SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF CHEMICAL AND FOOD TECHNOLOGY

Reference number: FCHPT-19990-50950

SYNTHESIS AND IMPLEMENTATION OF MODEL PREDICTIVE CONTROL

DISSERTATION THESIS

Bratislava, 2016 **Ing. Bálint Takács**

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF CHEMICAL AND FOOD TECHNOLOGY

SYNTHESIS AND IMPLEMENTATION OF MODEL PREDICTIVE CONTROL

DISSERTATION THESIS

FCHPT-19990-50950

Study program: Process Control
Study field number: 2621
Study field: 5.2.14 Automation
Workplace: Department of Information Engineering and Process Control
Supervisor: doc. Ing. Michal Kvasnica, PhD.

Bratislava, 2016

Ing. Bálint Takács

Slovak University of Technology in Bratislava

Institute of Information Engineering, Automation and Mathematics

Faculty of Chemical and Food Technology

DISSERTATION THESIS TOPIC

Author of thesis:	Ing. Bálint Takács
Study programme:	Process Control
Study field:	5.2.14. automation
Registration number:	FCHPT-19990-50950
Student's ID:	50950
Thesis supervisor:	doc. Ing. Michal Kvasnica, PhD.
Title of the thesis:	**Synthesis and Implementation of Model Predictive Control**
Date of entry:	01. 09. 2012
Date of submission:	31. 05. 2016

Ing. Bálint Takács
Solver

prof. Ing. Miroslav Fikar, DrSc.
Head of department

prof. Ing. Miroslav Fikar, DrSc.
Study programme supervisor

Acknowledgements

I would like to express my gratitude to my supervisor Associate Professor Michal Kvasnica for his help, guidance, patience which inspired me. I am grateful for the friendly environment and valuable discussions with him. Special thanks go to my family Hajnalka, Borbála and Vojtech Takács and also to my fiancée Éva Eperjessy for the endless support and strong nerves. My special thanks belongs to my friends and colleagues from the Institute of Information Engineering, Automation and Mathematics for their advice and helpful attitude. I will be grateful forever. Thank you all!

Bálint Takács
Bratislava, 2016

Abstract

This work deals with synthesis and implementation of model predictive control (MPC). More concretely, we focus our attention on explicit model predictive control. The advantage of explicit model predictive control over implicit one is its lower implementation requirements. Even though in the offline phase a difficult parametric programming problem must be solved to obtain the feedback law, in the online phase the evaluation of the control law boils down to a mere function evaluation. The main limitation of this approach, however, is its memory demand. Especially for systems with a large number of states and/or for MPC problems with a long prediction horizon the complexity of the explicit feedback law grows quickly. As a consequence, the controller is not implementable if its memory requirements exceed the hardware's threshold. For this reason, we propose two complexity reduction techniques to decrease the final memory footprint of explicit model predictive controllers.

To allow for a seamless transition from an MPC design environment (such as Matlab) to hardware implementation, in this thesis we furthermore propose a code generation technique. Its purpose is to generate a self-contained implementation of the explicit MPC algorithm in two programming languages – Python and JavaScript. The code generation module is designed in such a way that the generated code can be seamlessly embedded into target control applications written in these languages.

Last, but not least, in this thesis we showcast the design and implementation of model predictive control to two real-life processes. The first one aims at controlling the temperature of the plating solution of a hard chromium plating process. The second application shows the control of a fast electromechanical system which manipulates objects by a magnetic field. The common denominator of these two plants is their underlying nonlinear behavior.

Keywords: Explicit Model Predictive Control, Memory and Complexity Reduction, Code Generation, Model Predictive Control Implementation

Abstrakt

Táto práca sa zaoberá syntézou a implementáciou prediktívneho riadenia (MPC). Konkrétne, zamerali sme sa na explicitné prediktívne riadenie. Výhodou explicitného prediktívneho riadenia oproti implicitnému sú jeho nižšie implementačné požiadavky. Napriek tomu, že v predimplementačnej fáze sa musí vyriešit' náročný parametrický program na získanie zákona riadenia, v samotnej implementačnej fáze pozostáva vyhodnotenie regulátora iba z vyhodnotenia funkcie. Hlavným limitujúcim faktorom tejto metódy je však práve jeho pamät'ová náročnost'. Zložitost' explicitného zákona riadenia rastie rapídne rýchlo pre systémy s veľkým počtom stavových premmenných a s dlhým predikčným horizontom. Dôsledkom toho je regulátor neimplementovateľný, pretože jeho pamät'ová náročnost' prevyšuje možnosti hardvéru. Z tohto dôvodu sme navrhli dva rozličné postupy, ktoré znížia finálne pamät'ové zat'aženie explicitného prediktívneho regulátora.

Aby bolo možné zabezpečit' hladký prechod od návrhu MPC v programovom prostredí (ako je Matlab) až po finálnu hardvérovú implementáciu, v tejto práci sme d'alej navrhli techniku generovania kódu. Jej cieľom je vytvorenie sebestačného explicitného MPC algoritmu v dvoch programovacích jazykoch – Python a JavaScript. Modul generovania kódu je navrhnutý takým spôsobom, aby generovaný kód mohol byt' bez problémov vnorený do cieľových aplikácií napísaných v týchto jazykoch.

V neposlednom rade, v tejto práci uvedieme návrh a implementáciu prediktívneho riadenia na dvoch reálnych aplikáciách. Prvá z nich je zameraná na riadenie teploty roztoku chrómu v galvanizačnom procese. Druhá aplikácia predstavuje riadenie rýchleho elektromechanického systému, ktorý ovláda objekty cez magnetické pole. Spoločným menovateľom týchto dvoch procesov je ich nelineárne správanie sa.

Kľúčové slová: explicitné prediktívne riadenie, redukcia zložitosti, generovanie algoritmov, implementácia prediktívneho riadenia

Contents

List of Figures

List of Tables

List of Abbreviations

DMC	Dynamic Matrix Control
EMPC	Explicit Model Predictive Control
KKT	Karush-Kuhn-Tucker
LP	Linear Program
LQR	Linear Quadratic Regulator
LTI	Linear Time Invariant
MILP	Mixed Integer Linear Program
MIMO	Multiple-Input Multiple-Output
MIQP	Mixed Integer Quadratic Program
MPC	Model Predictive Control
MIP	Mixed Integer Program
NLP	Non-Linear Program
PID	Proportional Integral Derivative Controller
PLC	Programmable Logic Controller
PWA	Piecewise Affine
PWQ	Piecewise Quadratic
QP	Quadratic Program

SISO	Single-Input Single-Output
mp-LP	Multiparametric Linear Programming
mp-QP	Multiparametric Quadratic Programming

Chapter 1

Introduction

" Progress is man's ability to complicate simplicity. "

Thor Heyerdahl

Control theory has its origins both in mathematics and in engineering. It deals with influencing the behavior of dynamical systems around us. The main idea is to influence the system based on the information from measurements. The goal is to create an automated way of doing so, so-called controller, which takes care of the process control instead of humans. The purpose of the controller is to maintain some specific conditions, e.g. constant temperature in the chemical reactor or constant speed in case of cars. At the beginning, only simple approaches were used, such as rule-based controllers or PIDs. In the second half of the last century, more complex methods were introduced. One of these methods is Model Predictive Control (MPC). This control approach makes it possible to use the mathematical model of the real process to predict its future behavior. The information about the prediction is used to construct such a control policy which fulfills the given requirements. In order to obtain the values of the control actions, which are optimal related to some criteria, an optimization problem must be solved. Due to the lack of the computation power, this kind of control approach could only be implemented for processes with slow dynamics. After the time computers started to offer higher computation power which allowed aiming even systems with faster dynamics. The big changes came when Explicit Model Predictive Control was introduced. The

big benefit of this approach lies in its easy and cheap implementation since only operations such matrix multiplication and addition are required. There is no need for additional numerical solvers which makes the implementation simpler. Since in the evaluation phase there is no such operation as a division the validation of the final algorithm is also easier and faster. The Explicit Model Predictive Control is obtained by solving an optimization problem by parametric programming and the controller can be stored as a look-up table. The look-up table encodes a piecewise affine (PWA) function of the parameters and is defined over polytopic regions. This precomputed form of the optimization problem became desired in the industry.

1.1 Motivation

To implement explicit MPC, the parametric solution to the optimization problem must be saved in the memory of the target implementation hardware. The solution is defined over polytopic critical regions, which are given by a set of linear inequalities. The main limitation of the successful implementation is the method's memory demand. By increasing the value of the prediction horizon also the number of the critical regions increases. In order to reduce the memory demand, complexity reduction methods are investigated. Here, the goal is to reduce the number of critical regions. This could be done by using various approximation methods. The approximation simplifies the parametric solution but it also deteriorates the control quality. Therefore, there must be a compromise between the performance of the controller and its memory demand. After the final memory demand of the implementable controller fulfills the implementation requirements a so-called point location algorithm must be created to extract the optimal solution of the optimization problem from the precomputed lookup table. In order to make this step as easy as possible, the code generation module should be created to export the point location algorithm with the necessary information to the given programming language. The goal is to create such an interface which is user friendly, the generated code is self contained and could be easily merged with existing applications.

1.2 Goals of the Thesis

This dissertation thesis aims at the synthesis and implementation of Model Predictive Control techniques. The main objectives can be summarized as follows:

- Complexity and memory reduction by approximation techniques.

  Complexity reduction in explicit MPC is a crucial part of this work. In general, the goal is to develop such reduction techniques that are able to decrease the memory demand to the given implementation burden. In order to achieve a higher memory reduction approximation techniques used to be considered. However these methods could efficiently reduce the memory demand, the control performance also deteriorates. Therefore, the goal is to find a good compromise between the memory consumption and the controller's performance.

  The contributions of this thesis with respect to complexity reduction are based on the following publications which I have co-authored:

  – Takács, B. – Holaza, J. – Kvasnica, M. : NLP-based Derivation of Bounded Convex PWA Functions: Application to Complexity Reduction in Explicit MPC. In *Veszprém Optimization Conference: Advanced Algorithms*, Veszprém, Hungary, pp. 95–95, 2012.

  – Holaza, J. – Takács, B. – Kvasnica, M. : Complexity Reduction in Explicit MPC via Bounded PWA Approximations of the Cost Function. In *Selected Topics in Modelling and Control, Editors: Mikleš, J., Veselý, V., Slovak University of Technology Press, vol. 8*, pp. 27–32, 2012.

  – Takács, B. – Holaza, J. – Kvasnica, M. – Di Cairano, S. : Nearly-Optimal Simple Explicit MPC Regulators with Recursive Feasibility Guarantees. In *IEEE Conference on Decision and Control*, Florence, Italy, pp. 7089–7094, 2013.

  – Holaza, J. – Takács, B. – Kvasnica, M. : Synthesis of Simple Explicit MPC Optimizers by Function Approximation. In *Proceedings of the 19th International Conference on Process Control*, Štrbské Pleso, Slovakia, pp. 377–382, 2013.

  – Holaza, J. – Takács, B. – Kvasnica, M. : Simple Explicit MPC Controllers Based on Approximation of the Feedback Law. In *ACROSS Workshop on Cooperative Systems*, Zagreb, Croatia, pp. 48–49, 2014.

  – Holaza, J. – Takács, B. – Kvasnica, M. – Di Cairano, S.: Nearly optimal simple explicit MPC controllers with stability and feasibility guarantees.

In *Optimal Control Applications and Methods*, vol.6, 2015.

- Code generation tool to export the parametric solution.

  Here the goal is to bring explicit MPC closer to a wider community. In particular, we show how explicit MPC controllers can be exported to Python and JavaScript. To do so, we propose a new module in Multi-Parametric Toolbox, which exports the parametric solution to the chosen programming language. The export procedure should take care about that the generated file will contain all the necessary mathematical operation together with the information about the parametric solution. The correctness of the solution and the merging with existing application is validated on three test cases. Two of them cover Python - one applies explicit MPC to control of a computer game and the second one uses it for control of a quadcopter. With respect to JavaScript, we investigate the case of controlling, optimally, the temperature inside a building. Here, the explicit MPC controller runs in a web browser.

  The results for code generation as reported in this thesis are based on the following publications:

  – Takács, B. – Holaza, J. – Števek, J. – Kvasnica, M. : Export of Explicit Model Predictive Control to Python. In *Proceedings of the 20th International Conference on Process Control*, Štrbské Pleso, Slovakia, pp. 78–83, 2015.

  – Takács, B. – Števek, J. – Valo, R. – Kvasnica, M. : Python Code Generation for Explicit MPC in MPT. In *European Control Conference*, Aalborg, Denmark, 2016.

- Implementation of MPC on non-linear processes

  Although we always try to simplify the optimal control problem, in some cases too many simplifications lead to non-accurate models and lesser performance. Therefore, non-linear models are used, which can lead to a better description of the real process. The goal of this part of the thesis is to design and to implement various MPC-based control methods on two real-life processes that exhibit nonlinear behavior. The first process in which we are interested in is a hard chromium plating process, where the task is to control the temperature of the plating solution. The second process deals with the position control of

an object by changing the magnetic field around it. The results in this part of the thesis are, as of now, unpublished.

1.3 Overview of the Thesis

This thesis is aimed to decrease the computation demand of the model predictive control technique. The thesis is divided into 2 main parts. Part I discusses the theoretical background and concepts of model predictive control. Part II then reports our contributions. Here, after a concise overview of the contributions in Chapter 4, we first discuss two complexity reduction techniques in Chapter 5. Here the objective is to decrease the memory demand of explicit MPC solutions by reducing the number of regions over which the feedback law is defined. Two approaches are presented. One is based on approximating the optimal value function in such a way that a simpler controller can be recovered while preserving stability properties. The second method directly approximates the (complex) explicit MPC feedback law to find its simpler replacement. Outcomes of both approaches are documented on illustrative examples.

Chapter 6 then introduces the code generation framework for Python and JavaScript. The chapter introduces the framework from the users' perspective and discusses technical aspects of the automatically generated code. Three case studies are presented to illustrate the benefits of the proposed framework. The first one shows how to use explicit MPC to control a computer game in Python. In the second application, explicit MPC is synthesized and applied to control a quadcopter. Finally, the third application illustrates how a web-based version of explicit MPC, implemented in JavaScript, can be used to control the temperature in a building.

Finally, Chapter 7 reports the application of MPC to two real-life processes which exhibit nonlinear behavior. The first one deals with temperature control of a planting solution inside a hard chromium electroplating process. Three distinct MPC setups are considered. One is based on the full non-linear model of the plant and serves as a performance benchmark. Then, two setups based on linear models are considered. In the first one, the linear model is updated at each sampling instant via linearization around the current plant's conditions. The second approach uses one fixed linearization. The performance of all three approaches is verified via simulations. The second case study investigates the control of a magnetic ma-

nipulator which modifies the magnetic field as to move a metallic ball to desired locations. Here, besides the plant dynamics being nonlinear, the second difficulty is a high sampling frequency. To cope with these aspects we propose a two-layer MPC structure.

Part I

Theoretical basis

Chapter 2

Convex Optimization

Convex optimization finds its place in many today practical applications such process control, signal processing, data analysis, etc. It is a subfield of general optimization, which devotes attention to the problem of minimization of a convex function over convex set as defined in (2.1) (Boyd and Vandenberghe, 2004). In the following an equivalent but slightly different notation will be used for optimization problem (2.2). The reason why convex optimization is generally preferred over non-convex one is that any optimum of the convex problem is also a global optimum. This property makes convex problems easier to solve compared to their non-convex counterparts. Over the years, several effective methods, algorithms, and toolboxes (Gurobi Optimization (2015), GLPK (2012), ApS (2015)) have been developed to solve even large-scale convex optimization problems. An another advantage of convex optimization problems is that many engineering problems can be formulated in such a form, for instance, minimization of energy consumption of the given process or imposition of safety limits. Every optimization problem consists of two parts: the objective function (2.2a) and the constraints (2.2b).

$$f(x^\star) = \min\{f(x) \mid x \in \mathcal{X}\} \tag{2.1}$$

$$f(x^\star) = \min_x \; f(x) \tag{2.2a}$$

$$\text{s.t. } x \in \mathcal{X} \tag{2.2b}$$

For an optimization problem to be convex, the objective function must be a convex function and the constraints must have a form of a convex set. Properties of convex functions and convex sets are therefore reviewed in the sequel.

2.1 Convex Functions

A function f is a real-valued function, which maps the n-dimensional space of $\mathbb{R}^n$ into a scalar space $\mathbb{R}$. Based on the level of convexity we can distinguish two kinds of convex functions: non-strictly convex function and strictly convex functions.

Non-strictly convex function could be defined as follows

$$\forall x \neq y \in \mathcal{Z}, \forall t \in [0,1] : \; f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta) f(y). \tag{2.3}$$

A typical example of a non-strictly convex functions, which are very often used in optimization as a form objective function, are linear and affine functions. The line segment connecting two arbitrary points x, y from the domain $\mathcal{Z}$ of function f, is lying exactly on the value function f and not above it. The mentioned linear and affine functions are not only non-strictly convex functions but have the properties of non-strictly concave functions as well. The graphical representation of non-strict convex function could be seen in Fig. 2.1(b).

A strictly convex function could be defined in the following way

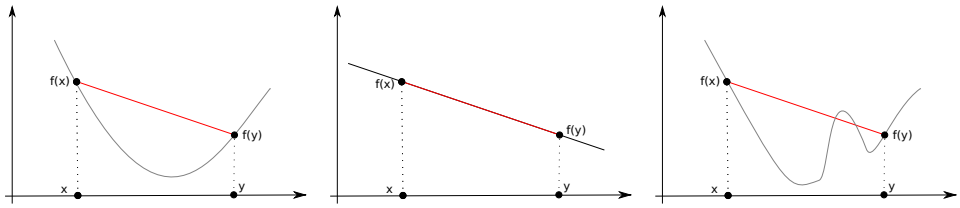
$$\forall x, y \in \mathcal{Z}, \forall t \in (0,1) : \; f(\theta x + (1-\theta)y) < \theta f(x) + (1-\theta) f(y). \tag{2.4}$$

For each point x, y from the domain $\mathcal{Z}$ of function f is strictly above the value function f. Graphical representation of a strict convex quadratic function depicted in Fig. 2.1(a).

In case the condition (2.3) or (2.4) not holds for the whole investigated domain of the function f than the given function is not convex. As it is evident from the Fig. 2.1(c), the non-convex function could feature multiple local minima, which complicates the search for the global optimum. Therefore in the sequel only convex functions are considered.

Norms

Norms are a special class of convex functions that are often used in optimal control problems. Every norm assigns a strictly positive distance to each vector except



(a) Strictly convex function (b) Non-strictly convex function (c) Non-convex function

Figure 2.1: Types of functions

the null vector. There exist several norms which express the distance in a different way. The general form of an arbitrary norm is defined as follows:

$$x = [x_1^\top, x_2^\top, ..., x_n^\top]^\top \tag{2.5}$$

$$||x||_p = (\sum_i |x_i|^p)^{1/p} \tag{2.6}$$

Each norm has to fulfill the following conditions:

1. triangle inequality (2.7)
2. zero vector (2.8)

All the norms satisfy the triangle inequality conditions, which states that the sum of two normed vectors is higher or at least equal than the norm of a sum of two vectors.

$$||x + y||_p \leq ||x||_p + ||y||_p \tag{2.7}$$

Furthermore, the norm can return 0 value only if the argument of the norm x is a zero vector, in other cases, the norm operation returns strictly positive values.

$$||x||_p > 0, \text{ if } x \neq \mathbf{0} \tag{2.8}$$

The general form of the p norm was defined in (2.5), by changing the parameter p we can get special types of norms. In this section three different kinds of norms will be introduced, namely the Manhattan norm, Euclidean norm, and maximum norm.

In case of Manhattan norm the parameter p is set to value 1. In this case, we can modify (2.5) to the following form (2.9). The Manhattan or 1-norm is a sum of absolute values of the vector x.

$$||x||_1 = (\sum_i |x_i|) \tag{2.9}$$

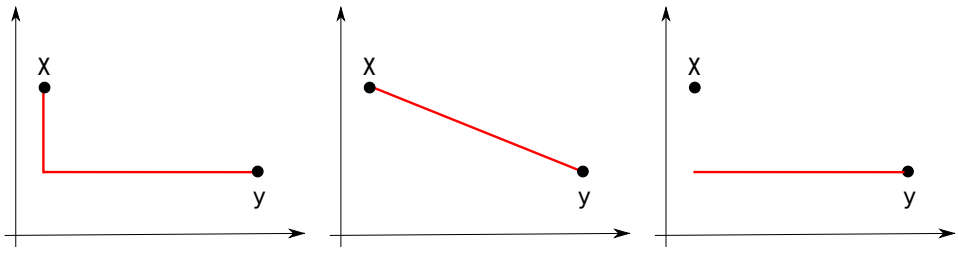
The other well-known norm often used in optimal control is the Euclidean norm (2.10). We obtain the definition of this norm if we substitute for parameter p value 2.

$$||x||_2 = \sqrt{\sum_i |x_i|} \tag{2.10}$$

Especially the square of the Euclidean norm is used often in optimization tasks.

The third mostly used norm in optimal control is the maximum norm. This kind of norm is used when the minimization of the possible worst case scenario is required.

$$||x||_\infty = \max_i |x_i| \tag{2.11}$$



(a) Manhattan (1) norm (b) Euclidian (2) norm (c) Maximum (∞) norm

Figure 2.2: Norms

All the above-mentioned norms lead to absolute value in the scalar case.

2.2 Convex Sets

As it was mentioned previously, in the case of the convex optimization we are interested in the feasible sets which are convex. Generally, a convex set could be defined as a part of the space $\mathcal{R}^n$, for which the whole connecting line segment for each chosen pair of points must be within the given set. Which means the set $S \subseteq \mathbb{R}^n$ is convex if and only if for any pair of $x \in S$ and $y \in S$ holds

$$\lambda x + (1 - \lambda)y \in S, \ \forall \lambda \in [0, 1]. \tag{2.12}$$

The condition presented in (2.12) does not hold for non-convex sets. The following convex sets are the most commonly used

- half-space
- hyperplane
- ellipsoid
- hyper-cube, hyper-rectangle
- polytope
- polyhedron

Very often in process control the sets are defined by half-spaces. In that case the set $S \subseteq \mathbb{R}^n$ will be linear in x, which in optimal control leads to linear constraints. In case of using hyper-planes, the defined set will be of the following form

$$S = \{x \mid a^\top x \leq b\}, \tag{2.13}$$

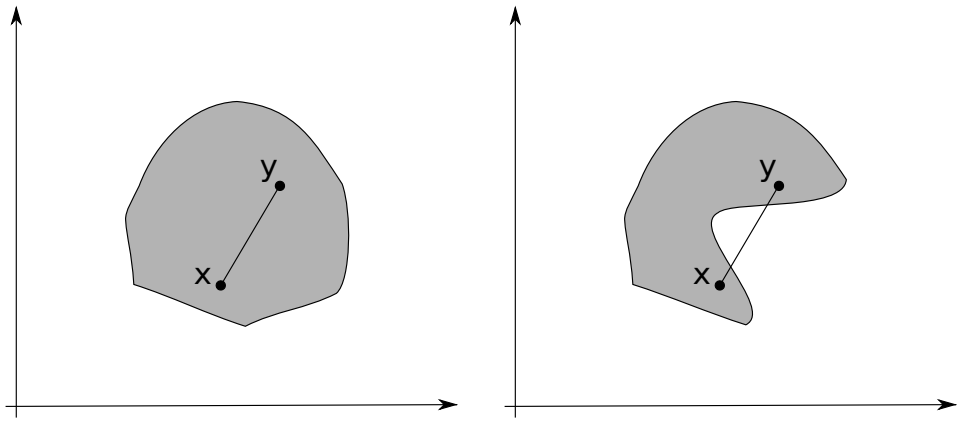
where $a \in \mathbb{R}^n$ is the normal vector of the half-space and $b \in \mathbb{R}$ is a constant part. In case of using set defined by hyperplanes $S \subseteq R^n$, the set could be described by

$$S = \{x \mid a^\top x = b\}, \tag{2.14}$$

where, similarly as in the previous case, $a \in \mathbb{R}^n$ is the normal vector of the hyperplane and $b \in \mathbb{R}$ is a constant part.

These two types of convex sets often arise in the context of model predictive control where they describe minimal/maximal bounds on predicted quantities (in the case of half-spaces), or equality constraints (in the case of hyperplanes).

The intersection of a final number of half-spaces defines polytopes or polyhedra, which are reviewed next.



(a) Convex set (b) Non-convex set

Figure 2.3: Sets

Polyhedra and Polytopes

A polyhedron P in the n-dimensional Euclidean space is defined as the intersection of finitely many half-spaces of the form (2.13). Specifically, consider m half-spaces $a_i^\top x \leq b_i$ with $a_i \in \mathbb{R}^n$ and $b_i \in \mathbb{R}$ for $i = 1, \ldots, m$. Then the set

$$P = \{x \mid a_1^\top x \leq b_1, \ldots, a_m^\top x \leq b_m\}, \tag{2.15}$$

is called a closed polyhedron. If all of the inequalities in (2.15) are strict, i.e.

$$P = \{x \mid a_1^\top x < b_1, \ldots, a_m^\top x < b_m\}, \tag{2.16}$$

then such a polyhedron is called open. If some of the inequalities are strict and some non-strict, such a polyhedron is neither open nor closed.

If the polyhedron contains a ray, then it is called to be unbounded. The polyhedron, which is bounded, therefore it can not contain any rays, is polytope. Polyhedra and polytopes are convex sets. Moreover, closed polytopes are so-called compact sets, because they are closed and bounded. Every polytope admits two representations, which are reviewed next.

Half-space and Vertex Representation of Polytopes

Polytope can be defined by two different ways. One representation, often called the half-space representation (or just $\mathcal{H}$-representation for short) defines a polytope as

the intersection of a finite number of half-spaces, i.e.,

$$P = \{x \mid Ax \leq b\}, \tag{2.17}$$

with $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$.

Another option is to represent a polytope as the convex hull of its vertices $v_1, \ldots, v_p$ with $v_i \in \mathbb{R}^n$ for $i = 1, \ldots, p$:

$$P = \left\{ x \;\middle|\; x = \sum_{i=1}^{p} \lambda_i v_i,\ 0 \leq \lambda_i \leq 1,\ \sum_{i=1}^{p} \lambda_i = 1 \right\}. \tag{2.18}$$

Such a representation is called the vertex representation, or the $\mathcal{V}$-representation for short.

It should be noted that the half-space representation could be transformed to the vertex representation and vice versa. The $\mathcal{H}$-to-$\mathcal{V}$ translation is called the vertex enumeration, while the opposite direction is called convex hull. Both operations are expensive since their runtime complexity is exponential in the worst case.

Minimial Representation of Polytopes

Minimal representation of a polytope means that removing of any halfspace or vertex would change its form. If removed part has not an effect on the form of the polytope then these constraints are called redundant. The elimination of redundant constraints has not any effect on the actual form of the polytope, but can fasten up the computation of optimization problem, since fewer constraints must be considered. The complexity of finding the minimal representation depends on the number of halfspaces or vertices because each halfspace/vertex must be checked. The checking cost is one Linear Program for each halfspace/vertex. The procedure for detecting redundant half-space of a polytope in the $\mathcal{H}$-representation is reported

as Algorithm 2.1.

Data: Half-spaces $a_i^\top x \leq b_i$, $i = 1, \dots n_h$ of a polytope.
for $i = 1, \dots, n_h$ **do**
    Let $J = \max\ a_i^\top x$ s.t. $a_j^\top x \leq b_j$, $\forall j \in \{1, \dots, n_h\} \setminus i$;
    if $J < b_i$ **then**
        halfspace $a_i^\top x \leq b_i$ is redundant
    else
        halfspace $a_i^\top x \leq b_i$ is not redundant
    end
end

Algorithm 2.1: Detection of redundant half-spaces.

The detection of redundant vertices in the $\mathcal{V}$-representation is presented in Algorithm 2.2.

Data: Vertices $v_1, \dots, v_{n_v}$ of a polytope.
for $i = 1, \dots, n_v$ **do**
    Solve the following linear programming problem:
    find λ
    s.t. $v_i = \sum_{j \neq i} \lambda_j v_j$,
        $0 \leq \lambda_j \leq 1$,
        $\sum_{j \neq i} \lambda_j = 1$.
    if *feasible* **then**
        vertex v_i is redundant
    else
        vertex v_i is not redundant
    end
end

Algorithm 2.2: Detection of redundant vertices

The result of Algorithm 2.1 is a minimal $\mathcal{H}$-representation of the given polytope, while the outcome of Algorithm 2.2 is the minimal $\mathcal{V}$-representation of the given polytope. Both algorithms come useful in case implementing EMPC in real hardware. In worst case each critical region consists of that many halfspaces how many constraints the MPC formulation has. Therefore, it is crucial from the implementation point of view to dispose of all the not necessary halfspaces or vertices.

2.3 Chebyshev's Ball

Chebyshev's ball is the largest hyperball inscribed to a polytope. Its properties can be used e.g. to detect whether a particular polytope is fully dimensional or not. If a polytope is not fully dimensional, we call it lower dimensional. An example of a lower-dimensional polytope in $\mathbb{R}^2$ is a line segment. If no ball inscribed to a particular polytope could be found, such a polytope is an empty set. Each Chebyshev's ball is characterized by its center point x_c and its radius r. These two parameters can be found by solving the following linear programming problem:

$$\max_{r,x_c} r \tag{2.19a}$$

$$\text{s.t. } a_i^\top x_c + ||a_i||_2 r \leq b_i,\ i = 1, \ldots, m, \tag{2.19b}$$

where m denotes the number of half-spaces which define the polytope. Note that problem (2.19) is always feasible even if the polytope is an empty set. In such a case a negative radius r is obtained. If $r = 0$ is the optimal solution to (2.19), the polytope is lower dimensional. Otherwise, if $r > 0$, the polytope is fully dimensional. In such a case the center point needs not to be unique, for example in the case of hyper-rectangle we can have multiple equally large inscribed circles.

2.4 Linear Programming

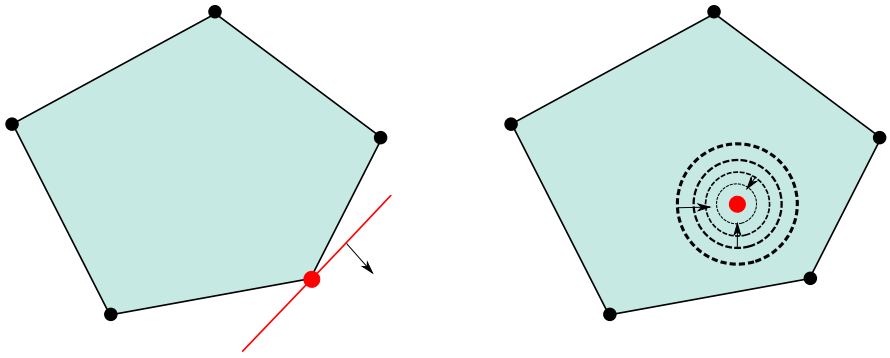
Linear programming (LP) is a special type of convex optimization problems with linear objective function and linear constraints (Boyd and Vandenberghe, 2004). A general formulation of an LP is given by

$$\min_x c^\top x \tag{2.20a}$$

$$\text{s.t. } Ax \leq b, \tag{2.20b}$$

$$Cx = d. \tag{2.20c}$$

Here, $x \in \mathbb{R}^n$ is the vector of optimization variables. The fitness of a particular choice of x is evaluated by the linear objective function (2.20a) with $c \in \mathbb{R}^n$. The optimal solution must satisfy equality and inequality constraints in (2.20b) and (2.20c) with $A \in \mathbb{R}^{m_i \times n}$, $b \in \mathbb{R}^{m_i}$, $C \in \mathbb{R}^{m_e \times n}$, $d \in \mathbb{R}^{m_e}$. A graphical interpretation of a linear programming problem is provided in Fig. 2.4(a).



(a) Linear Programming (b) Quadratic Programming

Figure 2.4: Linear and Quadratic Programming

The constraints determine the part of the search space where the problem is feasible. In general, this feasible set is a polyhedron. The number of equality constraints in (2.20b), i.e., m_e must be lower than the dimension of the problem. Otherwise, if $m_e = n$, the solution is uniquely determined by the equalities (provided the row rank of C is equal to m_e). If $m_e > n$ (more specifically, when the row rank of C is larger than n), then the problem is infeasible.

The goal of linear programming is to find the point in the feasible set where the objective function acquires the smallest possible value. Linear programs have a special property that their optimal solutions always lie on the vertex of the feasible set (which is a polyhedron). Even if the problem features multiple global optima, there always exists one on the vertex.

Besides the standard LP form in (2.20), another form is frequently used as well:

$$\min_x \; e^\top x \tag{2.21a}$$

$$\text{s.t. } Fx = g, \tag{2.21b}$$

$$x \geq 0. \tag{2.21c}$$

It is always possible to convert the standard form (2.20) into (2.21) and vice verse. The conversion from (2.20) into (2.21) is described next.

First, since (2.21) only assumes non-negative values of the decision variables,

we need to rewrite x as the difference of two non-negative quantities, i.e.,

$$x = x^+ - x^-, \tag{2.22}$$

with $x^+ \geq 0$ and $x^- \geq 0$. Subsequently, all inequality constraints in (2.20b) need to be converted into equalities. This can be achieved by introducing a non-negative vector of slack variables, i.e., $s \geq 0$, which translate $Ax \leq b$ into $Ax + s = b$. Putting it all together we obtain

$$\min\ c^\top (x^+ - x^-) \tag{2.23a}$$

$$\text{s.t. } A(x^+ - x^-) + s = b, \tag{2.23b}$$

$$C(x^+ - x^-) = d, \tag{2.23c}$$

$$0 \leq x^+, 0 \leq x^-, 0 \leq s \tag{2.23d}$$

By introducing the new optimization variable $y = [x^+, x^-, s]^\top$ we can rewrite (2.23) to

$$\min\ [c^\top \quad -c^\top \quad 0^\top] y \tag{2.24a}$$

$$\text{s.t. } \begin{bmatrix} A & -A & I \\ C & -C & 0 \end{bmatrix} y = \begin{bmatrix} b \\ d \end{bmatrix}, \tag{2.24b}$$

$$0 \leq y. \tag{2.24c}$$

Then the standard LP form (2.21) is recovered with

$$e = [c^\top \ -c^\top \ 0^\top]^\top, \tag{2.25a}$$

$$F = \begin{bmatrix} A & -A & I \\ C & -C & 0 \end{bmatrix}, \tag{2.25b}$$

$$g = \begin{bmatrix} b \\ d \end{bmatrix}. \tag{2.25c}$$

A feasible optimal solution to the given linear programming problem does not always exist. If the constraints in (2.20) are inconsistent it means the feasible domain of the given linear program is an empty set, in this case, the linear program is infeasible. If the domain or feasible set of the given linear program is not bounded in the direction of the gradient of objective function then there no optimal solution could be found. Linear programming problems could be solved by various algorithm. The less intelligent to solve it is to determine the vertices of the feasible

set, defined by constraints. After in each vertex to calculate the value of the objective function and to choose the minimum value. This approach is not usable if the dimension of the optimization problem is high, because just even to obtain the vertices of the feasible set is so-called vertex enumeration problem, which is not a straightforward operation. Another method which can be used to solve LP is so-called simplex method presented by George Dantzig. This method uses as a starting point one of the vertices and tries to find the path to the optimal vertex, without exploring all the vertices. In general, the LP problem could be defined as

$$\min_x \; c^\top x \tag{2.26a}$$

$$\text{s.t. } Ax \leq b \tag{2.26b}$$

this formulation should be transformed into

$$\max_z \; c^\top z \tag{2.27a}$$

$$\text{s.t. } Az \leq b, \tag{2.27b}$$

$$z \geq 0. \tag{2.27c}$$

It can be converted in three steps

- $Ax \leq b \Rightarrow \; Ax + Is = b, \; s \geq 0$ The inequality condition is transformed into equality constraints by introducing new nonnegative slack variables s.

