica, 41:1709–1721, October 2005.

[BBM98] BRANICKY, M.S., V.S. BORKAR and S.K. MITTER: *A unified framework for hybrid control: model and optimal control theory.* IEEE Trans. Automatic Control, 43(1):31–45, 1998.

[BBM00a] BEMPORAD, A., F. BORRELLI and M. MORARI: *Explicit Solution of LP-Based Model Predictive Control.* In *Proc. 39th IEEE Conf. on Decision and Control*, Sydney, Australia, December 2000.

[BBM00b] BEMPORAD, A., F. BORRELLI and M. MORARI: *Optimal Controllers for Hybrid Systems: Stability and Piecewise Linear Explicit Form.* In *Proc. 39th IEEE Conf. on Decision and Control*, Sydney, Australia, December 2000.

[BCM03a] BAOTIĆ, M., F. J. CHRISTOPHERSEN and M. MORARI: *Infinite Time Optimal Control of Hybrid Systems with a Linear Performance Index.* In *Proc. of the Conf. on Decision and Control, Maui, Hawaii, USA*, December 2003.

[BCM03b] BAOTIĆ, M., F.J. CHRISTOPHERSEN and M. MORARI: *A new Algorithm for Constrained Finite Time Optimal Control of Hybrid Systems with a Linear Performance Index.* In *European Control Conference*, Cambridge, UK, September 2003.

[Bem03] BEMPORAD, A.: *Hybrid Toolbox - User's Guide*, 2003.

[BFT01] BEMPORAD, A., K. FUKUDA and F.D. TORRISI: *Convexity Recognition of the Union of Polyhedra.* Computational Geometry, 18:141–154, April 2001.

[BFT04] BEMPORAD, A., C. FILIPPI and F. D. TORRISI: *Inner and outer approximation of polytopes using boxes.* Computational Geometry: Theory and Applications, 27(2):151–178, 2004.

[BGFB94] BOYD, S., L. EL GHAOUI, E. FERON and V. BALAKRISHNAN: *Linear Matrix Inequalities in System and Control Theory.* Studies in Applied Mathematics. SIAM, 1994.

[Bla99] BLANCHINI, F.: *Set invariance in control — A survey.* Automatica, 35(11):1747–1767, November 1999.

[BM99a] Bemporad, A. and M. Morari: *Control of Systems Integrating Logic, Dynamics, and Constraints.* Automatica, 35(3):407–427, March 1999.

[BM99b] Bemporad, A. and M. Morari: *Robust Model Predictive Control: A Survey.* In Garulli, A., A. Tesi and A. Vicino (editors): *Robustness in Identification and Control*, number 245 in *Lecture Notes in Control and Information Sciences*, pages 207–226. Springer-Verlag, 1999.

[BMDP02] Bemporad, A., M. Morari, V. Dua and E.N. Pistikopoulos: *The Explicit Linear Quadratic Regulator for Constrained Systems.* Automatica, 38(1):3–20, January 2002.

[Bon83] Boneh, A.: *Redundancy in Mathematical Programming*, volume 206 of *Lecture Notes in Economics and Mathematical Systems*, chapter PREDUCE - A Probabilistic Algorithm Identifying Redundancy by a Random Feasible Point Generator (RFPG). Springer-Verlag, 1983.

[Bor03] Borrelli, F.: *Constrained Optimal Control Of Linear And Hybrid Systems*, volume 290 of *Lecture Notes in Control and Information Sciences.* Springer, 2003.

[BT03] Baotić, M. and F.D. Torrisi: *Polycover.* Technical Report AUT03-11, Automatic Control Lab, ETHZ, Switzerland, 2003.

[BV04] Boyd, S. and L. Vandenberghe: *Convex Optimization.* Cambridge University Press, 2004. `http://www.stanford.edu/class/ee364/`.

[BZ00] Branicky, M.S. and G. Zhang: *Solving Hybrid Control Problems: Level Sets and Behavioral Programming.* In *Proc. American Contr. Conf.*, Chicago, Illinois USA, June 2000.

[CB99] Camacho, E.F. and C. Bordon: *Model Predictive Control.* Advanced Textbooks in Control and Signal Processing. Springer, London, 1999.

[Cer63] Cernikov, S.N.: *Contraction of finite systems of linear inequalities (in russian).* Doklady Akademiia Nauk SSSR, 152(5):1075–1078, 1963. (English translation in Societ Mathematics Doklady, Vol. 4, No. 5 (1963), pp.1520–1524).

