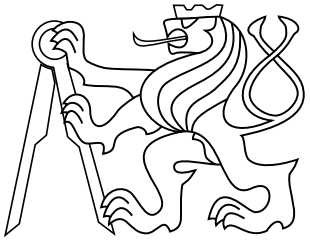
Master's Thesis

Fast Model Predictive Control with Long Prediction Horizon

Jiří Burant



April 2017

Advisor: Ing. Pavel Otta

Czech Technical University in Prague
Faculty of Electrical Engineering, Department of Control Engineering

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Burant Jiří**

Studijní program: Kybernetika a robotika
Obor: Systémy a řízení

Název tématu: **Rychlé prediktivní řízení s dlouhým horizontem predikce**

Pokyny pro vypracování:

1. Popiště prediktivní řízení a způsoby jeho řešení pomocí kvadratického programování.
2. Navrhněte efektivní řešení pro řízení s dlouhým horizontem predikce.
3. Odvoďte blokování vstupů pro vámi navržené řešení.
4. Implementujte a vyhodnoťte vámi navržené řešení.

Seznam odborné literatury:

[1] Y. Wang and S. Boyd, 'Fast Model Predictive Control Using Online Optimization,' in IEEE Transactions on Control Systems Technology, vol. 18, no. 2, pp. 267-278, March 2010.
[2] B. O'Donoghue, G. Stathopoulos and S. Boyd, 'A Splitting Method for Optimal Control,' in IEEE Transactions on Control Systems Technology, vol. 21, no. 6, pp. 2432-2442, Nov. 2013.
[3] O. Santin, 'Numerical Algorithms of Quadratic Programming for Model Predictive Control,' as Dissertation thesis at DCE, FEE, CTU Prague, 2016.
[4] Other scientific publications.

Vedoucí: Ing. Pavel Otta

Platnost zadání: do konce letního semestru 2017/2018

L.S.

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze, dne 10. 2. 2017

Acknowledgement

My thanks go mainly to my thesis supervisor Ing. Pavel Otta for his guidance throughout the work. I would also like to thank Ing. Ondřej Šantin PhD for his help and his work.

Finally, I would like to thank my family for supporting me through all my studies.

Declaration

I declare that I worked out the presented thesis independently and I quoted all used sources of information in accord with Methodical instructions about ethical principles for writing academic thesis.

Abstract

Tato práce poskytuje krátký úvod k řešení MPC řídícího problému a představuje efektivní algoritmus pro problémy s dlouhým predikčním horizontem. Výpočetní složitost řešení MPC problému roste podstatně s délkou predikčního horizontu. V případě klasické kondenzované formulace je růst s délkou predikčního horizontu kvadratický. Tato práce prezentuje nový algoritmus, využívající specifickou strukturu problému pro dosažení pouze lineární závislosti na délce predikčního horizontu. Navíc je zde prezentována i nová metoda blokování vstupů pro MPC problém, která výrazně redukuje výpočetní složitost.

Algoritmus je založen na modifikované metodě Aktivních množin, používající Schurovu metodu a metodu Nulového prostoru pro aplikaci omezení a pro řešení Newtonova kroku. Dále je využita Projekce Newtonova kroku pro aplikaci omezení. Všechny kroky algoritmu jsou detailně popsány a jejich výpočetní složitost je analyzována.

Výsledný algoritmus byl otestován na sadě systémů. Výsledky prokázaly lineární růst výpočetní náročnosti a také ukázaly, že algoritmus je vhodný pro velké systémy (se stovkami optimizačních proměnných).

Klíčová slova

Prediktivní řízení; Kvadratické programování; Metoda aktivních množin; Warm-start

Abstract

This thesis aims to give a brief introduction into solving of the MPC problem and develop an effective algorithm for problems with long prediction horizon. The computation cost of solving the MPC problem grows significantly with the length of the prediction horizon. In the case of classical dense formulation approach, the growth in the length of prediction is quadratic. This thesis presents a novel algorithm exploiting the special structure of the problem to obtain only linear dependency in the length of the prediction horizon. Moreover, a move blocking strategy for MPC problems in sparse form is presented to reduce the computation burden significantly.

The algorithm is based on a modified Active Set method, using Range Space method and Null Space method to apply the constraints and to solve the dual Newton step. Furthermore, a Newton Projection method is used for projection of the inequality constraints. All the steps of the algorithm are described in detail and their computation cost is analyzed.

The resulting algorithm has been tested on various problems. The results prove the linear growth of the computation load and also show that the algorithm is advantageous for large systems (hundreds of optimization variables).

Keywords

Model Predictive Control; Quadratic Programming; Active Set method; Warm-start

Contents

1 Introduction

Model Predictive Control (MPC) is an increasingly popular method of the discrete-time control. It presents an effective approach to system based control, offering powerful capabilities able to deal with wide range of problems and ensuring the best possible control, satisfying the physical limits of the system. A more detailed description follows, also more information about MPC can be found in [1],[2].

Model Predictive Control is traditionally popular in industrial processes. The first MPC-controlled systems emerged in the seventies in the chemical industry, [3]. The chemical industry was suitable for this type of control, because of slow dynamics and long process times, that gave then computers enough time to solve the optimization problem. As the technology progressed, the MPC strategy could be used for more demanding tasks and faster processes. According to [3], there has been major progress in the MPC field during the last two decades, with a number of improvements to the MPC algorithms. MPC regulators can be found for example in automotive industry, in automated control of the intelligent buildings or the power control [4].

The main task of the MPC is to find a control sequence, minimizing the optimality criterion with respect to the controlled system dynamics and additional constraints on optimized variables. Those additional constraints represent the physical limitations of the system, for example the valve cannot be opened more than fully.

An important concept related to the MPC is *receding horizon*. Receding horizon means that every sampling period the sequence of optimal control is re-optimized for the shifted horizon. Each step only the first control from the optimal control sequence is applied. The optimization is carried out again; it is parametrized by the currently measured state. Thus the feedback is implemented. The concept of receding horizon is illustrated in Figure 1.

The process of solving the optimization problem can be very computationally demanding since there might be many inputs and states of the system or long prediction horizon is required. The method presented in this paper is focused on decreasing the computational burden of problems with long prediction horizon.

In the last decade, many outstanding algorithms specialized on MPC have been developed, for example: [6],[7],[8]. With respect to the type of the problem form, the algorithms can be divided into two groups. Firstly, algorithms solving the problem in *condensed* form and secondly, algorithms solving the problem in the *sparse* form

In the condensed form, there are fewer optimization variables, but the problem itself is more complex (ill-conditioned). Alternatively, in the sparse form, there are more optimization variables, but the form has specific sparsity pattern with usually fewer nonzero elements and is less complex (well-conditioned). Note that for condensed problem generic solvers (e.g. quadprog [9] or MOSEK [10]) can be used. For the sparse form of the MPC problem, special algorithms *have to* be utilized to exploit the structure of the problem. That is why the condensed form was more popular in the past.

Algorithms dealing with the problem in the dense form are more common, and they are usually considered as a standard approach, [3]. Some of the important ones are briefly described in the following text.