- $x = x^+ - x^-$ Since the formulation in (2.27) assumes only nonnegative numbers, the negative numbers are obtained as a difference of two non negative numbers, therefore two new optimization variables must be introduced x^+ and x^-.

- $\min_x \; c^\top x \Rightarrow \max_x -c^\top x$ The minimization must be converted into maximization problem.

By assuming these 3 steps (2.26) could be transformed into:

$$\max_{x^+,x^-,s} \begin{bmatrix} -c^\top & c^\top & 0 \end{bmatrix} \begin{bmatrix} x^+ \\ x^- \\ s \end{bmatrix} \tag{2.28a}$$

$$\text{s.t.} \begin{bmatrix} A & -A & I \end{bmatrix} \begin{bmatrix} x^+ \\ x^- \\ s \end{bmatrix} = b, \tag{2.28b}$$

$$\begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x^+ \\ x^- \\ s \end{bmatrix} \geq 0 \tag{2.28c}$$

If the optimization problem is in 2.28 the so-called simplex tableaux is created and by pivot operations, which are elementary row operations the maximum value of the objective function is achieved. The iterative procedure is well documented in (Dantzig and Thapa, 2003).

2.5 Quadratic programming

We speak about quadratic programming if the objective function is quadratic subject to linear inequality and linear equality constraints:

$$\min \frac{1}{2} x^\top H x + c^\top x \tag{2.29a}$$

$$\text{s.t. } Ax \leq b, \tag{2.29b}$$

$$Cx = d \tag{2.29c}$$

For the problem to be convex, the objective function must be convex, which is the case when $H \succeq 0$, i.e., when the matrix H is positive semi-definite.

Quadratic programming is used when the distance or energy consumption must be minimized. In QP the optimal value does not have to lie on the vertex or edge of the polytope. The optimum can also be situated in the interior of the polytope. The solution of the QP is always unique, unlike in case the of LP. The optimization problem can in some applications be defined as LP with a quadratic part with relatively small weight in order to obtain the unique solution. QP can be solved by using the KKT conditions applying the active set method, which is covered in the next section.

2.6 Karush-Kuhn-Tucker conditions

In convex optimization, the Karush-Kuhn-Tucker (KKT) conditions are necessary and sufficient conditions for a solution LP/QP be optimal. The famous KKT conditions were named after Harold W. Kuhn and Albert W. Tucker, who published the optimality conditions in 1951, but later it was discovered that the conditions were introduced by William Karush in 1939 in his unpublished master's thesis. The KKT conditions could be divided into 4 parts: stationarity condition (2.31a), primal feasibility (2.31b, 2.31c), dual feasibility (2.31d) and complementary slackness (2.31e). The KKT conditions for optimization (2.30) are defined in (2.31). They play an important role in convex optimization, because it is possible to solve the system of KKT conditions analytically, and the solution is then also the solution for the given optimization problem. The KKT conditions are used to construct the parametric solution, this will be discussed in the section on Multiparametric programming.

$$\min\ f(z) \tag{2.30a}$$
$$\text{s.t. } g_i(z) \leq 0,\ \forall i = 1, \ldots, m \tag{2.30b}$$
$$h_j(z) = 0,\ \forall j = 1, \ldots, p \tag{2.30c}$$
$$z \in \mathcal{Z} \tag{2.30d}$$

$$\bigtriangledown f(z^\star) + \sum_{i=1}^{m} \lambda_i \bigtriangledown g_i(z^\star) + \sum_{j=1}^{p} \mu_j \bigtriangledown h_j(z^\star) = 0 \tag{2.31a}$$
$$g_i(z^\star) \leq 0,\ \forall i = 1, \ldots, m \tag{2.31b}$$
$$h_j(z^\star) = 0,\ \forall j = 1, \ldots, p \tag{2.31c}$$
$$\lambda_i{}^\star \geq 0,\ \forall i = 1, \ldots, m \tag{2.31d}$$
$$\lambda_i{}^\star g_i(z^\star) = 0, \forall i = 1, \ldots, m \tag{2.31e}$$

In the optimal point $z^\star$ the value of the Lagrangian is equal to the original problem since the equality constraints in optimal point are fulfilled as presented in (2.31c) and also inequality constraints do not change the value of the Lagrangian since (2.31e) holds. This problem is difficult to solve analytically as there is a nonlinear relation between λ_i and $g_i(z)$ as presented in (2.31e). This system could

be solved by so-called active set method, when the constraints are divided into the set of active and the remaining set of inactive constraints.

Optimization problem in (2.30) could be solved by applying the KKT method presented in (2.31). This method explores a different combination of active constraints. The downside of this method is that a large number of combination must be explored. In case of solving QP with the pure KKT method a possible number of constraints we need to consider

$$N_{\mathrm{AC}} = \sum_{i=0}^{n} \binom{m}{i} = \sum_{i=0}^{n} \frac{m!}{i!(m-i)!}, \tag{2.32}$$

where N_{AC} represents the number of possible active set combinations, m stands for the number of the constraints, variable n represents the number of optimization variables. In the worst case, we need to explore N_{AC} number of combination, which even for small systems is high number. For example, if we have only 5 optimization variables and for each variable we have 2 constraints the total amount of possible active constraints are 219 combinations. There also exists the clever way how to find the right combination of active set and not to explore all the combination. The name of this approach is an active set method. There exists several algorithms that use builtin heuristics to obtain good guess of the set of active constraints (Ferreau et al., 2014).

2.7 Mixed Integer Programming

Even if Mixed Integer Programming (MIP) does not belong to the family of convex optimization, it is frequently used to solve optimal control problems. This special class of non-linear programming allows for engineers to formulate problems, where not only continuous variables are considered. In general two types of MIP problems are considered. If the objective function is linear, we speak about Mixed Integer Linear Programming (2.33), while if the objective function contains a quadratic part Mixed Integer Quadratic Programming (2.34) must be solved.

$$\min_{x,\delta} \; c^\top x + d^\top \delta \tag{2.33a}$$

$$\text{s.t. } A_x x + A_\delta \delta \leq b, \tag{2.33b}$$

$$C_x x + C_\delta \delta = d, \tag{2.33c}$$

where variable $x \in \mathbb{R}^n$ and $c \in \mathbb{R}^n$, while $\delta \in \mathbb{R}^q$ and $d \in \mathbb{R}^q$. By assuming m inequality constraints the dimensions of the matrices are $A_x \in \mathbb{R}^{n \times m}$, $A_\delta \in \mathbb{R}^{q \times m}$ and $b \in \mathbb{R}^m$, while by considering p equality constraints the matrices have the following dimensions $C_x \in \mathbb{R}^{n \times p}$, $C_\delta \in \mathbb{R}^{q \times p}$ and $d \in \mathbb{R}^p$.

$$\min_{x,\lambda} \; x^\top H_1 x + x^\top H_2 \delta + \delta^\top H_3 \delta + d^\top \delta \tag{2.34a}$$

$$\text{s.t. } A_x x + A_\delta \delta \leq b, \tag{2.34b}$$

$$C_x x + C_\delta \delta = d, \tag{2.34c}$$

where matrices in the constraints by assuming m inequality and p equality constraints have the same dimension as in the case of (2.33). The weighting matrices have the following dimensions: $H_1 \in \mathbb{R}^{n \times n}$, $H_2 \in \mathbb{R}^{n \times q}$ and $H_3 \in \mathbb{R}^{q \times q}$. The property of convexity in the case of (2.33) and (2.34) is lost due to the presence of integer variables δ, which is the only difference in MIP's structure comparing to classical LP or QP problems. These problems could be solved by enumerating all possible combination of binary variables δ, and for each fixed combination solve a classical LP or QP in order to obtain the optimal value $x^\star$. Even if this seems to be a straightforward way of solving MIPs the number of combinations grows exponentially by the number of δ. Therefore, more advanced methods were developed to solve MIPs, such branch-and-bound or branch-and-cut (Ric, 2005), where the goal is to eliminate candidates, which will not contain the optimal solution. There exist several commercial solvers such Gurobi (Gurobi Optimization, 2015) or CPLEX (ILOG, 2003), which are able to solve MIP problems in an efficient way.

2.8 Nonlinear Programming

In some cases, there is no chance to formulate the optimization problem as a convex optimization problem. The sub-field of mathematical optimization dealing with such problems is so-called nonlinear optimization. Problems of such a type are solved by using nonlinear programming. The general formulation of nonlinear programming is presented in (2.35).

$$\min_{x} \; f(x) \tag{2.35a}$$

$$\text{s.t. } g(x) \leq 0, \tag{2.35b}$$

$$h(x) = 0, \tag{2.35c}$$

where f, h and g are nonlinear functions and potentially are non-convex. This could be solved by using the KKT conditions but in the case of non-convex optimization problems even the solution of the KKT conditions are only necessary conditions for the existence of optimum. This means that we can found the solution, which fulfills all the conditions, but the solution might be only a local optimum. This problem could be solved by so-called sequential quadratic programming. This method solves the QP approximation instead of (2.35) iteratively. After in each iteration, the solution is improved. By using the second order Taylor's expansion the function $f(x)$ could be locally approximated by

$$f(x) \approx f(x^s) + \nabla f(x^s)(x - x^s) + 1/2(x - x^s)^\top \nabla^2 f(x^s)(x - x^s). \tag{2.36}$$

Since the goal is to solve (2.35) by QP iteratively also functions $g(x)$ and $h(x)$ must be linear. In order to obtain linear constraints, the first order Taylor's expansion is used around the given linearization point.

$$g(x) \approx g(x^s) + \nabla g(x^s)(x - x^s) \tag{2.37a}$$
$$h(x) \approx h(x^s) + \nabla h(x^s)(x - x^s) \tag{2.37b}$$

Thus QP, which is the local approximation of (2.35) at the point x^s is of the following form

$$\min_x \; f(x^s) + \nabla f(x^s)(x - x^s) + \frac{1}{2}(x - x^s)^\top \nabla^2 f(x^s)(x - x^s) \tag{2.38a}$$
$$\text{s.t. } g(x^s) + \nabla g(x^s)(x - x^s) \leq 0, \tag{2.38b}$$
$$h(x^s) + \nabla h(x^s)(x - x^s) = 0. \tag{2.38c}$$

Since the objective function in (2.38) is quadratic and also the constraints are linear, it could be solved as QP. In each iteration, the quadratic approximation of (2.35) is created in point $x^\star$, where $x^\star$ is the solution of (2.38). The procedure of iterations is terminated if the solution of two consecutive steps fulfills the stopping criteria. This is in general defined as p norm of the differences between the solution, $||x^k - x^{k-1}||_p \leq \epsilon$. Variable ϵ represent the value of toleration. In some cases, it may happen that the hessian $\nabla^2 f(x^s)$ of the local quadratic approximation will no longer be positive definite. By adding the scaled identity matrix to the hessian we

can achieve that the approximated function will be always convex.

$$\min_x \ f(x^s) + \nabla f(x^s)(x - x^s) + \frac{1}{2}(x - x^s)^\top(\gamma I + \nabla^2 f(x^s))(x - x^s) \tag{2.39a}$$

$$\text{s.t. } g(x^s) + \nabla g(x^s)(x - x^s) \leq 0, \tag{2.39b}$$

$$h(x^s) + \nabla h(x^s)(x - x^s) = 0, \tag{2.39c}$$

where variable γ represents the scaling parameter and I stands for the identity matrix. By this scaling even if the approximation of (2.35) would lead to non-convex (concave) function, by adjusting the hessian at least a linear function approximation could be achieved.

2.9 Multiparametric Programming

In this section, we will discuss the idea of multiparametric programming (Gal and Nedoma, 1972; Willner, 1967), which is based on obtaining an explicit representation of MPC feedback law for a full scale of initial conditions (parameters). This is achieved by solving a given optimization problem off-line for all possible initial conditions. The resulting analytical solution will be encoded as a piecewise affine (PWA) function defined over the polytopic partition, composed of a series of critical regions (Borrelli, 2003).

The main task of multiparametric programming is to construct critical regions by using necessary and sufficient conditions of optimality. Over each region, the control law and the objective function is defined as a function of actual states. Let us consider the following multiparametric program:

$$J^\star(x) = \min_U \ J(U, x) \tag{2.40a}$$

$$\text{s.t. } GU \leq w + Ex, \tag{2.40b}$$

with $J^\star(x)$ as an optimal value of an objective function $J(U, x)$, U is a vector of optimization variables, x is a vector of parameters. Further, let us denote matrices $G \in \mathbb{R}^{m \times s}$, $w \in \mathbb{R}^m$, $E \in \mathbb{R}^{m \times n}$, where m represents the number of rows in constraints, s dimension of optimization variables, and n dimension of parameters.

During obtaining a critical region the constraints are divided into two parts. In each region different constraints are active (A) or inactive (NA), respectively. Let $I := \{1, \ldots, m\}$ be a set of m constraints in (2.40a), which can be split into active

constraints A and inactive constraints NA. A and NA are sub-sets of the set I, where $A \cup NA = I$. Therefor let us denote $G_A, w_A, E_A, G_{NA}, w_{NA}$ and E_{NA}

The constraint is said to be active when

$$G_j U = w_j + E_j x \tag{2.41}$$

and inactive when

$$G_j U < w_j + E_j x \tag{2.42}$$

The critical region is defined by

$$\mathcal{R}_A := \{x \in \mathcal{K} \mid I(x) = A\} \tag{2.43}$$

where $\mathcal{K}$ is part of parameters space in $\mathbb{R}^n$ in which we are interested in exploring, and $I(x) = A$ is interpreted as imposing as active constraints indexed by the index set A.

Depending on the type of the objective function $J(U, x)$, we distinguish between two types of multiparametric problems. If $J(U, x)$ is a linear function, then the problem in (2.40a) is called a multiparametric linear program (mp-LP). If $J(U, x)$ is a convex quadratic function, then (2.40a) is referred to as a multiparametric quadratic program (mp-QP). The construction of the explicit solutions to such problems is described next.

Multiparametric Linear Programming

Consider a multiparametric programming problem (2.40a) with a linear objective function of the form $c^\top U$, i.e.,

$$J^\star(x) = \min_U \; c^\top U \tag{2.44a}$$

$$\text{s.t. } GU \leq w + Ex. \tag{2.44b}$$

Our goal is to construct the optimizer $U^\star(x) = \arg\min_U \; c^\top U$ and the value function $J^\star(x)$ both as functions of the parameters x. These functions will be defined over a polytopic partition $\cup_i \mathcal{R}_i, i = 1, \ldots, k$ where $\mathcal{R}_i$ are polyhedral critical regions and k denotes the total number of critical regions.

Since the above mention problem is convex, the Karush-Kuhn-Tucker (KKT) conditions (Boyd and Vandenberghe, 2004) are necessary and sufficient conditions

for $U^\star$ to be a global minimizer of (2.44). The Lagrangian of the problem (2.44) is defined as

$$\mathcal{L}(U, x, \lambda) = c^\top U + \lambda^\top (GU - w - Ex). \tag{2.45}$$

Then the four KKT conditions are given by

$$\frac{\partial \mathcal{L}(U, x, \lambda)}{\partial U} = c + G^\top \lambda = 0, \tag{2.46a}$$

$$GU \leq w + Ex, \tag{2.46b}$$

$$0 \leq \lambda, \tag{2.46c}$$

$$\lambda_i (G_i U - w_i - E_i x) = 0. \tag{2.46d}$$

Here, (2.46a) is the stationarity condition, (2.46b) is the primal feasibility condition, (2.46c) represents dual feasibility, and (2.46d) is the complementary slackness condition. Furthermore, let us split 2.46b into active

$$G_A U - E_A x = w_A \tag{2.47}$$

and inactive

$$G_{NA} U - E_{NA} x < w_{NA} \tag{2.48}$$

parts. The optimizer $U^\star(x)$ can be derived from (2.46b) as

$$U^\star(x) = G_A^{-1}(E_A x + w_A) \tag{2.49}$$

Subsequently by substituting (2.49) into (2.44a) we obtain optimal value function as a function of parameter in a form of

$$J^\star(x) = c^\top G_A^{-1}(E_A x + w_A) \tag{2.50}$$

Finally, the critical region is defined as a union of all inequalities in (2.46). Since (2.46c) do not contribute any restriction to parameter we have that critical region can be obtained from (2.48) and (2.49) as:

$$\mathcal{R}_A = \{x \mid (G_{NA} G_A^{-1} E_A - E_{NA})x < w_{NA} - G_{NA} G_A^{-1} w_{NA}\}. \tag{2.51}$$

Clearly, the set in (2.51) describes an open polyhedron in the parametric space. For practical purposes, one can consider the closure of a critical region, which is obtained from (2.51) by replacing strict inequalities by non-strict ones.

Multiparametric Quadratic Programming

An another case of multiparametric programming is when the objective function is quadratic subject to linear constraints:

$$J(x) = \min_{U} \ \frac{1}{2}U^\top HU + x^\top FU \tag{2.52a}$$

$$\text{s.t. } GU \leq w + Ex, \tag{2.52b}$$

with $H = H^\top \succ 0$. Our goal is similarly as in multiparametric linear programming to construct an optimizer $U^\star = \arg\min_U \ \frac{1}{2}U^\top HU + x^\top FU$, value function $J^\star(x)$ and subsequently critical regions $\mathcal{R}_i$.

Hence the problem defined in (2.52) is convex, the KKT conditions provide necessary and sufficient conditions for $U^\star$ to be a global minimizer. The Lagrangian equation of problem defined in (2.52) is expressed by:

$$\mathcal{L}(U, x, \lambda) = \frac{1}{2}U^\top HU + x^\top FU + \lambda^\top(GU - w - Ex), \tag{2.53}$$

for which the KKT conditions are given by

$$\frac{\partial \mathcal{L}(U, x, \lambda)}{\partial U} = HU + F^\top x + G^\top \lambda = 0, \tag{2.54a}$$

$$GU \leq w + Ex, \tag{2.54b}$$

$$0 \leq \lambda, \tag{2.54c}$$

$$\lambda_i(G_i U - w_i - E_i x) = 0, \tag{2.54d}$$

where (2.54a) represents a stationarity condition, (2.54b) is the primal feasibility, (2.54c) is the dual feasibility and (2.54d) is the complementary slackness condition. Similarly as in (2.46) the equation (2.54b) is divided into the active

$$G_A U - E_A x = w_A, \tag{2.55}$$

and the inactive

$$G_{NA} U - E_{NA} x < w_{NA}, \tag{2.56}$$

constraints. The optimizer as a function of parameter $U^\star(x)$ can be derived from (2.54a) as

$$U^\star(x) = -H^{-1}(F^\top x + G^\top \lambda). \tag{2.57}$$

Since λ is the part of (2.57) it must be expressed as a function of the parameter to obtain optimizer only as a function of the parameter. The dual variable as a function of parameter is obtained by substituting (2.57) into (2.55):

$$\lambda = \underbrace{(-G_A H^{-1} G^\top)^{-1}(-G_A H^{-1} F^\top - E_A)}_{\gamma} x(t) \underbrace{-(G_A H^{-1} G^\top)^{-1} w_A}_{\delta} \tag{2.58}$$

afterward λ is then substituted into (2.57) to yield

$$U^\star(x) = -H^{-1}(F^\top x + G^\top(\gamma x + \delta)) = \underbrace{-H^{-1}(F^\top + G^\top)}_{\alpha} x \underbrace{-H^{-1} G^\top \delta}_{\beta}. \tag{2.59}$$

Subsequently by substituing (2.59) into (2.52a) the optimal expression of value function is obtained:

$$J^\star(x) = \frac{1}{2}(\alpha x + \beta)^\top H(\alpha x + \beta) + x(t)^\top F(\alpha x + \beta). \tag{2.60}$$

Finally, we need to determine the critical region where (2.59) and (2.60) are valid. The region is calculated as an intersection of primal and dual feasible sets, where the region of primal feasibility is defined as

$$P_p = \{x \in \mathcal{K} \mid (G_{NA}\alpha - E_{NA})x < w_{NA} - G_{NA} b\}, \tag{2.61}$$

and the region of dual feasibility by

$$P_d = \{x \in \mathcal{K} \mid \gamma x(t) \leq \beta\}. \tag{2.62}$$

The final critical region is then given as the intersection of P_p and P_d, i.e.,

$$\mathcal{R} = \{x \mid x \in P_p \cap P_d\}. \tag{2.63}$$

since P_p and also P_d are polytopic sets, the result of the intersection between them $\mathcal{R}$ is also polytopic set.

2.10 Properties of Multiparametric Programming

We obtain three expressions by solving the given optimization problem by parametric programming. The first is the feasible set $\mathcal{X}$, which is a domain of parameter x, where exists a feasible solution. Furthermore, the feasible set is a union of critical regions. Over these regions, the optimal solution and the objective functions are

defined The optimal solution $z^\star(x)$, which is a function linear function of parameter x is a PWA function defined over critical regions (2.64).

$$z^\star(x) = F_i x + g_i, \text{ if } x \in \mathcal{R}_i, \tag{2.64}$$

where $\mathcal{R}_i$ is the i-th critical region, while F_i and g_i are the local gain and affine part. The objective function is also defined over critical regions $J^\star(x)$ as function of parameter x. The properties of the objective function depends on what kind of multiparametric program was solved. In case mp-LP and mp-MILP the objective function is a PWA function and is of the following form:

$$J^\star(x) = c_i^\top x + d_i, \text{ if } x \in \mathcal{R}_i, \tag{2.65}$$

while in case of mp-QP and mp-MIQP it is a PWQ given as

$$J^\star(x) = x^\top H x + c_i^\top x + d_i, \text{ if } x \in \mathcal{R}_i. \tag{2.66}$$

In case mp-LP (2.65) or mp-QP (2.66) functions are continuous and convex. In case of mp-MILP (2.65) or mp-MIQP (2.66) the functions are possibly discontinuous and non-convex.

2.11 Summary

This chapter's aim was to present the fundamentals of convex optimization. The reasons why we are interested in convex optimization problems are that, if there exists a solution for a convex optimization problem then this solution represents the global optimum. However in the case of non-convex optimization, the result of the optimization problem could end in one of its local optima. The main convex optimization classes were introduced. In the next chapters, the theoretical fundamentals presented in this chapter will be used. The described optimization classes, such LP, QP, MILP, and MIQP are mainly used in MPC formulations. Furthermore, the construction of EMPC and its property was also discussed. The algorithms to extract the value of the optimal solution for the given initial parameter from the parametric solution will be discussed in the next chapter.

Chapter 3

Model Predictive Control

Due to its ability to handle system's limitations, MPC is a very popular control strategy nowadays that had to go through a long evolution. The beginning could be dated back to the work of Kalman (1960). In the '60s of the last century, he was working on a controller with an aim to compute an optimal input by quadratic penalization on a linear model based infinite horizon of states and inputs. Today, this method is known as linear quadratic regulation (LQR) (Kwakernaak and Sivan, 1972a). Between industrial technologists, LQR was not really popular. Main reasons were the absence of system's constraints in its formulation. Also nonlinear dynamic and large time delays caused problems as well. In the early '80s, so-called Dynamic Matrix Control (DMC) (Freedman and Bhatia, 1985) started to be used in the chemical and petrochemical fields. There was a huge demand for this approach because of its possibility to compute optimal control actions wherewith considerable amount of money could be saved. In the petrochemical field where products were measured in kilo or megatons, only a small improvement could lead to giant savings. The main reason for using this kind of control approach in chemical and petrochemical field was the fact that even if the strategy was computationally demanding, the system dynamics was slow enough. MPC computed the optimal input based on optimization that had to be performed in less time than the sampling time of the system. At that time, the stability was not theoretically determined (Mayne et al., 2000). The first version of MPC did not stabilize the process automatically. However, a long prediction horizon and stable systems were chosen to improve this

kind of control approach. Generally, the optimization problem is predefined and only the actual state values are necessary to obtain. Since MPC is a state based control approach it needs information of all states. In case that not all states are measurable, it requires an observer or estimator. In the beginning of this century a new approach was introduced, so-called Explicit Model Predictive Control (Bemporad et al., 2002). This method brought MPC to control fast mechanical and electrical systems. With all the benefits of MPC control technique, the Explicit MPC could be implemented in weaker hardware since the optimization problem was solved by using parametric programming in the offline phase, while in the online phase only function evaluation was required.

3.1 Model Predictive Control Formulation

Model predictive control problems can be formulated as optimization problems that consist of the following components:

- the objective function,
- constraints in form of equalities,
- constraints in form of inequalities,
- initial conditions.

The objective function usually penalizes the deviation of the plant's states from prescribed references or accounts for minimization of consumed energy (Maciejowski, 2002). Typically the mathematical model enters the optimization problem as equality constraints. Limits on predicted states, outputs, and inputs are converted to the inequality constraints. The initial condition supplies the optimization problem with the state measurements. If all constraints are linear, then they form a convex set. Moreover, if the objective function is also convex, the MPC problem can be solved as a convex optimization problem. If the objective function is quadratic, then the problem can be solved as a quadratic program. This is the case when squared 2-norms are used to penalize various quantities in the objective function. If the function employs 1- or ∞-norms, the problem can be solved as a linear programming problem. The purpose of this chapter is to provide an overview of mathematical formulations of MPC problems typically used. However, first we

review main advantages as well as limitations of MPC when compared to classical control approaches.

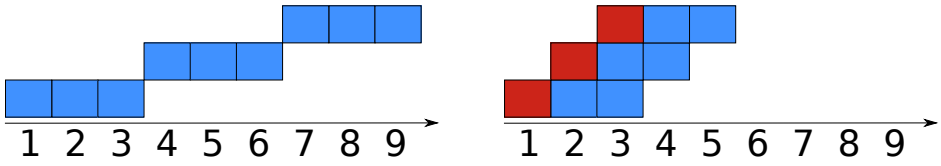
3.2 Comparison of MPC to Other Approaches

The model predictive control approach is used when systems constraints must be considering and the performance must be as good as possible. The classic control designs compared to MPC have several weaknesses. Widely used PID controllers have easy structure and few parameters which could be changed by operators in order to improve the controller's performance. In the case of bigger systems when multiple inputs and multiple outputs (MIMO) are considered then decoupling is us, which is not trivial. Moreover, PID controllers do not provide optimal control inputs. The input limitation can be saturated mechanically, but this can cause in some cases destabilization of systems. More advanced control approach is Linear Quadratic Regulator (LQR) which provides optimality (Kwakernaak and Sivan (1972b). LQR solves an optimization problem of minimization the state and inputs over infinity prediction horizon subject to a linear constraint which is the linear model of the system. LQR cannot handle other constraints, but it is applicable to MIMO systems without using any decoupling. However, MPC can handle MIMO systems and can incorporate several constraints in the form of equalities and inequalities. Although MPC uses only finite prediction horizon the benefits come from constraints satisfaction which makes it interesting (Maciejowski, 2002; Mayne et al., 2000).

There are two possible ways how to implement the result of the optimization which arises from MPC problem. The result is a sequence of optimal control inputs. The length of the sequence depends on the size of the control prediction horizon.

3.3 Open-Loop Implementation

If the whole sequence is applied to the system, the next measurement/estimation is done only after applying the last part of the computed control sequence. Then the optimization problem is solved again. At first, it looks like a good way to implement MPC, but if something unexpected happens between two measurements then controller does not know how to handle it since it has not any information about this event. The above mentioned approach can easily be demonstrated by



(a) Open loop (b) Closed loop

Figure 3.1: Implementation types

driving a car. We require from the cars leader staying on the road and does not violate any highway code. Now the prediction horizon can be specified as the distance ahead of a car which can be seen. The driver could open eyes just for the moment, evaluate the situation and immediately close them. He could open eyes again after passing the given part of the road. If an unexpected object appears, like another car or child, while drivers eyes are closed collision cannot be prevented. This is a so-called open-loop implementation of the calculated control sequence.

3.4 Closed-Loop Implementation

In the case of the closed loop approach, the whole calculated control sequence is not implemented to the process, but only the first element. After a new measurement is performed, the optimization is repeated.

3.5 MPC as an Optimization Problem

In MPC the information about the model of the controlled system is used to predict its responses to a particular choice of the control inputs (Jerez et al., 2010). Then optimization is employed to find the best control inputs that provide an optimal operation of the plant. In the sequel, we will assume that the controlled system can sufficiently well be modeled by a linear time-invariant system in the discrete time domain of the form

$$x^+ = Ax + Bu, \tag{3.1}$$

where $x \in \mathbb{R}^n$ is the vector of states, $u \in \mathbb{R}^m$ is the vector of control inputs, and x^+ denotes the successor state at the next sampling instant. The system in (3.1)

is assumed to operate subject to constraints

$$x \in \mathcal{X},\ u \in \mathcal{U}, \tag{3.2}$$

where $\mathcal{X} \subseteq \mathbb{R}^n$ and $\mathcal{U} \subseteq \mathbb{R}^m$ are polyhedra. More specifically, we assume

$$\mathcal{X} = \{x \mid Hx \leq K\}, \tag{3.3a}$$

$$\mathcal{U} = \{u \mid Lu \leq M\}, \tag{3.3b}$$

where (3.3a) and (3.3b) are the half-space representations of constraints on state and input variables.

Then the MPC optimization problem can be stated as

$$\min_{u_0,\ldots,u_{N-1}} \ \ell_{\mathrm{N}}(x_N) + \sum_{k=0}^{N-1} \ell(x_k, u_k) \tag{3.4a}$$

$$\text{s.t. } x_{k+1} = Ax_k + Bu_k, \tag{3.4b}$$

$$x_0 = x(t), \tag{3.4c}$$

$$x_k \in \mathcal{X}, \tag{3.4d}$$

$$u_k \in \mathcal{U}. \tag{3.4e}$$

$$x_N \in \mathcal{T}. \tag{3.4f}$$

where $\ell(.,.)$ represents the stage cost while ℓ_{N} is a terminal penalty with terminal constraint $\mathcal{T}$. In the sequel we will consider the specific case when the objective function is a quadratic function. For the sake of simplicity of the notation, we will not consider terminal constraints and terminal cost. In this case, the MPC is of the following form:

$$\min_{u_0,\ldots,u_{N-1}} \ \sum_{k=0}^{N-1} x_k^\top Q x_k + u_k^\top R u_k \tag{3.5a}$$

$$\text{s.t. } x_{k+1} = Ax_k + Bu_k, \tag{3.5b}$$

$$x_0 = x(t), \tag{3.5c}$$

$$x_k \in \mathcal{X}, \tag{3.5d}$$

$$u_k \in \mathcal{U}. \tag{3.5e}$$

where x_k and u_k represent the state and input variables at the k-th step of the prediction window, respectively. The predictions are obtained using the LTI prediction model defined in (3.5b), and initialized by the current state measurements

in (3.5c). Each predicted state and input variables has to fulfill restrictions due to constraints (3.5d) and (3.5e). Note that constraints (3.5b), (3.5d) and (3.5e) need to be imposed for each $k = 0, \ldots, N-1$. The purpose of the objective function in (3.5a) is to penalize the deviation of predicted states and inputs from the origin. Weighting matrices $Q = Q^\top \succeq 0$ and $R = R^\top \succ 0$ are tuning parameters to adjust the controllers performance.

The optimal control problem defined in (3.5) can be transformed into a quadratic program (Rossiter and Kouvaritakis, 2004; Scokaert and Rawlings, 1998). New variables are introduced in order to generalize the MPC problem.

$$X = \begin{bmatrix} x_0 \\ \vdots \\ x_{N-1} \end{bmatrix}, \quad \tilde{Q} = \begin{bmatrix} Q & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & Q \end{bmatrix}, \quad \tilde{R} = \begin{bmatrix} R & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & R \end{bmatrix} \tag{3.6a}$$

$$U = \begin{bmatrix} u_0 \\ \vdots \\ u_{N-1} \end{bmatrix}, \quad \tilde{H} = \begin{bmatrix} H & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & H \end{bmatrix}, \quad \tilde{L} = \begin{bmatrix} L & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & L \end{bmatrix} \tag{3.6b}$$

$$\tilde{M} = \begin{bmatrix} M \\ \vdots \\ M \end{bmatrix}, \quad \tilde{K} = \begin{bmatrix} K \\ \vdots \\ K \end{bmatrix}, \quad \tilde{E} = \begin{bmatrix} I \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{3.6c}$$

$$\tag{3.6d}$$

where matrices X and U represent the sequence of state and input variables, while matrices $\tilde{Q}$, $\tilde{R}$, $\tilde{H}$ and $\tilde{L}$ stand for the Kronecker product between the identity matrix and the given weighting matrices or left-hand side constraint matrices. Vectors $\tilde{M}$ and $\tilde{K}$ are the augmented vector to constrain the predicted state and input variables.

$$\tilde{A} = \begin{bmatrix} 0 & \cdots & \cdots & \cdots & 0 \\ A & 0 & \ddots & \ddots & \vdots \\ 0 & A & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & A & 0 \end{bmatrix}, \quad \tilde{B} = \begin{bmatrix} 0 & \cdots & \cdots & \cdots & 0 \\ B & 0 & \ddots & \ddots & \vdots \\ 0 & B & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & B & 0 \end{bmatrix} \tag{3.7a}$$

$\tilde{A}$ and $\tilde{B}$ represent the evolution of the dynamics of the system. This formulation is so called sparse formulation where the nonzero elements of the matrices are located on the diagonal.

$$\min\ X^\top \tilde{Q} X + U^\top \tilde{R} U \tag{3.8a}$$
$$\text{s.t. } X = \tilde{A}X + \tilde{B}U + \tilde{E}x(t), \tag{3.8b}$$
$$\tilde{H}X \leq \tilde{K}, \tag{3.8c}$$
$$\tilde{L}U \leq \tilde{M}. \tag{3.8d}$$

By introducing a new optimization variable $Z = [X^\top \quad U^\top]^\top$, optimization problem in (3.8) could be transformed into the standard QP form:

$$\min\ Z^\top \begin{bmatrix} \tilde{Q} & 0 \\ 0 & \tilde{R} \end{bmatrix} Z \tag{3.9a}$$
$$\text{s.t. } [I - \tilde{A} \quad -B]Z = \tilde{E}x(t), \tag{3.9b}$$
$$\begin{bmatrix} \tilde{H} & 0 \\ 0 & \tilde{L} \end{bmatrix} Z \leq \begin{bmatrix} \tilde{K} \\ \tilde{M} \end{bmatrix}. \tag{3.9c}$$

The formulation in (3.9) is also called a sparse MPC formulation with equality constraints. It is possible to eliminate the equality constraints by exploiting the LTI prediction equation in (3.1). Specifically, let us rewrite (3.1) into $x_{k+1} = Ax_k + Bu_k$. Then the first three predictions of the states are given by

$$x_1 = Ax_0 + Bu_0 \tag{3.10a}$$
$$x_2 = Ax_1 + Bu_1 = A(Ax_0 + Bu_0) + Bu_1 = A^2x_0 + ABu_0 + Bu_1 \tag{3.10b}$$
$$x_3 = Ax_2 + Bu_2 = A(A^2x_0 + ABu_0 + Bu_1) + Bu_2 = A^3x_0 + A^2Bu_0 + ABu_1 + Bu_2 \tag{3.10c}$$

As can be observed from (3.10), each predicted state is only a function of the initial condition x_0 and of the vector of predicted control inputs. We can generalize this procedure to give a compact representation of the k-th predicted state as

$$x_k = A^k x_0 + \sum_{i=0}^{k-1} A^{k-i-1} B u_i. \tag{3.11}$$

By defining $X = [x_0^\top, x_1^\top, \ldots, x_{N-1}^\top]^\top$ and $U = [u_0^\top, \ldots, u_{N-1}^\top]^\top$, we can compactly

rewrite the prediction equation as

$$X = \tilde{A}x(t) + \tilde{B}U, \tag{3.12}$$

which, more specifically, takes the following form:

$$\underbrace{\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{N-1} \end{bmatrix}}_{X} = \underbrace{\begin{bmatrix} I \\ A \\ A^2 \\ \vdots \\ A^{N-1} \end{bmatrix}}_{\tilde{A}} x(t) + \underbrace{\begin{bmatrix} 0 & 0 & \cdots & \cdots & 0 \\ B & 0 & \cdots & \cdots & 0 \\ AB & B & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ A^{N-2}B & A^{N-3}B & \cdots & B & 0 \end{bmatrix}}_{\tilde{B}} \underbrace{\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{N-1} \end{bmatrix}}_{U}. \tag{3.13a}$$

By exploiting (3.13) the MPC problem could be formulated as

$$\min\ (\tilde{A}x(t)) + \tilde{B}U)^\top \tilde{Q}(\tilde{A}x(t) + \tilde{B}U) + U^\top \tilde{R}U \tag{3.14a}$$

$$\text{s.t. } \tilde{H}(\tilde{A}x(t) + \tilde{B}U) \leq \tilde{K}, \tag{3.14b}$$

$$\tilde{L}U \leq \tilde{M}, \tag{3.14c}$$

which, after expansion, leads to

$$\min\ U^\top(\tilde{B}^\top \tilde{Q}\tilde{B} + \tilde{R})U + 2x(t)^\top \tilde{A}^\top \tilde{Q}\tilde{B}U \tag{3.15a}$$

$$\text{s.t. } \tilde{H}\tilde{B}U \leq (\tilde{K} - \tilde{H}\tilde{A}x(t)), \tag{3.15b}$$

$$\tilde{L}U \leq \tilde{M}. \tag{3.15c}$$

By introducing new variables

$$H = \tilde{B}^\top \tilde{Q}\tilde{B} + \tilde{R}, \tag{3.16a}$$

$$F = 2\tilde{A}^\top \tilde{Q}\tilde{B}, \tag{3.16b}$$

$$G = \begin{bmatrix} \tilde{H}\tilde{B} \\ \tilde{L} \end{bmatrix}, \quad w = \begin{bmatrix} \tilde{K} \\ \tilde{M} \end{bmatrix}, \quad E = \begin{bmatrix} \tilde{H}\tilde{A} \\ 0 \end{bmatrix} \tag{3.16c}$$

we obtain the final form of the so-called dense matrix formulation of MPC:

$$\min\ U^\top HU + x(t)^\top FU \tag{3.17a}$$

$$\text{s.t. } GU \leq w + Ex(t), \tag{3.17b}$$

where variable U represents the vector optimized variables and $x(t)$ the parameter. This form of the MPC problem is pre-constructed and the only unknown information is the value of parameter $x(t)$, which is usually obtained by measurement.