[CKJM07] Christophersen, F. J., M. Kvasnica, C. N. Jones and M. Morari: *Efficient Evaluation of Piecewise Control Laws defined over a Large Number of Polyhedra.* In *Proc. of the European Control Conference*, Kos, Greece, July 2007. Submitted.

[CLRS01] Cormen, T. H., C. E. Leiserson, R. L. Rivest and C. Stein: *Introduction to Algorithms.* MIT Press and McGraw-Hill, 2nd edition, 2001.

[CMT87] Clarke, D. W., C. Mohtadi and P. S. Tuffs: *Generalized Predictive Control–Part I. The Basic Algorithm.* Automatica, 23(2):137–148, 1987.

[CPS90] Cavalier, T.M., P.M. Pardalos and A.L. Soyster: *Modeling and integer programming techniques applied to propositional calculus.* Computers Opns Res., 17(6):561–570, 1990.

[dH94] Hertog, D. den: *Interior Point Approach to Linear, Quadratic and Convex Programming: Algorithms and Complexity.* Mathematics and Its Applications. Kluwer Academic Publishers, 1994.

[DPar] Dua, V. and E.N. Pistikopoulos: *An algorithm for the solution of multiparametric mixed integer linear programming problems.* Annals of Operations Research, to appear.

[dSvO00] de Berg, M., O. Schwarzkopf, M. van Kreveld and M. Overmars: *Computational Geometry: Algorithms and Applications.* Springer Verlag, 2nd edition, 2000.

[FGK93] Fourer, Robert, David M. Gay and Brian W. Kernighan: *AMPL: A Modeling Language for Mathematical Programming.* The Scientific Press (now an imprint of Boyd & Fraser Publishing Co.), Danvers, MA, USA, 1993.

[FLL00] Fukuda, K., T. M. Liebling and C. Lütolf: *Extended convex hull.* In *12th Canadian Conference on Computational Geometry*, page 5764, July 2000.

[FP96] FUKUDA, K. and A. PRODON: *Double description method revisited.* Combinatorics and Computer Science, 1120:91111, 1996. ftp://ftp.ifor.math.ethz.ch/pub/fukuda/reports/ddrev960315.ps.gz.

[FTCMM02] FERRARI-TRECATE, G., F. A. CUZZOLA, D. MIGNONE and M. MORARI: *Analysis of discrete-time piecewise affine and hybrid systems.* Automatica, 38:2139–2146, 2002.

[Fuk97] FUKUDA, K.: *Cdd/cdd+ Reference Manual*, December 1997. www.cs.mcgill.ca/~fukuda/soft/cdd_home/cdd.html.

[Fuk00] FUKUDA, K.: *Polyhedral computation FAQ*, 2000. On line document. Both html and ps versions available from http://www.ifor.math.ethz.ch/staff/fukuda.

[Gal95] GAL, T.: *Postoptimal Analyses, Parametric Programming, and Related Topics.* de Gruyter, Berlin, 2nd ed. edition, 1995.

[GBTM04] GRIEDER, P., F. BORRELLI, F. D. TORRISI and M. MORARI: *Computation of the constrained infnite time linear quadratic regulator.* Automatica, 40:701–708, 2004.

[Gey05] GEYER, T.: *Low Complexity Model Predictive Control in Power Electronics and Power Systems.* Dr. sc. thesis, ETH Zurich, Zurich, Switzerland, March 2005. Available from http://control.ee.ethz.ch/index.cgi?page=publications;action=details;id=2124.

[GKBM03] GRIEDER, P., M. KVASNICA, M. BAOTIĆ and M. MORARI: *Low Complexity Control of Piecewise Affine Systems with Stability Guarantee.* Technical Report AUT03-13, Automatic Control Lab, ETHZ, Switzerland, 2003. Available from http://control.ee.ethz.ch/research/publications/publications.msql?

[GKBM04] GRIEDER, P., M. KVASNICA, M. BAOTIĆ and M. MORARI: *Low Complexity Control of Piecewise Affine Systems with Stability Guarantee.* In *Proc. of the American Control Conference*, pages 1196–1201, Boston, USA, June 2004. Available from http://control.ee.ethz.ch/research/publications/publications.msql?

[GKBM05] GRIEDER, P., M. KVASNICA, M. BAOTIC and M. MORARI: *Stabilizing low complexity feedback control of constrained piecewise affine systems.* Automatica, 41, issue 10:1683–1694, October 2005.