Richter's FGP [11] – a Fast Gradient Projection algorithm solving condensed prob-

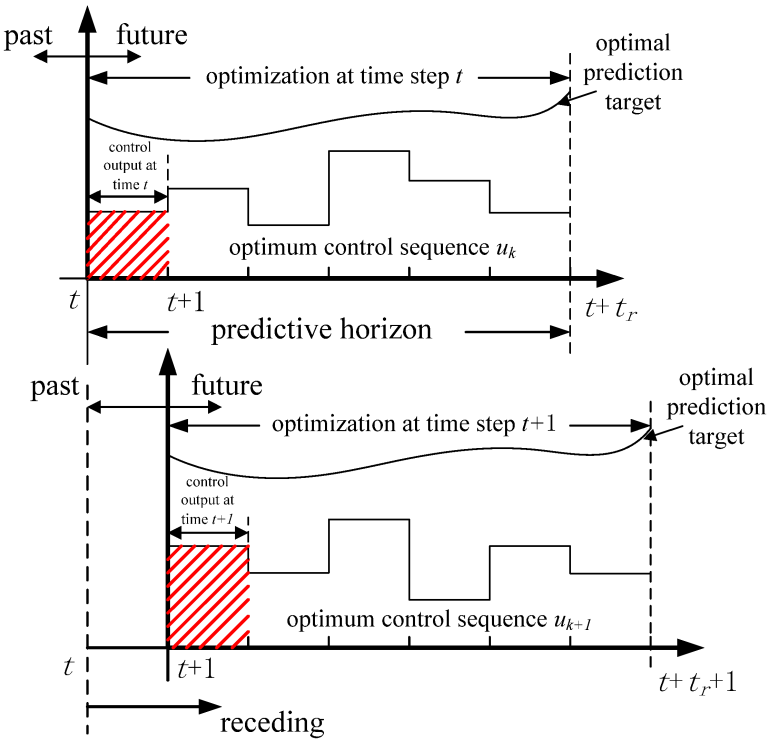


Figure 1 Receding horizon principle, [5]

lem form. Intended originally for AC-DC converters, requiring short prediction horizons (benchmarked with the length of horizon five), and small computation times (tens of μs) with an inexpensive hardware solution.

qpOASES [8] – a highly optimized implementation of Multiparametric Active Set method [12]. The algorithm of qpOASES was initially developed at KU Leuven. Currently, it is supported mainly by Heidelberg University and ABB Corporate Research. The software of qpOASES is versatile in a sense that it can be used both as a standalone application or with a conjunction with third-party software, such as MATLAB. The MATLAB interface for qpOASES was used in this work as a reference solution for the testing of the developed algorithm.

Quadprog [9] – a QP solver, which can be found in the Optimization toolbox of MATLAB. It uses the Interior Point method algorithm for finding the solution. It is a generic QP solver, meaning that it is not developed specifically for MPC problems. On the other hand, it can solve a large portfolio of problems and often is used as a reference when developing custom solvers.

FiOrdOs [13] – a Matlab toolbox for automated C-code generation of first-order methods. It features both the Gradient method and the Fast Gradient method. Since this is a first order solver, the iterations are cheap, but typically a large number of iterations is required to find the solution.

NPP [7]– a novel algorithm based on the Active Set method and optimized for the MPC problems with box constrained inputs. The algorithm combines the Active Set strategy with the proportioning test to decide when to change the active set. The minimization in each iteration is solved using the Newton step strategy. This algorithm forms the basis for the sparse form algorithm presented in this thesis.

Recently, several solvers for the sparse form of the MPC problem, exploiting problem structure have been developed.

fastMPC [15] – a Barrier Interior Point algorithm using Newton step solved by tailored Cholesky decomposition. It has been developed specifically for MPC and presented in a journal paper. This novel approach has the advantage of having complexity only linear in the prediction horizon.

qpDUNES [14] – an implementation of dual Newton step solving the MPC problem via Dual Decomposition. The Dual Decomposition allows to exploit the block-banded structure of the problem matrices, similarly as Interior Point methods, yet allowing the warm-starting capabilities of the Active Set method. A brief overview of the basic concepts can be found in [6].

FORCES [16] – a code generated Interior Point method optimization algorithm. It generates library-free ANSI-C code that is, according to the authors, several orders of magnitude faster and smaller than most other solvers. It has been designed specifically for real-time control. FORCES is suitable for several classes of optimization problems, apart from the quadratic programming (QP) problems, it is suitable for linear programming (LP) and quadratically constrained QPs (QCQP) problems.

ADMMmpc [17] – an Alternating Direction Method of Multipliers (ADMM) [18] algorithm using Interior Point method in the inner loop. This method became only recently used for solving the QPs but is becoming widely popular, as it is suitable for solving the sparse structured problems.

The contribution of this paper is two-fold. Firstly, it is the modification of the NPP algorithm from [7] suitable for the problems in sparse form with long prediction horizons. Several major modifications have been done, to change the problem formulation. Most importantly, the procedure of solving the Newton step has been changed, following the procedure from [15], where a similar approach has been presented for a Barrier Interior Point method. This Newton step approach is used to obtain linear dependency in prediction horizon instead of the cubic one. Also, the projection of the Newton step had to be changed to fit the sparse form of the problem.

Secondly, it is the formulation of a move blocking strategy for the sparse form of the MPC problem, which significantly reduces the dimension of the problem and can be used in combination with other sparse form solvers as well.

1.1 Organization of this Thesis

In the Chapter 1, the MPC strategy is presented, giving an introduction to the topic and explaining basic concepts. In the Chapter 2, the MPC control problem is presented. Moreover, several forms of MPC problem are presented along with basic concepts. In the Chapter 3, several methods for solving the resulting constrained QP are introduced. In the Chapter 4, the modified algorithm is described along with the move blocking strategy. In Chapter 5, implementation details are given and several tests are described. Finally, in Chapter 6 a conclusion of this work is given along with the suggestions for future improvements.

1.2 Notation

Throughout the paper, following notation is used

$\boldsymbol{v}$ – bold latin small letter denotes vector,

$\boldsymbol{M}$ – bold latin capital letter denotes matrix,

$\mathbb{M}$ – doubled latin capital letter denotes a matrix composed of other matrices,

$\boldsymbol{\lambda}$ – bold small greek letter denotes vector changed in every iteration of the algorithm,
$\boldsymbol{\Gamma}$ – greek capital letter denotes matrix changed in every iteration of the algorithm,
$\boldsymbol{A}^T$ – superscript T denotes transpose of matrix (or vector),
n_x – letter n with subscript denotes length of x,
$\boldsymbol{A} = \boldsymbol{A}^T$ – matrix equal to its transpose denotes symmetric matrix,
$\boldsymbol{A} > 0$ – denotes positive definite matrix,
$\mathcal{R}^m$ – denotes vector space of size m,
$\mathcal{R}^{m \times n}$ – denotes matrix space of size m times n,
$\Delta\boldsymbol{x}$ – symbol $\Delta\boldsymbol{x}$ denotes step in the given variable,
$\boldsymbol{x}^-$ – minus symbol in superscript denotes the value of the variable in current iteration,
$\mathcal{O}$ – denotes asymptotic complexity,
$\boldsymbol{A} \otimes \boldsymbol{B}$ – denotes kronecker product of the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$,
blkdiag$(\boldsymbol{A}, \boldsymbol{B})$ – denotes new matrix formed by stacking the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ on the diagonal,
FLOP – Floating-point operation, used to measure the computational complexity.

2 Model Predictive Control

The general procedure of MPC design is explained by the chart in Figure 2.

First of all, the system state model is described, the weight coefficients are set, and the constraints are determined. Then, the MPC cost function is formulated, usually in the form of a quadratic function with weights on inputs and states. The cost function is minimized, under the constraints. This thesis is focused on problems with the quadratic criterion and simple linear constraints on inputs.

The MPC problem is transformed into a QP problem to find the optimal sequence of control inputs in given time period k. Consecutively, this QP problem is solved. In the course of this work, a problem with inequality constraints on inputs is assumed. Therefore the QP problem has to be solved using an iterative method.

The QP problem is solved in every iteration; it is parametrized by current states and weights. A part of the solution of the QP problem is applied as control input back to the system, the rest is discarded and the algorithm continues with the next time sample.