3.5.1 Tracking

The objective function in (3.4a) forces the control inputs to converge the system towards the origin, while all the constraints are respected. However often in practice, a non-zero reference is required. Such a formulation is generally called a tracking MPC setup. The goal is to track state reference x_{ref}. In this case, the MPC is of the form

$$\min \sum_{k=0}^{N-1} (||Q_x(x_k - x_{\text{ref}})||_p + ||Q_u u_k||_p) \tag{3.18a}$$

$$\text{s.t. } x_{k+1} = Ax_t + Bu_k, \tag{3.18b}$$

$$x_0 = x(t), \tag{3.18c}$$

$$x_k \in \mathcal{X}, \tag{3.18d}$$

$$u_k \in \mathcal{U}. \tag{3.18e}$$

Alternatively, this problem can be reformulated in order to ensure that the outputs of the system will track certain output reference y_{ref}. Here, we assume that the output equation is linear, i.e.,

$$y = Cx + Du, \tag{3.19}$$

with $y \in \mathbb{R}^q$ denoting the vector of system's outputs, which can be constrained by $y \in \mathcal{Y}$ where $\mathcal{Y} \subseteq \mathbb{R}^q$ is a polyhedron. The corresponding MPC problem then becomes

$$\min \sum_{k=0}^{N-1} (||Q_y(y_k - y_{\text{ref}})||_p + ||Q_u u_k||_p) \tag{3.20a}$$

$$\text{s.t. } x_{k+1} = Ax_k + Bu_k, \tag{3.20b}$$

$$y_k = Cx_k + Du_k, \tag{3.20c}$$

$$x_0 = x(t), \tag{3.20d}$$

$$x_k \in \mathcal{X}, \tag{3.20e}$$

$$u_k \in \mathcal{U}, \tag{3.20f}$$

$$y_k \in \mathcal{Y}. \tag{3.20g}$$

Note that this formulation is not able to eliminate the steady-state offset. This fact is caused by the minimization of inputs in the objective function. The objective

function tries to push inputs to zero. By this action states/outputs never could reach reference, just getting as close as it is possible. If offset-free tracking is required the optimization problem must be reformulated. The first method is called steady-state approach. Here, we minimize the difference between the predicted inputs and the corresponding steady-state values. The steady-state input u_{ss}, along with the corresponding steady-state value of the state vector can be calculated from

$$x_{ss} = Ax_{ss} + Bu_{ss}, \tag{3.21a}$$

$$y_{\text{ref}} = Cx_{ss} + D_{ss}. \tag{3.21b}$$

To solve for x_{ss} and u_{ss} from (3.21), we can rewrite the two equations into the compact matrix form

$$\begin{bmatrix} I - A & -B \\ C & D \end{bmatrix} \begin{bmatrix} x_{ss} \\ u_{ss} \end{bmatrix} = \begin{bmatrix} 0 \\ y_{\text{ref}} \end{bmatrix}, \tag{3.22}$$

from which x_{ss} and y_{ss} can be computed relatively easily, while we consider that the matrix in the left-hand-side is invertible.

Then the offset-free tracking MPC problem is given by

$$\min \sum_{k=0}^{N-1} (||Q_y(y_k - y_{\text{ref}})||_p + ||Q_u(u_k - u_{ss})||_p), \tag{3.23a}$$

$$\text{s.t. } x_{k+1} = Ax_k + Bu_k, \tag{3.23b}$$

$$y_k = Cx_k + Du_k, \tag{3.23c}$$

$$x_0 = x(t), \tag{3.23d}$$

$$x_k \in \mathcal{X}, \tag{3.23e}$$

$$u_k \in \mathcal{U}, \tag{3.23f}$$

$$y_k \in \mathcal{Y}. \tag{3.23g}$$

Another possibility to eliminate the tracking offset is to employ the so-called Δu formulation. Here, instead of minimizing the control inputs directly, we optimize over their increments, i.e.,

$$\Delta u_k = u_k - u_{k-1}. \tag{3.24}$$

The objective function then becomes

$$\min \sum_{k=0}^{N-1} (||Q_y(y_k - y_{\text{ref}})||_p + ||Q_{\Delta u}\Delta u_k||_p). \tag{3.25}$$

Moreover, the constraints need to be modified such that they are a function of the increments. This can be achieved by defining an auxiliary dynamical system

$$\underbrace{\begin{bmatrix} x_{k+1} \\ u_k \end{bmatrix}}_{\tilde{x_{k+1}}} = \underbrace{\begin{bmatrix} A & B \\ 0 & I \end{bmatrix}}_{\tilde{A}} \underbrace{\begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix}}_{\tilde{x_k}} + \underbrace{\begin{bmatrix} B \\ I \end{bmatrix}}_{\tilde{B}} \underbrace{\Delta u_k}_{\tilde{u_k}}, \tag{3.26a}$$

$$y_k = \underbrace{\begin{bmatrix} C & D \end{bmatrix}}_{\tilde{C}} \begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix} + \underbrace{D}_{\tilde{D}} \Delta u_k. \tag{3.26b}$$

Then the complete MPC tracking problem becomes

$$\min \sum_{k=0}^{N-1} (||Q_y(y_k - y_{\text{ref}})||_p + ||Q_{\Delta u} \Delta u_k||_p), \tag{3.27a}$$

$$\text{s.t. } \tilde{x}_{k+1} = \tilde{A}\tilde{x_k} + \tilde{B}\Delta u_k, \tag{3.27b}$$

$$y_k = \tilde{C}\tilde{x_k} + \tilde{D}\Delta u_k, \tag{3.27c}$$

$$\tilde{x_t} = \begin{bmatrix} x(t) \\ u(t-1) \end{bmatrix}, \tag{3.27d}$$

$$\tilde{H}_{\Delta u}\tilde{x}_k \leq \tilde{K_{\Delta u}} \quad \tilde{H}_y\tilde{y_k} \leq \tilde{K}_y, \tag{3.27e}$$

where

$$x_t \in \mathcal{X} = H_x x_{t+k} \leq K_x, \tag{3.28a}$$

$$u_t \in \mathcal{U} = H_u u_{t+k} \leq K_u, \tag{3.28b}$$

$$\underbrace{\begin{bmatrix} H_x & 0 \\ 0 & H_u \end{bmatrix}}_{\tilde{H}_{\Delta u}} \tilde{x_{t+k}} \leq \underbrace{\begin{bmatrix} K_x \\ K_u \end{bmatrix}}_{\tilde{K}_{\Delta u}}. \tag{3.28c}$$

Practical applications usually have slew rate limits, which means they are not able to make big changes in control action. For example, the valve cannot be fully open at one time and the next step totally closed. This kind of approach can radically decrease the lifetime of the equipment. Therefore Δu formulation can be changed by adding one additional constraint, which bounds the changes between control actions:

$$\Delta u_{min} \leq \Delta u \leq \Delta u_{max}, \tag{3.29}$$

where $\Delta u_{\min}$ is a minimal decrease in the value of the control variable, subsequently $\Delta u_{\max}$ represents the maximal increase in the value of the control variable in one control step.

3.5.2 Move Blocking

The complexity of MPC problems grows proportionally with the value of the prediction horizon. This complexity can be decreased in two ways. One option is to decrease the value of the prediction horizon. However, doing so has a negative impact on the performance of the MPC strategy. An alternative way is to split the prediction horizon into two parts (Maciejowski, 2002). The first part, called the control horizon (N_c) specifies the number of optimized control inputs. Then in the remainder of the prediction window, from steps N_c to N, one assumes that the control inputs are fixed to the last computed value. This allows to reduce the number of optimization variables. Needless to say, if N_c is too short compared to N, one can expect such a controller to exhibit a large amount of suboptimality. The corresponding mathematic formulation of the move blocking MPC formulation is given by

$$\min \sum_{k=0}^{N_c-1} \left(\|Qx_k\|_p + \|Q_u u_k\|_p\right) + \sum_{k=Nc}^{N-1} \|Qx_k\|_p \tag{3.30a}$$

$$\text{s.t. } x_{k+1} = Ax_k + Bu_k, \tag{3.30b}$$

$$x_0 = x(t), \tag{3.30c}$$

$$x_k \in \mathcal{X}, \tag{3.30d}$$

$$u_k \in \mathcal{U}, \tag{3.30e}$$

$$u_{Nc+k} = u_{Nc-1}, \tag{3.30f}$$

where the last constraint represents the blocking condition.

3.5.3 Soft Constraints

Sometimes MPC formulation can lead to infeasibility. This is usually caused by so-called hard constraints. These kind of constraints must be satisfied at all time. In the industry, it can represent safety boundary for pressure or maximum opening of valve cant be more than 100 %. There exist another type of constraints which can cause that the MPC problem will be always feasible. This kind of approach is

used when a small violation is allowed. Hard constraints can be softened by adding non-negative slack variables. However, these slack variables need to be penalized in the objective function as to discourage MPC from violating the constraints too often. The corresponding mathematical formulation of the MPC problem is given as

$$\min \sum_{k=0}^{N-1} (\|Q_y y_k\|_p + \|Q_u u_k\|_p + Q_{s,y} S_{y,k} + Q_{s,u} S_{u,k} \tag{3.31a}$$

$$\text{s.t. } x_{k+1} = Ax_k + Bu_k, \tag{3.31b}$$

$$y_k = Cx_k + Du_k, \tag{3.31c}$$

$$x_0 = x(t), \tag{3.31d}$$

$$u_{\min} - s_{u,k} \leq u_k \leq u_{\max} + s_{u,k}, \tag{3.31e}$$

$$y_{\min} - s_{y,k} \leq y_k \leq y_{\max} + s_{y,k}, \tag{3.31f}$$

$$s_{u,k} \geq 0, \tag{3.31g}$$

$$s_{y,k} \geq 0, \tag{3.31h}$$

where $s_{u,k}$ and $s_{y,k}$ are the slack variables which soften the constraints on the k-th predicted input and output, respectively.

3.6 Explicit Model Predictive Control

Even though several efficient solvers exist to solve the MPC problem quickly at each sampling time, the runtime of the optimization can still be prohibitive when targeting implementation of MPC to systems with fast dynamics. Moreover, these solvers are usually applicable only on expensive hardware, which is not so common in the industry. Hence, the industry requires hardware which will permanently work for long years in the unfavorable environment, like dusty, acidic or moist environment. Due to these reasons, programmable logic controllers (PLCs) are predominant in the industrial setup. This type of implementation hardware typically provides only small computational power and a limited amount of memory. To implement MPC on PLCs is therefore challenging. One way to accomplish this task is to employ explicit pre-compute the optimal solution to a given MPC optimization problem for all possible values of the initial condition (Bemporad et al., 2002). Subsequently, the pre-computed solution is stored in the simple hardware and is used to identify optimal control inputs on-line for a particular value of the initial condition. It can

be shown that for a rich class of MPC problems, specifically for those which can be formulated as linear or quadratic problems, such a pre-computed solution takes a form of a piecewise affine function. The identification of optimal control inputs thus boils down to a mere function evaluation. Such a task can be done quickly even with limited computational resources. This approach often called the explicit MPC approach (Kvasnica (2011)), is elaborated next. Specifically, we show how to obtain the analytical representation of the piecewise affine optimizer and review its properties.

The construction of the explicit controller is a complex procedure. In practical application we are interested in parameters only in a given polytopic set $\mathcal{K}$:

$$\mathcal{K} = \{x \mid Tx \leq V\}. \tag{3.32}$$

At the beginning one LP with equations (2.44a, 2.44b) is solved for $x(t) = x_0$ and $x_0 \in \mathcal{K}$. Afterward the indices of active and inactive constraints are determined. Subsequently, the objective function (2.50) and input (2.49) are calculated as functions of parameter x_0. The critical region where the above mentioned affine properties are valid is computed by (2.51). Since the explicit solution usually consists of more than one region, this procedure must be carried out repeatedly. This procedure is given in Algorithm 3.1. The algorithm does not only construct the polytopic critical regions, but also constructs the PWA control law and also the PWA/PWQ objective function defined over those critical regions. With such an information in hand, everything is ready for the implementation phase.

3.7 Point Location Algorithms

The parametric solution of the optimization problem presented in (2.40a) is in the form of PWA function defined over polytopic regions. The way of obtaining the optimal value of the optimization problem as a function of the parameter is called a point location problem. In the following section, three different point location algorithms will be presented. Namely, the sequential search, extended search via objective function and binary search tree will be discussed. Other methods include works of Nguyen et al. (2015) if the objective function is linear; Herceg et al. (2013b) and Wang et al. (2007) exploited adjacency list that is inherently associated with the polytopic partition of the PWA function. However, a common drawback of the referenced approaches is that they can only be applied to continuous feedback laws.

Data: mp-LP / mp-QP formulation per (2.44) or (2.52)
Result: Multiparametric solution
initialization: $x_0 \in \mathcal{K}$, $\mathcal{Q} = \{x_0\}$;
while $\mathcal{Q} \neq \{\}$ **do**
  $x_0 = \mathcal{Q}_1$
  $\mathcal{Q} = \mathcal{Q} \setminus \mathcal{Q}_1$
  Solve LP equations (2.44a, 2.44b) or equations (2.52a, 2.52b) for $x = x_0$
  Obtain the optimizer $U^\star$ from (2.49)/(2.59)
  Obtain the value function $J^\star$ from (2.50) /(2.60)
  Obtain the critical region $\mathcal{R}$ from (2.51)/(2.63)
  Find a point on each edge of the critical region
  Make small step in the direction of normal vector of edge to obtain points x_{new}
  if $x_{new} \in \mathcal{K} \wedge x_{new} \notin \cup_i \mathcal{R}_i$ **then**
    $\mathcal{Q} = \mathcal{Q} \cup x_{new}$
  end
end

Algorithm 3.1: mp-LP / mp-QP algorithm

3.7.1 Sequential Search

The sequential search is the easiest of the point location algorithms. Each critical region is defined in the form of linear inequalities (3.33).

$$R_i = \{x \mid H_i x \leq h_i\} \tag{3.33}$$

If the optimization problem has a solution then there is at least one critical region R_i which contains the parameter x. The goal is to find the index of the region i, where the point x is located. If the PWA control law is a continuous function the values of the optimal control action even on the edges of the critical regions are identical. Therefore, it is enough to find the first critical region, which contains the parameter x. In the worst case, the point location algorithm needs to go through all of the M critical regions. After finding the critical region there is no need to continue in searching and the point location algorithm can be terminated.

Data: Feedback laws F_i, g_i, critical regions $\mathcal{R}_i$, number of regions M, state measurement x
Result: Optimal control input

```
for i = 1, ..., M do
    if x ∈ R_i then
        return u* = F_i x + g_i
    end
end
```

Algorithm 3.2: Sequential search for continuous functions

The procedure of sequential search is described in Algorithm 3.2 where F_i represents the affine part of the control law. In general, it can be stored as a matrix or a two-dimensional array. Variable g_i stands for the affine part of the control law and it can be stored as a vector or one-dimensional array. Critical region $\mathcal{R}_i$ represents the i-th critical region. It can be represented as an abstract class as it is implemented in the Multi-Parametic Toolbox or as a combination of matrix and vector, respectively: one two-dimensional array for the left-hand side of the halfspaces and one one-dimensional array for the right-hand side of the halfspaces (3.33). Variable M represents the number of critical regions. x represents the parameter of the optimization problem. In the context of MPC it contains information about the state variables. It is a vector or one-dimensional array. The

result of the Algorithm 3.2 is the optimal control input, which is the solution of the general MPC problem defined in (3.4). This has a form of the vector type variable or one-dimensional array.

3.7.2 Extended Sequential Search

The point location algorithm defined in the previous section should be used only for a parametric solution with continuous control law. Generally, after applying some of the complexity reduction techniques on the parametric solution the final approximated control law does not need to have a property of continuity. This means on the edges/boundaries of the neighboring critical regions the values of the calculated control actions are not identical. The another way when discontinuity occurs is when the critical regions overlap. This may happen with hybrid prediction models (Camacho et al., 2009). The lack of continuity may also happen when the system has inputs with binary characteristics. In the practical application due to some noises, it can lead to mechanical stresses, which can influence the control performance and also shortens the life span of the systems actuators. The way how such a discontinuity of the PWA control law could be tackled is presented in this section. The idea of solving such a problem lies in finding the indices of all regions which contain the parameter x. Based on the set of indices the objective function is evaluated and the index with the least calculated value is used to obtain the optimal control action. The procedure of the extended sequential search is defined in Algorithm 3.3, where variables F_i, g_i, M and $\mathcal{R}_i$ has the same structure as in Algorithm 3.2. The only added variable is presented by $\mathcal{I}$, which is an extendable array like variable.

At the beginning, the set of admissible indices $\mathcal{I}$ is set to empty. After each iteration, the next critical region is checked. If the parameter x fulfills conditions $H_i x \leq h_i$ then the index i is added to the set $\mathcal{I}$. In the penultimate step, the index is chosen by defined criteria $i^\star$. The algorithm returns optimal control action, based on an affine function defined over the $i^\star$ critical region.

Based on the cardinality of set $\mathcal{I}$ 3 different scenarios may occur:

1. $|\mathcal{I}| = 0$, i.e., the set $\mathcal{I}$ is empty;
2. $|\mathcal{I}| = 1$, i.e., the set contains exactly one index;
3. $|\mathcal{I}| > 1$, i.e., there are several candidate critical regions.

Data: Feedback laws F_i, g_i, critical regions $\mathcal{R}_i$, number of regions M, query point x
Result: Optimal control input
$\mathcal{I} \leftarrow \emptyset$
for $i = 1, \ldots, M$ **do**
  if $x \in \mathcal{R}_i$ **then**
    $\mathcal{I} \leftarrow \mathcal{I} \cup \{i\}$
  end
end
Select $i^\star \in \mathcal{I}$
Output: $u^\star = F_{i^\star} x + g_{i^\star}$

Algorithm 3.3: Sequential search with tiebreaks

In the first case when the cardinality of the set $\mathcal{I}$ is empty, it means no feasible solution was found during the investigation of the critical regions. In this case generally, a NaN (Not a Number) string value is returned which indicates the optimization problem for the given parameter x is not located in the feasible domain.

If the cardinality of set $\mathcal{I}$ is 1, it means only one region contains the given investigated parameter x. This happens when parameter x is located in the strict interior of the critical region.

If there are more than one critical region that contains the parameter then x is situated at the boundaries of two neighboring critical regions. Since the value of optimal control action in each region could be different an optimal objective function's values is used to choose the index of the correct region. Based on the form of the objective function of the optimization problem the cost function of the parametric solution could be of the form

$$J_i(x) = \alpha_i x + \beta_i \tag{3.34}$$

in the case of PWA function and

$$J_i(x) = x^\top \gamma_i x + \alpha_i x + \beta_i \tag{3.35}$$

in case of PWQ function. The right index i is then selected by evaluating the PWA or PWQ function, but only over regions with indices being in set $\mathcal{I}$. Afterward,

the index which corresponds to the minimum value is used to calculate to optimal control action. The index $i^\star$ representing the minimal value is presented in (3.36)

$$i^\star = \arg\min_{i \in \mathcal{I}} J_i(x). \tag{3.36}$$

Both of the point location methods could be relatively easily implemented in low-level and even in high-level programming languages. The procedures presented in Algorithm 3.2 and Algorithm 3.3 are division free, which means the procedure of validation is also easier since there is no need to investigate division by zero. The only required operations in the presented algorithms to obtain the optimal control input are matrix multiplication and addition.

3.7.3 Binary Search Tree

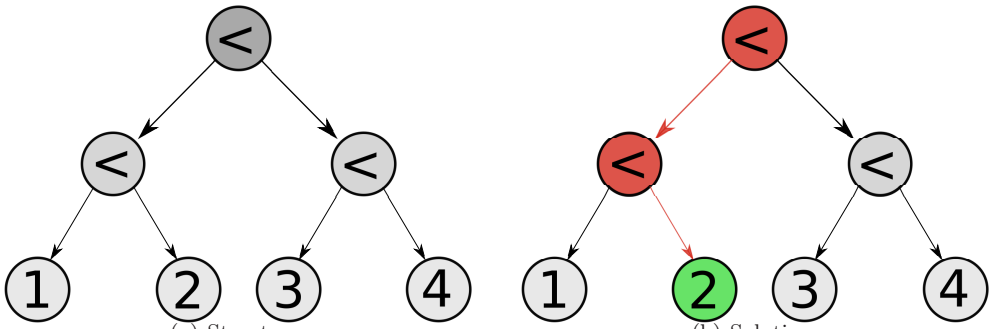
Binary search tree is a fundamentally different approach to the previously presented point location algorithms. In the case of Algorithm 3.2 we need to go through every critical region and in Algorithm 3.3 every time all critical regions must be investigated. The idea of binary search tree presented in Tøndel et al. (2003) is to split the set critical regions based on their geometric properties into smaller groups. The investigated groups are chosen based on simple comparable conditions. If the binary search tree is symmetrical the number of critical regions compared to the required operations are exponentially higher. For example, if the parametric solution consists of 8 regions and the binary search tree is symmetrical only 3 comparison operations are required to obtain the right index of the region. In this case, the minimal number of steps could be computed as follows

$$2^k \geq M, \tag{3.37}$$

where M represents the number of regions and k is the number of required comparison operations. The graphical representation of the structure of the binary search tree is depicted in Fig.3.2(a) and in Fig.3.2(b).

In the case of binary search tree the following properties used to be considered:

1. size of the tree (number of all nodes)
2. roots (number of logical connection between nodes)
3. number of leaves (last nodes, doesn't contain logical test)



(a) Structure. (b) Solution.

Figure 3.2: Binary search tree.

Data: Binary tree T, control laws F_i, g_i defined over regions $\mathcal{R}_i$, state measurement $x(t)$

Result: Optimal control input

$i = 1$
while $True$ **do**
  if $T(i)x(t) \leq 0$ **then**
    $i \leftarrow T(i, left)$
  end
  else
    $i \leftarrow T(i, right)$
  end
  if $i < 0$ **then**
    $i^\star \leftarrow abs(i)$
    $u^\star(t) \leftarrow F_{i^\star} x(t) + g_{i^\star}$
    $break$
  end
end

Output: $u^\star(t) = F_{i^\star} x(t) + g_{i^\star}$

Algorithm 3.4: Binary search tree

4. depth (number of levels)

The number of leaves represents the individual critical regions, while the nodes contain information for the logical test. Each subtree has all the properties of binary search trees. The most interesting information is the depth, which is the indication how many logical tests must be performed in the worst case in order to obtain the optimal control action. Each logical test in the best case allows to skip the half of the remaining unchecked regions. Therefore, this method is the most effective evaluation of the parametric solution. The algorithm of evaluation of binary search tree is presented in Algorithm 3.4 where $T(i)$ represents the information for the logical tests saved in the nodes of the binary search tree, and it could be saved as matrix type variable or two-dimensional array. If variable i is set to the negative value, it means the iteration reaches the leaf of the tree and the control action $u^\star$ could be calculated by the evaluating the local affine function. However, the online computation is much more computationally demanding as in the case of sequential search methods. The construction of the tree, in case the number of regions is relatively high and/or the dimension of the investigated parametric space is also higher, is computationally more demanding. Since the construction of the tree similarly as a computation of the parametric solution is done in the offline phase, the binary search tree method in the online phase does not suffer from any disadvantages. Therefore, it is popular and desired method.

All of the above algorithms mentioned are suitable for control of a chosen application or process. In the following chapter the goal is to introduce the details of implementation of the above mentioned point location methods in different programming languages in order to control nontrivial processes.

3.8 Summary

This chapter briefly described the MPC control technique. It has many advantages and thanks to its natural capability to handle systems constraints became a widely adopted control approach in the academic field and industry. The appropriately defined objective function purpose is to minimize the necessary cost of the inputs to the systems. However the computational cost of the control action by MPC technique is the main drawback, EMPC could bring this advanced approach to the low-cost hardware. Since during the evaluation of EMPC does not require

complicated operations, such matrix inversion the implementation is relatively easy. The way of extracting the optimal control action for different initial parameters is so called point location problem. Three different point location approaches were also introduced. Each of them has different computation demand. The user should choose from the mentioned methods based on the parametric solution. In case the number of regions is small and the control law is continuous the sequential search method is preferred is preferred. If the control law is not a continuous function the extended sequential search may be a good choice. In case the number of regions is high it is worth to consider binary search tree.

Part II

Contributions

Chapter 4

Overview of Contributions

This part of this thesis introduces our novel theoretical and practical results. Theoretical novelties are summarized in Chapter 5. Here, two schemes for reducing the complexity of explicit MPC controllers are presented. The first approach reported in Section 5.1 reduces the complexity in two steps. First, the optimal PWA value function is approximated by a different PWA function in such a way that the new value function preserves guarantees of closed-loop stability and is simpler compared to the original one. The simpler approximation is found by solving a nonlinear optimization problem which originates from Karush-Kuhn-Tucker optimality conditions. Once solved, the solution then encodes a simpler PWA value function from which the (simple) feedback law can be recovered using triangulation. This results in a simple PWA feedback law which, when applied in a closed loop, provides guarantees of recursive constraint satisfaction and closed-loop stability.

The second complexity reduction approach, summarized in Section 5.2, finds a simple PWA feedback law by directly approximating the original (complex) explicit MPC controller. The idea here is first to synthesize a simpler polytopic partition over which the new controller will be defined. This is done by solving a simpler MPC problem with a smaller value of the prediction horizon. To guarantee that the new problem covers all feasible states, we employ control invariant sets as state constraints. To find the local controllers associated with each region, we solve a convex function approximation problem. To do so, we rely on triangulation of the original regions into simplices, followed by employing a well-known function

approximation technique. By approximating the original feedback law in an optimal fashion we are able to reduce the amount of suboptimality. Results of such a procedure are demonstrated on an illustrative example.

Subsequently, in Chapter 6 we present our new code generation framework which serves to automatically generate executable code for explicit MPC feedback strategies. We cover two scenarios. In the first one described in Section 6.1 we show how to export explicit MPC into Python. The main idea here is to extend the scope of explicit MPC beyond Matlab. We show how the code is generated from the users' perspective and describe in detail its function. The merging of the exported code with the target Python-based application is illustrated in two real-life case studies. The first one, reported in Section 6.1.1, shows the application of Python-based explicit MPC to control a popular computer game. In this section we guide the reader through all steps, starting from the modeling of the game, through the construction of a suitable MPC controller up to its implementation to the game. Results reported in the section show that MPC-based players are superior to their human counterparts. The second application in Section 6.1.2 then demonstrates explicit MPC design and implementation for a quadcopter. Experimental results are provided to demonstrate the approach. Export of explicit MPC to the JavaScript language is then covered by Section 6.2. Here, the objective is to allow explicit MPC to run inside a web browser and to utilize the computational power of computers and hand-held devices such as phones or tablets. The section explains the generated code and shows how it can be used. Applicability of the approach is then demonstrated on a simulation case study which involves the control of a temperature inside a building with the explicit MPC controller running inside a web browser.

Finally, Chapter 7 presents two real-life case studies. The first one, presented in Section 7.1, deals with MPC design and implementation for a hard chromium plating process. Due to the underlying nonlinear nature of the plant, we investigate three different MPC setups. The first one is based on the full nonlinear model of the plant and serves as a performance benchmark. Needless to say, implementation of MPC in such a case requires solving non-convex nonlinear optimization problems, which is a time-consuming task. Therefore, two alternatives are investigated. One, called *linearized MPC*, employs a linear approximation of the nonlinearities which is updated at each sampling instant given the current plant's conditions. However, because the prediction model is constantly updated, this prohibits the use of explicit

MPC which requires a fixed prediction model. Therefore the second approach employs a fixed linearization to design the controller. The performance of all three setups is compared together with the required implementation effort. The second case study in Section 7.2 investigates a magnetic manipulator system. Here, the task is to control currents injected into coins which in turn create a shaped magnetic field that moves a metallic ball. The objective is to control the currents in an optimal fashion such that the ball follows a prescribed reference. To cope with nonlinearities we split the control design task into two parts. One uses a linear model of the motion to design the required forces on the ball. The second part then uses an inverse model of the nonlinearity to compute currents for individual coils such that they create a required force field.

Chapter 5

Complexity Reduction in Explicit MPC

Explicit model predictive control (EMPC) becomes a preferred control strategy when there is a need to implement MPC on a hardware with limited capabilities or for systems with fast dynamics. The main advantage of EMPC is the fact the optimization is shifted offline. At this stage, the optimization problem is solved by parametric programming as it was introduced in Section 2.9. The solution is of the form of a piecewise affine (PWA) function defined over polytopic regions. Then, in the online phase, there is no need to perform numerical optimization. Instead, optimal control actions are obtained by evaluating the PWA function for current state measurements. This evaluation is done by solving the point location problem, as described in Section 3.7. This method requires only simple matrix operations such as summation and multiplications, and therefore EMPC is suitable also for hardware with limited computational resources.

The main limitation of EMPC, however, lies in its complexity, most usually expressed in term of the number of regions over which the PWA feedback law is defined. For large prediction horizons and/or for systems with a large number of states the complexity can quickly become prohibitive from a hardware point of view. Therefore, it is of imminent importance to synthesize simple explicit MPC controllers with a reasonably low number of regions. In this chapter, we present two techniques for finding simple explicit MPC feedback laws.

Technically speaking, we aim at solving the following problem: we are given a (complex) explicit MPC controller whose number of regions exceeds storage limits. The task is to replace this complex controller by a simpler one, defined over a smaller number of regions. Moreover, we require the simpler feedback law to exhibit certain desired properties, such as closed-loop stability, recursive constraint satisfaction, and a mitigated suboptimality.

In the sequel, we present two ways of solving this problem. In Section 5.1 we first approximate the optimal value function as to find a new one of lower complexity. Since the approximated function is bounded from above and from below by the optimal cost, it maintains closed-loop stability and recursive constraint satisfaction. Subsequently, the (simple) feedback law is recovered from the approximated cost function via triangulation. The second approach presented in Section 5.2 directly approximates the original (complex) feedback law by a simpler PWA function. The approximation is done in such a way that the amount of suboptimality, represented as the integrated squared error between the original feedback and its approximation, is minimized.

5.1 Bounded PWA Approximation of the Cost Function

In this section, a complexity reduction technique will be introduced which uses bounded PWA approximation of the cost function. This result was published in (Holaza et al., 2012). The result of this approximation technique is a simpler cost function which is bounded from below and above and this simpler function is defined over a smaller number of critical regions. It is possible to recover the feedback law from the new cost function in the form of PWA function defined over polytopic regions by triangulation and interpolation (Jones et al., 2007). One nonlinear programming problem (NLP) needs to be solved to obtain the coefficients of the approximated cost function. The presented method is suitable only for objective functions which are piecewise affine.

Let us consider an MPC problem with the following setup:

$$J^\star(x_0) = \min_{u_0,\dots,u_{N-1}} \quad \sum_{k=0}^{N-1} \|Q_x x_k\|_p + \|Q_u u_k\|_p \tag{5.1a}$$

$$\text{s.t.} \quad x_{k+1} = Ax_k + Bu_k, \tag{5.1b}$$

$$x_k \in \mathcal{X}, \tag{5.1c}$$

$$u_k \in \mathcal{U}, \tag{5.1d}$$

$$x_N \in \mathcal{T}, \tag{5.1e}$$

with $p = 1$ (cf. (2.9)) or $p = \infty$ (cf. (2.11)). We also consider polytopic state constraints $\mathcal{X}$, input constraints $\mathcal{U}$ and a polytopic terminal set $\mathcal{T}$. Control problem defined in (5.1) can be easily transformed into a mp-LP problem of the form (2.44) and solved parametrically using an mp-LP algorithm as shown in Section 2.9. The parametric solution to (5.1) then yields a piecewise affine optimizer $u_0^\star(x_0)$, i.e.,

$$u^\star(x) := \begin{cases} F_1 x + g_1 & \text{if } x \in \mathcal{R}_1 \\ \vdots & \\ F_R x + g_R & \text{if } x \in \mathcal{R}_R, \end{cases} \tag{5.2}$$

along with a PWA representation of the optimal cost function $J^\star(x_0)$ with

$$J^\star(x) := \begin{cases} c_1^\top x + d_1 & \text{if } x \in \mathcal{R}_1 \\ \vdots & \\ c_R^\top x + d_R & \text{if } x \in \mathcal{R}_R. \end{cases} \tag{5.3}$$

Here, $\mathcal{R}_i$, $i = 1, \dots, R$ are the underlying polytopic regions of these two functions with R denoting the number of regions. If the penalty matrices Q_x and Q_u, along with the terminal set $\mathcal{T}$ are designed in a proper way, the cost function in (5.3) is a Lyapunov function.