[GLPM03] GRIEDER, P., M. LÜTHI, P. PARILLO and M. MORARI: *Stability & Feasibility of Receding Horizon Control.* In *European Control Conference*, Cambridge, UK, September 2003.

[GM03] GRIEDER, P. and M. MORARI: *Complexity Reduction of Receding Horizon Control.* In *Proc. 42th IEEE Conf. on Decision and Control*, Maui, Hawaii, USA, December 2003.

[GNAE93] GROSSMANN, R., A. NERODE, A.P.RAVN and H. RISCHEL (EDS.): *Hybrid Systems.* Springer Verlag, New York, 1993. no. 736 in LCNS.

[Gon97] GONDZIO, J.: *Presolve Analysis of Linear Programs Prior to Applying an Interior Point Method.* OSRA Journal on Computing, 9(1):73–91, 1997.

[GPM89] GARCIA, C.E., D.M. PRETT and M. MORARI: *Model Predictive Control: Theory and Practive - A Survey.* Automatica, 25(3):335–348, 1989.

[GPM03] GRIEDER, P., P. PARILLO and M. MORARI: *Robust Receding Horizon Control - Analysis & Synthesis.* In *Proc. 42th IEEE Conf. on Decision and Control*, Maui, Hawaii, USA, December 2003.

[Gri04] GRIEDER, P.: *Efficient Computation of Feedback Controllers for Constrained Systems.* PhD thesis, Automatic Control Laboratory, ETH Zurich, 2004.

[Grü00] GRÜNBAUM, B.: *Convex Polytopes.* Springer-Verlag, Second edition, 2000.

[GT91] GILBERT, E. G. and K. T. TAN: *Linear systems with state and control constraints: the theory and applications of maximal output admissible sets.* IEEE Trans. on Automatic Control, 36(9):1008–1020, 1991.

[GTM03] GEYER, T., F. D. TORRISI and M. MORARI: *Efficient Mode Enumeration of Compositional Hybrid Models.* In *Proc. of the Intern. Workshop* *on Hybrid Systems: Computation and Control*, volume 2623 of *Lecture Notes in Computer Science*, pages 216–232. Springer-Verlag, 2003.

[GTM04] GEYER, T., F. D. TORRISI and M. MORARI: *Optimal Complexity Reduction of Piecewise Affine Models Based on Hyperplane Arrangements.* In *Proc. on the American Control Conference*, pages 1190–1195, Boston, Massachusetts, USA, June 2004.

[HDB01] HEEMELS, W. P. M., B. DE SCHUTTER and A. BEMPORAD: *Equivalence of hybrid dynamical models.* Automatica, 37(7):1085–1091, 2001.

[Hee99] HEEMELS, W. P. M. H.: *Linear Complementarity Systems: A Study in Hybrid Dynamics.* PhD thesis, Technische Universiteit Eindhoven, Eindhoven, The Netherlands, November 1999.

[Her96] HERMANN, O.: *Regelung eines ball and plate systems.* Diploma thesis, ETH Zurich, Zurich, Switzerland, 1996.

[HSB$^+$90] HILL, H. F., L. SAEGER, R. BJURSTROM, G. DONALDSON, C. R. CHAPMAN and R. JACOBSON: *Steady -State Infusions of Opioids in Human Volunteers. I. Pharmacokinetic Tailoring.* Pain, 43:57–67, 1990.

[ILO04] ILOG, INC.: *CPLEX 9.0 User Manual.* Gentilly Cedex, France, 2004.

[JGR05] JONES, C. N., P. GRIEDER and S. V. RAKOVIĆ: *A Logarithmic Solution to the Point Location Problem for Closed-Form Linear MPC.* In *IFAC World Congress*, Prague, Czech Republic, July 2005.

[JHW02] JULOSKI, A., M. HEEMELS and S. WEILAND: *Observer Design for a Class of Piece-Wise Affine Systems.* In *Proc. 41th IEEE Conf. on Decision and Control*, Las Vegas, USA, 2002.

[JKM04] JONES, C.N., E.C. KERRIGAN and J.M. MACIEJOWSKI: *Equality Set Projection: A new algorithm for the projection of polytopes in halfspace representation.* Technical Report CUED Technical Report CUED/F-INFENG/TR.463, Department of Engineering, Cambridge University, UK, 2004. `http://www-control.eng.cam.ac.uk/ cnj22/`.