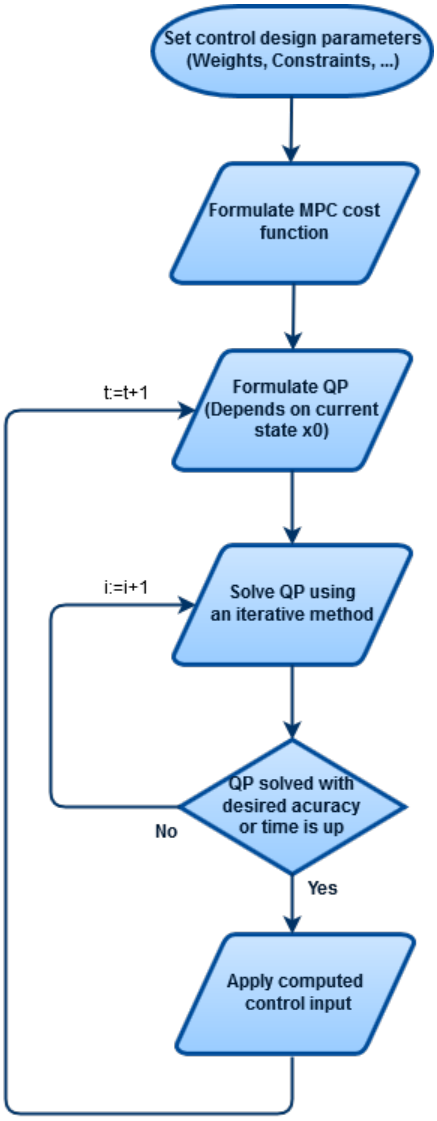


Figure 2 MPC control flowchart, t is discrete time step

2.1 Problem Description

The cost function has quadratic form and consists of the weighted sum of optimization variables. In most of the cases, those are states $\boldsymbol{x}_k$ and inputs $\boldsymbol{u}_k$, often weights on outputs $\boldsymbol{y}_k$ [19] or input change $\Delta\boldsymbol{u}_k$ can also be present, but they are omitted here for the sake of symplicity.

The following cost function is used in this thesis,

$$J(\boldsymbol{x}_k, \boldsymbol{u}_k) = \frac{1}{2}\boldsymbol{x}_{n_p}^T \boldsymbol{Q}_{n_p} \boldsymbol{x}_{n_p} + \frac{1}{2}\sum_{k=0}^{n_p-1} \begin{bmatrix} \boldsymbol{u}_k \\ \boldsymbol{x}_k \end{bmatrix}^T \begin{bmatrix} \boldsymbol{R} & \boldsymbol{S}^T \\ \boldsymbol{S} & \boldsymbol{Q} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_k \\ \boldsymbol{x}_k \end{bmatrix}. \tag{1}$$

Subscript k denotes time instant of the prediction horizon, n_p denotes the length of the prediction horizon. Symbol $\boldsymbol{x}_k \in \mathcal{R}^{n_x}$ is a vector of states, $\boldsymbol{u}_k \in \mathcal{R}^{n_u}$ is a vector of inputs. Matrices $\boldsymbol{Q}, \boldsymbol{Q}_{n_p}, \boldsymbol{R}, \boldsymbol{S}$ are the weight matrices.

The matrices $\boldsymbol{Q}, \boldsymbol{R}, \boldsymbol{S}$ are design parameters, and they are explicitly formulated, based on the performance demands, also the following condition must hold to ensure

that the problem is convex and has one global minimum,

$$\begin{bmatrix} \boldsymbol{R} & \boldsymbol{S}^T \\ \boldsymbol{S} & \boldsymbol{Q} \end{bmatrix} > 0.$$

The matrix $\boldsymbol{Q}_{n_p}$ is the weight on the final state, which can be obtained as the solution to the discrete-time algebraic Riccati equation, i.e.

$$\boldsymbol{Q}_{n_p} = \boldsymbol{Q} + \boldsymbol{A}^T\boldsymbol{Q}_{n_p}\boldsymbol{A} - (\boldsymbol{A}^T\boldsymbol{Q}_{n_p}\boldsymbol{B} + \boldsymbol{S})(\boldsymbol{R} + \boldsymbol{B}^T\boldsymbol{Q}_{n_p}\boldsymbol{B})^{-1}(\boldsymbol{B}^T\boldsymbol{Q}_{n_p}\boldsymbol{A} + \boldsymbol{S}^T).$$

The term is equivalent to extending the prediction horizon to infinity, as shown in [20].

Furthermore, the discrete linear time-invariant (LTI) system is assumed

$$\boldsymbol{x}(t+1) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t), \quad t > 0.$$

The full space problem consists of the cost function and problem constraints and can be formulated as

$$\begin{aligned} \min_{\boldsymbol{x}_k, \boldsymbol{u}_k} \quad & J(\boldsymbol{x}_k, \boldsymbol{u}_k) \\ \text{s.t.} \quad & \boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k, \quad \boldsymbol{x}_0 = \boldsymbol{x}(t) && \text{(2a)} \\ & \underline{\boldsymbol{u}}_k \leq \boldsymbol{u}_k \leq \overline{\boldsymbol{u}}_k, \quad k = 0, \ldots, n_p - 1. && \text{(2b)} \end{aligned}$$

The constraints on the solution are given by the dynamics of the controlled system, the limitations of the actuators and sensors and also by the requirements on the design and performance. The equation (2a) represents dynamics of the controlled system. The equation (2b) expresses constraints on inputs. The constraints can also be divided into two groups, *equality* constraints, and *inequality* constraints. Equality constraints are present in every MPC control problem, they represent the dynamics of the controlled system,

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k,$$

where $\boldsymbol{A}$, $\boldsymbol{B}$ are state-space matrices, $\boldsymbol{x}_k$ and $\boldsymbol{x}_{k+1}$ are states in the respective time samples and $\boldsymbol{u}_k$ is input at time instant k. Another reason for equality constraints might be the requirement on certain states to have a constant value, for example, to be zero at the end of the prediction horizon.

Inequality constraints can express physical limitations of actuators or saturations of the states. They can also limit the output of the system or the input change. In this thesis, only linear constraints on inputs are assumed.

$$\boldsymbol{A}\boldsymbol{u} \leq \boldsymbol{b} \tag{3}$$

Throughout this work, special attention is given to the *box constraints* on inputs are used. Box constraints are the simplest version of the linear constraints from (3). They are represented as upper and lower limits on inputs taken element-wise

$$\underline{\boldsymbol{u}}_k \leq \boldsymbol{u}_k \leq \overline{\boldsymbol{u}}_k.$$

Such constraints are common in the case of independent inputs, for example, two independent ventils in a pipeline.

The cost function (1) can be rewritten into the vector form

$$J(\boldsymbol{x}, \boldsymbol{u}) = \frac{1}{2} \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{x} \end{bmatrix}^T \begin{bmatrix} \mathbb{R} & \mathbb{S}^T \\ \mathbb{S} & \mathbb{Q} \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{x} \end{bmatrix}, \tag{4}$$

where the vectors and matrices are formed by stacking the values for every k, forming larger vectors and matrices as follows $\boldsymbol{x} = [\boldsymbol{x}_1^T, \boldsymbol{x}_2^T, \ldots, \boldsymbol{x}_{n_p}^T]^T$, $\boldsymbol{u} = [\boldsymbol{u}_0^T, \boldsymbol{u}_1^T, \ldots, \boldsymbol{u}_{n_p-1}^T]^T$, $\mathbb{Q} = \text{blkdiag}(\boldsymbol{I}_{n_p-1} \otimes \boldsymbol{Q}, \boldsymbol{Q}_{n_p}), \mathbb{R} = \boldsymbol{I}_{n_p} \otimes \boldsymbol{R}, \mathbb{S} = (0, I_{n_p-1} \otimes \boldsymbol{S})$.

Such vector form is more convenient for later derivations.

2.2 MPC Forms

In this section, the problem (2) is transformed into the quadratic programming optimization problem, to solve the MPC task. There are several ways of achieving that, the most common being the dense and the sparse form.

2.2.1 Condensed Form

One can formulate the *condensed* optimization problem. Such form is common in practice, it is used for example in [11], [8], [9].