The goal is to find an another PWA continuous function $\widetilde{J}(x)$ which is defined over Q regions with $Q < R$. We also require $\widetilde{J}(x)$ to be a Lyapunov function. This property can be obtained if $\widetilde{J}(x)$ is bounded from below by $J^\star(x)$ and from above by $J^\star(x) + \ell(x,u)$ where $\ell(x,u) = \|Q_x x\|_p + \|Q_u u\|_p$ is the stage cost. Formally, we require that

$$J^\star(x) \leq \widetilde{J}(x) \leq J^\star(x) + \ell(x,u). \tag{5.4}$$

If $\widetilde{J}(x)$ satisfying (5.4) can be found, the associated simple feedback law $\widetilde{u}(x)$ can be recovered using triangulation and interpolation.

Our objective is hence to find $\widetilde{J}(x)$ which is bounded as in (5.4) for all states x. To simplify the exposition, we define the lower bound $J^\star(x)$ as

$$J_{low}(x) = \underline{c}_i^\top x + \underline{d}_i \text{ if } x \in \mathcal{R}_i,\ i = 1, \ldots, R, \tag{5.5}$$

and the upper bound, i.e., $J^\star(x) + \ell(x, u)$, by

$$J_{up}(x) = \overline{c}_i^\top x + \overline{d}_i \text{ if } x \in \mathcal{R}_i,\ i = 1, \ldots, R. \tag{5.6}$$

We aim at finding the function

$$\widetilde{J}(x) = \max_i \{\alpha_i^\top x + \beta_i\}, \tag{5.7}$$

i.e., to find the parameters α_i and β_i, $i = 1, \ldots, Q$, such that $\widetilde{J}(x)$ of (5.7) satisfies (5.4) for all x from the domain of $J^\star(x)$.

Bounding a PWA function from above, i.e., to get $\widetilde{J}(x) \leq J_{up}(x)$ is easy once we know the vertices of regions $\mathcal{R}_i$ over which $J_{up}(x)$ is defined. In particular, denote by $\mathcal{V}_i$ the vertices of $\mathcal{R}_i$ for $i = 1, \ldots, R$. Then $\widetilde{J}(x)$ from (5.7) is upper bounded by $J_{up}(x)$ if and only if

$$\max_k \{\alpha_k^\top v_j + \beta_k\} \leq \overline{c}_i^\top v_j + \overline{d}_i, \forall v_j \in \mathcal{V}_i. \tag{5.8}$$

To get rid of the maximum in (5.8), we enforce the constraints to hold for all $k = 1, \ldots, Q$:

$$\alpha_k^\top v_j + \beta_k \leq \overline{c}_i^\top v_j + \overline{d}_i, \forall v_j \in \mathcal{V}_i, \forall k, \forall i. \tag{5.9}$$

This is a set of linear constraints in α_i and β_i since the vertices v_j are considered to be constants.

Bounding $\widetilde{J}(x)$ from below by $J^\star(x)$ is less straightforward. Even when x is restricted to be contained in $\mathcal{R}_i$, the problem is to guarantee that

$$\underline{c}_i^\top v_j + \underline{d}_i \leq \widetilde{J}(x), \forall x \in \mathcal{R}_i \tag{5.10}$$

holds. Rewriting (5.10) gives

$$\widetilde{J}(x) - \underline{c}_i^\top v_j - \underline{d}_i \geq 0, \forall x \in \mathcal{R}_i. \tag{5.11}$$

Then the relation in (5.11) holds if and only if

$$z^\star = \min_{x \in \mathcal{R}_i} (\widetilde{J}(x) - \underline{c}_i^\top v_j - \underline{d}_i) \tag{5.12}$$

satisfies

$$z^\star \geq 0. \tag{5.13}$$

By introducing a new scalar variable $\epsilon \in \mathbb{R}$ it is possible to rewrite $\widetilde{J}(x)$ in the following way

$$\widetilde{J}(x) = \min\{\epsilon \mid \epsilon \geq \alpha_k^\top x + \beta_k, \forall k\}. \tag{5.14}$$

Then we cast (5.12) as

$$z^\star = \min\{\epsilon - \underline{c}_i^\top x - \underline{d}_i \mid \epsilon \geq \alpha_k^\top x + \beta_k, A_i x \leq b_i, \forall k\}, \tag{5.15}$$

where A_i and b_i are the matrices/vectors of the half-space representation of region $\mathcal{R}_i$ as in $\mathcal{R}_i = \{x \mid A_i x \leq b_i\}$.

The set of constraints defined in (5.15) are nonlinear in unknown variables α_k, β_k, ϵ and x. The optimization problem in (5.15) can be rewritten by using the Karush-Kuhn-Tucker conditions as

$$\alpha_k^\top x^\star + \beta_k - \epsilon^\star \leq 0, \tag{5.16a}$$
$$A_i x^\star \leq b_i, \tag{5.16b}$$
$$\lambda_k^\star \geq 0, \tag{5.16c}$$
$$\mu^\star \geq 0, \tag{5.16d}$$
$$(\alpha_k^\top x^\star + \beta_k)^\top \lambda_k^\star = 0, \tag{5.16e}$$
$$(A_i x^\star - b_i)^\top \mu^\star = 0, \tag{5.16f}$$
$$1 - \sum_{k=1}^{N} \lambda_k^\star = 0, \tag{5.16g}$$
$$-\underline{c}_i + \sum_{k=1}^{Q} (\alpha_k^\top \lambda_k^\star) + A_i^\top \mu^\star = 0. \tag{5.16h}$$

Here, (5.16h) represents the derivation of the Lagrangian, (5.16a) and (5.16b) express the primal feasibility conditions and (5.16c) together with (5.16d) the dual feasibility conditions. The complementary slackness condition is presented in (5.16e) and (5.16f). Note that constraints (5.16) are nonlinear in the decision variables ϵ, λ, μ, x, α and β.

Let $\epsilon^\star$ and $x^\star$ be the feasible solution to (5.16). By representing $z^\star = \epsilon^\star - \underline{c}_i^\top x^\star - \underline{d}_i$, we can rewrite (5.11) using (5.13) as

$$\epsilon^\star - \underline{c}_i^\top x^\star - \underline{d}_i \geq 0. \tag{5.17}$$

Relations (5.16) together with (5.13) therefore encode the lower bounding problem $J^\star(x) \leq \widetilde{J}(x)$ for all x in the domain of $J^\star(x)$. If, moreover, (5.9) is also considered, then a feasible solution to the joint problem yields the parameters α_k, β_k, $k = 1, \ldots, Q$ of an approximate cost function $\widetilde{J}(x)$ that satisfies (5.4). The final problem is in the form

$$\alpha_k^\top v_j + \beta_k \leq \overline{c}_i^\top v_j + \overline{d}_i, \quad \forall v_j \in \mathcal{V}_i,\, k = 1, \ldots, N, \tag{5.18a}$$

$$\epsilon_i^\star - \underline{c}_i^\top x_i^\star - \underline{d}_i \geq 0, \tag{5.18b}$$

$$\alpha_k^\top x_i^\star + \beta_k - \epsilon_i^\star \leq 0, \quad k = 1, \ldots, N, \tag{5.18c}$$

$$A_i x_i^\star \leq b_i, \tag{5.18d}$$

$$\lambda_{i,k}^\star \geq 0, \quad k = 1, \ldots, N, \tag{5.18e}$$

$$\mu^\star \geq 0, \tag{5.18f}$$

$$(\alpha_k^\top x_i^\star + \beta_k)^\top \lambda_{i,k}^\star = 0, \quad k = 1, \ldots, N, \tag{5.18g}$$

$$(A_i x_i^\star - b_i)^\top \mu^\star = 0, \tag{5.18h}$$

$$1 - \textstyle\sum_{k=1}^{N} \lambda_{i,k}^\star = 0, \tag{5.18i}$$

$$-\underline{c}_i + \textstyle\sum_{k=1}^{N} (\alpha_k^\top \lambda_{i,k}^\star) + A_i^\top \mu_i^\star = 0, \tag{5.18j}$$

which is a nonlinear programming problem.

Once a feasible solution α_k, β_k, $k = 1, \ldots, Q$ is found for some fixed value of Q, the simple feedback law $\widetilde{u}(x)$ can be recovered from $\widetilde{J}(x)$ using triangulation and interpolation (Jones et al., 2007). To recover it, we need to obtain the vertices of the new critical regions $\mathcal{V}_i$, where index i distinguishes between the individual regions. We calculate the optimal control action in each vertex $\mathcal{V}_{i_j}$ by evaluating the original optimal control law $U_{\mathrm{opt}_{\mathcal{V}_i}}$. The parameters such as the slope and the affine part of the new control law are calculated as

$$\begin{bmatrix} \alpha_{\mathrm{aprx}_i} \\ \beta_{\mathrm{aprx}_i} \end{bmatrix} = U_{\mathrm{opt}_{\mathcal{V}_i}} \begin{bmatrix} \mathcal{V}_i \\ \mathbf{1} \end{bmatrix}^{-1}, \tag{5.19}$$

where α_{aprx_i} and β_{aprx_i} represent the coefficients of local slope and affine part in the i-th region of the approximated control law.

In the sequel the procedure is illustrated on example.

5.1.1 Example

Let us consider a system with unstable with the following dynamics

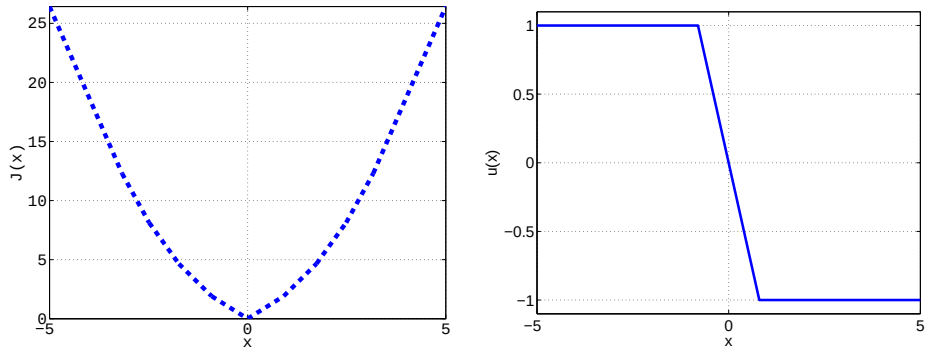
$$x_{k+1} = 1.1x_k + u_k. \tag{5.20}$$

We have designed MPC control problem for this system and considered constraints on states and input variable of the form

$$-1 \leq u \leq 1, \tag{5.21a}$$

$$-5 \leq x \leq 5, \tag{5.21b}$$

the length of the prediction horizon $N = 5$, with weightings $Q_x = 1$ and $Q_u = 1$ and Manhattan-norm $p = 1$. This kind optimal control problem can be solved as mp-LP, since the objective function is linear and all the constraints are linear as well. The result of this optimization problem is a PWA function of the objective function and the control law is defined over 10 regions. The optimal cost function and the optimal control law are depicted in Fig. 5.1.



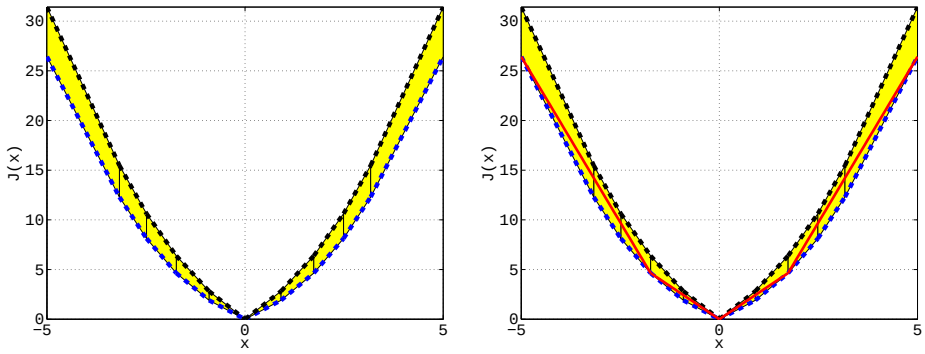
(a) Optimal cost function $J^*(x)$ (b) Optimal control law $u^\star(x)$

Figure 5.1: Original EMPC controller.

The safe area, which is bounded below by the original objective function and from above by the objective function shifted by the value of stage cost is depicted in Fig 5.2(a), while the simplified one is shown in Fig. 5.2(b).

Both, the original control law and the approximated control law are depicted in Fig. 5.4(a).

The controllers were also compared in simulations. The result, which represents the different evolution of the controlled system is depicted in Fig. 5.4(b). For the



(a) Tube (b) Approximation

Figure 5.2: Approximation procedure for unstable system.

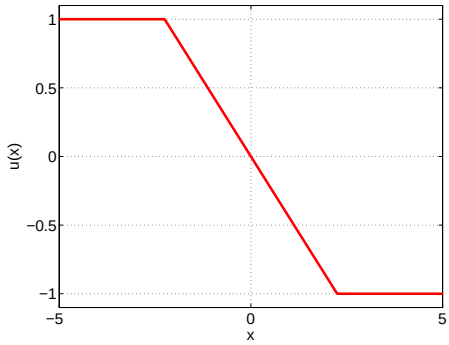


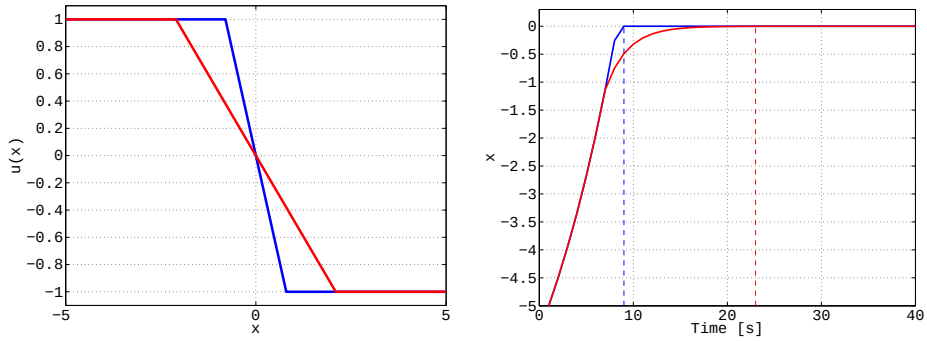
Figure 5.3: Approximated control law.

chosen initial point $x_0 = -5$ both of the investigated controllers were able to regulate the system, however the approximated controller is slower.

Table 5.1 summarizes a difference between the original and the approximated controller. In this particular case the achieved memory reduction was 60%. However the approximated controller has worse performance, is still provides constraint satisfaction and able to move the system to the desired origin.

5.1.2 Conclusions

In this section, a complexity reduction method was introduced. The proposed method uses the properties of the objective function. This method is applicable



(a) Control laws

(b) Simulation results

Figure 5.4: Comparison of unstable system control laws: (a) the optimal (blue line) and the suboptimal (red line) control laws; (b) the optimal (blue line) and the suboptimal (red line) system responses. The regulation time is shown by dashed lines with corresponding colors.

Properties	$u^*(x)$	$\widetilde{u}(x)$
Number of regions	10	4
Memory footprint (Regions)	80	32
Memory footprint (Control law) (bytes)	80	32
Memory footprint (Total) (bytes)	160	64
Computation time (10^{-5} s)	7.9	3.8

Table 5.1: Memory and runtime for the unstable system

only if the objective function is convex and PWA affine function. In order to achieve memory reduction, the new objective function must be created over a smaller number of polytopic regions. Their number is the main memory consumer in case of EMPC. Therefore, our goal is to reduce them. The new objective function is created by using the proposed approximation technique. The goal of approximation was formulated as NLP problem since there is a product in optimized variables. The result of the optimization contains the values of individual slope and affine part of the new PWA objective function. After the approximation procedure, the control law is recovered. The main limitation of this approach is its computation demand in higher state space. Since the number of optimized variables increases

with the dimension of the state space dimension, the approach is suitable only for smaller systems.

5.2 Nearly-Optimal Simple Explicit MPC

The results described in this section are one of the outcomes of the joint work with Juraj Holaza from our Institute. This section is devoted to complexity reduction method which is suitable also for higher dimensions of the state space. We present a procedure that finds a simple PWA feedback law by directly approximating the original (complex) explicit MPC controller. The key idea here is first to synthesize a simpler polytopic partition over which the new controller will be defined. This is done by solving a simpler MPC problem with a smaller value of the prediction horizon. To guarantee that the new problem covers all feasible states, we employ control invariant sets as state constraints. To find the local controllers associated with each region, we solve a convex function approximation problem. To do so, we rely on triangulation of the original regions into simplices, followed by employing a well-known function approximation technique. By approximating the original feedback law in an optimal fashion we are able to reduce the amount of suboptimality. Results of such a procedure are demonstrated on an illustrative example. The results presented in this section are based on our work published in Takács et al. (2013).

5.2.1 Problem Statement

We aim to control linear discrete-time systems in the state-space form

$$x_{k+1} = Ax_k + Bu_k, \tag{5.22}$$

with $x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$. The system in (5.22) is subject to state and input constraints

$$x_k \in \mathcal{X},\ u_k \in \mathcal{U}, \tag{5.23}$$

where $\mathcal{X} \subset \mathbb{R}^n$, $\mathcal{U} \subset \mathbb{R}^m$ are polytopes that contain the origin in their respective interiors.

For the given model in (5.22) and constraints in (5.23), we formulate optimiza-

tion problem of MPC as follows

$$\mu = \arg\min \sum_{k=0}^{N-1} \ell(x_k, u_k) \tag{5.24a}$$

$$\text{s.t. } x_{k+1} = Ax_k + Bu_k, \tag{5.24b}$$

$$u_k \in \mathcal{U}, \tag{5.24c}$$

$$x_0 \in \mathcal{C}_\infty,\ x_1 \in \mathcal{C}_\infty, \tag{5.24d}$$

$$x_0 = x(t), \tag{5.24e}$$

where u_k, x_k are predictions of the control inputs and states, respectively, at the time step $t + k$. Next, Q_u and Q_x are weighting matrices, Ndenotes prediction horizon and $\mathcal{C}_\infty$ is a maximal control invariant set. Maximal control invariant set $\mathcal{C}_\infty$ can be defined as a set of all initial conditions $x_0 \in \mathcal{X}$ from which there exists at least one control input $u \in \mathcal{U}$ such that, for all future time steps, states of (5.23) are contained inside of this set $x_{k+1} \in \mathcal{C}_\infty$, i.e. can by formulated as

$$\mathcal{C}_\infty = \{x_0 \in \mathcal{X} \mid \forall k \in \mathbb{N} :\ \exists u_k \in \mathcal{U} \text{ s.t.} x_{k+1} = Ax_k + Bu_k \in \mathcal{X}\},$$

and computed iteratively. By solving optimization problem in (5.24) via parametric programming, we obtain explicit solution $\mu : \mathbb{R}^n \to \mathbb{R}^m$ in a form of

$$\mu = F_j x + g_j \text{ if } x \in \mathcal{R}_j,\ j = 1, \ldots, M, \tag{5.25}$$

which is a PWA function defined over M regions, with local affine gains $F_j \in \mathbb{R}^{m\times n}$ and $g_j \in \mathbb{R}^m$. In sequel we define the problem, which illustration can be found in Fig. 5.5.

Consider that we are given an explicit MPC feedback law as in (5.1). We aim to devise a new explicit controller $\widetilde{\mu} : \mathbb{R}^n \to \mathbb{R}^m$ defined as

$$\widetilde{\mu}(x) = \widetilde{F}_i x + \widetilde{g}_i \text{ if } x \in \widetilde{\mathcal{R}}_i,\ i = 1, \ldots, \widetilde{M}, \tag{5.26}$$

with $\widetilde{F}_i \in \mathbb{R}^{m\times n}$, $g_i \in \mathbb{R}^m$ and $\widetilde{M} < M$ such that following statements hold:

R1: for each $x \in \text{dom}(\mu)$ the approximated controller provides recursive constraints satisfaction, on both input and state constraints, which means for all the future time instances the system remains in within the state constraints. $\forall t \in \mathbb{N}$ we have that $\widetilde{\mu}(x(t)) \in \mathcal{U}$ and $Ax(t) + B\widetilde{\mu}(x(t)) \in \mathcal{X}$;

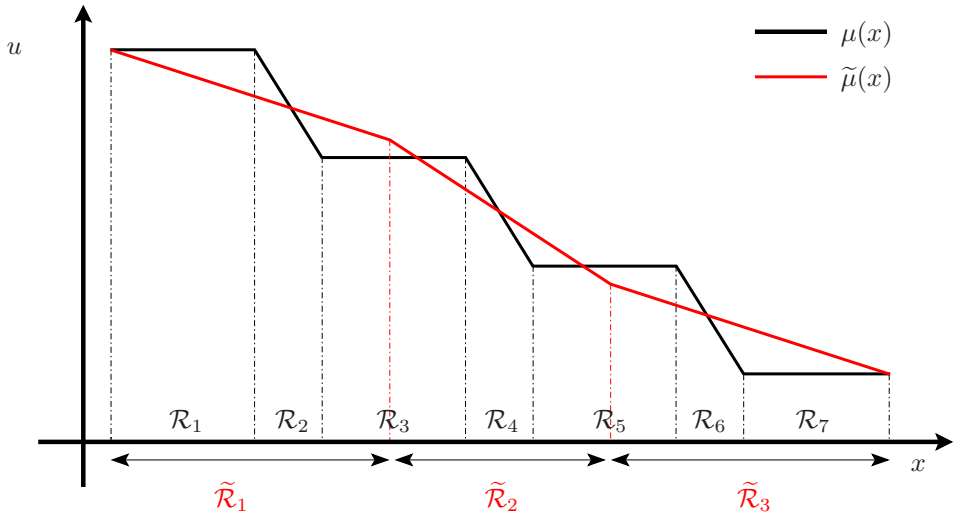


Figure 5.5: The function $\mu(\cdot)$, shown in black, is given. The task is to synthesize the function $\widetilde{\mu}(\cdot)$, shown in red, which is less complex (here it is defined just over 3 regions instead of 7 for $\mu(\cdot)$) and minimizes the integrated square error (5.27).

R2: $\widetilde{\mu}(\cdot)$ is constructed such the integrated squared error between PWA functions $\mu(\cdot)$ and $\widetilde{\mu}(\cdot)$, the domain of $\mu(\cdot)$, Ω, is minimized:

$$\min \int_{\Omega} \|\mu(x) - \widetilde{\mu}(x)\|_2^2 \, \mathrm{d}x, \tag{5.27}$$

where $\mathrm{d}x$ represents the Lebesgue measure of Ω, see Baldoni et al. (2010).

Such an approximated control law $\widetilde{\mu}(\cdot)$ does not have an a-priori guarantee of closed-loop stability, but it can be a-posteriori certified by constructing a suitable Lyapunov function, see e.g. Ferrari-Trecate et al. (2002). Moreover, in order to render the origin an equilibrium of system (5.22), the requirements should be extended by the constraint $\widetilde{\mu}(0) = 0$.

5.2.2 Memory Reduction Technique

We approach to solve the problem in Section 5.2.1 via two steps. In the first step we will construct a new (less complex) polytopic partition, which will be subsequently used in the second step, where we find new local gains of $\widetilde{\mu}(\cdot)$ above each region $\tilde{\mathcal{R}}_i$.

Polytopic Partition

Here, we aim to devise a new partition of polytopes $\cup_i \tilde{\mathcal{R}}_i$, such that $\widetilde{M} < M$ and the domain of the original controller and the new one are identical, i.e., $\cup_i \tilde{\mathcal{R}}_i = \cup_j \mathcal{R}_j$ with $i = 1, \ldots, M$ and $j = 1, \ldots, \widetilde{M}$.

We suggest to obtain $\cup_i \tilde{\mathcal{R}}_i$ by solving (5.24) again but with a shorter prediction horizon. Thanks to the maximal control invariant in (5.24d) the feasible sets in different prediction horizon length are identical, the difference is in the number $\tilde{M}$ of regions $\tilde{\mathcal{R}}_i$.

We note that the parametric solution of (5.24), with a shorter prediction horizon, already yields a simpler explicit feedback law, which satisfies requirement R1. However, since condition R2 is not guaranteed, i.e. degradation of performance is not minimized, thus we remove this controller and preserve only its polytopic partition.

Function Fitting

In what follows, we show how to optimize new affine gains of $\widetilde{\mu}$, such that both conditions R1 and R2 hold. We propose to address this goal by following optimization problem:

$$\min_{\widetilde{F}_i, \widetilde{g}_i} \int_{\widetilde{\mathcal{R}}_i} \|\mu(x) - \widetilde{\mu}(x)\|_2^2 \, \mathrm{d}x \tag{5.28a}$$

$$\text{s.t. } \forall x \in \widetilde{\mathcal{R}}_i : \begin{cases} \widetilde{F}_i x + \widetilde{g}_i \in \mathcal{U}, \\ Ax + B(\widetilde{F}_i x + \widetilde{g}_i) \in \mathcal{C}_\infty, \end{cases} \tag{5.28b}$$

where the objective function (5.28a) imposes condition R1, the first constraint in (5.28b) ensures recursive satisfaction of inputs and states constraints, i.e. condition R2. There are, however, two main obstacles, which prohibit application of the optimization problem in (5.28).The first issue represents the integral over a polyhedron $\widetilde{\mathcal{R}}_i$ in (5.28a) over which $\mu(x)$ may be still a PWA function. And the second issue is that constraints in (5.28b) have to hold $\forall x \in \widetilde{\mathcal{R}}_i$, thus for infinite number of points.

To tackle the first issue, we propose to reformulate (5.28a) into the following form

$$\min_{\widetilde{F}_i, \widetilde{g}_i} \sum_{j \in \mathcal{J}_i} \int_{\mathcal{Q}_{i,j}} \|(F_j x - g_j) - (\widetilde{F}_i x + \widetilde{g}_i)\|_2^2 \, \mathrm{d}x, \tag{5.29}$$

where $\mathcal{Q}_{i,j}$ denotes intersections between region $\tilde{\mathcal{R}}_i$ and $\mathcal{R}_j$, e.i.

$$\mathcal{Q}_{i,j} = \tilde{\mathcal{R}}_i \cap \mathcal{R}_j, \forall j \in \{1, \dots, M\}. \tag{5.30}$$

Next, to obtain an analytic expression for the integral, we use the result of (Lasserre and Avrachenkov, 2001), extended by (Baldoni et al., 2010), where authors have proposed an integration formula of a homogeneous polynomial f of degree d in n variables as

$$\int_{\Delta} f(y)\mathrm{d}y = \gamma \sum_{1 \le i_1 \le \cdots \le i_d \le n+1} \sum_{\epsilon \in \{\pm 1\}^d} \epsilon_1 \cdots \epsilon_d f(\textstyle\sum_{k=1}^{d} \epsilon_k s_{i_k}), \tag{5.31}$$

where $s_1, \dots, s_{n+1}$ are the vertices of an n-dimensional simplex Δ and

$$\gamma = \frac{\mathrm{vol}(\Delta)}{2^d d! \binom{d+n}{d}}, \tag{5.32}$$

with $\mathrm{vol}(\Delta)$ denoting the volume of the simplex. Note that formula in (5.31) can not be directly used in (5.28) as polyhedral intersections in (5.30) are not necessary simplices. We approach this problem by dividing each intersections $\mathcal{Q}_{i,j}$ into simplices $\Delta_{i,j,1}, \dots, \Delta_{i,j,K}$ with $\mathrm{int}(\Delta_{i,j,k_1}) \cap \mathrm{int}(\Delta_{i,j,k_2}) = \emptyset$ for all $k_1 \neq k_2$, and $\cup_k \Delta_{i,j,k} = \mathcal{Q}_{i,j}$. Then we can rewrite (5.29) as a sum of the integrals evaluated over each simplex:

$$\min_{\widetilde{F}_i, \widetilde{g}_i} \sum_{j \in \mathcal{J}_i} \sum_{k=1}^{K} \int_{\Delta_{i,j,k}} \|(F_j x - g_j) - (\widetilde{F}_i x + \widetilde{g}_i)\|_2^2 \, \mathrm{d}x, \tag{5.33}$$

where K is the number of simplices tessellating $\mathcal{Q}_{i,j}$. Next, integral error in (5.29) is not homogeneous function. To see this, expand $f(x) := \|(F_j x + g_j) - (\widetilde{F}_i x + \widetilde{g}_i)\|_2^2$ to $f(x) := x^T Q x + r^T x + q$ with

$$Q = F_j^T F_j - 2F_j \widetilde{F}_i + \widetilde{F}_i^T \widetilde{F}_i, \tag{5.34a}$$

$$r = 2(F_j^T \widetilde{g}_i + \widetilde{F}_i^T \widetilde{g}_i - \widetilde{F}_i^T g_j - F_j^T \widetilde{g}_i), \tag{5.34b}$$

$$q = g_j^T g_j - 2g_j^T \widetilde{g}_i + \widetilde{g}_i^T \widetilde{g}_i. \tag{5.34c}$$

Now, we can see that $f(x)$ is a quadratic function with optimized variables $\widetilde{F}_i$ and $\widetilde{g}_i$. Let us now split $f(x)$ into its monomials as follows:

$$\int_{\Delta} f(x) = \int_{\Delta} f_{\mathrm{quad}}(x) + \int_{\Delta} f_{\mathrm{lin}}(x) + \int_{\Delta} f_{\mathrm{const}}, \tag{5.35}$$

where $f_{\mathrm{quad}}(x) := x^T Q x$, $f_{\mathrm{lin}} := r^T x$ and $f_{\mathrm{const}} := q$. Moreover, integrand $\mathrm{d}x$ is neglected for simplicity. Now, as $f(x)$ in (5.35) represents a set of homogeneous

polynomials of degree 2, 1 and 0, respectively, which are defined over simplices Δ, we can use the analytic formula in (5.31) to integrate error in (5.33).

Now, we will aim to tackle the second problem, where we need to ensure that constraints in (5.28b) hold $\forall x \in \widetilde{\mathcal{R}}_i$. Since both sets $\mathcal{U} = \{u \,|\, H_{\mathrm{u}} u \leq h_{\mathrm{u}}\}$ and $\mathcal{C}_\infty = \{x \,|\, H_{\mathrm{c}} x \leq h_{\mathrm{c}}\}$ are assumed to be polytopes, they can be represented as a set of linear inequalities with substitution $u = \widetilde{F}_i x + \widetilde{g}_i$ as follows

$$\forall x \in \widetilde{\mathcal{R}}_i : f(x) \leq 0, \tag{5.36}$$

with

$$f(x) := \begin{bmatrix} H_{\mathrm{u}} \widetilde{F}_i \\ H_{\mathrm{c}}(A + B\widetilde{F}_i) \end{bmatrix} x + \begin{bmatrix} H_{\mathrm{u}} \widetilde{g}_i - h_{\mathrm{u}} \\ H_{\mathrm{c}} \widetilde{g}_i - h_{\mathrm{c}} \end{bmatrix}. \tag{5.37}$$

By denoting $\mathcal{V}_i = \{v_{i,1}, \ldots, v_{i,n_{v,i}}\}$, $v_{i,j} \in \mathbb{R}^n$ to be a set of all vertices of $\widetilde{\mathcal{R}}_i$, we have that (5.36) holds if $f(v_{i,j}) \leq 0$ is satisfied for all vertices.

The final form of the optimization problem could be expressed as

$$\min_{\widetilde{F}_i, \widetilde{g}_i} \sum_{j \in \mathcal{J}_i} \sum_{k=1}^{K} \int_{\Delta_{i,j,k}} \|(F_j x - g_j) - (\widetilde{F}_i x + \widetilde{g}_i)\|_2^2 \, \mathrm{d}x, \tag{5.38a}$$

$$\text{s.t. } \forall v_{i,\ell} \in \mathrm{vert}(\widetilde{\mathcal{R}}_i) : \begin{cases} \widetilde{F}_i v_{i,\ell} + \widetilde{g}_i \in \mathcal{U}, \\ A v_{i,\ell} + B(\widetilde{F}_i v_{i,\ell} + \widetilde{g}_i) \in \mathcal{C}_\infty, \end{cases} \tag{5.38b}$$

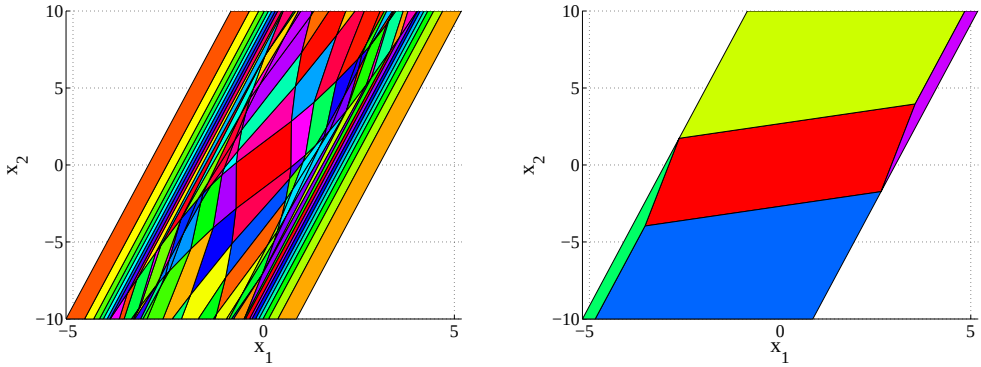
where $\mathrm{vert}(\widetilde{\mathcal{R}}_i)$ enumerates all vertices of the corresponding polytope.

5.2.3 Illustrative Example

The above described method will be demonstrated on a second order, discrete-time, linear time-invariant system. The dynamics of the system could be expressed as

$$x(t+1) = \begin{bmatrix} 0.9539 & -0.3440 \\ -0.4833 & -0.5325 \end{bmatrix} x(t) + \begin{bmatrix} -0.4817 \\ -0.5918 \end{bmatrix} u(t), \tag{5.39}$$

This system has also state $-10 \leq x_i(t) \leq 10$, $\forall i \in \{1, 2\}$ and input $-0.5 \leq u(t) \leq 0.5$ and input constraint. The values of weighting matrices for the complex explicit controller are $Q_x = I_{2\times 2}$, $Q_u = 2$ and the length of the prediction horizon is $N = 20$. Its explicit representation was defined over $M = 127$ critical regions and it is depicted in Fig. 5.6(a). Several different prediction horizons were used to construct the new domain $\cup_i \tilde{\mathcal{R}}_i$. Especially $N = \{1, 2, 3, 4\}$ values were considered,



(a) 127 regions of $\mu(\cdot)$. (b) 5 regions of $\widetilde{\mu}(\cdot)$.

Figure 5.6: Regions of the complex controller $\mu(\cdot)$ and of the approximate feedback $\widetilde{\mu}(\cdot)$

Table 5.2: Complexity and suboptimality comparison for the example.