[Joh03] JOHANSSON, M.: *Piecewise Linear Control Systems: A Computational Approach*, volume 284 of *Lecture Notes in Control and Information Sciences*. Springer-Verlag, 2003.

[Jon05] JONES, C. N.: *Polyhedral Tools for Control*. PhD thesis, University of Cambridge, Cambridge, U.K., July 2005.

[JR98] JOHANNSON, M. and A. RANTZER: *Computation of piece-wise quadratic Lyapunov functions for hybrid systems*. IEEE Trans. Automatic Control, 43(4):555–559, 1998.

[KA02] KOUTSOUKOS, X. D. and P. J. ANTSAKLIS: *Design of stabilizing switching control laws for discrete and continuous-time linear systems using piecewise linear Lyapunov functions*. Int. J. Control, 75(12):932–945, 2002.

[KBM96] KOTHARE, M. V., V. BALAKRISHNAN and M. MORARI: *Robust constrained model predictive control using linear matrix inequalities*. Automatica, 32(10):1361–1379, 1996.

[KC01] KOUVARITAKIS, B. and M. CANNON: *Nonlinear Predictive Control: Theory and Practice*. The Institution of Electrical Engineers, London, UK, 2001.

[Ker00] KERRIGAN, E. C.: *Robust Constraints Satisfaction: Invariant Sets and Predictive Control*. PhD thesis, Department of Engineering, The University of Cambridge, Cambridge, England, 2000.

[KG98] KOLMANOVSKY, I. and E. G. GILBERT: *Theory and Computation of Disturbance Invariant Sets for Discrete-Time Linear Systems*. Mathematical Problems in Egineering, 4:317–367, 1998.

[KGB04] KVASNICA, M., P. GRIEDER and M. BAOTIĆ: *Multi-Parametric Toolbox (MPT)*, 2004. Available from http://control.ee.ethz.ch/~mpt/.

[KGBC06] KVASNICA, M., P. GRIEDER, M. BAOTIĆ and F. J. CHRISTOPHERSEN: *Multi-Parametric Toolbox (MPT) User's Manual*. Automatic Control Laboratory, ETH Zurich, March 2006.

[KGBM03a] KVASNICA, M., P. GRIEDER, M. BAOTIĆ and M. MORARI: *Multi-Parametric Toolbox (MPT)*. In *Proc. of the Intern. Workshop on Hybrid Systems: Computation and Control*, volume 2993 of *Lecture Notes in Computer Science*, pages 448–462, Pennsylvania, Philadelphia, USA, March 2003. Springer-Verlag.

[KGBM03b] KVASNICA, M., P. GRIEDER, M. BAOTIĆ and M. MORARI: *Multi-Parametric Toolbox (MPT)*. Technical Report AUT03-17, Automatic Control Lab, ETHZ, Switzerland, 2003.

[KM02] KERRIGAN, E. C. and D. Q. MAYNE: *Optimal control of constrained, piecewise affine systems with bounded disturbances*. In *Proc. 41st IEEE Conference on Decision and Control*, Las Vegas, Nevada, USA, December 2002.

[KM03] KERRIGAN, E. C. and J. M. MACIEJOWSKI: *On robust optimization and the optimal control of constrained linear systems with bounded state disturbances*. In *In Proc. 2003 European Control Conference*, Cambridge, UK, September 2003.

[KS90] KEERTHI, S. S. and K. SRIDHARAN: *Solution of parametrized linear inequalities by fourier elimination and its applications*. J. Opt. Theory and Applications, 65(1):161–169, 1990.

[Lö03] LÖFBERG, J.: *Minimax approaches to robust model predictive control*. PhD thesis, Linköping University, Linköping, Sweden, 2003. Diss. No. 812.

[Lag01] LAGERBERG, A.: *A literature survey on control of automotive powertrains with backlash*. Technical Report R013/2001, Department of Signals and Systems, Chalmers University of Technology, Göteborg, Sweden, December 2001.

[Lag04] LAGERBERG, A.: *Control and Estimation of Automotive Powertrains with Backlash*. PhD thesis, Department of Signals and Systems, Chalmers University of Technology, Göteborg, Sweden, 2004.

[LE03a] LAGERBERG, A. and B. EGARDT: *Backlash gap position estimation in automotive powertrains.* In *European Control Conference*, Cambridge, UK, 2003.