First, the relation between $\boldsymbol{u}$ and $\boldsymbol{x}$ is transformed into a compact matrix form using the system dynamics, i.e.

$$\boldsymbol{x} = \mathbb{P}\boldsymbol{x}_0 + \mathbb{V}\boldsymbol{u}, \tag{5}$$

where matrices are

$$\mathbb{P} = \begin{bmatrix} \boldsymbol{A} \\ \boldsymbol{A}^2 \\ \boldsymbol{A}^3 \\ \vdots \\ \boldsymbol{A}^{n_p} \end{bmatrix}, \mathbb{V} = \begin{bmatrix} \boldsymbol{B} & 0 & 0 & \ldots & 0 \\ \boldsymbol{A}\boldsymbol{B} & \boldsymbol{B} & 0 & \ldots & 0 \\ \boldsymbol{A}^2\boldsymbol{B} & \boldsymbol{A}\boldsymbol{B} & \boldsymbol{B} & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{A}^{n_p-1}\boldsymbol{B} & \boldsymbol{A}^{n_p-2}\boldsymbol{B} & \boldsymbol{A}^{n_p-3}\boldsymbol{B} & \ldots & \boldsymbol{B} \end{bmatrix}.$$

By substituting (5) into criteria (4), the resulting form of the problem is obtained as

$$\begin{aligned} \min_{\boldsymbol{u}} \quad & \frac{1}{2}\boldsymbol{u}^T \widehat{\boldsymbol{H}} \boldsymbol{u} + \boldsymbol{u}^T \widehat{\boldsymbol{f}} \\ \text{s.t.} \quad & \underline{\boldsymbol{u}} \leq \boldsymbol{u} \leq \overline{\boldsymbol{u}}, \end{aligned}$$

where $\widehat{\boldsymbol{H}} = \mathbb{V}^T\mathbb{Q}\mathbb{V} + \mathbb{R} + \mathbb{S}^T\mathbb{V} + \mathbb{V}^T\mathbb{S}$ and $\widehat{\boldsymbol{f}} = \mathbb{V}^T\mathbb{Q}\mathbb{P}\boldsymbol{x}_0 + \mathbb{S}^T\mathbb{P}\boldsymbol{x}_0$.

The problem then is solved only in control variables $\boldsymbol{u}$, thanks to that the dimension of the problem is reduced significantly. The size of the Hessian $\widehat{\boldsymbol{H}}$ is $n_p n_u \times n_p n_u$. On the other hand, the problem has a complicated structure.

2.2.2 Sparse Form

Another form of the optimization problem is the *sparse* form. The resulting matrices are larger than in the case of the dense form, but they have a specific structure with large number of non-zero elements, which can be exploited.

The sparse formulation is becoming more popular in control community in the recent years, as new algorithms exploiting the specific structure are being developed, for example [14] or [15].

Instead of substituting for $\boldsymbol{x}$, the problem is solved in both variables, i.e.

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{u}} \quad & J(\boldsymbol{x},\boldsymbol{u}) \\ \text{s.t.} \quad & 0 = \mathbb{A}\boldsymbol{x} + \mathbb{B}\boldsymbol{u} + \boldsymbol{d} \\ & \underline{\boldsymbol{u}} \leq \boldsymbol{u} \leq \overline{\boldsymbol{u}}. \end{aligned} \tag{6}$$

The structure of the matrices and the vector $\boldsymbol{d}$ is following,

$$\mathbb{A} = \begin{bmatrix} -\boldsymbol{I} & & & & \\ \boldsymbol{A} & -\boldsymbol{I} & & & \\ & \boldsymbol{A} & -\boldsymbol{I} & & \\ & & \ddots & \ddots & \\ & & & \boldsymbol{A} & -\boldsymbol{I} \end{bmatrix}, \mathbb{B} = \begin{bmatrix} \boldsymbol{B} & & & \\ & \boldsymbol{B} & & \\ & & \boldsymbol{B} & \\ & & & \ddots \\ & & & \boldsymbol{B} \end{bmatrix}, \boldsymbol{d} = \begin{bmatrix} \boldsymbol{A}\boldsymbol{x}_0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Note that the prediction matrices from the previous Section 2.2.1 are actually

$$\mathbb{P}\boldsymbol{x}_0 = -\mathbb{A}^{-1}\boldsymbol{d}, \quad \mathbb{V} = -\mathbb{A}^{-1}\mathbb{B}.$$

2.2.3 Sparse Condensed Form

The forms mentioned above are the most common in practice, but different approaches can be found in the literature. In this Section, an interesting hybrid approach of condensed and sparse forms is described. The full description can be found in [21].

The basic idea of this approach is to introduce a state feedback control law

$$\boldsymbol{u}_k = \boldsymbol{K}\boldsymbol{x}_k + \boldsymbol{w}_k, \text{ for } k = 0, 1, 2, ..., n_p - 1, \tag{7}$$

where $\boldsymbol{K}$ is a gain matrix of the state feedback and $\boldsymbol{w}_k$ is the new vector of inputs. Note that no actual pre-stabilizing feedback is implemented in the end. The law (7) is just a trick to reformulate the problem.

The dynamics of the system (2a) are rewritten as

$$\boldsymbol{x}_{k+1} = (\boldsymbol{A} + \boldsymbol{B}\boldsymbol{K})\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{w}_k = \boldsymbol{A}_K\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{w}_k. \tag{8}$$

Using (8), the dynamics over prediction horizon n_p are setup in a similar way as in condensed formulation

$$\boldsymbol{x} = \mathbb{A}_K\boldsymbol{x}_0 + \mathbb{B}_K\boldsymbol{w},$$

where $\boldsymbol{w} = [\boldsymbol{w}_0^T, \boldsymbol{w}_1^T, \boldsymbol{w}_2^T, \ldots, \boldsymbol{w}_{n_p-2}^T, \boldsymbol{w}_{n_p-1}^T]^T$ is the sequence of new inputs,

$$\mathbb{A}_K = \begin{bmatrix} \boldsymbol{A}_K \\ \boldsymbol{A}_K^2 \\ \vdots \\ \boldsymbol{A}_K^{r-1} \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \mathbb{B}_K = \begin{bmatrix} \boldsymbol{B} & 0 & 0 & \dots & 0 & 0 & 0 \\ \boldsymbol{AB} & \boldsymbol{B} & 0 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \boldsymbol{A}_K^{r-1}\boldsymbol{B} & \boldsymbol{A}_K^{r-2}\boldsymbol{B} & \boldsymbol{A}_K^{r-3}\boldsymbol{B} & \ddots & \vdots & \vdots & \vdots \\ 0 & \boldsymbol{A}_K^{r-1}\boldsymbol{B} & \boldsymbol{A}_K^{r-2}\boldsymbol{B} & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \boldsymbol{A}_K^{r-1}\boldsymbol{B} & \ddots & \vdots & \vdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \vdots \\ \vdots & \dots & \dots & \dots & \boldsymbol{AB} & \boldsymbol{B} & 0 \\ 0 & \dots & \dots & \dots & \boldsymbol{A}^2\boldsymbol{B} & \boldsymbol{AB} & \boldsymbol{B} \end{bmatrix}.$$

The symbol r is the controllability index of the matrices $\boldsymbol{A}, \boldsymbol{B}$. The weight matrices of the cost criterion from (1) are replaced by new matrices,

$$\begin{aligned} \boldsymbol{Q}_K &= \boldsymbol{Q} + \boldsymbol{K}^T\boldsymbol{R}\boldsymbol{K} + \boldsymbol{S}^T\boldsymbol{K} + \boldsymbol{K}^T\boldsymbol{S} \\ \boldsymbol{S}_K &= \boldsymbol{S} + \boldsymbol{K}^T\boldsymbol{R} \\ \boldsymbol{R}_K &= \boldsymbol{R}. \end{aligned}$$

The problem then can be expressed similarly as,

$$\min_{\boldsymbol{v}} \quad \frac{1}{2}\boldsymbol{w}^T\widetilde{\boldsymbol{H}}\boldsymbol{w} + \boldsymbol{w}^T\widetilde{\boldsymbol{f}},$$

where $\widetilde{\boldsymbol{H}} = \mathbb{B}_K^T\mathbb{Q}\mathbb{B}_K + (\mathbb{K}\mathbb{B}_K + \boldsymbol{I})^T(\mathbb{R}(\mathbb{K}\mathbb{B}_K + \boldsymbol{I}) + \mathbb{S}^T\mathbb{B}_K) + (\mathbb{B}_K^T\mathbb{S}(\mathbb{K}\mathbb{B}_K + \boldsymbol{I}), \widetilde{\boldsymbol{f}} = \boldsymbol{x}_0^T\mathbb{A}_K^T(\mathbb{Q}\mathbb{B}_K + \mathbb{S}(\mathbb{K}\mathbb{B}_K + \boldsymbol{I}) + \mathbb{K}^T(\mathbb{R}(\mathbb{K}\mathbb{B}_K + \boldsymbol{I}) + \mathbb{S}\mathbb{B}_K))$.