N	# of regions	Subopt. simple	Subopt. approximated
1	3	60.8 %	25.1 %
2	5	32.9 %	18.0 %
3	11	11.4 %	8.3 %
4	17	6.9 %	1.7 %

while the number of corresponding regions were $\tilde{M} = \{3, 5, 11, 17\}$. The new regions for prediction horizon $N = 2$ are presented in Fig. 5.6(b). After that another explicit controllers were constructed with the same prediction horizon for comparison purposes. A closed-loop simulations for 10000 was performed with equidistantly spaced initial conditions from the domain of $\mu(\cdot)$. In each simulation we have evaluated the performance criterion $J_{\text{sim}} = \sum_{i=1}^{N_{\text{sim}}} x_i^T Q_x x_i + u_i^T Q_u u_i$ for $N_{\text{sim}} = 100$. The results are reported in Table 5.2.

The performance of the controller constructed via approximation is better compared to the simple controller assuming the same prediction horizon. It is important to mention that the performance of the approximated controller compared to the simple controller is higher, it's memory demand remains the same. Therefore this approach biggest benefit is the chance to increase the performance of the controller while the requirements remain the same.

5.2.4 Conclusions

In this section a complexity reduction technique was introduced. The memory demand of the parametric solution can be reduced in 2 steps. In the first step a new partition is created by solving an optimization problem, which comes from MPC problem formulation with a shorter prediction horizon. It is mandatory that the domain of the EMPC controllers with longer and shorter horizon to be the same. The approximation is performed over the parametric solution with fewer critical regions. The result of the approximation is the new control law, which is defined over a smaller number of critical regions. Moreover, the performance should be better than that of EMPC controllers with smaller prediction horizon. Thus, a significant memory reduction can be achieved even in the higher dimensional state space.

5.3 Summary

In this part of the work two complexity reduction techniques were presented. The first introduced method is applicable only for problems which could be solved as mp-LP. Since the form of the objective function is a PWA function the presented approximation method can be implemented. Another function is created by shifting PWA objective function by the value of stage cost. The created tube is located between the original PWA objective function and the shifted one. If it is possible to find an another PWA function, which is situated in this tube then a new control law can be created by triangulation and interpolation. The goal is to find such a function, which is defined over a smaller number of polytopic regions as the original parametric solution. We look for the slopes and affine parts of this PWA function. The problem can be formulated and solved as NLP. Since the computational complexity is high, this method is applicable mainly for lower dimensional problems.

The second introduced method uses the information about the function of control law. The idea is to create an approximated control law, but over less amount of regions. This approach could be divided into two procedures. In the first procedure the original MPC problem is solved and subsequently the same MPC with lower prediction horizon as well. It is important to incorporate the maximal control invariant set to the MPC constraints. This ensures that the feasible domain will

be always the same. Approximation of the control law is performed over the less complex domain. This is achieved by dividing the polytopic critical regions into simplices and by approximating the control law over the simplices. The approximation error is calculated as a sum of integrals over the simplices. The goal is find the local gain and affine part. This problem can be formulated as QP. The new approximated controller has better performance than the original simple control law, while its memory demand is lower than the complex controller's one.

We have shown that in both cases significant memory can be saved by applying the discussed approaches. This added value could bring advanced control technique such MPC even to low-cost hardware. However by using approximation techniques the memory demand is reduced, the user should realize that there is always trade-off between the memory demand and the performance of the controller.

Chapter 6

Code Generation for Explicit MPC

Explicit model predictive control (EMPC) as presented in Section 3.6 allows to implement MPC as a feedback policy just by evaluating the parametric solution to a given MPC optimization problem. This evaluation is performed by solving a point location problem using one of the algorithms reported in Section 3.7. The purpose of this chapter is to show how, for a particular EMPC solution, these algorithms can be automatically generated in the form of an executable code with little to no user effort.

Specifically, we present a new code generation module for the Multi-Parametric Toolbox (MPT) (Herceg et al., 2013a), which is a Matlab-based tool for setting up, solving, and analyzing EMPC controllers. Previously, MPT was only able to export EMPC solutions either to a pure Matlab code or to the ANSI-C language. However, nowadays we see a big expansion of other programming languages being used to create control-oriented applications. Therefore, the new code generation module allows to export EMPC controllers to the Python and JavaScript languages.

These languages were selected for multiple reasons. They are freely available, have a large and vital community, offer all the necessary mathematical routines to facilitate the EMPC implementation, and are progressive. However, only a small subset of the adopters of these languages are control engineers. Therefore, the aim of the proposed code generation framework is to give to the hand of a large community a tool for embedding high-quality control algorithms, represented by EMPC, into their applications in an easy-to-understand fashion.

This chapter is composed of two parts. First, in Section 6.1 we present the Python code generation framework. We guide the reader through all the steps that are necessary to synthesize an EMPC solution in the MPT toolbox and to generate the corresponding Python code that evaluates the EMPC solution. The procedure is illustrated on two real-life examples. The first one, described in Section 6.1.1, is a popular computer game where we show how to design and implement an MPC-based player. The second application in Section 6.1.2 deals with the control of a quadcopter. The JavaScript-based code generation is then introduced in Section 6.2 and illustrated on a simulation case study involving the control of a temperature inside a building.

6.1 Export to Python

In this section we present a new Python-based code generation module for the MPT toolbox. The module generates a self-contained implementation of various point-location algorithms that are used to evaluate a given EMPC solution for a known initial condition. Specifically, the module can generate Python versions of following point-location algorithms:

1. the sequential search algorithm for continuous feedback laws, cf. Section 3.7.1,
2. the extended sequential search procedure for discontinuous feedback laws, cf. Section 3.7.2,
3. the binary tree algorithm from Section 3.7.3.

The input to the code-generation framework is a Matlab object of the MPT toolbox that encodes the explicit representation of the PWA feedback law $u^\star(x)$ given by

$$u^\star(x) = F_i x + g_i \text{ if } x \in \mathcal{R}_i, \tag{6.1}$$

and the PWA or PWQ value function $J^\star(x)$, i.e., either

$$J^\star(x) = \alpha_i x + \beta_i \text{ if } x \in \mathcal{R}_i \tag{6.2}$$

in the case of a PWA value function or

$$J^\star(x) = x^\top \gamma_i x + \alpha_i x + \beta_i \text{ if } x \in \mathcal{R}_i \tag{6.3}$$

in the case of a PWQ value function. In all cases, $\mathcal{R}_i = \{x \mid A_i x \leq b_i\}$ are the polytopic regions of these functions.

Once these functions are computed using parametric programming and aggregated into a single object, the Python version of the sequential point-location algorithm of Section 3.7.1 can be obtained by a single line of code:

```
toPython(solution, 'myctrl', 'primal', 'first-region', library)
```

Here, `solution` is a Matlab variable which denotes the MPT object that contains the functions $u^\star(x)$ and $J^\star(x)$. Details on how to construct this object will be presented in Sections 6.1.1 and 6.1.2. Moreover, `'myctrl'` is a string which denotes the name of the file which should be generated. The suffix `'.py'` will be added automatically. In this example, the generated code will therefore be written into `'myctrl.py'`. Next, `'primal'` means that we want to export the primal optimizer, i.e., the feedback law $u^\star(x)$. Note that the module can export evaluation code for arbitrary PWA/PWQ functions besides the feedback law (e.g., the dual optimizer). The `'first-region'` string means that we want to use the sequential point-location algorithm as described in Section 3.7.1 that is stopped once the first region containing the given initial condition is found. We remind the reader that for this algorithm to work, the exported function must be continuous. Finally, the `library` can either be `library='numpy'` or `library='numpy-free'`. In the former case, the point-location algorithm will utilize the freely available `numpy` library to perform all necessary matrix/vector operations. In the latter case, the exported code does not depend on any external libraries and is therefore self-contained.

An example of the automatically generated code for `library='numpy'` is shown next:

```
from numpy import *
import math
def empc(x):
   xh=matrix([[0],[0],[0],[-1]],dtype=object)
   for i in range(0,len(x)):
      xh[i]=x[i]
   nx=3;nz=5;nu=1;num_regions=233;
   H=matrix([[-0.3881,0.8137,-0.4324,-0.0008],...])
   ni=matrix([[1],[9],[16],...])
   fF=matrix([[0.0051,0.0999],...])
```

```
fg=matrix([[0.0005],...])
for i in range(0,num_regions):
   if (H[ni[i]-1:ni[i+1]-1,:]*xh<=1e-08).all():
      z=fF[i*nz:(i+1)*nz,:]*x+fg[i*nz:(i+1)*nz]
      return z[0:nu]
   if i==num_regions-1:
      z=matrix([[float('NaN')]])
      return z
```

The code defines a Python function `empc` that takes one input argument - the initial condition `x` for which we want to evaluate the primal optimizer, i.e., to obtain $u^\star(x)$. Inside, the function defines the aggregated half-space representation of the critical regions in the variable `H`. Specifically, we have that

$$\mathcal{R}_i = \{x \mid A_i x \leq b_i\} \tag{6.4}$$

is rewritten as

$$\mathcal{R}_i = \{x \mid A_i x - b_i \leq 0\}, \tag{6.5}$$

which can then be aggregated as

$$\mathcal{R}_i = \{x \mid \underbrace{[A_i \; b_i]}_{H_i} \underbrace{\begin{bmatrix} x \\ -1 \end{bmatrix}}_{\tilde{x}} \leq 0\}. \tag{6.6}$$

The extended vector $\tilde{x}$ is denoted by `xh` in the code. After defining the half-spaces of the individual regions in the variable `H`, the number of half-space of each region in `ni`, and the parameters of local affine feedback laws in `tF` and `tg`, the code then iterates over all regions sequentially and checks whether $x \in \mathcal{R}_i$. If the condition is satisfied, $u^\star$ is computed by $u^\star = F_i x + g_i$ and the search is aborted. If no regions containing x is found, `NaN` is returned to indicate infeasibility.

If `library='numpy-free'` is used, the generated code no longer depends on the `numpy` library. Instead, the code contains two additional functions, called `mult` and `add`, which implement matrix/vector multiplication and addition, respectively. The idea here is for the code to be completely self-contained and not to depend on any external libraries. An example of the automatically generated code is provided next.

```
def mult(X,Y):
```

```
   result = [[sum(a*b for a,b in zip(X_row,Y_col)) for Y_col
      in zip(*Y)] for X_row in X]
   return result
def add(X,Y):
   result = [[X[i][j] + Y[i][j]  for j in range(len(X[0]))]
      for i in range(len(X))]
   return result
def empc(x):
   xh = list(x)
   xh.append([-1])
   nx=3;nz=5;nu=1;num_regions=233;
   H=[[-0.3881,0.8137,-0.4324,-0.0008],...]
   ni=[[1],[9],[16],...]
   fF=[[0.0051,0.0999],...]
   fg=[[0.0005],...]
   for i in range(0,num_regions):
      if max(mult(H[ni[i]-1:ni[i+1]-1],xh))[0]<=1e-08:
         z=add(mult(fF[i*nz:(i+1)*nz],x),fg[i*nz:(i+1)*nz])
         return z[0:nu]
      if i==num_regions-1:
         z=[[float('NaN')]]
         return z
```

The price to be paid when using the library-free option is a deteriorated computational speed. Specifically, the `numpy` library offers advanced and fast algorithms to perform matrix/vector multiplications while our library-free version only uses a simplistic approach to the problem.

The extended sequential algorithm described in Section 3.7.2 can be automatically generated as follows:

```
toPython(solution, 'myctrl', 'primal', 'obj', library)
```

The only difference to the preceding case is that the fourth input argument was changed from `'first-region'` to `'obj'`. Here the `'obj'` string refers to the cost function $J^\star(x)$ being used as a criterion to resolve conflicts if multiple regions $\mathcal{R}_i$ contain the initial condition x. This can easily happen e.g. when the feedback law is discontinuous. In such a case the value of the cost function for all these colliding

regions are evaluated and the region with the lowest value of the cost is used to compute the optimal control action. The automatically generated code for this type of implementation algorithm looks as follows:

```
tH=matrix([[1.7156,0.3652,3.7800],...]);
tF=matrix([[0.0051,0.0999],...]);
tg=matrix([[0.0005],...]);
tb=array([]);
for i in range(0,num_regions):
    if (H[ni[i]-1:ni[i+1]-1,:]*xh<=1e-08).all():
        tv=tF[i,:]*x+tg[i];
        tv=tv+x.T*tH[i*nx:(i+1)*nx,:]*x;
        tb=insert(tb,shape(tb),i);
        tb=insert(tb,shape(tb),squeeze(asarray(tv)));
num_reg=shape(tb);num_reg=num_reg[0]/2;
tb=tb.reshape(num_reg,2);
if tb.sum()!=0:
    i=int(tb[tb[:,1].argmin(),0]);
    z=fF[i*nz:(i+1)*nz,:]*x+fg[i*nz:(i+1)*nz];
    return z[0:nu]
else:
    z=matrix([[float('NaN')]])
    return z
```

Here, the variables `tH`, `tF`, and `tg` are the quadratic, linear and, constant parameters of the (in this case PWQ) value functions. The `for` cycle together with the `if` condition are used to check whether parameter `x` is a part of the given region. If there is such a region, then the value of the objective function is saved into variable `tv`, while the index of the region is saved into `tb`. If the `list tb` is empty there is no feasible solution for the given initial point `x`. If there are more elements in `list tb` the values of the objective functions are compared, and the index `i` with the minimum value is chosen. The index is used to compute the optimal control action from the local affine law.

The last option is to generate the Python code of the binary tree point location algorithm as described in Section 3.7.3. This is achieved by calling

```
toPythonTree(tree, 'mytree', 'primal', 'library')
```

where `tree` is an MPT variable which contains the pre-compute binary tree, `'mytree'` is the name of the exported file with the `'.py'` suffix being added automatically, and `'primal'` is the name of the function we wish to export. In this case `'primal'` refers to the primal optimizer, i.e., to the feedback law $u^\star(x)$. The exported code that the following form:

```
from numpy import *
import math
def empctree(x):
    xh=matrix([[0],[0],[-1]],dtype=object)
    for i in range(0,len(x)):
        xh[i]=x[i]
    T=matrix([[0.2080,0.9781,-0.1899,2,55],...])
    fF=matrix([[0.0051,0.0999],...]);
    fg=matrix([[0.0005],...]);
    i=0;nz=5;nx=2;nu=1;z=matrix([[float('NaN')]]);
    while True:
        h=T[i,0:3]*xh
        if h<=0:
            i=T[i,3]
        else:
            i=T[i,4]
        if i<0:
            i=-i
            i=i-1
            z=fF[i*nz:(i+1)*nz,:]*x + fg[i*nz:(i+1)*nz]
            return z[0:nu]
        elif i==0:
            return z
        i=i-1
```

Here, the variable `T` contains information about the values of the logical tests in each node of the binary search tree. In the `while` loop logical tests are performed and based on the result this test `h` the index `i` of the next node is obtained. If the value of index `i` is a negative value it means that the algorithm achieved the leave of the tree. In this case the local control action could be implemented.

To embed the automatically generated codes into the target Python application all that needs to be done is first to include the code via

```
import empc
```

or

```
import empctree
```

in the case of the binary tree, followed by calling the corresponding function, i.e.

```
u = empc(x)
```

in the case of the sequential search procedure, or

```
u = empctree(x)
```

for the binary tree.

In the following two sections the code generation procedure is illustrated on two examples. The first one, described in Section 6.1.1, is a popular computer game where we show how to design and implement an MPC-based player. The second application in Section 6.1.2 is concerned with the control of a quadcopter.

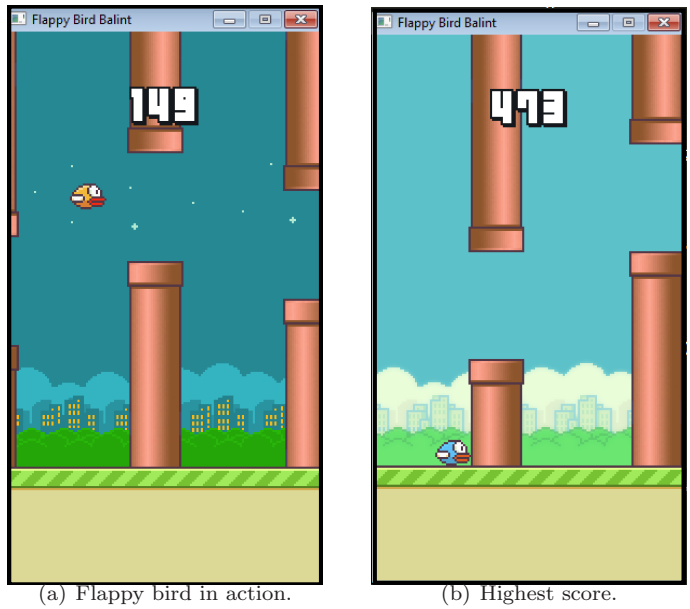
6.1.1 Flappy Bird

Flappy Bird is a popular computer game, where the player's goal is to control the altitude of the bird while try to keep away from the obstacles. The obstacles are graphically presented as pipes, while there is a gap between the upper and the lower pipe. The height of the pipes are generated randomly, and there is no information about the future values of the pipes height. The number of successfully passed pipes are the indicators of the score. The screenshots of the investigated game is provided in Figure 6.1(a). The game is running in fixed sampling frequency.

Modeling

The goal was to create a controller, which can replace a human player. At first, the mathematical model of the dynamics was proposed, which was defined by a simple affine difference equation presented in (6.7).

$$x_{k+1} = x_k + bu_k - f, \tag{6.7}$$



(a) Flappy bird in action. (b) Highest score.

Figure 6.1: Screenshots of the Flappy Bird game.

State variable x_k stands for the actual altitude of the bird, while x_{k+1} is the altitude in the next step. The input of the system represented by variable u_k is considered to have binary values. Specially, u_k can acquire two different values

$$u_k \in \{0, 1\}. \tag{6.8}$$

The constants b and f were extracted from the game. The value of the affine part of dynamics in (6.7) was set to $f = -9$, which means the bird is permanently falling in each step with the same value. In order to compensate the falling the value of variable b was set to 18. This means if the control is not activated the bird's altitude decreases by 9 pixels, while if the control is activated the height increases by 9 pixels. In order to create as simple MPC as possible instead of obstacle avoidance reference tracking technique was implemented. Such an implementation does not require to use time varying constraints, while the time varying reference was calculated as a middle point of the gap located between the lower and the upper pipe.

MPC Design

After the construction of the mathematical model the designed optimal control problem is of the following form

$$\min \sum_{k=0}^{N-1} \|x_k - x_{\text{ref}}\|_1, \tag{6.9a}$$

$$\text{s.t. } x_{k+1} = x_k + bu_k - f, \tag{6.9b}$$

$$u_k \in \{0, 1\}, \tag{6.9c}$$

$$x_0 = x(t). \tag{6.9d}$$

Since the constraints of MPC presented in (6.9) are not convex due to binary character of the input variable (6.9c), it cannot be solved as a convex optimization problem. Fortunately this non-convexity has a special form and the optimal control problem defined in (6.9) can be formulated as a mixed-integer programming problem, which is easier to solve than a general non-convex optimization problem. This optimal control problem can easily be implemented by using only a few lines of code in Matlab by using the features of MPT

```
model = LTISystem('A', 1, 'B', 18, 'f', -9)
model.u.with('binary')
model.x.with('reference')
model.x.reference = 'free'
model.x.penalty = OneNormFunction(1)
N = 4
mpc = MPCController(model, N)
```

The function `LTISystem` in the first line is a built-in function of the Multi-Parametric Toolbox and it creates an instance of the object `LTISystem` that describes the affine system (6.9). After the instance in created additional properties can be set up. At first the binary nature of the input variable is defined, with the second line. After the next line changes the objective function in order to activate the reference tracking mode, and since the reference during the control is not constant it can be set by the following line the free. This means that it can be changed in every control step. In the model definition the final step is to set the weighting matrices of the objective function. In this particular case we use 1-norm for penalizing the distance of state variable from the desired setpoint.

After the all necessary information about the model are set and the prediction horizon is defined a built-in `MPCController` function is called to create an instance which contains information about the optimal control problem defined in (6.9). This function transforms the defined optimal control problem into an optimization problem in such a form, which is feasible for solvers. In this case the MPC problem can be solved by online solvers by calling `mpc.evaluate(x)` command. In order to receive explicit MPC, the optimization problem arise from (6.9) needs to be solved parametrically. This procedure requires numerous mathematical operations which especially in higher dimensions are not straightforward. The way of obtaining explicit form of MPC was presented in Section 2.9.Thanks to Multi-Parametric Toolbox this not straightforward procedure can be performed by evaluating one single line of code

```
empc = mpc.toExplicit()
```

This convenient way of defining the MPC problem and subsequent user friendly way of constructing the parametric solution is one of the biggest advantages of Multi-Parametric Toolbox. The next step of creating the binary search tree from explicit representation is done similarly by evaluating a single line in Matlab

```
tree = BinTreePolyUnion(empc.optimizer)
```

where variable `tree` contains all the information necessary for the proper evaluation. The code generation module works on the similar manner.

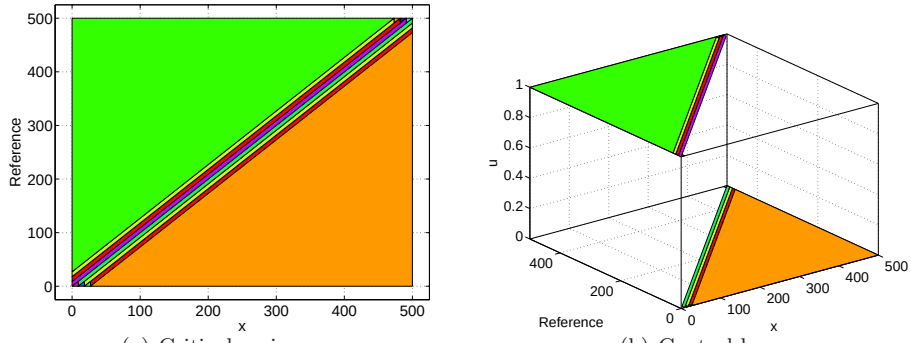
Code Generation and Implementation

In order to obtain the point location algorithm based on binary search tree the export function must be executed

```
toPythonTree(tree, 'flappympctree', 'primal')
```

By this command a new file `flappympctree.py` is created in the actual folder which contains the generated algorithm and data. The merging of the generated code into the application was extremely easy. Basically it works on the plug-and-play principle. The only thing what was changed is just the condition, which checked if the human player pressed the button

```
if event.type == KEYDOWN
```



(a) Critical regions. (b) Control law

Figure 6.2: Flappy: Binary input.

was replaced by a function evaluation

```
if flappympctree([[altitude],[reference]]) == 1
```

This means the merging of the code with the actual application could be done only by changing the function. The parametric solution is depicted in Fig. 6.2 , where Fig. 6.2(a) represents the critical regions and Fig. 6.2(b) the control law defined over the critical regions.

This controller was then run several times and the reached scores are summarized in Table 6.1. As can be seen from the table, the MPC-based artificial player significantly outperforms the performance of human-based players by one order of magnitude. This demonstrates viability of the proposed approach.

Summary

The goal of this section was to introduce the code export module in action. We wanted to show how could be MPC implemented in a non typical application like a computer game. We presented a way how to defined the mathematical model of the bird's dynamics. In this particular case the binary search tree point location method was used. The generated code was inserted into the application and subsequently it was evaluated by a single line of code. The final phase was the test of the designed controller. 3 human players were asked to try to control the height of the bird. They had a possibility to try it several times. Table 6.1 summarizes the average and the highest scores achieved by the human players and also the MPC based artificial controller.

Table 6.1: Achieved score in Flappy Bird

Players	Average score	Best score
Human no. 1	16	33
Human no. 2	14	19
Human no. 3	9	17
MPC	182	473

As it can be seen the artificial controller reached incomparably higher scores than the human players. This is due to the advantage of better reaction time. The goal in this application was not to design the best possible controller as in general used to be the goal, but the way of introducing a user friendly method how to merge an advanced control technique with an arbitrary application in Python.

6.1.2 Quadrocopter

The results described in this section are one of the outcomes of the joint work with Juraj Števek from our Institute. In the previous section the EMPC control technique was implemented on the computer game. Since the computer game was running in the fixed frequency, which means each operation were performed in the given order regardless of their computation time, we did not have to care too much about the computation time. In this section we aimed an another type of application, where the fast evaluation of EMPC comes to be useful. The system on which we wanted to demonstrate the further applicability of the presented code generation module is an Ar.Drone2 quadrotor (Bristeau et al., 2011), which is depicted in Fig 6.3. In order to demonstrate the applicability of the code generation module the control of the yaw angle will be presented in sequel.

AR.Drone2 is a commercial quadrotor platform that is freely available in numerous retail stores. The quadrotor is fitted with four propellers driven by four brushless motors. The supply is provided by Lithium-Polymer battery (1000-1500 mAh). The quadrotor is also equipped with rich set of sensors: 3-axis gyroscope, accelerometers, ultrasound altimeter, electronic compass, pressure sensor, QVGA downward camera (320x240, 60fps). Additionally the drone has a front 720p HD 93° wide-angle camera with 640x480 resolution at 30fps. Information from these sensors is used in the on-board embedded control. The on-board system ensures

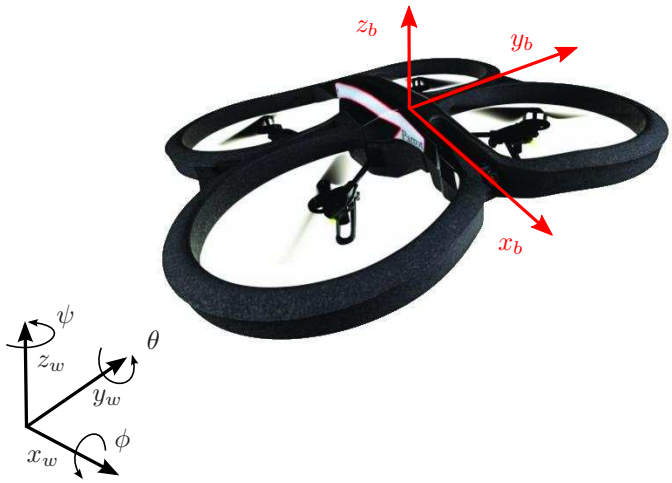


Figure 6.3: The AR.Drone2 and its coordinate systems. The arrows of the rotation angles correspond to positive values.

gravity compensation and stabilization of the drone. Some details about technology inside the AR.Drone2 are given by Bristeau et al. (2011). The normalized control command for the on-board system is

$$u = (u_\phi, u_\theta, u_{\dot{\psi}}, u_{\dot{z}})^\top \in [-1, 1]^4 \tag{6.10}$$

where u_ϕ is the reference for roll ϕ, u_θ is the reference for pitch θ, $u_{\dot{\psi}}$ is the reference for yaw rate $\dot{\psi}$ and $u_{\dot{z}}$ is the reference for vertical speed $\dot{z}$.

The on-board software runs three control loops. The innermost control loop (Angular Rate Control Loop – ARCL) drives rotational speed of propellers by a reference (proportional control). The second loop (Attitude Control Loop – ACL) with PI control maintains required attitude angles (given by the user or zero in a hovering state). The output of this loop serves as the reference for ARCL. The outermost loop (Hovering Control Loop – HCL) regulates the drone to the hovering state.

The communication with AR.Drone2 is provided by WiFi. After powered on, the AR.Drone2 becomes a wifi hotspot and an external client may connect to it. The communication is realized over three channels. The first channel (UDP 5554) serves for sending drone's data from sensors and the current state estimate (Navdata). Through the second channel (UDP 5555), the video stream from one of

two cameras is processed. The client's commands (e.g. takeoff, land, desired roll, pitch, yaw speed and vertical speed) are transferred through the command channel (UDP 5556). The AR.Drone2 is controlled using a device with a mobile operation system (Android, iOS, Windows Phone) equipped by FreeFlight application, see e.g. ardrone2.parrot.com/support-android. The application offers to control the drone manually or to use a navigation with a GPS module extension. The application shows the video stream from the front camera (or the bottom camera) in the real time in such a way the user has a view from drone's perspective. Robot Operating System (ROS) (M. Quigley and Faust, 2009) was used as an interface to control and communicate with the drone. It was developed at Stanford Artificial Intelligence Laboratory and it is a collection of numerous frameworks for creating a robot control software. ROS offers Python interface to implement different user defined algorithms to control the drone. For the data exchange a built in message package is used. The message is a special type of data structure, which stores data.

Modeling

Raffo et al. (2010) presented a standard approach to describe the quadrotor's dynamics. It is based on Newton-Euler and Euler-Lagrange equations. The resulting model comprises a set of nonlinear and unstable differential equations. The vector of angular speeds $(\omega_1, \omega_2, \omega_3, \omega_4)^\top$ of four propellers is the input of such model. Bresciani (2008) proposed a simple (quadratic) transformation of propellers' speeds that lead to an input vector that consists of four basic movements (throttle, roll, pitch, yaw). Parameters of such model are quadrotor's mass, gravitational acceleration, inertias, friction constants etc. The identification of these parameters is provided by a series of identification experiments presented by Bresciani (2008); Qianying (2014).

In the case of AR.Drone2, the input vector (6.10) consists of references for roll, pitch, yaw rate and vertical speed. The on-board control software drives these variables to the reference values. Since the details of this control are unknown, it is considered as a part of the system. We consider the state vector

$$\vec{x} = (x, y, z, \dot{x}, \dot{y}, \dot{z}, \phi, \theta, \psi, \dot{\psi})^\top \in \mathbb{R}^{10} \tag{6.11}$$

where $(x, y, z)^\top$ is the position of the quadrotor (in m) and $(\dot{x}, \dot{y}, \dot{z})^\top$ the velocity (in m/s), both in the world frame. Next, $(\phi, \theta, \psi)^\top$ is the attitude (in rad). Finally,

$\dot{\psi}$ is the angular velocity around the z_w axis (in rad/s). We consider the state space model of the form

$$\begin{aligned} \dot{\vec{x}} &= \vec{f}(\vec{x}, \vec{u}) + \vec{w} \\ \vec{z} &= \vec{h}(\vec{x}) + \vec{v} \end{aligned} \tag{6.12}$$

where $\vec{f}$ and $\vec{h}$ are differentiable nonlinear functions and $\vec{w}$ and $\vec{v}$ represent multivariate Gaussian noise.

Here we adopt the idea introduced by Engel et al. (2012). We approximate the horizontal acceleration vector $(\ddot{x}, \ddot{y})^\top$ by a nonlinear transformation of the attitude vector. The transformation results from the following idea.

The acceleration in the horizontal direction is roughly proportional to the horizontal force acting on the quadrotor

$$\begin{pmatrix} \ddot{x} \\ \ddot{y} \end{pmatrix} \sim \vec{f}_{\text{acc}} - \vec{f}_{\text{drag}} \tag{6.13}$$

where $\vec{f}_{\text{acc}}$ denotes an accelerating force and $\vec{f}_{\text{drag}}$ denotes an aerodynamic drag force. The drag force is related to the velocity in the horizontal direction and $\vec{f}_{\text{acc}}$ relates to the tilt angle (the projection of z_b-axis onto the horizontal plane).

$$\ddot{x} = K_1(\sin\psi \sin\phi \cos\theta - \cos\psi \sin\theta) - K_2\dot{x} \tag{6.14}$$

$$\ddot{y} = K_3(-\cos\psi \sin\phi \cos\theta - \sin\psi \sin\theta) - K_4\dot{y} \tag{6.15}$$

Under a condition of a near-hover flight, it is possible to model and identify the forward, lateral, yaw, and altitude dynamics as four independent sub-systems. Engel et al. (2012) and Krajník et al. (2011) model the AR.Drone's internal control and dynamics as four independent linear systems

$$\dot{\phi} = K_5 u_\phi - K_6\phi, \tag{6.16}$$

$$\dot{\theta} = K_7 u_\theta - K_8\theta, \tag{6.17}$$

$$\ddot{\psi} = K_9 u_{\dot{\psi}} - K_{10}\dot{\psi}, \tag{6.18}$$

$$\ddot{z} = K_{11} u_{\dot{z}} - K_{12}\dot{z}. \tag{6.19}$$

V120 tracks the quadrotor in user's defined world frame. The direct observation is given by

$$\boldsymbol{h}(\vec{x}) = (x, y, z, \phi, \theta, \psi)^\top. \tag{6.20}$$

Identification of yaw and height dynamics are simpler problems compared to horizontal dynamics. In this case, we look for parameters of an unstable second order system. We may use basic procedures of the MATLAB's System identification toolbox (Ljung, 2011) to find the unknown parameters or we may proceed in the same way like in the two previous dynamics.

The corresponding NLS optimization problem for the yaw dynamic is of the form:

$$\begin{aligned}
\min_{\bar{K}} \; & Q_{LS} = \sum_{i=1}^{N} (\dot{\psi}_i - \tilde{\dot{\psi}}_i)^2 + (\psi_i - \tilde{\psi}_i)^2 \\
s.t. : & \bar{K}_{\min} \leq \bar{K} \leq \bar{K}_{\max}, \\
& \frac{\mathrm{d}\psi}{dt} = \dot{\psi} \\
& \frac{\mathrm{d}^2\psi}{dt^2} = K_9 u_{\dot{\psi}} - K_{10}\dot{\psi}, \\
& 0 = \begin{pmatrix} \psi(t_0) & \dot{\psi}(t_0) \end{pmatrix}^\top, \\
& u_{\dot{\psi}} = \Pi(t), \quad t \in [t_0, t_f],
\end{aligned} \tag{6.21}$$

where $\bar{K} = (K_9, K_{10})^\top$, $\bar{K}_{\min}$ and $\bar{K}_{\max}$ denote the lower and upper bound for the parameter vector, respectively. N is the number of measurements.

The NLS optimization problem for the altitude dynamic is of the form:

$$\begin{aligned}
\min_{\bar{K}} \; & Q_{LS} = \sum_{i=1}^{N} (\dot{z}_i - \tilde{\dot{z}}_i)^2 + (z_i - \tilde{z}_i)^2 \\
s.t. : & \bar{K}_{\min} \leq \bar{K} \leq \bar{K}_{\max}, \\
& \frac{\mathrm{d}z}{dt} = \dot{z} \\
& \frac{\mathrm{d}^2\psi}{dt^2} = K_{11} u_{\dot{z}} - K_{12}\dot{z}, \\
& 0 = \begin{pmatrix} z(t_0) & \dot{z}(t_0) \end{pmatrix}^\top, \\
& u_{\dot{z}} = \Pi(t), \quad t \in [t_0, t_f].
\end{aligned} \tag{6.22}$$

where $\bar{K} = (K_{11}, K_{12})^\top$, $\bar{K}_{\min}$ and $\bar{K}_{\max}$ denote the lower and upper bound for the parameter vector, respectively. N is the number of measurements.