[LE03b] LAGERBERG, A. and B. EGARDT: *Estimation of backlash with application to automotive powertrains.* In *Proceedings of the 42nd IEEE Conference on Decision and Control*, Maui, Hawaii USA, 2003.

[LE05] LAGERBERG, A. and B. EGARDT: *Model Predictive Control of Automotive powertrains with backlash.* In *16th IFAC World Congress, Prague*, Prague, Czech Republic, July 2005.

[LHWB04] LAZAR, M., W.P.M.H. HEEMELS, S. WEILAND and A. BEMPORAD: *Stabilization conditions for model predictive control of constrained PWA systems.* In *Proc. 43th IEEE Conf. on Decision and Control*, Bahamas, December 2004.

[Lju06] LJUNG, L.: *System Identification Toolbox, User's Guide, Version 6.* The MathWorks, Inc., 2006.

[Löf03] LÖFBERG, J.: *Minimax approaches to robust model predictive control.* PhD thesis, Linköping University, Sweden, April 2003.

[Löf04] LÖFBERG, J.: *YALMIP*, 2004. Available from http://control.ee.ethz.ch/~joloef/yalmip.php.

[LR03] LINCOLN, B. and A. RANTZER: *Relaxed Optimal Control of Piecewise Linear Systems.* In *IFAC Conference on Analysis and Design of Hybrid Systems*, Saint-Malo, France, 2003.

[LTS99] LYGEROS, J., C. TOMLIN and S. SASTRY: *Controllers for reachability specifications for hybrid systems.* Automatica, 35(3):349–370, 1999.

[Mac02] MACIEJOWSKI, J.M.: *Predictive Control with Constraints.* Prentice Hall, 2002.

[Mak01] MAKHORIN, A.: *GLPK - GNU Linear Programming Kit*, 2001. http://www.gnu.org/directory/libs/glpk.html.

[MATC00] MERCADANTE, S., E. ARCURI, W. TIRELLI and A. CASUCCIO: *Analgesic Effect of Intravenous Ketamine in Cancer Patients on Morphine Therapy: A Randomized, Controlled, Double-Blind, Crossover, Double-Dose Study.* Journal of Pain and symptom Management, 20:246–252, 2000.

[May01] MAYNE, D. Q.: *Control of Constrained Dynamic Systems.* European Jornal of Control, 7:87–99, 2001.

[MBB03] MORARI, M., M. BAOTIC and F. BORRELLI: *Hybrid Systems Modeling and Control.* European Journal of Control, 9(2-3):177–189, 2003.

[MFTM00] MIGNONE, D., G. FERRARI-TRECATE and M. MORARI: *Stability and stabilization of piecewise affine and hybrid systems: An LMI approach.* In *Proc. 39th IEEE Conf. on Decision and Control*, December 2000.

[Mig02] MIGNONE, D.: *Control and Estimation of Hybrid systems via Mathematical Optimization.* PhD thesis, Automatic Control Labotatory – ETH, Zurich, 2002.

[ML99] MORARI, M. and J.H. LEE: *Model predictive control: past, present and future.* Computers & Chemical Engineering, 23(4–5):667–682, 1999.

[MM01] MIGNONE, D. and N. MONACHINO: *The Total Three Tank Tutorial Text.* Technical Report, 2001.

[MPL91] MAX, M.B., R.K. PORTENOY and E. LASKA: volume 18 of *Advances in pain research and therapy.* Raven, New York, 1991.

[MR02] MAYNE, D. Q. and S. RAKOVIĆ: *Optimal control of constrained piecewise affine discrete-time systems using reverse transformation.* In *Proc. 41st IEEE Conference on Decision and Control*, Las Vegas, Nevada, USA, December 2002.

[MR03] MAYNE, D. Q. and S. RAKOVIĆ: *Model Predicitive Control of Constrained Piecewise-Affine Discrete-Time Systems.* Int. J. of Robust and Nonlinear Control, 13(3):261–279, April 2003.

[MRRS00] MAYNE, D. Q., J.B. RAWLINGS, C.V. RAO and P.O.M. SCOKAERT: *Constrained model predictive control: Stability and Optimality.* Automatica, 36(6):789–814, June 2000.

[MSS$^+$00] MINTO, C. F., T. W. SCHNIEDER, T. G. SHORT, K. M. GREGG, A. GENTILINI and S. L. SHAFER: *Response Surface Model for Anesthetic Drug Interactions.* Anesthesiology, 6:1603–1616, 2000.