The main drawback of this formulation is that the input constraints are modified and cannot be expressed as box constraints anymore.

2.3 Move Blocking

Move blocking is a strategy used in MPC to decrease the computational complexity by decreasing the degree of freedom of the optimization problem. The idea is to reduce the number of independent inputs by fixing the input to be constant over several time steps, [22]. As a consequence, a new concept is introduced. The *control horizon* indicates the number of independent input vectors with size n_c lesser or equal to the length of the prediction horizon. Note that in the case of no move blocking, the control horizon is identical to the prediction horizon. Then the control can be separated into several blocks of different lengths n_B.

Supposing the problem is in the dense formulation, the input blocking can be done by introducing a transformation matrix $\mathbb{G}$, which transforms the original vector $\boldsymbol{u}$ of n_p input vectors into a smaller vector $\boldsymbol{v}$ of n_c vectors. This matrix $\mathbb{G}$ consists of zeros and ones with exactly one nonzero element in each row, also the causality for the inputs must be preserved, meaning that the order of the inputs must be preserved. The box constraints remain untouched. More can be found in [23], [22] or [24]. An example of such transformation for four input blocks $n_B = [1, 2, 2, 1]$ follows,

$$\begin{bmatrix} \boldsymbol{u}_0 \\ \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \\ \boldsymbol{u}_3 \\ \boldsymbol{u}_4 \\ \boldsymbol{u}_5 \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} & & & \\ & \boldsymbol{I} & & \\ & \boldsymbol{I} & & \\ & & \boldsymbol{I} & \\ & & \boldsymbol{I} & \\ & & & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_0 \\ \boldsymbol{v}_1 \\ \boldsymbol{v}_2 \\ \boldsymbol{v}_3 \end{bmatrix}.$$

This example is demonstrated in Figure 3.

The black dots represent the non-blocked inputs, the white dots represent blocked inputs and the red lines represent bounds on the inputs.

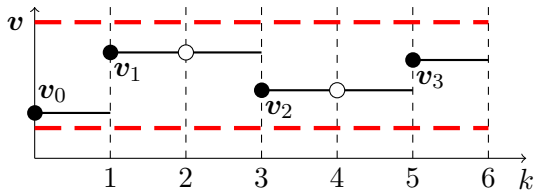


Figure 3 Move Blocking

This move blocking strategy is commonly used for MPC problems in dense formulation, but to our knowledge, there is no extension of this method for problems in the sparse form.

3 Quadratic Programming

3.1 Problem Description

Quadratic programming (QP) is a form of a mathematical optimization problem. The goal is to minimize quadratic cost function, subject to given linear constraints. A description of this type of optimization can be found in [25]. Regardless of the form, linear MPC with quadratic cost leads to a constrained QP optimization problem. The framework of QP optimization is well understood, and there is a variety of algorithms and solvers dealing with these kinds of problems, for example *Quadprog* [9] or *FORCES* [16].

The problem of QP can be written in general way as

$$\begin{aligned} \min_{\boldsymbol{z}} \quad & \frac{1}{2}\boldsymbol{z}^T\boldsymbol{H}\boldsymbol{z} + \boldsymbol{f}^T\boldsymbol{z} \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{z} \leq \boldsymbol{b}, \end{aligned} \tag{9}$$

where $\boldsymbol{H} > 0, \boldsymbol{H} \in \mathcal{R}^{n_z \times n_z}$ is the Hessian, vector $\boldsymbol{f} \in \mathcal{R}^{n_z}$ is the linear part of the problem, $\boldsymbol{z} \in \mathcal{R}^{n_z}$ is the optimization variable. Matrix $\boldsymbol{A} \in \mathcal{R}^{m \times n_z}$ has full row rank, $\boldsymbol{b} \in \mathcal{R}^m$ is vector of constraints, m is number of constraints.

Gradient of (9) $\boldsymbol{g}(\boldsymbol{z})$, is also of importance. It is defined as the first derivative of the criterion, i.e.

$$\boldsymbol{g}(\boldsymbol{z}) = \boldsymbol{H}\boldsymbol{z} + \boldsymbol{f}.$$

3.2 Algorithms

In this Section, several of the most common algorithms for solving the QP problem (9) are briefly described. They are iterative algorithms, that transfer the problem with inequality constraints into a series of linear equality constrained problems.

3.2.1 Interior Point Method

The Interior Point method (IP) was introduced by N. Karmarkar in 1984 [26]. His contribution was an algorithm for solving a linear programming problem with linear equality constraints. In the 90s, the Primal-Dual Interior Point method was introduced [27]. This invention led to a considerable improvement of the performance. More about history of IP methods in [28]. A detailed description of the method can be found for example in [25].

The IP method uses the *logarithmic barrier* to transform the problem. Logarithmic barrier for the QP problem from (9) has the following form,

$$\phi(\boldsymbol{z}) = -\sum_{i=1}^{m} \log(-\boldsymbol{A}\boldsymbol{z} + \boldsymbol{b}).$$

Logarithmic barrier serves as an approximation of the inequality constraints and allows to add them directly into the objective of (9). The problem solved in each iteration is then,

$$\min_{\boldsymbol{z}} \quad \kappa \left(\frac{1}{2} \boldsymbol{z}^T \boldsymbol{H} \boldsymbol{z} + \boldsymbol{c}^T \boldsymbol{z} \right) + \phi(\boldsymbol{z}). \tag{10}$$

The parameter κ is a scalar setting the weight of the barrier against the original criterion. It starts with a small value and is gradually increased in each iteration. The algorithm starts with a sub-optimal solution, that is strongly influenced by the logarithmic barrier. As the influence of the barrier is reduced, the solution in every iteration is getting closer to the true solution of the original problem. This progress is illustrated in the Figure 4. The red line is the progress of the IP method and the black lines represent the constraints of the problem.

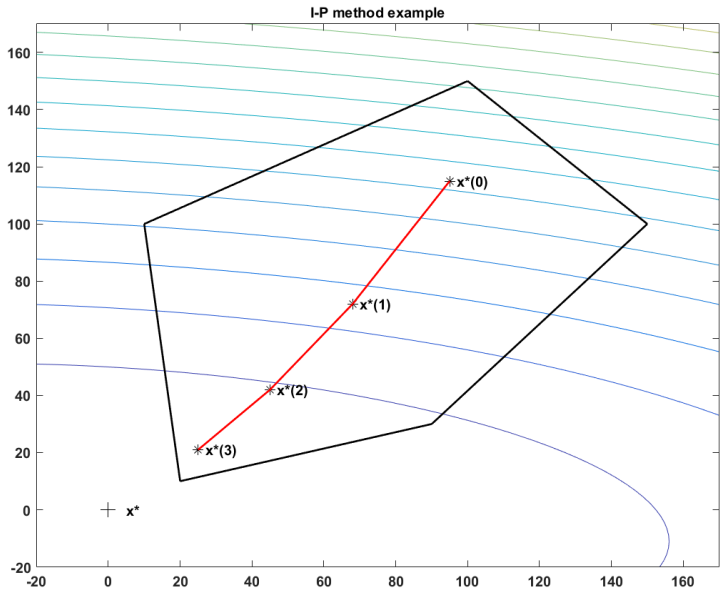


Figure 4 Progress of the IP method

An important condition on proper functionality is that the algorithm starts in a feasible point. Thanks to the logarithmic barrier and because the constrained area is convex; the solution remains feasible in every iteration. The algorithm ends, when the desired accuracy of the solution is achieved.