The onboard software implements basic control loops based on the measurements from various sensors, such as accelerometers, ultrasound sensor, barometer,

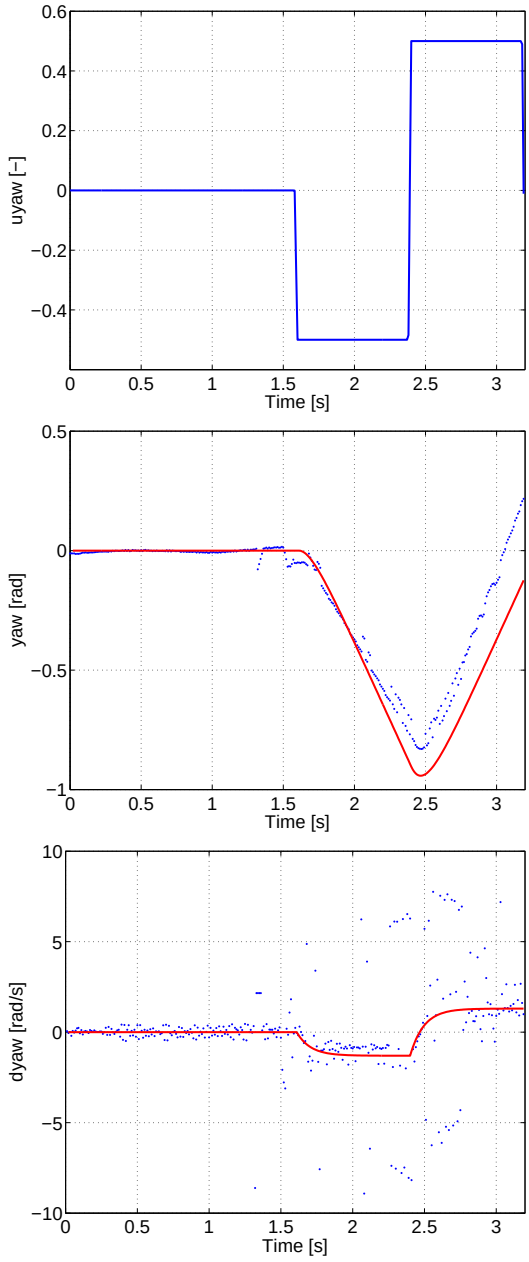


Figure 6.4: The result of the yaw dynamics identification experiment. Blue – measurements, Red – model.

and electronic compass. It controls the vertical speed $\dot{z}$, yaw speed $\dot{\psi}$, roll ϕ and pitch angle θ based on references it acquires via WiFi. The MPC objective is to devise these references in an optimal fashion. Specifically, the control inputs generated by MPC are:

- $u_{\dot{z}}$, the control command for velocity in vertical axis;
- $u_{\dot{\psi}}$, the control command for rotation velocity in z axis;
- u_{ϕ}, the control command for rotation over the x axis;
- u_{θ}, the control command for rotation over the y axis.

All inputs are normalized and constrained by $-1 \leq u \leq 1$.

In this section we control the yaw angle with respect to the global frame. The simplified mathematical model of the Ar.Drone's rotational dynamic over the z-axis was presented in Engel et al. (2012):

$$\ddot{\psi} = K_9 u_{\dot{\psi}} - K_{10}\dot{\psi}, \tag{6.23}$$

where $\dot{\psi}$ represents the linear acceleration over the z-axis. The parameters K_9, K_{10} are model gains that have to be experimentally identified. The first term $K_9 u_{\dot{\psi}}$ represents accelerating force, and $K_{10}\dot{\psi}$ represents drag force in (6.23).

Applying the forward Euler discretization to (6.23), the discrete time model of the system takes the form

$$\underbrace{\begin{bmatrix}\dot{\psi}_{k+1}\\ \psi_{k+1}\end{bmatrix}}_{x_{k+1}} = \underbrace{\begin{bmatrix}1-K_{10}T_s & 0\\ T_s & 1\end{bmatrix}}_{A}\underbrace{\begin{bmatrix}\dot{\psi}_k\\ \psi_k\end{bmatrix}}_{x_k} + \underbrace{\begin{bmatrix}K_9 T_s\\ 0\end{bmatrix}}_{B}\underbrace{u_{\dot{\psi}_k}}_{u_k} \tag{6.24}$$

where T_s is the sampling period. To achieve tracking properties, the system in

Table 6.2: Estimated parameters and deviations.

Parameters	Values ± std	Units
K_9	27.21 ±11.46	s^{-1}
K_{10}	10.45 ±5.17	rad s^{-1}

(6.24) was augmented into the form

$$\tilde{x}_{k+1} = \underbrace{\begin{bmatrix} A & B \\ 0 & I \end{bmatrix}}_{\tilde{A}} \tilde{x}_k + \underbrace{\begin{bmatrix} B \\ I \end{bmatrix}}_{\tilde{B}} \Delta u_k, \tag{6.25}$$

where $\tilde{x}_k = \begin{bmatrix} \dot{\psi}_k & \psi & u_{k-1} \end{bmatrix}^\top$.

Control

Using the Multi-Parametric Toolbox we have formulated the following MPC problem, the objective of which is to manipulate the rotational velocity such that the yaw angle tracks a prescribed reference. This is achieved by formulating the problem as follows:

$$\min \sum_{k=0}^{N-1} \Delta y_k^\top Q_{y_k} \Delta y + \Delta u_k^\top Q_u \Delta u_k, \tag{6.26a}$$

$$\text{s.t. } \tilde{x}_{k+1} = \tilde{A}\tilde{x}_k + \tilde{B}\Delta u_k, \tag{6.26b}$$

$$y_k = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tilde{x}_k, \tag{6.26c}$$

$$\Delta y = y - y_{\text{ref}}, \tag{6.26d}$$

$$\Delta u_k = u_k - u_{k-1}, \tag{6.26e}$$

$$-0.005 \leq \Delta u_k \leq 0.005, \tag{6.26f}$$

$$-1 \leq u_k \leq 1, \tag{6.26g}$$

$$\tilde{x}_{min} \leq \tilde{x}_k \leq \tilde{x}_{max}, \tag{6.26h}$$

$$x_0 = x(t). \tag{6.26i}$$

The model gains are $K_9 = 27.21$, $K_{10} = 10.45$, while the state penalty matrix $Q_y = \left[\begin{smallmatrix} 500 & 0 \\ 0 & 2000 \end{smallmatrix}\right]$ and $Q_u = 1$ were chosen experimentally. The weights were adjusted to put emphasis to ψ and slightly restrict $\dot{\psi}$. The prediction horizon $N = 5$ and sampling time $T_s = 20\,\text{ms}$ were used.

The explicit MPC feedback was then generated using parametric solvers contained in Multi-Parametric Toolbox. The solution consisted of 281 regions. The controller was subsequently exported to Python code using the `toPython` function and embedded into the control software represented by the ROS platform.

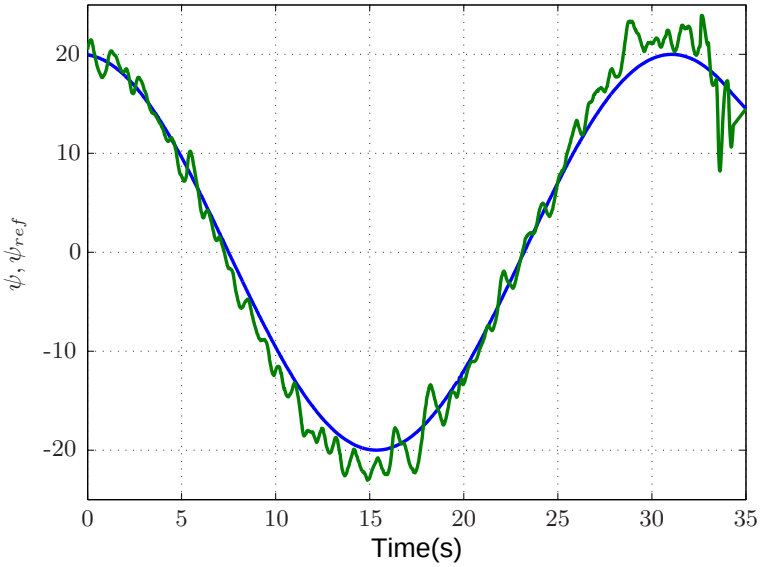


Figure 6.5: Controlled yaw angle (green) and the corresponding reference (blue).

The controller was tested in a real-time experiment with control period $\sim 20\,\text{ms}$. The real control period varied from 15 to 27 milliseconds, which introduced additional disturbances into the control loop.

The MPC problem was defined by using the features of Multi-Parametric Toolbox in the same way as it was presented in the previous application. The parametric solution was subsequently exported to Python by the `toPython` function and embedded into control interface in ROS platform.

The variation of the control period was caused mostly by the varying evaluation time of the sequential search. Results of the tracking experiment are shown in Figs. 6.6 and 6.5. We have chosen not a single reference point, but a curve as can be seen in Fig.6.5. This represents the horizontal turning. As can be seen, the explicit MPC controller rejects disturbances and achieves the tracking objective by bringing ψ to a harmonic reference. The small velocity oscillations were caused by air turbulence and varying control period.

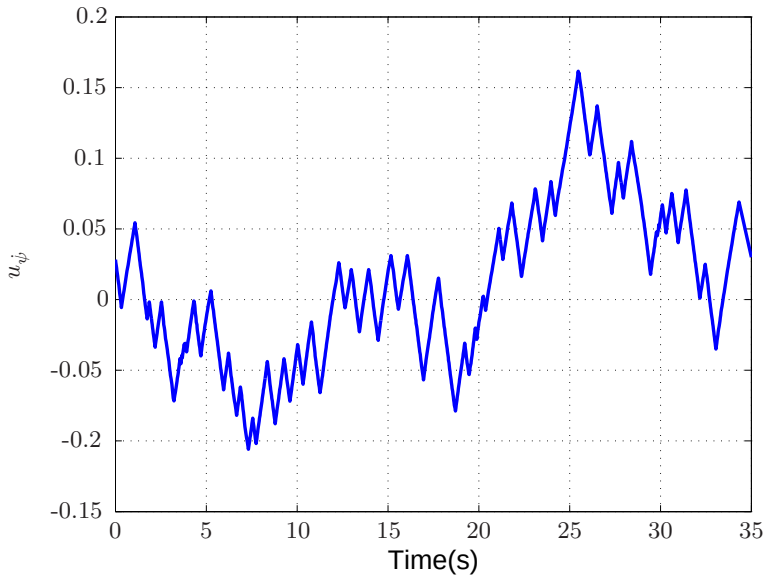


Figure 6.6: Control input for the Ar.Drone example.

6.1.3 Summary

In this section a the advantage of the new code generation module in Multi-Parametric Toolbox was primarily introduced. The presented module allows to the final user of the toolbox to extract EMPC controllers into Python. The generated file also contains the point location algorithm. Furthermore, such a generated code is directly implementable into different applications. In this section two different applications were introduced. The first one is a popular computer game, where the goal was to control altitude with MPC, while in the second application we aimed to control the yaw angle of the quadrocopter. In both cases the MPC control technique was successfully implemented.

6.2 Export to JavaScript

In the recent days JavaScript became a popular programming language. It can be used as front-end and even as a back-end tool. However it is a scripting language, it also allows to use the features of object oriented programming. The main reason of using JavaScript is that it allows to complete procedures or requests in the user's

browser without sending messages back and forth to the server. Doing something directly in the browser can be much faster than sending a message to the server and getting a reply. It is almost always a noticeable time lag, and can take many seconds. Generally JavaScript is used to create interactions and dynamic effects, but it could be used as a part of application outside web pages. It becomes so powerful language that even computer games, visualization or 3D rendering could be developed in JavaScript. The reason of choosing this language is its unique possibility to easily merge with web based applications. JavaScript is used by every company, which has a website. There exist several frameworks, such as

- JSON - easier-to-use alternative to XML for storing and exchanging data
- AngularJS - extends HTML with new attributes (Ryanair, iTunes Connect, PayPal Checkout, ...)
- jQuery - simpler event handling

and many more, which allows fast and convenient algorithm prototyping. Our goal was to use the web based environment as a communication tool for control application. Therefore we proposed to create a module for an export of the point location methods into JavaScript programming language. With this act we can bring advanced control techniques such MPC to the web based environment. We believe that there is a need for simple code export, which brings the power of the parametric optimization into web based environment. Since JavaScript could be used for both front-end and back-end development, the possibilities for implementation seem to be endless. The generated code should be fully self-contained, which means there is no need for external libraries. The generated function should contain all the necessary matrix operation and it should work on plug-and-play system, where the user just needs to attach the generated file with `'.js'` extension to it's application. The principle of code generation of JavaScript has similar features as was presented in code generation to Python. The only difference is in the name of the built in function, `toJavaScript`, for the code export. The function generates two different files, one JavaScript file which includes the information about the parametric solution, the point location algorithm and the necessary functions for matrix operations. There are also free libraries that allow to use matrix operations, but those libraries code could be measured in megabytes. Therefore we created our own functions and only those are exported together with the parametric solution.

The second file is a pure HTML file, where the user can manually test the exported controller. To our best knowledge, there has not been any simple way to merge MPC control strategies into existing JavaScript applications in an easy fashion. To demonstrate the benefit of the export procedure to JavaScript a temperature control for a single zone building application will be introduced in sequel.

In Matlab the export procedure could be performed by evaluating the following line:

```
toJavaScript(solution, 'buildingmpc','primal','algorithm')
```

Where variables `solution`, `'primal'` and `'algorithm'` has the same options and properties like it was presented in Section 6.1. Furthermore `'buildingmpc'` is the name of the generated function. If this string based variable does not contain the `.js` extension it is added automatically.

The exported file id of the following form: the created file contains the exported sequential search approach, presented in Section 3.7.1 with the information about the parametric solution.

```
function buildingmpc(x){
var nx=7;
var nz=5;
var nu=1;
var H=[[-0.27105,0.96257,3.5677,...],...];
var ni=[[1],...];
var fF=[[5.5511e-17,...],...];
var fg=[[0],...];
x = trans(x);
var xh = x.slice();
xh.push([-1]);
var z = [];
var num_regions = 35;
for (var i = 0; i < num_regions; i++) {
    if(mulM(subst(H,ni[i]-1,ni[i+1]-1,0,nx+1),xh).
      every(isGreaterZero)){
        z = addM(mulM(subst(fF,i*nz,(i+1)*nz,0,nx),x),
           subst(fg,i*nz,(i+1)*nz,0,1));
        break
```

```
    }
}
```

where variable `nx` stands for state, `nz` represent the number of optimized variables, while `nu` stands for the number of inputs. Variables `H`, `ni`, `fF` and `fg` are array type variables and contain information about the parametric solution. The exported code contains sequential search algorithm. The used functions in the algorithm performs the following operations:

- `mulM` - matrix multiplication
- `addM` - matrix addition
- `subst` - selects the given part of the array
- `isGreaterZero` - element wise comparison of the array
- `trans` - transposition

All the mentioned functions are also the part of the generated function. In case the user is interested in the extended sequential search, presented in Section 3.7.2, it can be exported by executing the following command:

```
opt.toJavaScript('buildingmpc','primal','obj')
```

The exported function is saved in `buildingmpc.js` and is of the following form:

```
function buildingmpc(x){
var nx=7;
var nz=5;
var nu=1;
var H=[[-0.27105,0.96257,3.5677,...],...];
var ni=[[1],...];
var fF=[[5.5511e-17,...],...];
var fg=[[0],...];
var tH=[[2.2734,0.98655,...],...];
var tF=[[1.0313,1.0519,...],...];
var tg=[[0.48411],...];
x = trans(x);
var xh = x.slice();
```

```
xh.push([-1]);
var tb = [];
var z = 0;
var k = 0;
var num_regions = 35;
for (i=0;i < num_regions; i++){
   if (mulM(subst(H,ni[i]-1,ni[i+1]-1,0,
          nx+1),xh).every(isGreaterZero)){
      z = addM(mulM(subst(tF,i,i+1,0,nx),x),subst(tg,i,i+1,0,1));
      z = addM(z,mulM(mulM(trans(x),
         subst(tH,i*nx,(i+1)*nx,0,nx)),x));
      tb[k] = [];
      tb[k][0] = i;
      tb[k][1] = z;
      k++;}
}
if (tb.length==0)
   {return nu2nan(nu)}
else{
   i = tb[index(subst(tb,0,tb.length,1,2))][0];
   z=addM(mulM(subst(fF,i*nz,(i+1)*nz,0,nx),x),
      subst(fg,i*nz,(i+1)*nz,0,1));
   return subst(z,0,nu,0,1)}
}
```

The functions used in this function are same that we described in the previous code sample. The result of both generated functions is the optimal control action, which is the same for the same initial parameter `x` (3.4c) regardless on what kind of point location algorithm is used to extract it from the parametric solution. The initial parameter `x` enters to the point generated file `u = buildingmpc(x)` as an one dimensional array, while the sequence of control actions `u`, which is returned by function `buildingmpc(x)` is a one dimensional array.

6.2.1 Building Temperature Control

This part presents the results of an actual cooperation with Ján Drgoňa and Martin Kalúz from our Institute. Temperature control in the last years became a topic of common interest since high amount of energy is used to create sufficient conditions in the buildings. This high amount of energy is consumed by heating, ventilation and cooling the air. In order to save energy two things can be done. Better thermal isolation could be installed on the building in order to reduce its energy consumption. This is time consuming and requires big financial investments. On the other hand more advanced control methods should be used to control the thermal comfort in the building. In the literature there exist several ways how to deal with the design of advanced control approaches. Results of successful implementation of MPC by assuming weather forecast were introduced in Oldewurtel et al. (2012). Another approach which uses stochastic MPC formulation is discussed in Ma et al. (2012). The combination of the weather forecast and stochastic MPC is demonstrated in Oldewurtel et al. (2010). Experimental results on a real building control are described in Široký et al. (2011). Results presented in Klaučo et al. (2014) introduce the implementation of MPC-like feedback policy by using machine learning tools. Another paper (Klaučo and Kvasnica, 2014) describes how to implement EMPC with so-called PMV index. This index considers not only temperature, but it can also combine more factors such as humidity, ventilation in order to describe the person's thermal comfort. In general, the goal is to create such a controller which maximizes the thermal comfort while there is an emphasis on energy consumption minimization.

As an application to demonstrate the advantage of code generation to JavaScript a thermal control of a single zone building was considered. In this set up we assumed that the exported EMPC controller is located in the mobile phone, while the communication with the simulation model which represents the behavior of the building is performed by WebSocket. This protocol provides full-duplex communication channel.

The proposed control loop is depicted in Fig. 6.7.

In this case study we did not have a chance to control a real building, but the simulation of the building model that runs in the hardware-in-the-loop fashion. The building runs on the Arduino YÚN micro-controller 6.8 and it is implemented in a form of discrete differential equations. The model is evaluated in real time, but

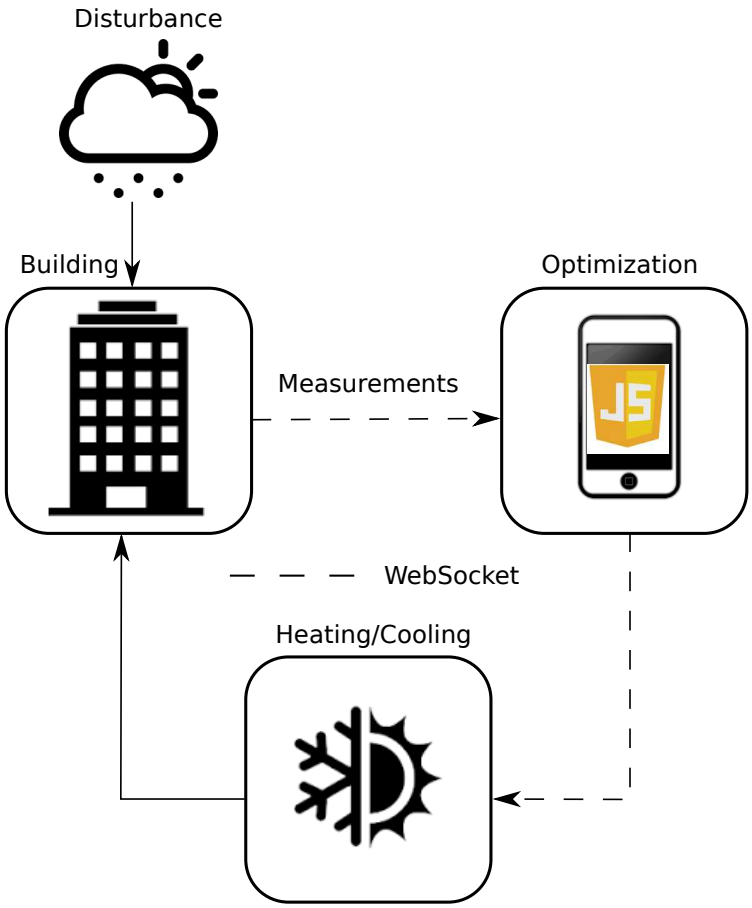


Figure 6.7: Control loop.

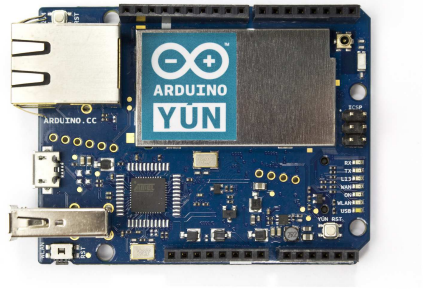


Figure 6.8: Arduino YÚN.

time is fasten up in order to shorten the simulation time. Arduino YÚN has two different computing environments. The first is an 8-bit AVR microprocessor, where the building's model is running. This communicates via serial port with another integrated ARM computer with a light Linux distribution for embedded systems OpenWrt-Yun.

Both of them are located on one motherboard. On this system a Python based software is running, which ensures two tasks:

- handling a communication with the client

  On the client side the EMPC is running. It uses HTTP protocol for a full duplex WebSocket communication.

- data transmission

  This ensures the communication between the control application and the low level AVR microcontroller.

In our simulation setup the simulation data was stored on the Micro-SD card, and after the end of the simulation the data could be downloaded and studied after the experiment. The actual values of the mathematical model, which are located in the Arduino YÚN were sent by the communication link to the web based application, where based on the received information the EMPC controller is evaluated. Subsequently, the optimal control action is sent back to the Arduino YÚN, where the control action is implemented to the mathematical model.

The thermodynamical behavior of the single zone building can be compactly represented by a linear time-invariant state-space model in the discrete time domain

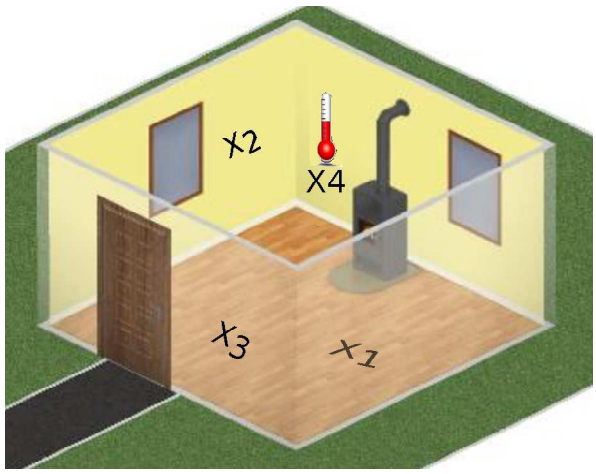


Figure 6.9: Representation of state variables.

as follows

$$x_{k+1} = Ax_k + Bu_k + Ed_k, \tag{6.27a}$$

$$y_k = Cx_k. \tag{6.27b}$$

The $A \in \mathbb{R}^{4\times4}$, $B \in \mathbb{R}^{4\times1}$, $E \in \mathbb{R}^{4\times3}$ are the state-update matrices, whose values were extracted from MATLAB toolbox ISE (van Schijndel, 2005):

$$A = 10^{-3} \cdot \begin{bmatrix} -0.020 & 0 & 0 & 0.020 \\ 0 & -0.020 & 0.001 & 0.020 \\ 0 & 0.001 & -0.056 & 0 \\ 1.234 & 2.987 & 0 & -4.548 \end{bmatrix},$$

$$B = 10^{-3} \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0.003 \end{bmatrix}, \; E = 10^{-3} \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0.055 & 0 & 0 \\ 0.327 & 0.003 & 0.001 \end{bmatrix}.$$

The physical meanings of the used variables are depicted in Table 6.3.

The idea was to control the temperature of the building by using explicit MPC and WebSocket communication. In this part we focus on the design of the EMPC

Table 6.3: Description of the variables.

Notation	Units	Description
x_1	$[°C]$	floor temperature
x_2	$[°C]$	internal facade temperature
x_3	$[°C]$	external facade temperature
x_4	$[°C]$	internal room temperature
u	$[W]$	heat flow
d_1	$[°C]$	external temperature
d_2	$[W]$	heat generated by the occupancy
d_3	$[W]$	solar radiation

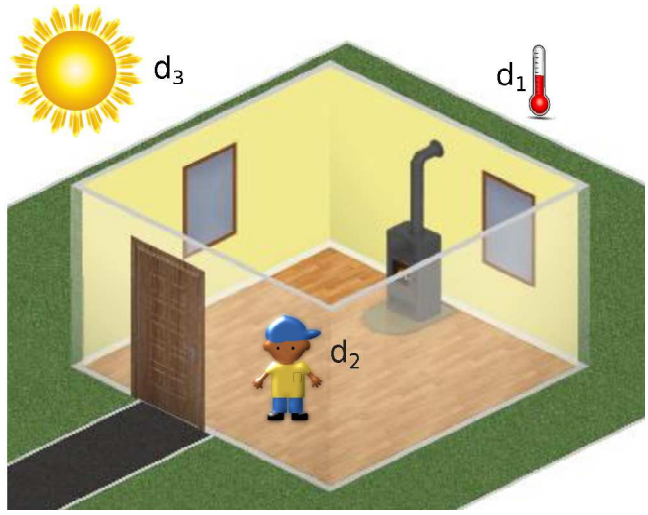


Figure 6.10: Representation of disturbance variables.

controller. The designed MPC problem which assumes model in (6.27) is of the following form

$$\min \sum_{k=0}^{N-1} Qu_k^2 + Ss_k \tag{6.28a}$$

$$\text{s.t. } x_{k+1} = Ax_k + Bu_k + Ed_k, \tag{6.28b}$$

$$u_k \in \mathcal{U}, \tag{6.28c}$$

$$x_k \in \mathcal{X}, \tag{6.28d}$$

$$s_k \in \mathcal{S}, \tag{6.28e}$$

$$T_{\text{ref}} - \epsilon - s_k \leq Cx_k, \tag{6.28f}$$

$$T_{\text{ref}} + \epsilon + s_k \geq Cx_k, \tag{6.28g}$$

$$x_0 = x(t), \tag{6.28h}$$

where x_k represents the vector of state variables, u_k stand for the input variable while s_k represents the slack variable. The slack variable could acquire only non-negative values. Since the reference is not defined in the objective function, this formulation of MPC does not have tracking properties. There the goal is to maintain the indoor temperature in the specified range around the reference. The width of the zone could be changed by manipulating with the value of ϵ. In case of decreasing it's value, the formulation is more restrictive, while in the other case the controller has bigger freedom. In the objective function the heating and the cooling cost are minimized and the value of the slack variable is penalized. Since weighting matrix value $S = 100000$ and $Q = 1$ the controller's goal is to maintain the temperature in the defined zone and to minimize the energy consumption. The prediction horizon was set to $N = 5$. The definition of MPC problem could be constructed in the same way as it was demonstrated in the previous sections. The parametric solution consisted of 190 critical regions in the 8-dimensional parametric space. The only difference is the exporting function's name. The code generation could be performed by evaluating

```
toJavaScript( empc, 'building', 'primal', 'first-region');
```

where `empc` represents the parametric solution, while `'building'` represents the string which is a user defined name of the generated file. In this case `building.js` and `building.html` are created. The `html` file's purpose is to check if the controller

returns the valid answers. The JavaScript file could be imported to arbitrary `html` file by a single line of code.

```
<script src="building.js" type="text/javascript"></script>
```

Together with the point location algorithm 144 KB was used to store the EMPC controller to the exported file `building.js`.

In the simulation scenario the reference temperature was set to 21 °C and while $\epsilon = 0.5$ °C was considered. This means the temperature can be within the interval from 20.5 °C to 21.5 °C. The simulation time was set to 1 year, while unpredictable disturbances were considered. The evolution of the indoor temperature is depicted in Fig. 6.11 and the and the manipulated variable is presented in Fig. 6.12. The influence of the disturbance variables, the external temperature, the solar radiation and heat generated by occupancy are depicted in Fig. 6.13, Fig. 6.14 and Fig. 6.15. It can be seen that the designed MPC control fulfills its work and the indoor temperature mainly remains in the desired area. However, there are several small violations of the tube constraints that were caused by the external disturbances, but the control performance is still acceptable. Table 6.4 summarizes the control performance of the designed MPC controller. As we can see, MPC violates the state constraints for the indoor temperature several times. The reason was that the controller goal was to minimize the energy consumption, and therefore the temperature was as close to the constraints as possible. Due to the external disturbances, the MPC controller was not able to fulfill state constraints all the time. In order to stay in the predefined control interval the values of the slack variables were minimized.

Table 6.4: Control performance.

Heating cost	12795.73 kWh
Cooling cost	62.12 kWh
Total cost	12857.85 kWh
% of constraints satisfaction	99.71 %

Summary

In this section of the work we demonstrated code generation module which can export a parametric solution of the given optimization problem together with a

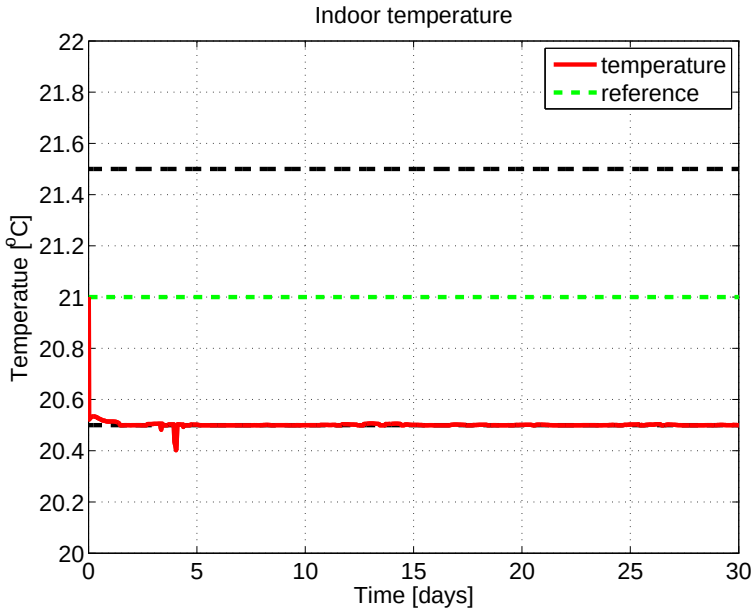


Figure 6.11: Evolution of the indoor temperature.

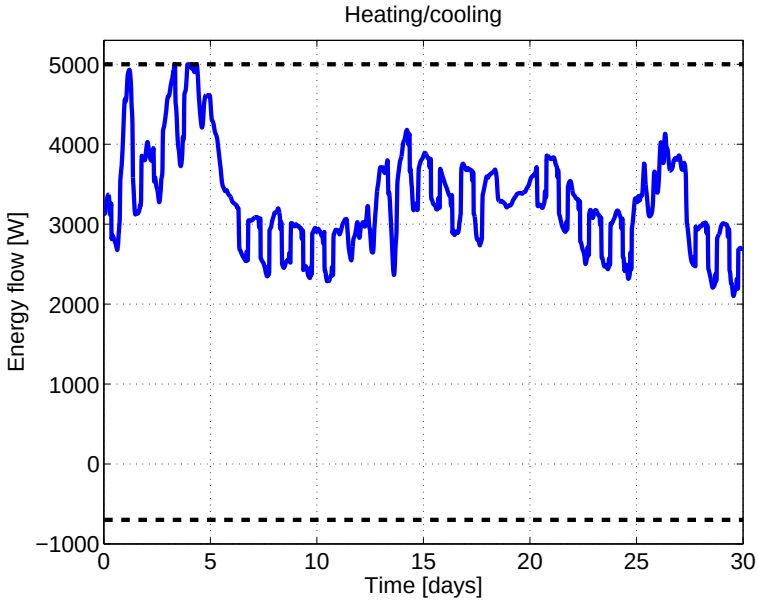


Figure 6.12: Heating and cooling.

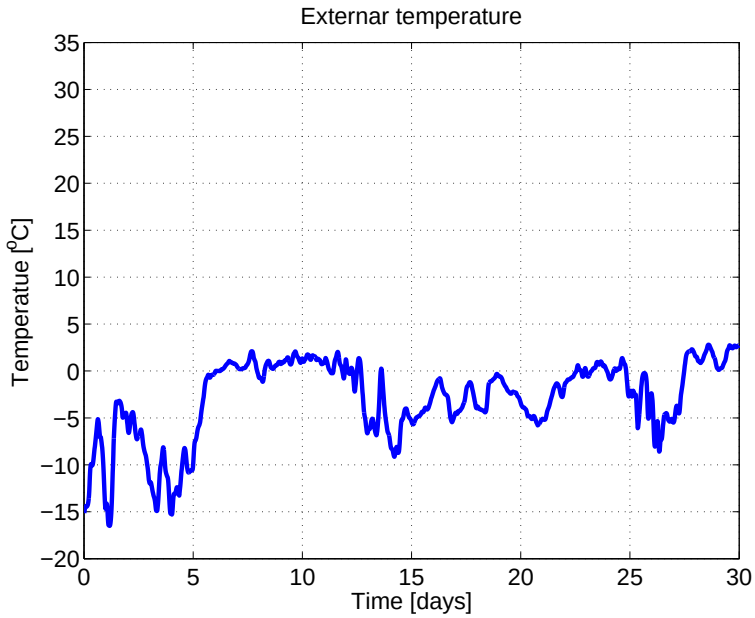


Figure 6.13: External temperature.

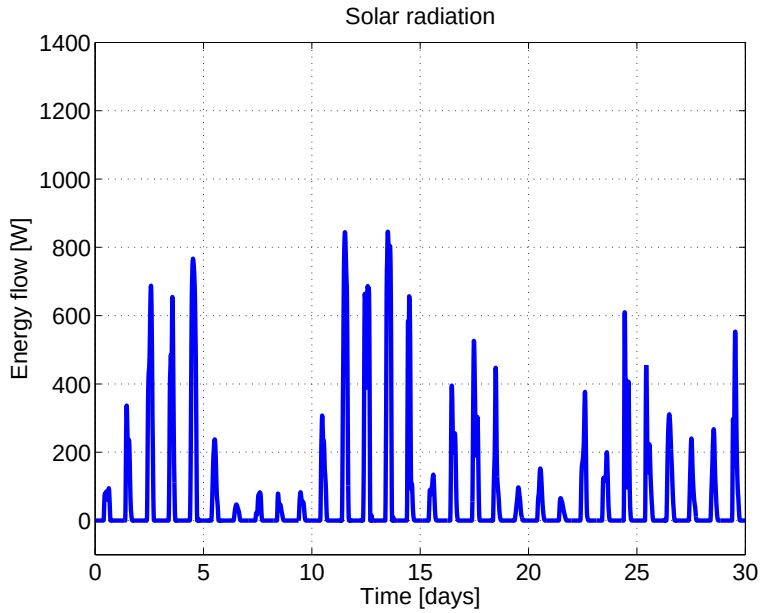


Figure 6.14: Solar radiation.