[Ne97] NORDIN, M. and *et. al.*: *New models for backlash and gear play.* International journal of adaptive control and signal processing, 11:49–63, 1997.

[Neu04] NEUMAIER, A.: *Complete Search in Continuous Global Optimization and Constraint Satisfaction.* In ISERLES, A. (editor): *Acta Numerica*, Lecture Notes in Control and Information Sciences. Cambridge University Press, 2004.

[NG02] NORDIN, N. and P-O GUTMAN: *Controlling mechanical systems with backlash-a survey.* Automaitca, 38(10):1633–1649, 2002.

[NW88] NEMHAUSER, G.L. and L.A. WOLSEY: *Integer and Combinatorial Optimization.* Wiley, 1988.

[PLC$^+$87] PEDRAZ, J.L., J. M. LANAC, M. B. CALVO, C. MURIEL, J. HERNANDEZ-ARBEIZA and A. DOMINGUEZ-GIL: *Pharmacokinetic and Clinical Evaluation of Ketamine Aministered by I.V. and Epidural Routes.* International journal of clinical pharmacology, 25:77–80, 1987.

[QB97] QIN, S.J. and T.A. BADGEWELL: *An overview of industrial model predictive control technology.* In *Chemical Process Control - V*, volume 93, no. 316, pages 232–256. AIChe Symposium Series - American Institute of Chemical Engineers, 1997.

[RBBM06] ROSTALSKI, PH., TH. BESSELMANN, M. BARIC and M. MORARI: *A Hybrid Approach to Modeling, Control and State estimation of Mechanical Systems with Backlash.* Technical Report, June 2006.

[RG92] RAMAN, R. and I. E. GROSSMANN: *Integration of logic and heuristic knowledge in MINLP optimization for process synthesis.* Computers chem. Engng., 16(3):155–171, 1992.

[RGK+04] Raković, S. V., P. Grieder, M. Kvasnica, D. Q. Mayne and M. Morari: *Computation of Invariant Sets for Piecewise Affine Discrete Time Systems subject to Bounded Disturbances.* In *IEEE Conference on Decision and Control*, pages 1418–1423, December 2004.

[RKM03] Raković, S. V., E. C. Kerrigan and D. Q. Mayne: *Reachability computations for constrained discrete-time systems with state- and input-dependent disturbances.* In *Proc. 42nd IEEE Conference on Decision and Control*, Maui, Hawwai, USA, December 2003.

[Ros03] Rossiter, J. A.: *Model-based predictive control, a practical approach.* CRC Press, 2003.

[Ser88] Serra, J.: *Image Analysis and Mathematical Morphology, Vol II: Theoretical advances.* Academic Press, 1988.

[SGE+03] Sveticic, G., A. Gentilini, U. Eichenberger, M. Lunginbül and M. Curatolo: *Combinations of morphine with ketamine for patient controlled analgesia. A new optimization method.* Anesthesiology, 98:1195–205, 2003.

[SLG+04] Suard, R., J. Löfberg, P. Grieder, M. Kvasnica and M. Morari: *Efficient Computation of Controller Partitions in Multi-Parametric Programming.* In *Proc. 43th IEEE Conf. on Decision and Control*, pages 3643–3648, Bahamas, December 2004.

[Sno97] Snoeyink, J.: *Point Location.* In Goodman, J. E. and J. O'Rouke (editors): *Handbook of Discrete and Computational Geometry*, chapter 30, pages 558–574. CRC Press, Boca Raton, New York, 1997.

[Son81] Sontag, E. D.: *Nonlinear regulation: The piecewise linear approach.* IEEE Trans. on Automatic Control, 26(2):346–358, April 1981.

[Son96] Sontag, E.D.: *Interconnected automata and linear systems: A theoretical framework in discrete-time.* In Alur, R., T.A. Henzinger and E.D. Sontag (editors): *Hybrid Systems III - Verification and Control*, number 1066 in *Lecture Notes in Computer Science*, pages 436–448. Springer-Verlag, 1996.

[Stu99] Sturm, J.F.: *Using SeDuMi 1.02, A MATLAB Toolbox for Optimization over Symmetric Cones.* Optimization Methods and Software, pages 625–653, October 1999.

[SZKM05] Sartori, V., E. Zanderigo, M. Kvasnica and M. Morari: *Optimal Infusion of intravenous morphine and ketamine - A Mixed-Integer Linear Programming application.* In *IEEE Engineering in Medicine and Biology Society*, Shanghai, China, September 2005.