Solvers using Interior Point method are for example *fastMPC* [15] or *FORCES* [16]. A basic algorithm of Barrier Interior Point method, taken from [25] follows.

Algorithm 1 Interior Point Method, [25]

1: $\boldsymbol{z} := \boldsymbol{z}_0$ strictly feasible, $\mu > 1$, $\kappa := \kappa_0 > 0$, tolerance $\epsilon > 0$
2: **while** $m/\kappa < \epsilon$ **do**
3: $\quad \boldsymbol{z}^* :=$ solution of (10);
4: $\quad \boldsymbol{z} := \boldsymbol{z}^*$;
5: $\quad \kappa := \kappa\mu$;
6: **end while**

3.2.2 Active Set Method

The Active Set method (AS) presents another approach to solving the QP. It is also an iterative optimization method that transforms the problem with inequality constraints into a series of problems with equality constraints. In each iteration, the so-called *working set*, a set of constraints active in the current iteration is changed. A constraint is either added or removed from this set.

First of all, the step in the optimized variable is introduced as

$$\boldsymbol{z} = \boldsymbol{z}^{-} + \Delta\boldsymbol{z},$$

where $\boldsymbol{z}^{-}$ is the value from the last iteration and $\Delta\boldsymbol{z}$ is the current step.
In each iteration, the optimization problem

$$\begin{aligned} \min_{\Delta\boldsymbol{z}} & \frac{1}{2}\Delta\boldsymbol{z}^T\boldsymbol{H}\Delta\boldsymbol{z} + \Delta\boldsymbol{z}^T\boldsymbol{g}(\boldsymbol{z}^{-}) \\ \text{s.t.} \quad & \boldsymbol{A}^j\Delta\boldsymbol{z} = 0, \end{aligned} \tag{11}$$

where $\boldsymbol{A}^j$ is the matrix of active constraints in the j-th iteration, is solved.
In the case $\Delta\boldsymbol{z} = 0$, it is possible that the algorithm found the solution. If they are not satisfied a constraint needs to be removed from the working set, and the algorithm continues. Otherwise, the solution is found.
If $\Delta\boldsymbol{z} \neq 0$, the step needs to be applied. To ensure that the algorithm does not leave the feasible set, the maximal length of such step in i-th component, α_i has to be computed. It expresses the distance from the constraints along the computed direction. This parameter is computed using following relation,

$$\alpha_i \leq \frac{b_i - \boldsymbol{A}_i^j\boldsymbol{z}^{-}}{\boldsymbol{A}_i^j\Delta\boldsymbol{z}}.$$

In case of simple box constraints, which is the case of this work, this relation can be simplified as,

$$\alpha_i \leq \frac{b_i - z_i^{-}}{\Delta z_i}, \tag{12}$$

where b_i is the i-th coordinate of the $\boldsymbol{b}$ vector and $\boldsymbol{A}_i^j$ is the i-th row of matrix $\boldsymbol{A}^j$. The parameter α is chosen as the largest number, satisfying the inequality for each i. If $\alpha < 1$, the step cannot be applied fully and a blocking constraint for which the α_i is the smallest has to be added to the working set. If $\alpha \geq 1$, no constraint is added, and the step is applied fully with $\alpha = 1$.

If $\alpha = 1$ and the active set is correct, the result is the optimum $\boldsymbol{z}^*$.

When the final optimal solution is found, also the active set consisting exactly of the blocking constraints is found. An example of the progress of Active Set method is depicted in Figure 5. The red line symbolises the progress of the method and the black line represents the bounds on the problem.
An extensive description of this method can be found in the literature (e.g. [29]).
A general algorithm is summarized in Algorithm 2.

Algorithm 2 Active Set Method, [29]

$\boldsymbol{z}_0$ strictly feasible;
$\mathcal{A}^0$ initial active set at $\boldsymbol{z}_0$;
for $j < j_{max}$ **do**
  $\Delta \boldsymbol{z} :=$ solution of (11);
  if $\Delta \boldsymbol{z} = 0$ **then**
    if KKT conditions are satisfied **then**
      $\boldsymbol{z}^* := \boldsymbol{z}^-$;
      break;
    else
      $\boldsymbol{z}^{j+1} := \boldsymbol{z}^-$;
      Remove a constraint from $\mathcal{A}^j$;
    end if
  else
    Compute α_i from (12);
    $\boldsymbol{z}^- := \boldsymbol{z}^- + \alpha_i \Delta \boldsymbol{z}$;
    if $\alpha_i < 1$ **then**
      Add a constraint to $\mathcal{A}^j$;
    else
      $\mathcal{A}^{j+1} := \mathcal{A}^j$;
    end if
  end if
end for

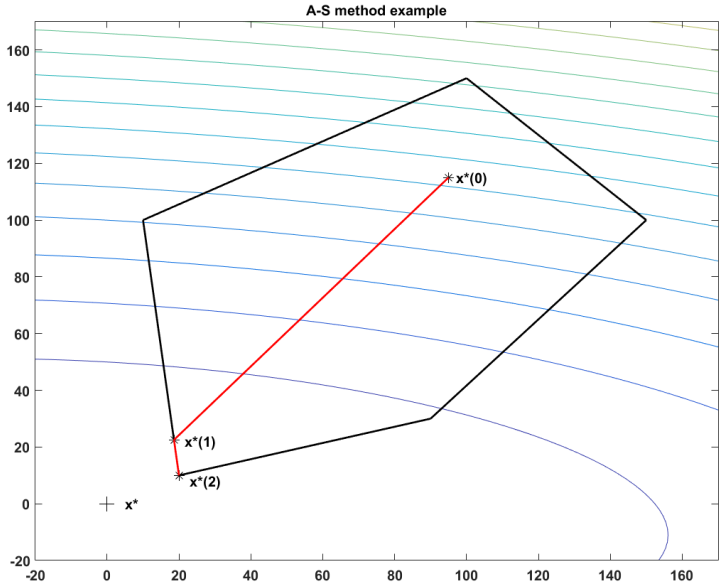


Figure 5 Progress of the Active Set method

3.2.3 Alternating Direction Method of Multipliers

The Alternating Direction Method of Multipliers (ADMM) is based on the Dual Ascent method but combines it with the method of the Augmented Lagrangian, which brings

robustness to the algorithm. The main advantage of this algorithm is the possibility to decompose the problem into smaller ones, which can be solved in parallel as in [18].