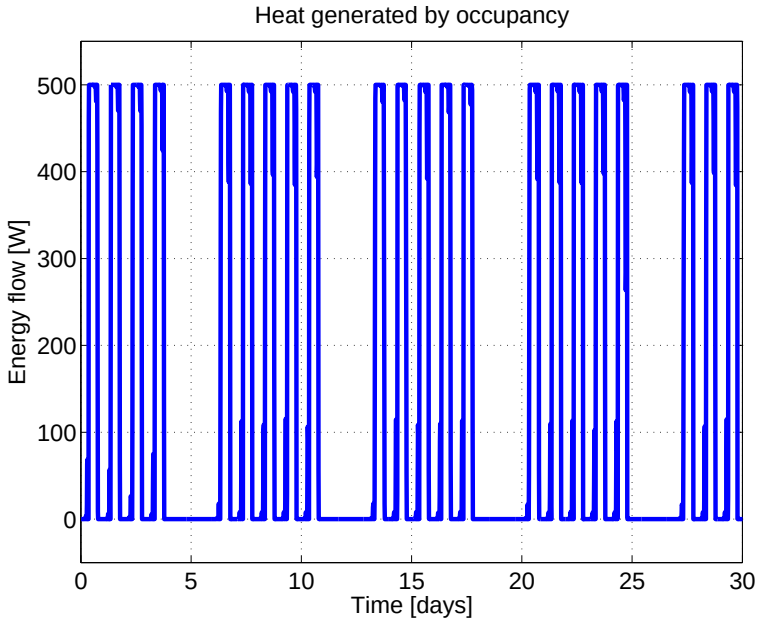


Figure 6.15: Heat generated by occupancy.

point location algorithm into JavaScript. The presented module can bring advanced control techniques like MPC into the web based environment. To propagate the power of the module, we introduced an application for thermal comfort control of the single zone building. The MPC algorithm was running on the separate device while the communication was performed by WebSocket control technique.

6.3 Summary

In this part of the work we have proposed a tool which manages an export of the parametric solution. We have shown how could be the parametric solution of the given optimization problem exported into two different programming languages such a Python and JavaScript. The reason of choosing these programming languages is the fact there have not existed an easy implementation of advanced control technique such as MPC in a convenient matter. Both languages offer interesting possibilities, while development of applications is convenient. It was shown that the merging of the exported controller with an actual application could be

done in a plug-and-play principle. In the case of both languages the imported controllers could be easily evaluated by only calling one single line of code. All the necessary operations which are required to extract the optimal control action for the particular value of the parameters are also defined in the exported function.

Chapter 7

Case Studies

This chapter deals with implementation of MPC policy into two non-linear systems, namely electroplating process and magnetic manipulation. In the first case the goal is to control the temperature of the plating solution subject to process input and output constraints. The task for the magnetic manipulator is to control the position of the ball in two-dimensional space. The magnetic field influencing the position of the ball is generated by the series of coils. The coils are located below the ball and the magnetic field can be changed by using different currents. We show that a well performing MPC policy design is not an easy task as both aforementioned processes have non-linear dynamics.

7.1 Electroplating Process

The results described in this chapter are one of the outcomes of the joint work with Surasit Tanthadiloke and Paisan Kittisupakorn from the Chulalongkorn University in Bangkok, Thailand.

Electroplating is a process that uses electric current to reduce dissolved metal cations so that they form a coherent metal coating on an electrode. Electroplating is primarily used to change the surface properties of objects by creating a thin new surface, which provides better corrosion protection.

The process consists of three components, the cathode, the anode, and the electrolyte solution. The cathode represents the object to be treated. The anode is

made of metal to be plated and the electrolyte contains dissolved metal salts and some other ions. The free ions ensure the flow of the electricity. Both anode and cathode are dipped into the solution. The power supply sends the direct current to the anode, which starts to dissolve metal ions in the electrolyte. The dissolved metal on the cathode creates a new surface. The thickness of the new surface can be adjusted by changing the operation time or by changing the applied current. Electroplating can change the physical, chemical, and mechanical properties of the workpiece.

The aim is to control the temperature of the plating solution to a specific value. For this purpose, three different MPC setups are presented and compared. The goal is to find an acceptable comparison between the complexity of the optimization problem and the performance of the controller. Components of the electroplating process are illustrated in Fig. 7.1.

The cooling tower together with the reservoir tank cools down the water's temperature. The power supply ensures the right value of the electric current, while the plating bath contains the anodes, workpiece as a cathode and the plating solution. During the operational time of the electroplating process, a certain temperature of the plating solution must be maintained. A high temperature can cause surface burning. The unwanted burning area must additionally be polished, which increases production costs. However, if the temperature is too low the thickness of the new surface will not be sufficient. In the following section, the control methods will be presented from the most difficult to the easiest formulation, from the computation point of view.

7.1.1 Mathematical model

The mathematical model was constructed by Surasit Tanthadiloke using mass and energy balances (Tanthadiloke et al., 2013). In order to obtain a valid mathematical model of the hard chromium plating process, a real data of conventional operating conditions and parameters were collected from the Siam Hard Chrome Co., which is a company in Thailand dealing with hard chromium plating. The nonlinear

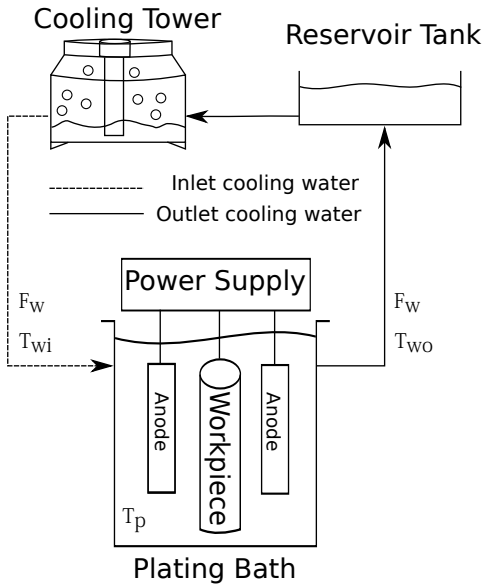


Figure 7.1: Electroplating process.

behavior of the electroplating process is defined as follows

$$\frac{\mathrm{d}T_p}{\mathrm{d}t} = \frac{IV - U_o A_{ht}(\Delta T_{lm}) - Q_{loss}}{\rho_p C p_p V_p}, \tag{7.1a}$$

$$\frac{\mathrm{d}T_{wo}}{\mathrm{d}t} = \frac{U_o A_{ht}(\Delta T_{lm}) + F_w(T_{wi} - T_{wo})}{\rho_w C p_w L_{tube} A_o}, \tag{7.1b}$$

$$\Delta T_{lm} = \frac{(T_p - T_{wi}) - (T_p - T_{wo})}{\ln \frac{(T_p - T_{wi})}{(T_p - T_{wo})}}, \tag{7.1c}$$

$$U_o = \frac{1}{\frac{A_o}{A_i h} + \frac{A_o \ln(d_o/d_i)}{2\pi K L_{tube}} + \frac{1}{h}}, \tag{7.1d}$$

$$Q_{loss} = \alpha_{Q_{loss}} A_{sur} T_p^{\beta_{Q_{loss}}}, \tag{7.1e}$$

where the first state of the system is the temperature of the plating solution T_p, the second state is the temperature of the outlet cooling water from the plating bath T_{wo} and the control input is the inlet flow of the cooling water to the plating bath F_w. We are interested in maintaining the output of the process T_p at the desired reference. Just to simplify the differential equations (7.1a) and (7.1b), the expression for the logarithmic mean temperature difference T_{lm} in (7.1c) and the overall heat transfer coefficient U_o in (7.1d) were separated. Variable Q_{loss} in (7.1e)

represents the heat loss from the plating bath. The nonlinear behavior is caused by the logarithmic mean temperature difference (7.1c) and also by heat loss (7.1e). The parameters used in (7.1) are listed in Table 7.1.

Table 7.1: Process variables and parameters.

Parameters	Units	Values
Electric current (I)	A	3500
Voltage (V)	V	5.5
Outer diameter of heat exchanger tube (d_o)	m	0.0254
Inner diameter of heat exchanger tube (d_i)	m	0.0224
Outside surface area for heat transfer (A_o)	m^2	5.067×10^{-4}
Inside surface area for heat transfer (A_i)	m^2	$3.942x10^{-4}$
Heat transfer area of heat exchanger (A_{ht})	m^2	1.489
Heat exchanger length (L_{tube})	m	18.66
Overall heat transfer coefficient (U_o)	kW/m^2 °C	0.58484
Density (ρ)	kg/m^3	
- Plating solution (ρ_p)		1174.4
- Cooling water (ρ_w)		992.25
Specific heat capacity of plating solution (Cp)	kJ/kg.°C	
- Plating solution (Cp_p)		4.9172
- Cooling water (Cp_w)		4.181
Electroplating bath volume (V_p)	m^3	11.35
Electroplating bath temperature (T_p)	°C	-
Temperature of the out water stream (T_{wo})	°C	-
Flow of the out water stream (F_w)	m^3s^{-1}	-
Heat loss from a plating bath (Q_{loss})	KJ/s	-
Coefficient of heat loss ($\alpha_{Q_{loss}}$)	-	1.96×10^{-5}
Coefficient of heat loss ($\beta_{Q_{loss}}$)	-	2.8806

The process (7.1) is subjected to physical limitations on the flow of the cooling water

$$1 \times 10^{-4}\,\mathrm{m^3 s^{-1}} \leq F_w \leq 2 \times 10^{-3}\,\mathrm{m^3 s^{-1}}, \tag{7.2}$$

as well as limitations on both temperatures T_p and T_{wo} as

$$30.0\,^\circ\mathrm{C} \leq T_p \leq 70.0\,^\circ\mathrm{C}, \tag{7.3a}$$

$$26.5\,^\circ\mathrm{C} \leq T_{wo} \leq 95.0\,^\circ\mathrm{C}. \tag{7.3b}$$

Furthermore, we assume that the cooling unit is able to cool down the recycled water's temperature to $T_{wi} = 26.5\,^\circ\mathrm{C}$. During the electroplating procedure anodes and the object, which should be treated, are connected to the power supply. The value of the voltage and the electric current are chosen based on the material of the object and the desired thickness of the new surface. During the electroplating procedure, due to the supplied electricity, the plating solution's temperature starts to increase. The temperature in the plating bath is crucial for the successful electroplating. In case when the temperature is over the desired value the flow of cooling inlet water is increased. Furthermore the water had to circulate consistently, which means even if the temperature of the plating solution is lower than the desired the minimal water flow remains. The objective is to maintain the temperature of the plating solution at the desired $50\,^\circ\mathrm{C}$ (Zitko et al., 2010). Our goal is to design such a MPC controller, which is able to maintain the temperature in the plating bath, while there will not large overshoots in the flow of the cooling water and the physical limitation of the system will be satisfied.

Three different MPC setups will be presented in sequel. In each of them, we use the same form of the objective function, where we penalize the tracking error of the controlled output and the increment of the control input. The reference temperature was set to $50\,^\circ\mathrm{C}$ and the simulation time was set to 5100 seconds, which is a real process time received from the industry. The initial temperature of the plating solution was set to $47\,^\circ\mathrm{C}$, while the temperature of cooling water on the outlet was set to $30\,^\circ\mathrm{C}$. In the simulations, plant model mismatch was considered, by changing parameters U_o and Q_{loss} by $\pm 10\%$ compared to their nominal values. The prediction horizon in each cases was set to $N = 5$ and with sampling period $T_s = 15\mathrm{s}$.

7.1.2 Nonlinear MPC

In the first presented MPC setup we consider quadratic penalization on the difference between the current and desired temperature of the plating solution and also the value of the increment of the control action is penalized by a quadratic

function. The weighting matrices Q and R are scalars since only one output and one input is considered. First we formulate a non-linear MPC problem as

$$\min_{u_0,\dots,u_{N-1}} \sum_{k=0}^{N-1} Q(y_k - y_{\text{ref}})^2 + R\Delta u_k^2 \tag{7.4a}$$

$$\text{s.t. } x_{k+1}^1 = x_k^1 + T_s \frac{IV - U_o A_{ht}(\Delta T_{lm}) - Q_{loss}}{\rho_p C p_p V_p}, \tag{7.4b}$$

$$x_{k+1}^2 = x_k^2 + T_s \frac{U_o A_{ht}(\Delta T_{lm}) + F_w(T_{wi} - x_k^2)}{\rho_w C p_w L_{tube} A_o}, \tag{7.4c}$$

$$\Delta T_{lm} = \frac{(x_k^1 - T_{wi}) - (x_k^1 - x_k^2)}{\ln \frac{(x_k^1 - T_{wi})}{(x_k^1 - x_k^2)}}, \tag{7.4d}$$

$$U_o = \frac{1}{\frac{A_o}{A_i h} + \frac{A_o \ln(d_o/d_i)}{2\pi K L_{tube}} + \frac{1}{h}}, \tag{7.4e}$$

$$Q_{loss} = \alpha_{Q_{loss}} A_{sur} x_k^{1^{\beta_{Q_{loss}}}}, \tag{7.4f}$$

$$y_k = C x_k, \tag{7.4g}$$

$$x_{\min} \leq x_k \leq x_{\max}, \tag{7.4h}$$

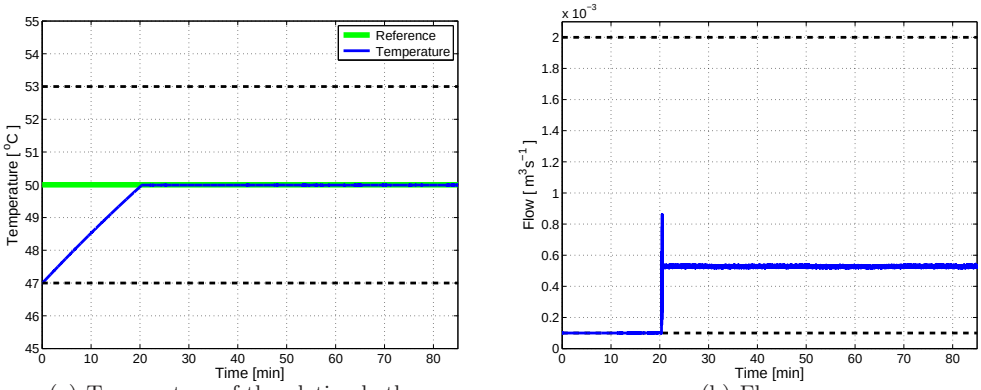
$$u_{\min} \leq u_k \leq u_{\max}, \tag{7.4i}$$

$$\Delta u_k = u_k - u_{k-1}, \tag{7.4j}$$

$$x_0 = x(t), \tag{7.4k}$$

$$u_{-1} = u(t-1), \tag{7.4l}$$

where constraints (7.4b)-(7.4j) are enforced for all $k = 0, \dots, N-1$. Constraint (7.4b) and (7.4c) represent the discretized model of (7.1), where we assumed $x^1 = T_p$ and $x^2 = T_{wo}$. We used forward Euler's discretization scheme. The objective function is quadratic and the constraints are not linear, therefore control problem defined in 7.5 must be solved by a non-linear numerical solver. The optimization problem was solved in Matlab (MATLAB, 2010) by using a built in `fmincon` solver. We used a built-in `interior-point` algorithm to solve (7.4). The values of penalty matrices are $Q = 1$ and $R = 100$. Those values were chosen experimentally. We choose integral squared error for performance index Its value for this setup is ISE $= 3.5566 \times 10^4$. The evolution of the system is depicted in Fig 7.2(a) and Fig 7.2(b). Since we had a nonlinear optimization problem a non-linear numerical solver was used, which is almost impossible to implement in hardware with limited computation resources. As it can be seen in Fig. 7.2(a) the controller was able to move the system and maintain the desired temperature. However the evolution of



(a) Temperature of the plating bath. (b) Flow.

Figure 7.2: Evolution of the process controlled by (7.4)

the temperature of the plating solution seems to be a slow procedure, we need to realize that the volume of the plating bath is $V_p = 11.35$ m^3 and also a specific heat capacity of the plating solution is high $Cp_p = 4.9172$ kJ/kg.°C. This is the reason of the high regulation time, which is around 20 minutes. Also the control action presented in Fig. 7.2(b) fulfilled the saturation. The small oscillation of the control action was caused by model mismatch.

7.1.3 Linearized MPC

Our second approach used a simplified formulation of the process that was linearized around the actual state in every sampling time. This gives rise to a time-variant model with matrices $A(t)$, $B(t)$ and $f(t)$.

Thanks to this additional linearization the MPC problem could be formulated

as QP and solved more effectively.

$$\min_{u_0,\dots,u_{N-1}} \sum_{k=0}^{N-1} Q(y_k - y_{\text{ref}})^2 + R\Delta u_k^2 \tag{7.5a}$$

$$\text{s.t. } x_{k+1} = A(t)x_k + B(t)u_k + f(t), \tag{7.5b}$$

$$y_k = Cx_k, \tag{7.5c}$$

$$x_{\min} \leq x_k \leq x_{\max}, \tag{7.5d}$$

$$u_{\min} \leq u_k \leq u_{\max}, \tag{7.5e}$$

$$\Delta u_k = u_k - u_{k-1}, \tag{7.5f}$$

$$x_0 = x(t), \tag{7.5g}$$

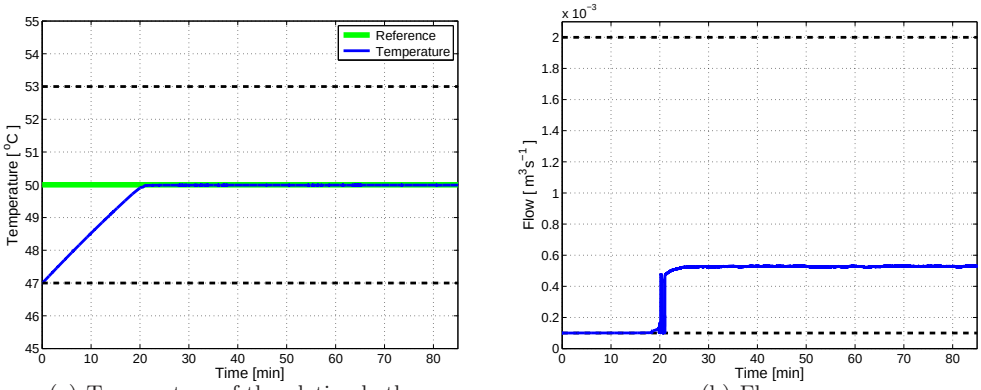
$$u_{-1} = u(t-1). \tag{7.5h}$$

We used Gurobi optimization solver to solve the given QP. Using Yalmip Toolbox (Löfberg, 2004) the persistent linearization could be easily implemented, since the parameter varying optimization problem can be defined by the built in `optimizer` function. This allows us to update the time varying parameters, $A(t)$, $B(t)$ and $f(t)$, of the mathematical model defined in (7.5b) in each simulation step. The values of the weighting matrices in the simulation were set to $Q = 2.5$ and $R = 90$. The value of the ISE is 3.5640×10^4, and the process progress is presented in Fig 7.3(a) and Fig 7.3(b). As can be seen, the evolution of the temperature of the plating solution is similar to the previous setup. Since the mathematical model used for simulation has time varying parameters U_o and Q_{loss}, there will be always mismatch between the model for simulation and the model used for prediction purposes.

In this example we showed that, for such a system with slow dynamics even the persistently linearized model is suitable to achieve control goals.

7.1.4 Linear MPC

After realizing that, by decreasing the computational complexity we are still able to create a simpler controller with an acceptable performance, we wanted to explore further simplification possibilities. In order to obtain computationally the cheapest possible controller only one linearized model was used. This means, we eliminate the linearization scheme, which was presented in the previous section. We linearized



(a) Temperature of the plating bath. (b) Flow.

Figure 7.3: Evolution of the process controlled by (7.5)

the system in it's stable equilibrium (7.6),

$$T_p^s = 50.0009\,^\circ\mathrm{C} \tag{7.6a}$$

$$T_{wo}^s = 34.2110\,^\circ\mathrm{C} \tag{7.6b}$$

$$F_w^s = 5.2607 \times 10^{-4}\,\mathrm{m^3 s^{-1}} \tag{7.6c}$$

and we obtained variables A, B and f. The whole optimization problem could be precomputed by parametric programming, which means there is no need for external solvers, just a mere function evaluation is required to obtain the solution. This approach has the lowest online computation demand and fastest online evaluation time.

$$\min_{u_0,\ldots,u_{N-1}} \sum_{k=0}^{N-1} Q(y_k - y_{\mathrm{ref}})^2 + R\Delta u_k^2 \tag{7.7a}$$

$$\text{s.t. } x_{k+1} = Ax_k + Bu_k + f, \tag{7.7b}$$

$$y_k = Cx_k, \tag{7.7c}$$

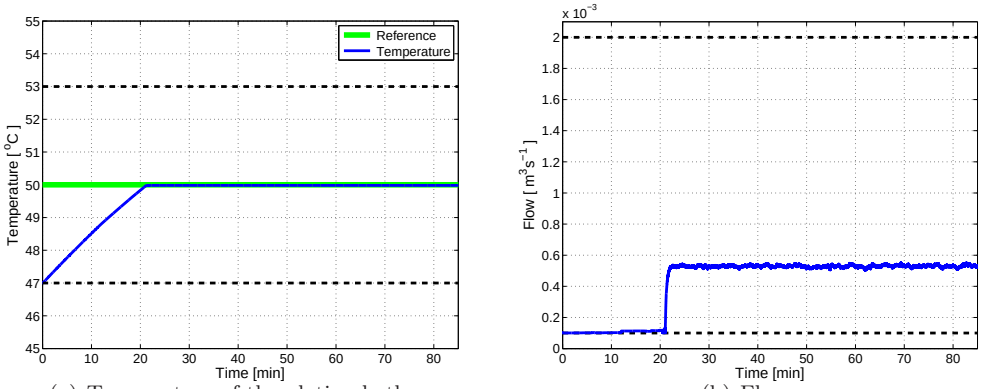
$$x_{\min} \leq x_k \leq x_{\max}, \tag{7.7d}$$

$$u_{\min} \leq u_k \leq u_{\max}, \tag{7.7e}$$

$$\Delta u_k = u_k - u_{k-1}, \tag{7.7f}$$

$$x_0 = x(t), \tag{7.7g}$$

$$u_{-1} = u(t-1). \tag{7.7h}$$



(a) Temperature of the plating bath. (b) Flow.

Figure 7.4: Evolution of the process controlled by (7.7)

Since the system has slow dynamics this control design was also sufficient from the point of constraint violation and control performance. The values of the weighting matrices are $Q = 1$ and $R = 10$ while ISE $= 3.5566 \times 10^4$. The temperature and control variable evolution are depicted in in Fig 7.4(a) and Fig 7.4(b). As can be seen even this simple MPC setup was able to control and maintain the temperature to the desired 50°C. The results showed that even the MPC based on one linear model is suitable for this system after appropriate tuning.

7.1.5 Summary

In this section the hard chromium electroplating process was presented. The parameters of the process are values received from the industry. Three MPC setups were presented. Each of the three designed MPC control techniques has different performance and computation requirements. The first discussed method most accurately describes the behavior of the systems, but since the model is non-linear a non-linear optimization problem must be solved in order to obtain the optimal values of the control action. The next presented method uses persistent linearization, where the computational requirements are lower since only convex QP problem need to be solved, but also an additional linerization scheme was added in order to update the matrices of the linear model. The third presented method considered a linear model, which was linearized in the steady state operation conditions. All the described approaches were able to control the system without violating the

Table 7.2: Control performance comparison.

Method	Non-linear	Linearized	Linear
Fmincon [s]	0.290565	-	-
Gurobi [s]	-	0.009075	-
Linearization [s]	-	0.002716	-
Point Location [s]	-	-	0.001981
Total time [s]	0.290565	0.011791	0.001981
ISE [-]	3.5566×10^4	3.5640×10^4	3.6009×10^4

constraints and the differences in the chosen performance index were minor. Since the system has slow dynamics it is enough to use linear MPC, which has the lowest computational requirements. Table 7.2 summarizes the results of the proposed MPC setups. The last MPC setup is also implementable on a PLC like devices since there is no need for external numerical solvers.

In this section, we considered only one batch operation. In the future, it would be interesting to consider multiple batches, where the thickness of the requested surface on the treated material will not be identical. In order to change the thickness of the surface, the value of the electric current are voltage should be changed. In this case, the controller should incorporate with another parameters, which would be the value of the electric current and the voltage. Moreover, since it is not a continuous process, also the manipulation time to withdraw the plated material and time needed to immerse the another workpiece should be considered in the future.

7.2 Magnetic Manipulation

The result presented in this section are the outcome of the joint work with Jiří Zemánek from the Czech Technical University in Prague. We will demonstrate MPC implementation to a process with extremely fast dynamics. This section represents only the first step of an interesting experiment and control design. The future plans also involve size reduction of the device, manipulation of more than one object at the same time, and extension from 2D to 3D. This could bring new possibilities and new control challenges. We believe that in a not so distinct future

the principle of magnetic manipulator could be used in modern non-invasive surgery techniques.

7.2.1 Device Description and Problem Formulation

The magnetic manipulator depicted in Fig. 7.5 consists of four separate modules. Each module contains four coils and the control electronics which is located on the bottom. Power supply connectors are located on each side of the module. This allows to simply extend the magnetic manipulator platform. A surface with touch foil is located over the set of coils. This pressure-sensitive plate is used to obtain the actual position of the object, which is located above the coils. Furthermore it is also possible to detect the position of the objects by an optical system. The objects are recognized by color thresholds. The physical dimension of the plate is 20 by 20 centimeters and the coils are arranged in a regular 4-by-4 grid. The device can operate at a frequency up to 1 kHz.

Each coil can be activated separately. The only stable equilibrium points are located above the coils. The position of the object without feedback can only be moved to the center of the coil. There is also an unstable equilibrium point between the coils. However, if only two neighboring coils are activated there is no chance to move the object to the unstable equilibrium.

The objective is to design an MPC controller which will manipulate the current supplied to each coil such that the ball follows a prescribed trajectory. The reason why MPC is employed stems from the fact that constraints play an important role (Bemporad et al. (2004)). Specifically, the controller must ensure that the ball never falls off the plate and that the calculated currents stay within their physical limits.

7.2.2 Control Synthesis

For the sake of clarity of explanation, in this section we first present control design assuming one-dimensional motion of the ball along a line. In this setup, only 4 coils along the line will be activated. Under such conditions the model of the motion of

Figure 7.5: MAGMAN - Magnatic Manipulator Device

the ball is given by

$$\dot{p}_x = v_x, \tag{7.8a}$$

$$\dot{v}_x = \gamma \sum_{i=1}^{4} F(p_x, i) T_i, \tag{7.8b}$$

where p_x represents the position of the ball, and v_x denotes the ball's velocity. Moreover, T_i is the current supplied to the i-th coil, and $F(p_x, i)$ is the magnetic force generated by the i-th coil as a function of the ball's position. The final force is then obtained by a superposition principle. Finally, γ is a system's constant. The following analytic representation of the function $F(p_x, i)$ was obtained by experimental identification by our colleagues in Prague:

$$F(p_x, i) = \frac{-0.1857(p_x - i)}{((p_x - i)^2 + 0.3303)^3}, \tag{7.9}$$

which assumes that the ball's position p_x is normalized to the interval [0, 5]. If $p_x = 1$, then the ball is directly above the first coil, if $p_x = 2$, then it is above the center point of the second coil, etc.

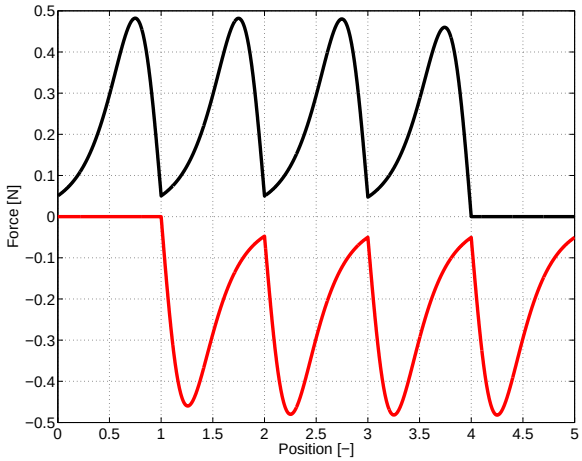


Figure 7.6: Force as function of position. Red line represents the minimal and black line the maximal force

By assuming that the minimal applicable current is zero, and the maximal limit is 0.8 (expressed in dimensionless, normalized units), one can plot the minimal and maximal force which the array of 4 coils can generate as in Fig. 7.6. Any force between the two functions shown in Fig. 7.6 can be generated by an appropriate selection of the respective currents. As can be seen, the area of generate forces forms a non-convex set, which represents a nonlinear constraint on decision variables (the currents) in the control problem.

The problem of employing (7.8) in MPC is due to its nonlinearity. Specifically, $F(p_x, i)$ is a nonlinear function, and the model also features a product between $F(p_x, i)$ and the control inputs T_i. To avoid nonlinearity, we propose to replace the nonlinear term by a new decision variable, which will represent the required acceleration of the ball, i.e.,

$$a_x = \sum_{i=1}^{4} F(p_x, i) T_i. \tag{7.10}$$

Using this substitution, the model in (7.8) turns into a linear system of the form

$$\begin{bmatrix} \dot{p}_x \\ \dot{v}_x \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} p_x \\ v_x \end{bmatrix} + \begin{bmatrix} 0 \\ \gamma \end{bmatrix} a_x, \tag{7.11a}$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} p_x \\ v_x \end{bmatrix}, \tag{7.11b}$$

which is a well-known double integrator system.

With the model (7.11) at hand, standard convex MPC can be used to look for optimal accelerations required for the ball to reach a certain position. However, the constraints on the acceleration have to be chosen in a conservative fashion as not to exceed the allowed authority depicted in Fig. 7.6. Another disadvantage of this approach is that the MPC controller only provides optimal accelerations, not the coils' currents. To find the currents that generate a particular acceleration, we propose to solve the following linear programming problem:

$$\min \sum_{i=1}^{4} T_i \tag{7.12a}$$

$$\text{s.t. } a_x = \sum_i F(p_x, i) T_i. \tag{7.12b}$$

It is important to notice that in (7.12) both the acceleration a_x as well as the current position p_x are assumed to be known. Once p_x is known, the quantity $F(p_x, i)$ can be evaluated from (7.9). Therefore the value of $F(p_x, i)$ is treated as a constant in (7.12), and thus the constraint is linear. Minimizing the sum of currents provides the least possible excitation of the magnets required to create a certain acceleration.

The overall implementation is therefore as follows:

- obtain information about current states p_x and v_x;
- obtain the optimal acceleration a_x by solving an MPC problem;
- solve for T_i, $i = 1, \ldots, 4$ from the LP in (7.12);
- apply all currents T_i to the device;
- request new state measurement at the next sampling instant and repeat from the beginning.

The procedure discussed above can be easily extended to the two-dimensional scenario where the position of the ball is controlled along both axis. In particular, the nonlinear model then takes the form

$$\dot{p}_x = v_x, \tag{7.13a}$$

$$\dot{v}_x = \gamma \sum_{i=1,j=1}^{4} F_x(p_x, p_y, i, j) T_{i,j}, \tag{7.13b}$$

$$\dot{p}_y = v_y, \tag{7.13c}$$

$$\dot{v}_y = \gamma \sum_{i=1,j=1}^{4} F_y(p_x, p_y, i, j) T_{i,j}, \tag{7.13d}$$

where p_x and p_y denote, respectively, positions of the ball along the x- and along the y-axis. Moreover, $F_x(\cdot)$ is a force along the x-axis generated by the coil situated at coordinates i and j (the first coordinate corresponds to the x-axis, the other one to the y-axis), and $F_y(\cdot)$ is the force along the y-axis generated by the corresponding coil. The force functions are given by

$$F_x(p_x, p_y, i, j) = \frac{-0.1857(p_x - i)}{((p_x - i)^2 + (p_y - j)^2 + 0.3303)^3}, \tag{7.14a}$$

$$F_y(p_x, p_y, i, j) = \frac{-0.1857(p_y - j)}{((p_x - i)^2 + (p_y - j)^2 + 0.3303)^3}. \tag{7.14b}$$

To eliminate the nonlinear terms from (7.13), we propose to introduce additional variables a_x and a_y, which effectively denote accelerations along the corresponding axis:

$$a_x = \sum_{i=1,j=1}^{4} F_x(p_x, p_y, i, j) T_{i,j}, \tag{7.15a}$$

$$a_y = \sum_{i=1,j=1}^{4} F_y(p_x, p_y, i, j) T_{i,j}. \tag{7.15b}$$

The the system's model simplifies to

$$\begin{bmatrix} \dot{p}_x \\ \dot{v}_x \\ \dot{p}_y \\ \dot{v}_y \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_x \\ v_x \\ p_y \\ v_y \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ \gamma & 0 \\ 0 & 0 \\ 0 & \gamma \end{bmatrix} \begin{bmatrix} a_x \\ a_y \end{bmatrix}, \tag{7.16a}$$

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} p_x \\ v_x \\ p_y \\ v_y \end{bmatrix}, \tag{7.16b}$$

which is linear. It can thus be employed to synthesize an MPC controller as described in previous sections. Note, however, that such an MPC controller produces accelerations a_x and a_y as decision variables. To convert the accelerations into currents, the following linear programming problem needs to be solved after the MPC problem:

$$\min \sum_{i=1,j=1}^{4} T_{i,j} \tag{7.17a}$$

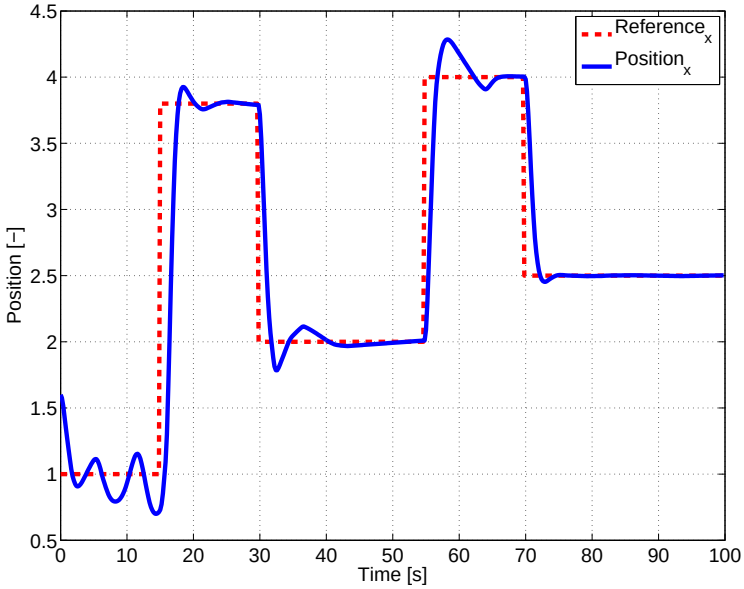
$$\text{s.t. } a_x = \sum_{i=1,j=1}^{4} F_x(p_x, p_y, i, j) T_{i,j}, \tag{7.17b}$$

$$a_y = \sum_{i=1,j=1}^{4} F_y(p_x, p_y, i, j) T_{i,j}. \tag{7.17c}$$

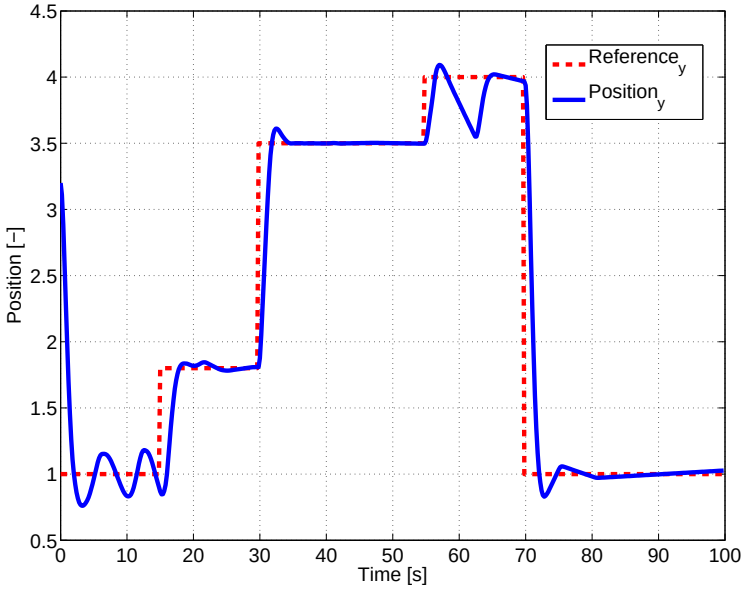
$$\tag{7.17d}$$

Again, it is important to note that p_x and p_y are known at the time problem (7.17) is solved. Thus $F_x(p_x, p_y, i, j)$ and $F_y(p_x, p_y, j, j)$ are constant and therefore the constraints in (7.17) are linear.