[SZS⁺05] Sartori, V., E. Zanderigo, G. Sveticic, P.M. Schumacher, T. Bouillon, M. Morari and M. Curatolo: *A New Drug Interaction Model: Application to the Combination of intravenous Morphine and Ketamine.* Technical Report AUT05-03, Automatic Control Lab, ETHZ, Switzerland, 2005.

[TB02] Torrisi, F.D. and A. Bemporad: *HYSDEL — A Tool for Generating Computational Hybrid Models.* Technical Report AUT02-03, ETH Zurich, 2002. Submitted for publication on IEEE Trans. on Control Systems Technology.

[TBB⁺02] Torrisi, F.D., A. Bemporad, G. Bertini, P. Hertach, D. Jost and D. Mignone: *HYSDEL - User Manual.* Technical Report, August 2002.

[TJB01] Tøndel, P., T.A. Johansen and A. Bemporad: *An Algorithm for Multi-Parametric Quadratic Programming and Explicit MPC Solutions.* In *Proc. 40th IEEE Conf. on Decision and Control*, Orlando, Florida, December 2001.

[TJB03a] Tøndel, P., T.A. Johansen and A. Bemporad: *Evaluation of piecewise affine control via binary search tree.* Automatica, 39(5):945–950, 2003.

[TJB03b] Tøndel, P., T.A. Johansen and A. Bemporad: *Further results on Multi-parametric quadratic programming.* In *Proceedings 42nd IEEE Conference on Decision and Control*, Maui, Hawaii, USA, December 2003.

[TLS00] TOMLIN, C.J., J. LYGEROS and S. S. SASTRY: *A Game Theoretic Approach to Controller Design for Hybrid Systems.* Proc. of the IEEE, 88, July 2000.

[TM99] TYLER, M.L. and M. MORARI: *Propositional logic in control and monitoring problems.* Automatica, 35(4):565–582, 1999.

[TM00] THE MATHWORKS, INC.: MATLAB® – *The Language of Technical Computing.* The MathWorks, Inc., Natwick, NA, 2000.

[Tøn00] TØNDEL, P.: *Constrained Optimal Control via Multiparametric Quadratic Programming.* PhD thesis, Department of Engineering Cybernetics, NTNU, Trondheim, Norway, 2000.

[Tor02] TORRISI, F. D.: *Hybrid System DEscription Language (HYSDEL),* 2002. Available from http://control.ee.ethz.ch/~hybrid/hysdel/.

[Tor03] TORRISI, F.: *Modeling and Reach-Set Computation for Analysis and Optimal Control of Discrete Hybrid Automata.* PhD thesis, Automatic Control Laboratory, ETH Zurich, 2003.

[Ver03] VERES, S. M.: *Geometric Bounding Toolbox (GBT) for* MATLAB. Official website: http://www.sysbrain.com, 2003.

[Vid93] VIDYASAGAR, M.: *Nonlinear Systems Analysis.* Prentice Hall, 2nd edition, 1993.

[vS00] VAN DER SCHAFT, A. and H. SCHUMACHER: *An Introduction to Hybrid Dynamical Systems*, volume 251 of *Lecture Notes in Control and Information Sciences.* Springer-Verlag, 2000.

[Wei] WEISSTEIN, ERIC W.: *MathWorld-A Wolfram Web Resource.* http://mathworld.wolfram.com/.

[Wil93] WILLIAMS, H.P.: *Model Building in Mathematical Programming.* John Wiley & Sons, Third Edition, 1993.

[Zie94] ZIEGLER, G. M.: *Lectures on Polytopes.* Springer, 1994.

[ZLF+94] ZACNY, J.P., J. LANCE LICHTOR, D. FLEMMING, D. W. COALSON and W. K. THOMPSON: *A Dose-Response Analysis of the Subjective, Psychomotor and Physiological Effects of Intravenous Morphine in Healthy Volunteers.* Journal of Pharmacology and Experimental Therapeutics, 268:1–9, 1994.

[ZSS+05] ZANDERIGO, E., V. SARTORI., G. SVETICIC, T. BOUILLON, P. SCHUMACHER, M. CURATOLO and M. MORARI: *A New Model for Drug Interactions and Optimal Drug Dosing.* 2005. Submitted to IEEE EMBS 2005, Shanghai, China.