Dual Ascent method

Considering a problem in the even more general form

$$\begin{aligned} \min_{\boldsymbol{z}} \quad & f(\boldsymbol{z}) \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{z} = \boldsymbol{b}, \end{aligned}$$

the constraints are integrated to the objective using Lagrange multipliers,

$$\mathcal{L}(\boldsymbol{z}, \boldsymbol{\lambda}) = f(\boldsymbol{z}) + \boldsymbol{\lambda}^T(\boldsymbol{A}\boldsymbol{z} - \boldsymbol{b}).$$

The solution is then found in the dual variable using the Gradient method to maximize the dual function $h(\boldsymbol{\lambda})$,

$$\max_{\boldsymbol{\lambda}} h(\boldsymbol{\lambda}),$$

where

$$h(\boldsymbol{\lambda}) = \inf_{\boldsymbol{z}} \mathcal{L}(\boldsymbol{z}, \boldsymbol{\lambda}).$$

The algorithm solves the problem by alternating two steps. In the first step, $\boldsymbol{z}$ is minimized. In the second step, $\boldsymbol{\lambda}$ is updated.

$$\begin{aligned} \boldsymbol{z}^{j+1} &= \text{argmin}_{\boldsymbol{z}} \mathcal{L}(\boldsymbol{z}, \boldsymbol{\lambda}^j) \\ \boldsymbol{\lambda}^{j+1} &= \boldsymbol{\lambda}^j + \alpha(\boldsymbol{A}\boldsymbol{z}^{j+1} - \boldsymbol{b}) \end{aligned} \tag{14}$$

Method of Multipliers

The Dual Ascent method converges only under strong assumptions. To make the Dual Ascent method more robust, Augmented Lagrangian,

$$\mathcal{L}_\rho(\boldsymbol{z}, \boldsymbol{\lambda}) = f(\boldsymbol{z}) + \boldsymbol{\lambda}^T(\boldsymbol{A}\boldsymbol{z} - \boldsymbol{b}) + \frac{\rho}{2}||\boldsymbol{A}\boldsymbol{z} - \boldsymbol{b}||_2^2 \tag{15}$$

is introduced.
The Augmented Lagrangian (15) is then solved similarly as in (14), but instead of general α, penalty parameter ρ is used.

$$\begin{aligned} \boldsymbol{z}^{j+1} &= \text{argmin}_{\boldsymbol{z}} L_\rho(\boldsymbol{z}, \boldsymbol{\lambda}^j) \\ \boldsymbol{\lambda}^{j+1} &= \boldsymbol{\lambda}^j + \rho(\boldsymbol{A}\boldsymbol{z}^{j+1} - \boldsymbol{b}) \end{aligned}$$

Alternating Direction Method of Multipliers

Suppose the following problem in variables $\boldsymbol{z}, \boldsymbol{v}$ with separable objective functions $f(\boldsymbol{z})$ and $l(\boldsymbol{v})$.

$$\begin{aligned} &\min_{\boldsymbol{z},\boldsymbol{v}} f(\boldsymbol{z}) + l(\boldsymbol{v}) \\ &\text{s.t.} \quad \boldsymbol{A}\boldsymbol{z} + \boldsymbol{B}\boldsymbol{v} = \boldsymbol{c} \end{aligned}$$

Augmented Lagrangian is in this case,

$$L_\rho(\boldsymbol{z}, \boldsymbol{\lambda}, \boldsymbol{v}) = f(\boldsymbol{z}) + l(\boldsymbol{v}) + \boldsymbol{\lambda}^T(\boldsymbol{A}\boldsymbol{z} + \boldsymbol{B}\boldsymbol{v} - \boldsymbol{c}) + \frac{\rho}{2}||\boldsymbol{A}\boldsymbol{z} + \boldsymbol{B}\boldsymbol{v} - \boldsymbol{c}||_2^2.$$

Thanks to the separable objectives, the computations can be done in parallel. Also this formulation can be used to solve QP. The complete algorithm of ADMM follows.

Algorithm 3 ADMM Method

1: $\boldsymbol{z} := \boldsymbol{z}_0$, $\boldsymbol{\lambda} := \boldsymbol{\lambda}_0$, $\boldsymbol{v} := \boldsymbol{v}_0$ feasible
2: **for** $j < j_{max}$ **do**
3: $\quad \boldsymbol{z}^{j+1} = \text{argmin}_{\boldsymbol{z}} L_\rho(\boldsymbol{z}, \boldsymbol{\lambda}^j, \boldsymbol{v}^j)$;
4: $\quad \boldsymbol{v}^{j+1} = \text{argmin}_{\boldsymbol{v}} L_\rho(\boldsymbol{z}^{j+1}, \boldsymbol{\lambda}^j, \boldsymbol{v})$;
5: $\quad \boldsymbol{\lambda}^{j+1} = \boldsymbol{\lambda}^j + \rho(\boldsymbol{A}\boldsymbol{z}^{j+1} + \boldsymbol{B}\boldsymbol{v}^{j+1} - \boldsymbol{c})$;
6: **end for**

3.2.4 Gradient Projection Method

The Gradient Projection method can be senn as the Active Set method, but it addresses the problem of slow active set change rate. The basic Active Set method allows to add or drop at most one constraint from the working set in each iteration. In case, that the final active set is dramatically different from the starting set (especially in the case of large problems), the number of iterations can be high.

The Gradient Projection algorithm allows large changes in the active set in each iteration, using the *projection step*,

$$\mathcal{P}(\boldsymbol{z} - \alpha \boldsymbol{g}(\boldsymbol{z})), \alpha > 0, \tag{18}$$

where parameter α sets the size of the step.

The step (18) projects the new solution onto the feasible set, resulting in feasible solution and also in the new working set because the active constraints are identified in the process. The parameter α must be chosen such that the algorithm converges. In case of QP problem

$$\alpha < \frac{1}{L},$$

where L is the largest eigenvalue of $\boldsymbol{H}$. In case of simple bound constraints, the projection is found fairly easily as,

$$\mathcal{P}(\boldsymbol{z}) = \text{median}(\underline{\boldsymbol{z}}, \boldsymbol{z}, \overline{\boldsymbol{z}}),$$

where median is taken element-wise. Symbols $\underline{\boldsymbol{z}}, \overline{\boldsymbol{z}}$ are lower and upper bounds on $\boldsymbol{z}$. Basic Gradient Projection method algorithm follows.

Algorithm 4 Gradient Projection Method

1: $\boldsymbol{z} := \boldsymbol{z}_0$ feasible, $0 < \alpha < \alpha_{max}$
2: **for** $j < j_{max}$ **do**
3: $\quad \boldsymbol{z}^+ = \boldsymbol{z}^j - \alpha \boldsymbol{g}(\boldsymbol{z}^j)$;
4: $\quad \boldsymbol{z}^{j+1} = \text{median}(\underline{\boldsymbol{z}}, \boldsymbol{z}^+, \overline{\boldsymbol{z}})$;
5: **end for**

3.3 Warm-start

Warm-start is a technique aimed to reduce the computational load of a QP solver. It tries to find a good initial estimate of the solution, to reduce the number of iterations. Warm-start uses the optimal sequence from previous step or additional information from current step to obtain such estimate. Warm-start is often used in case of AS methods and Gradient methods.

The main problems of the warm-start are the uncertainty in the system model and unknown disturbances entering the system. Those issues have to be taken into account, and several strategies for dealing with them have been developed [30].

Consider the QP problem from (9). The solution of this problem in time instant t is denoted as $\boldsymbol{z}^*(t) = [\boldsymbol{z}_1^{*T}(t), \boldsymbol{z}_2^{*T}(t), ..., \boldsymbol{z}_N^{*T}(t)]^T$. Similarly, optimal solution from the previous step is $\boldsymbol{z}^*(t-1)$.

Two approaches are presented here; they can also be combined.

3.3.1 Warm-start using Previous Solution

The easiest method of warm-start (used e.g. in [17]) is shifting the sequence $\boldsymbol{z}^*(t-1)$ by one element and duplicating the last term, to obtain a new sequence of the same length. Then the initial element is $\boldsymbol{z}^0(t) = [\boldsymbol{z}_2^{*T}(t-1), \boldsymbol{z}_3^{*T}(t-1)..., \boldsymbol{z}_N^{*T}(t-1), \boldsymbol{z}_N^{*T}(t-1)]^T$.

Such method is computationally inexpensive, but it also does not exploit any further information except for the optimal sequence from the previous step. Therefore, it is susceptible to the disturbances and errors in the model. On the other hand, in case there are no disturbances present and the model is precise, the solution is near optimal. The difference from optimal solution in such case occurs only in the last term, [30].

3.3.2 Warm-start using New Measurement

In the case of the MPC, another option can be realized. The idea is to use the information about the current state to improve the initial guess via the LQ regulator. Such method does not apply to general QP because the LQ regulator is specific to linear systems and quadratic criterion.