The procedure reported above was first verified by a simulation, results of which are shown in Fig. 7.7(a), Fig. 7.7(b) and Fig. 7.8. Subsequently, the MPC approach was also implemented on the real device, albeit only in the one-dimensional scenario. Several changes of the reference positions were made during the experiment and the results are presented in Figures 7.9(a) and 7.9(b). The oscillation around the set point was caused by the sensor and of course the small imperfection of the balls and plates surface.



(a) Position of ball on the x axis



(b) Position of ball on the y axis

Figure 7.7: Simulation results in two dimensional case

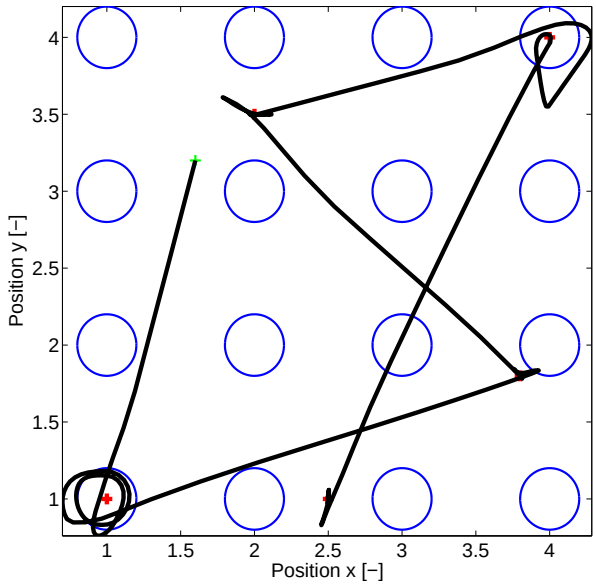
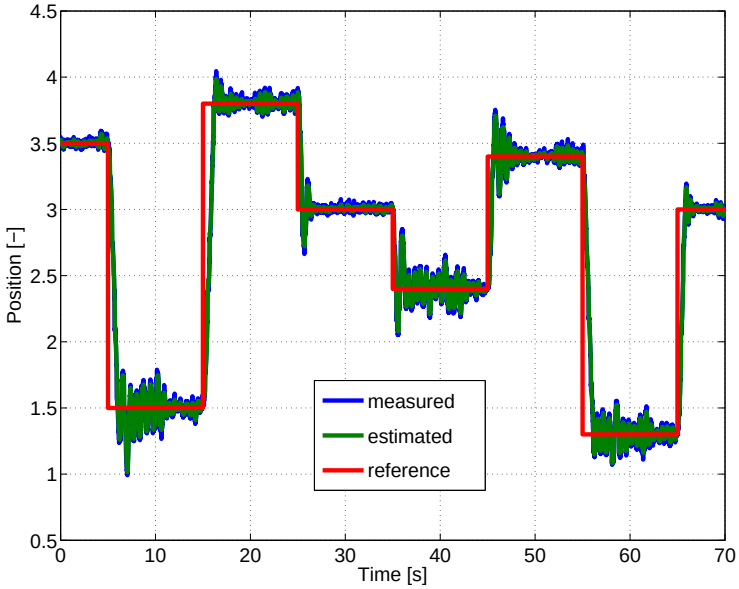
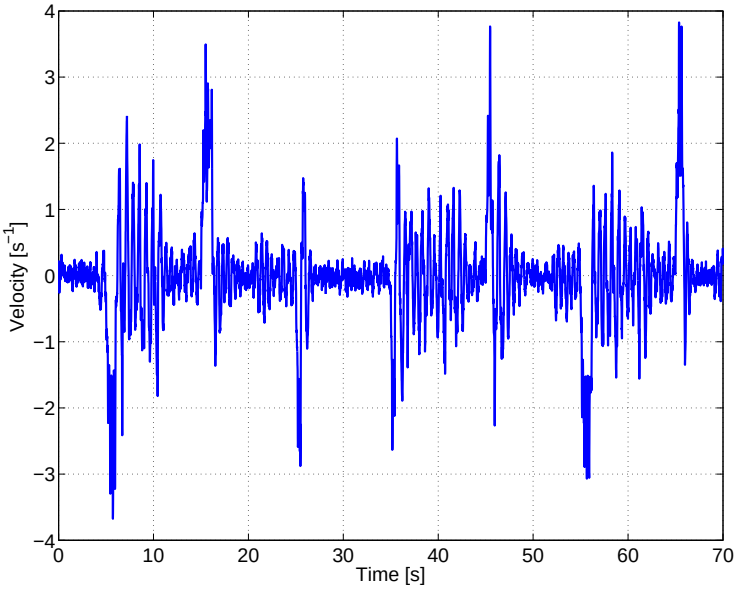


Figure 7.8: Ball's position in 2D



(a) Position of ball



(b) Velocity of ball

Figure 7.9: Experimental results

7.3 Summary

In this part of the work two different systems were introduced with several MPC setups. We have shown that the control performance which is sufficient enough could be achieved by appropriate tuning. However in the case of the electroplating process we expected that the control evolution will be incomparably better in case of assuming non-linear model of the system compared to those methods, where we considered only linear approximations of the systems behavior. Thanks to the systems slow dynamics and the limitation of the control variable it was shown that in some cases even simpler MPC could be implemented in order to achieve the desired performance. In case of the second system it was shown how could be a non-linear MPC problem formulation divided into separate parts in order to reduce the complexity of the optimization problem. Since this system has a relatively high sampling frequency the computation time and effort required to solve the non-linear MPC are higher we were looking for model simplification. In case of such a decomposition only convex problem need to be solved, which can be precomputed by parametric programming. The EMPC was shown to be fast enough to control to magnetic manipulation systems. In the real case scenario oscillation around the desired reference could be detected. This could be caused by imperfection on the ball's and plates surface or the badly identified parameters of the considered model.

Chapter 8

Conclusions and Future Research

In this work we investigated the possibilities of Model Predictive Control technique in different applications. The beginning of the work is dedicated to complexity reduction techniques that are useful when the advanced control technique must be implemented on the real device. Different approaches were presented. Since each application needs different approach during the synthesis there is no universal method to obtain the desired complexity.

The next part was dedicated to a code generation, where the goal was to create a new module to Multi-Parametric Toolbox, which is able to export the parametric solution of the optimization problem in programming languages such Python and JavaScript. The profit of the new module is demonstrated in 3 different applications. On the computer game, where MPC took place as an artificial player. The second application deals with the control of Ar.Drone2's yaw angle, while the last application introduces the possibility to use the code generation for temperature control in the single zone building.

The last part of the work deals with the problem of implementing MPC in various nonlinear process. Different simplification techniques were used in order to implement controller with the lower computation demand.

Motivation for future work can be summarized in three points:

1. Complexity reduction of explicit model predictive control

   However interesting and applicable results were presented, there are still possibilities to reduce the memory or computational demand of the final para-

metric solution. With the effective reduction technique the industry could use the remaining computational resources for another applications. Our goal is to merge the discussed method with recent approach presented in (Kvasnica et al., 2015), which does not require to store the critical regions in the form of $\mathcal{H}$-representation. However until now this method is applicable only for the original control parametric solution, we believe that further improvement is possible.

2. Code generation

   With the new code generation module in hand we are able to export the parametric solution to different programming languages. Python and even JavaScript are popular programming languages, and thanks to the new module the implementation is easy. The merging with an actual applications can be done in a plug-and-play manner. With this module the benefits of the parametric programming could be used in non-trivial or not directly control based applications, like computer games or web based applications. Until now the export procedure works only if the defined MPC problem could be solved as a LP, QP, MILP or MIQP, because only the parametric solution of the listed optimization problems have the form of PWA defined over polytopic regions. In the future it would be beneficial to consider an export module for systems, which have non-linear dynamics and the final optimization problem leads to non-linear programming problem. For this purpose the new module should be created, which can handle even nonlinear systems. After the appropriate formulation of the optimization problem, the export procedure should generate a file with the definition of the non-linear optimization problem together with the appropriate solver. We plan to add an another logical switch, which will take care about the choosing of appropriate numerical solver. The goal will again to create such an export procedure, which create a self consistent file, which would contain all the necessary mathematical operations to obtain the optimal control action for the given parameter. This could increase the number of potential users of the Multi-Parametric Toolbox, since even systems with non-linear dynamics could be handled in the future. Furthermore since the code generation module presented in this work aim web based languages, is would be interesting to create such an application, which has decentralized structure. On each end of this structure an electronic device would be

located and by using it's free computational power a larger scale problems could be solved. Recent problems like intelligent house control techniques, which combines numerous control actions and measurable factors, could also find attractive the code generation. We possible could use the free computational power of the mobile devices or computers to calculate and send the optimal control actions via internet connection to the actuators. In the future our goal is bring together different technologies to make it possible to create a revolutionary applications.

3. Nonlinear Model Predictive Control

   Generally in case of non-linear MPC different simplification techniques used to be used in order to make the computation of the optimal control action more tractable. Usually linearization of the non-linear mathematical model used to be done around the given operation point. In some other cases based on more operation points a PWA mathematical model used to be introduced in order to approximate a non-linear behavior of the real system. In the future it would be interesting not to simplify the mathematical model and obtain the real optimal control action. It would be beneficial to compare the performance of the linear, PWA or non-linear models. Therefore our goal is to explore the possibilities of methods like direct single-shooting, direct multiple-shooting or orthogonal collocation. Each of the methods uses different techniques to solve the construction NLP. In the future it would be beneficial to generalize these methods based on their computational requirements and control performance for different systems. It could be a good survey for the future scientist.

Bibliography

Mixed-integer programming for control. pages 2676–2683, 2005. 42

MOSEK ApS. *The MOSEK optimization toolbox for MATLAB manual. Version 7.1 (Revision 28).*, 2015. URL http://docs.mosek.com/7.1/toolbox/index.html. 27

V. Baldoni, N. Berline, J. A. De Loera, M. Köppe, and M. Vergne. How to integrate a polynomial over a simplex. *Mathematics of Computation*, 80(273):297, 2010. 90, 92

Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38: 3–20, 2002. doi: 10.1016/S0005-1098(01)00174-1. 52, 63

Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Computer Aided Control Systems Design, 2004 IEEE International Symposium*, 38:284–289, 2004. doi: 10.1109/CACSD.2004.1393890. 146

F. Borrelli. *Constrained Optimal Control of Linear and Hybrid Systems*, volume 290 of *Lecture Notes in Control and Information Sciences*. Springer-Verlag, 2003. 44

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, Cambridge, Cambridge, 2004. 27, 35, 45

Tommaso Bresciani. *Modelling, Identification and Control of a Quadrotor Helicopter*. Msc. thesis, Lund University, 2008. 111

P.J. Bristeau, F. Callou, D. Vissière, and N. Petit. The Navigation and Control technology inside the AR . Drone micro UAV. In *Proceedings of the 18th IFAC World Congress, 2011*, volume 18, pages 1477–1484, 2011. ISBN 9783902661937. doi: 10.3182/20110828-6-IT-1002.02327. URL http://www.ifac-papersonline.net/Detailed/47959.html. 109, 110

E.F. Camacho, D.R. Ramírez, D. Limón, and T. Álamo. Model predictive control techniques for hybrid systems. {*IFAC*} *Proceedings Volumes*, 42:1 – 13, 2009. 67

George Dantzig and Mukund Thapa. *Linear Programming 2: Theory and Extensions*. Springer-Verlag, 2003. 39

J. Engel, J. Sturm, and D. Cremers. Camera-based navigation of a low-cost quadrocopter. In *IEEE International Conference on Intelligent Robots and Systems*, pages 2815–2821, 2012. 112, 115

G. Ferrari-Trecate, F. A. Cuzzola, D. Mignone, and M. Morari. Analysis of discrete-time piecewise affine and hybrid systems. 38(12):2139–2146, December 2002. 90

H.J. Ferreau, C. Kirches, A. Potschka, H.G. Bock, and M. Diehl. qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, 2014. 41

Richard W. Freedman and Alok Bhatia. Adaptive dynamic matrix control: Online evaluation of the dmc model coefficients. *American Control Conference, 1985*, pages 220–225, 1985. 51

T. Gal and J. Nedoma. Multiparametric linear programming. *Management Science*, 18:406–442, 1972. 44

GLPK. *GLPK (GNU Linear Programming Kit)*, 2012. URL http://www.gnu.org/software/glpk/. 27

Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2015. URL http://www.gurobi.com. 27, 42

M. Herceg, M. Kvasnica, C.N. Jones, and M. Morari. Multi-Parametric Toolbox 3.0. In *Proc. of the European Control Conference*, pages 502–510, Zürich, Switzerland, July 17–19 2013a. http://control.ee.ethz.ch/~mpt. 97

M. Herceg, S. Mariethoz, and M. Morari. Evaluation of piecewise affine control law via graph traversal. In *European Control Conference*, pages 3083–3088, Zurich, Switzerland, July 2013b. ISBN 9783033039629. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6669667. 64

J. Holaza, B. Takács, and M. Kvasnica. *Selected Topics in Modelling and Control*, chapter Complexity Reduction in Explicit MPC via Bounded PWA Approximations of the Cost Function, pages 27–32. Number 8. Slovak University of Technology Press, 2012. URL http://www.kirp.chtf.stuba.sk/publication_info.php?id_pub=1356. 80

Inc ILOG. Cplex 8.0 user manual. 2003. URL http://www.ilog.fr/products/cplex/. 42

J.L. Jerez, E.C. Kerrigan, and G.A. Constantinides. A condensed and sparse qp formulation for predictive control. *Decision and Control and European Control Conference*, pages 5217–5222, 2010. doi: 10.1109/CDC.2011.6160293. 54

C.N. Jones, M. Baric, and M. Morari. Multiparametric Linear Programming with Applications to Control. *European Journal of Control*, 13(2-3):152–170, March 2007. 80, 84

Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960. 51

M. Klaučo and M. Kvasnica. Explicit mpc approach to pmv-based thermal comfort control. In *53rd IEEE Conference on Decision and Control*, volume 53, pages 4856–4861, Los Angeles, California, USA, December 15-17, 2014 2014. URL http://www.kirp.chtf.stuba.sk/publication_info.php?id_pub=1551. 123

M. Klaučo, J. Drgoňa, M. Kvasnica, and S. Di Cairano. Building temperature control by simple mpc-like feedback laws learned from closed-loop data. In *Preprints of the 19th IFAC World Congress Cape Town*

(South Africa) August 24 - August 29, 2014, pages 581–586, 2014. URL http://www.kirp.chtf.stuba.sk/publication_info.php?id_pub=1527. 123

T Krajník, V Vonásek, D Fišer, and J Faigl. AR-drone as a platform for robotic research and education. In *Proceedings of Communications in Computer and Information Science (CCIS)*, 2011. 112

M. Kvasnica. *Selected Topics on Constrained and Nonlinear Control. Workbook*, chapter Multi-Parametric Toolbox, pages 101–170. STU Bratislava - NTNU Trondheim, 2011. URL http://www.kirp.chtf.stuba.sk/publication_info.php?id_pub=1057. 64

M. Kvasnica, B. Takács, J. Holaza, and S. Di Cairano. On region-free explicit model predictive control. In *54rd IEEE Conference on Decision and Control*, volume 54, pages 3669–3674, Osaka, Japan, December 15-18, 2015 2015. URL http://www.kirp.chtf.stuba.sk/publication_info.php?id_pub=1675. 158

Huibert Kwakernaak and Raphael Sivan. *Linear Optimal Control Systems*. Wiley-Interscience, 1972a. 51

Huibert Kwakernaak and Raphael Sivan. *Linear Optimal Control Systems*. Wiley-Interscience, 1972b. 53

J.B. Lasserre and K.E. Avrachenkov. The Multi-Dimensional Version of $\int_a^b x^p dx$. *The American Mathematical Monthly*, 108(2):151–154, 2001. 92

L Ljung. System identification toolbox. *The Matlab user's guide*, 1:237, 2011. 113

J. Löfberg. Yalmip : A toolbox for modeling and optimization in matlab. *CACSD Conference*, 2004. 142

B.P. Gerkey M. Quigley, K. Conley and J. Faust. ROS: an open-source Robot Operating System. In *Proceedings of ICRA Workshop on Open Source Software*, 2009. URL https://www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf. 111

Y. Ma, S. Vichik, and F. Borrelli. Fast stochastic mpc with optimal risk allocation applied to building control systems. In *2012 IEEE 51st IEEE Conference on*

Decision and Control (CDC), pages 7559–7564, Dec 2012. doi: 10.1109/CDC.2012.6426251. 123

J. Maciejowski. *Predictive Control with Constraints*. Tokyo Denki University Press, Prentice Hall, 2002. 52, 53, 62

MATLAB. *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts, 2010. 140

D. Q. Mayne, J.B. Rawling, C.V. Rao, and P. O. M. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36:789–814, 2000. doi: 10.1016/S0005-1098(99)00214-9. 51, 53

Ngoc Anh Nguyen, Sorin Olaru, and Pedro Rodriguez-Ayerbe. On the complexity of the convex liftings-based solution to inverse parametric convex programming problems. In *Control Conference (ECC), 2015 European*, pages 3428–3433. IEEE, 2015. 64

F. Oldewurtel, A. Parisio, C. N. Jones, M. Morari, D. Gyalistras, M. Gwerder, V. Stauch, B. Lehmann, and K. Wirth. Energy efficient building climate control using stochastic model predictive control and weather predictions. In *Proceedings of the 2010 American Control Conference*, pages 5100–5105, June 2010. doi: 10.1109/ACC.2010.5530680. 123

Frauke Oldewurtel, Alessandra Parisio, Colin N. Jones, Dimitrios Gyalistras, Markus Gwerder, Vanessa Stauch, Beat Lehmann, and Manfred Morari. Use of model predictive control and weather forecasts for energy efficient building climate control. *Energy and Buildings*, 45:15 – 27, 2012. ISSN 0378-7788. doi: http://dx.doi.org/10.1016/j.enbuild.2011.09.022. URL http://www.sciencedirect.com/science/article/pii/S0378778811004105. 123

Li Qianying. *Grey-Box System Identification of a Quadrotor Unmanned Aerial Vehicle*. PhD thesis, Delft University of Technology, 2014. 111

Guilherme V. Raffo, Manuel G. Ortega, and Francisco R. Rubio. An integral predictive/nonlinear H-inf control structure for a quadrotor helicopter. *Automatica*, 46:29–39, 2010. 111

J.A. Rossiter and B. Kouvaritakis. Constrained stable generalised predictive control. *Control Theory and Applications, IEE Proceedings D*, 140:243–254, 2004. 56

P.O.M Scokaert and J.B. Rawlings. Constrained linear quadratic regulation. *Control Theory and Applications, IEE Proceedings D*, 43:1163–1169, 1998. doi: 10.1109/9.704994. 56

B. Takács, J. Holaza, M. Kvasnica, and S. Di Cairano. Nearly-optimal simple explicit mpc regulators with recursive feasibility guarantees. In *IEEE Conference on Decision and Control*, pages 7089–7094, Florence, Italy, 2013. URL http://www.kirp.chtf.stuba.sk/publication_info.php?id_pub=1469. 88

S. Tanthadiloke, P. Kittisupakorn, and M. Mujtaba. Improvement of production performance of a hard chromium electroplating via operational optimization. In *Proceeding of the 5th Regional Conference on Chemical Engineering, Thailand*, 2013. 136

P. Tøndel, T.A. Johansen, and A. Bemporad. Evaluation of piecewise affine control via binary search tree. *Automatica*, 39(5):945–950, 2003. 69

A. van Schijndel. Integrated heat, air and moisture modeling and simulation in hamlab. In *IEA Annex 41 working meeting, Montreal, May*, 2005. 126

Jan Široký, Frauke Oldewurtel, JiĹ™Ă Cigler, and Samuel PrĂvara. Experimental analysis of model predictive control for an energy efficient building heating system. *Applied Energy*, 88(9):3079 – 3087, 2011. ISSN 0306-2619. doi: http://dx.doi.org/10.1016/j.apenergy.2011.03.009. URL http://www.sciencedirect.com/science/article/pii/S0306261911001668. 123

Y. Wang, C. Jones, and J. Maciejowski. Efficient point location via subdivision walking with application to explicit MPC. In *Proc. European Control Conf.*, pages 447–453, 2007. ISBN 9789608902855. 64

L.B. Willner. On Parametric Linear Programming. *SIAM Journal on Applied Mathematics*, 15(5):1253–1257, September 1967. 44

L. Zitko, G. Cushnie, P. Chalmer, and R. Taylor. Hard chrome plating traing course. 2010. 139

Author's Publications

List of publications of the author in the field of complexity reduction and implementation of model predictive control technique is as follows:

- Articles in journals indexed in Current Contents Database

  1. Holaza, J. – Takács, B. – Kvasnica, M. – Di Cairano, S.: Nearly optimal simple explicit MPC controllers with stability and feasibility guarantees. In *Optimal Control Applications and Methods*, vol.6, 2015.

- Articles in international conference proceedings

  1. Takács, B. – Holaza, J. – Kvasnica, M. : NLP-based Derivation of Bounded Convex PWA Functions: Application to Complexity Reduction in Explicit MPC. In *Veszprém Optimization Conference: Advanced Algorithms*, Veszprém, Hungary, pp. 95–95, 2012.
  2. Holaza, J. – Takács, B. – Kvasnica, M. : Complexity Reduction in Explicit MPC via Bounded PWA Approximations of the Cost Function. In *Selected Topics in Modelling and Control, Editor(i): Mikleš, J., Veselý, V., Slovak University of Technology Press, vol. 8*, pp. 27–32, 2012.
  3. Takács, B. – Holaza, J. – Kvasnica, M. – Di Cairano, S. : Nearly-Optimal Simple Explicit MPC Regulators with Recursive Feasibility Guarantees. In *IEEE Conference on Decision and Control*, Florence, Italy, pp. 7089–7094, 2013.

4. Holaza, J. – Takács, B. – Kvasnica, M. : Simple Explicit MPC Controllers Based on Approximation of the Feedback Law. In *ACROSS Workshop on Cooperative Systems*, Zagreb, Croatia, pp. 48–49, 2014.
5. Holaza, J. – Takács, B. – Kvasnica, M. : Verification of Performance Bounds for A-Posteriori Quantized Explicit MPC Feedback Laws. In *Preprints of the 19th IFAC World Congress Cape Town*, Cape Town, South Africa, pp. 1035–1040, 2014.
6. Kvasnica, M. – Takács, B. – Holaza, J. – Di Cairano, S. : On Region-Free Explicit Model Predictive Control. In *54rd IEEE Conference on Decision and Control*, Osaka, Japan, pp. 3669–3674, 2015.
7. Kvasnica, M. – Holaza, J. – Takács, B. – Ingole, D. : Design and Verification of Low-Complexity Explicit MPC Controllers in MPT3. In *European Control Conference*, Linz, Austria, pp. 2600–2605, 2015.
8. Holaza, J. – Takács, B. – Kvasnica, M. : Safety Verification of Implicitly Defined MPC Feedback Laws. In *European Control Conference*, Linz, Austria, pp. 2552–2557, 2015.
9. Takács, B. – Števek, J. – Valo, R. – Kvasnica, M. : Python Code Generation for Explicit MPC in MPT. In *European Control Conference*, Aalborg, Denmark, 2016.

- Articles in domestic conference proceedings

1. Holaza, J. – Takács, B. – Kvasnica, M. : Synthesis of Simple Explicit MPC Optimizers by Function Approximation. In *Proceedings of the 19th International Conference on Process Control*, Štrbské Pleso, Slovakia, pp. 377–382, 2013.
2. Kvasnica, M. – Takács, B. – Holaza, J. – Ingole, D. : Reachability Analysis and Control Synthesis for Uncertain Linear Systems in MPT. In *Proceedings of the 8th IFAC Symposium on Robust Control Design*, Bratislava, Slovakia, pp. 302–307, 2015.
3. Ingole, D. – Holaza, J. – Takács, B. – Kvasnica, M. : FPGA-Based Explicit Model Predictive Control for Closed-Loop Control of Intravenous Anesthesia. In *Proceedings of the 20th International Conference on Process Control*, Štrbské Pleso, Slovakia, pp. 42–47, 2015.

4. Takács, B. – Holaza, J. – Števek, J. – Kvasnica, M. : Export of Explicit Model Predictive Control to Python. In *Proceedings of the 20th International Conference on Process Control*, Štrbské Pleso, Slovakia, pp. 78–83, 2015.

Curriculum Vitae

Bálint Takács

Date of Birth: May 19, 1988
Citizenship: Slovakia
Email: balint.takacs1@gmail.com
Homepage: http://www.kirp.chtf.stuba.sk/∼takacs

Education

- B.S. Process Control, Slovak University of Technology, 2010.
  - *Minors*: Optimal Process Control and Chemical Engineering.
- M.S. Process Control, Slovak University of Technology, 2012
  - *Minors*: Model Predictive Control, Convex Optimization
- Ph.D. Process Control, Slovak University of Technology, *expected* 2016
  - *Majors*: Synthesis and Implementation of Model Predictive Control
  - *Minors*: Convex Optimization
  - *Advanced Study*: Design MPC controller on hard chromium plating process at Chulalongkorn University, Bangkok, Thailand, January 2016 – April 2016

– *Advanced Study*: Code Generation of the Parametric Solution of the Optimization Problem at Czech Technical University, Prague, Czech Republic, May 2016 – July 2016

Research Fields

- Explict Model Predictive Control, Complexity Reduction, Code Generation

Miscellaneous

Computer Skills

- C/C++, Python, Matlab, Simulink, PHP, HTML, XML
- Windows, Unix/Linux, MS Office, Keynote, LaTeX
- Siemens Simatic, Foxboro

Language Skills

- Hungarian, Slovak, English

Resumé

Predkladaná dizertačná práca pojednáva o syntéze a implementácii prediktívneho riadenia. Prediktívne riadenie je moderný prístup, ktorý používa matematický opis reálneho procesu na predpoveď jeho budúceho správania. V porovnaní s klasickými prístupmi, výhodou prediktívneho riadenia je schopnosť predpovedať vývoj systému v čase. Na základe toho je možné určiť taký akčný zásah, aby mohli byť dodržané vopred určené obmedzenia na stavové, výstupné a vstupné veličiny. Okrem týchto obmedzení sa dajú zakomponovať aj zložitejšie relácie. Kvalitu regulačného priebehu môžeme ovplyvniť správne zvolenou účelovou funkciou. Po úspešnom naformulovaní problému optimálneho riadenia sa daná formulácia premietne do tvaru optimalizačnej úlohy, ktorú je možné vyriešiť pomocou optimalizačných nástrojov. Musí sa však vyriešiť opakovane za daný vzorkovací čas. Samotný výpočet je časovo náročný a vyžaduje vhodný výpočtový výkon. Niekedy nie je možné zvýšiť maximálny čas alebo zabezpečiť silnejší hardvér. Týmto obmedzeniam sa dokážeme vyhnúť implementáciou explicitného prediktívneho riadenia, ktoré predstavuje predpočítanie optimalizačného problému pre všetky možné riešiteľné počiatočné hodnoty. Takýto predpočítaný optimalizačný problém nepotrebuje v implementačnej fáze žiadne dodatkové numerické nástroje na riešenie, vyžaduje iba maticové násobenie a sčítanie. Počas riešenia nie je potrebné rátať inverzie alebo používať delenie, tým pádom je samotná implementácia jednoduchšia a správnosť výsledného algoritmu sa dá jednoduchšie overiť. Kvôli nízkym implementačným požiadavkám je táto forma riadenia veľmi populárna a žiadaná v priemysle.

Nevýhodou explicitného riešenia je jeho pamäťová náročnosť. Explicitný predik-

tívny regulátor je výsledkom parametrického riešenia optimalizačného problému formou PWA funkcie, ktorá je definovaná nad polytopickými (mnohostennými) regiónmi. Pre správne vyhodnotenie je potrebné si uložit' informácie o všetkých regiónoch. Ich množstvo exponenciálne rastie s dĺžkou predikčného horizontu a v čím vyššej dimenzii je definovaný parametrický priestor, tým viac pamäte je potrebnej na uloženie daného regiónu. Konečná pamät'ová stopa výsledného regulátora môže prekročit' fyzikálne hranice daného zariadenia a tým pádom sa nedá implementovat'. Našim cieľom je navrhnút' rôzne techniky na zníženie zložitost parametrického riešenia optimalizačného problému. Prvý prístup prezentovaný v tejto práci využíva vlastnosti účelovej funkcie. V prípade, že tvar účelovej funkcie optimalizačného problému je lineárny, tak aj samotné parametrické riešenie bude po častiach afínne. V prípade, že sa hodnota účelovej funkcie zvýši o danú hodnotu (stage cos), vytvorí sa trubica, v ktorej je zaručená stabilita systému. Cieľom je nájst' novú po častiach afínnu funkciu, ktorá sa nachádza vo vymedzenej trubici. Tento problém sa dá naformulovat' ako nelineárny optimalizačný problém. Nevýhodou tohto prístupu je náročnost' riešenia optimalizačného problému aj pri nižších dimenziách. Ako d'alší spôsob na redukciu zložitosti parametrického riešenia uvažujeme v tejto práci o aproximácii zákona riadenia. Pri tomto postupe sa znova skonštruuje parametrické riešenie s nižším predikčným horizontom, aby sa ziskali nové kritické regióny, nad ktorými by sa mal vytvorit' aproximovaný akčný zákon. Ak chceme, aby bola únia kritických regiónov rovnaká pri formulácii optimalizačného problému, použije sa maximálna pozitívna invariantná množina. Určia sa prieniky jednotlivých regiónov, ktoré sa delia d'alej na simplexy. Nad simplexami už môžeme aplikovat' aproximáciu zákona riadenia využiteľnú už aj vo vyšších dimenziách. Tieto aproximačné metódy síce znižujú výkon regulátora, ale aspoň zabezpečia splnenie pamät'ových kritérií na úspešnú implementáciu.

Pri implementácii explicitného prediktívneho riadenia je potrebné uložit' parametrické riešenie ako aj vytvorit' algoritmus, ktorý na základe vstupného parametra vyextrahuje riešenie z optimalzačného problému. Existuje viacero druhov algoritmov. Táto práca sa zaoberá s tromi najznámejším prístupmi:

1. sekvenčné vyhľadávanie
2. rozšírené sekvenčné vyhľadávanie
3. binárny strom

Pri sekvenčnom prehľadávaní algoritmus vyšetruje, či daný bod patrí do regiónu alebo nie. V prípade, že sa bod nenachádza v regióne, jeho index as použije na výpočet akčného člena z uloženého afínneho zákona riadenia. V nasledovnom kroku sa celý proces začína od začiatku. V prípade rozšíreného sekvenčného vyhľadávania sa okrem vyšetrovania, či daný bod patrí do regiónu, ešte vyhodnotí hodnota účelovej funkcie v danom bode. Následne sa vyberie index regiónu, v ktorom je hodnota vyhodnotenej účelovej funkcie najnižšia. Tento prístup sa využíva vtedy, keď samotná funkcia akčného zákona nie je spojitá alebo sa prekrývajú regóny. Posledná možnosť je najefektívnejšia spomedzi troch spomenutých. Pri prehľadávaní binárneho stromu krok za krokom, sa dá vylúčiť polovica neprehľadaných regiónov. Táto metóda je využívaná vtedy, keď parametrické riešenie pozostáva z veľkého množstva regiónov. Ako príklad uvedieme 1000 regiónov. V tomto prípade, na zistenie indexu požadovaného regiónu je potrebných iba 10 porovnávacích operácií. Kým už samotné vyhľadávanie je veľmi efektívne, skonštruovanie binárneho stromu hlavne vo vyšších dimenziách je časovo i výpočtovo náročné. Cieľom tejto práce je navrhnúť nový modul do Multi-Parametrického Toolbox-u, ktorý by zabezpečil automatické generovanie zvoleného algoritmu a potrebnej informácie o parametrickom riešení. Modul bol navrhnutý tak, aby si mohol koncový užívateľ veľmi jednoducho vyberať zo spomenutých algoritmov. Na exportovanie parametrického riešenia stačí v programovom prostredí Matlab vyhodnotiť iba jednu funkciu. Táto funkcia vytvorí na základe požiadaviek od užívateľa nový súbor, ktorý bude obsahovať všetky potrebné informácie. Ďalší vygenerovaný súbor je sebestačný, čo znamená, že nepotrebuje ďalšie externé knižnice pri implementácii. Modul podporuje export do programového prostredia Python a JavaScript, kým názvy funkcií zabezpečujúce export v Matlabe sú:

1. toPython
2. toJavaScript

Obe funkcie požadujú argumenty ako parametrické riešenie optimalizačného problému, názov vygenerovanej funkcie a typ prehľadávacieho algoritmu. Po zadaní správnych argumentov vytvorí modul nový súbor: v prípade exportu do Pythonu s príponou `.py`, v prípade exportu da JavaScript-u s príponou `.js` a `.html`. Súbor HTML slúži už len na kontrolu, či export prebehol úspešne. V prípade obidvoch jazykov je spájanie už s existujúcimi aplikáciami veľmi jednoduché. Implementovateľnosť modulu je ukázaná na troch odlišných aplikáciách. Na jednej známej

počítačovej hre sme sa snažili navrhnúť prediktívne riadenie na riadenie výšky. Druhá aplikácia sa zaoberá riadením lietajúceho zariadenia s názvom Ar.Drone2, kým posledná aplikácia ukazuje možnosť využitia exportovacieho modulu na riadenie tepelného komfortu v budove.

Posledné kapitoly práce opisujú implementáciu prediktivného riadenia na dvoch odlišných lineárnych systémoch. Jeden systém predstavuje problém riadenia galvanizácie a druhý riadenie polohy guličky pomocou magnetického pola. Oba systémy sú nelineárne a pri návrhu prediktívneho riadenia boli použité rôzne prístupy na vysporiadanie sa s nelinearitou.

Ako ďalšie pokračovanie tejto práce uvažujeme o pokračovaní vo vývoji exportovacieho modulu tak, aby umožnil implementáciu nielen explicitného prediktívneho riadenia, tým pádom by koncový užívateľ mohol využívať benefity aj explicitného, aj online prístupu. Výsledky jednotlivých sekcií, ktoré vznikli v spolupráci s kolegami z České Technické Učení v Praze, Chulalongkorn University in Bangkok a Slovenskej Technickej Univerzity v Bratislave, ešte nebol odpublikované. Spomenuté výsledky sú zo sekcií opisujúcich riadenie tepelného komfortu, riadenie galvanizácie a riadenie guličky pomocou magnetického pola, ktoré so spoluautormi plánujeme publikovať po doladení detailov v blízkej budúcnosti.