The LQ regulator computes optimal solution for the unconstrained problem, which serves as an estimate of the constrained problem. Of course, the feasibility of this estimate has to be ensured by projecting the solution onto the constraints (as proposed in [30]).

Firstly, it is needed to extract the input part $\boldsymbol{u}^*(t)$ and the state part $\boldsymbol{x}^*(t)$ from the optimized vector $\boldsymbol{z}^*(t)$ (in case of condensed form $\boldsymbol{z}^*(t) = \boldsymbol{u}^*(t)$).

Recall the form of the discrete-time system.

$$\boldsymbol{x}(t+1) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t), \quad t > 0.$$

For the criterion (1), the solution of the constrained LQR is following,

$$\begin{aligned} \boldsymbol{K} &= (\boldsymbol{R} + \boldsymbol{B}^T \boldsymbol{Q}_{n_p} \boldsymbol{B})^{-1} (\boldsymbol{S}^T + \boldsymbol{B}^T \boldsymbol{Q}_{n_p}) \boldsymbol{A} \\ \boldsymbol{u}^*(t) &= -\mathcal{P}(\boldsymbol{K}\boldsymbol{x}(t)), \end{aligned}$$

where $\boldsymbol{K}$ is the Kalman gain of the regulator and $\mathcal{P}$ symbolises projection of the inputs onto the constraints. In case of simple box constraints on inputs, the projection can be done easily,

$$\boldsymbol{u}^* = \text{median}(\underline{\boldsymbol{u}}, -\boldsymbol{K}\boldsymbol{x}(t), \overline{\boldsymbol{u}}),$$

where the median is taken element-wise.

The LQR, in this case, is used iteratively, to find the control on full control horizon.

4 Tailored Algorithm for MPC with Long Prediction Horizon

4.1 Newton Projection with Proportioning

The Newton Projection with Proportioning (NPP) algorithm, presented in [7] represents a new iterative method, based on the Active Set algorithm. The results, from [7], are promising, although there is a major drawback of this method - it can solve only optimization problems with no equality constraints. From the view of the MPC, this means only problems in the dense form.

The main contribution of this work is the modification of the NPP algorithm for MPC with the long prediction horizon. The idea is to exploit the structure of the sparse problem, similarly as in [25]. Such approach leads to a smaller computational complexity fo the algorithm, it becomes linear with respect to the length of the prediction horizon, while in the original dense form the complexity is cubic. Several basic ingredients of the original NPP algorithm had to be modified to achieve that.

Algorithm 5 Newton Projection with Proportioning, [7]

$\boldsymbol{z} := \boldsymbol{z}_0$ feasible, $\boldsymbol{g}(\boldsymbol{z}_0) = \boldsymbol{H}\boldsymbol{z}_0 + \boldsymbol{f}$, $\Gamma > 0$;
while $||\boldsymbol{\mu}(\boldsymbol{z}^-)|| \leq \epsilon$ **do**
  if $||\boldsymbol{\beta}(\boldsymbol{z}^-)|| \geq \Gamma||\boldsymbol{\phi}(\boldsymbol{z}^-)||$ **then**
    // Proportional $\boldsymbol{z}^-$
    Obtain working set;
  else
    // Non-Proportional $\boldsymbol{z}^-$
    Obtain working set only for free gradient;
  end if
  $\Delta\boldsymbol{z}$:= Solution of Newton step;
  $\alpha_f := \max\{\alpha : \boldsymbol{z}^- + \alpha\Delta\boldsymbol{z}, \text{s.t.}(\underline{\boldsymbol{z}} < \boldsymbol{z} < \overline{\boldsymbol{z}})\}$
  if $\alpha_f < 1$ **then**
    //Expansion step
    $[\boldsymbol{z}^-, \boldsymbol{g}(\boldsymbol{z}^-)]$:= Solution of PLS;
  else
    $\boldsymbol{z}^- := \boldsymbol{z}^- + \Delta\boldsymbol{z}$;
    $\boldsymbol{g}(\boldsymbol{z}^-) := \boldsymbol{g}(\boldsymbol{z}^-) + \boldsymbol{H}\Delta\boldsymbol{z}$;
  end if
end while
$\boldsymbol{z}^* := \boldsymbol{z}^-$;

Proportioning

Proportioning step is used in the framework of the NPP algorithm to decide whether to remove or add the constraints from the working set.

First, consider the gradient $\boldsymbol{g}(\boldsymbol{z}) = \boldsymbol{H}\boldsymbol{z}^{-} + \boldsymbol{f}$. Next, consider working set $\mathcal{A}$, set of currently active constraints.
Then *chopped gradient* $\boldsymbol{\beta}(\boldsymbol{z})$ and *free gradient* $\boldsymbol{\phi}(\boldsymbol{z})$ concepts are defined as,

$$\begin{aligned}\boldsymbol{\phi}_i(\boldsymbol{z}) &= 0, \text{for } i \in \mathcal{A}, \text{ else } \boldsymbol{\phi}_i(\boldsymbol{z}) = \boldsymbol{g}_i(\boldsymbol{z}) \\ \boldsymbol{\beta}_i(\boldsymbol{z}) &= \boldsymbol{g}_i(\boldsymbol{z})^{\#}, \text{for } i \in \mathcal{A}, \text{ else } \boldsymbol{\beta}_i(\boldsymbol{z}) = 0,\end{aligned}$$

where

$$\boldsymbol{g}_i(\boldsymbol{z})^{\#} = \begin{cases}\max\{\boldsymbol{g}_i(\boldsymbol{z}), 0\}, & \text{if } \boldsymbol{z}_i = \overline{\boldsymbol{z}}_i \\ \min\{\boldsymbol{g}_i(\boldsymbol{z}), 0\}, & \text{if } \boldsymbol{z}_i = \underline{\boldsymbol{z}}_i\end{cases}.$$

Together, they form the *projected gradient* $\boldsymbol{\mu}(\boldsymbol{z}) = \boldsymbol{\phi}(\boldsymbol{z}) + \boldsymbol{\beta}(\boldsymbol{z})$. The solution $\boldsymbol{z}$ satisfies the KKT conditions if and only if $\boldsymbol{\mu}(\boldsymbol{z}^*) = 0$.

By comparing the norm of $\boldsymbol{\beta}$ and $\boldsymbol{\phi}$, one can decide whether the step is *Proportional* or *Non-Proportional*.
In case of the Proportional step,

$$\mathcal{A} = \{i : \{\boldsymbol{z}_i = \overline{\boldsymbol{z}}_i\} \text{ or } \{\boldsymbol{z}_i = \underline{\boldsymbol{z}}_i\}\}.$$

In case of the Non-Proportional step,

$$\mathcal{A} = \{i : (\{\boldsymbol{z}_i = \overline{\boldsymbol{z}}_i\} \text{ or } \{\boldsymbol{z}_i = \underline{\boldsymbol{z}}_i\}) \text{ and } \boldsymbol{\beta}_i(\boldsymbol{z}) = 0\}.$$

The Proportional step is taken, if

$$||\boldsymbol{\beta}(\boldsymbol{z})|| \geq \Gamma ||\boldsymbol{\phi}(\boldsymbol{z})||,$$

otherwise, the step is Non-Proportional.

The parameter Γ sets the weight between the free and the chopped gradient. The bigger is Γ, the more weight is put on the free gradient $\boldsymbol{\phi}$, therefore preferring the Non-Proportional step and vice versa. The default value of Γ is 1.

Face problem

Face problem, as defined in [7] is a name for the reduced problem, which is to be solved in every iteration of the Active Set method.

The restricted problem can be written as,

$$\begin{aligned}&\min_{\boldsymbol{z}} \frac{1}{2}\boldsymbol{z}^T\boldsymbol{H}\boldsymbol{z} + \boldsymbol{f}^T\boldsymbol{z}, \\ &\text{s.t. } \boldsymbol{I}_A\boldsymbol{z} = \boldsymbol{e},\end{aligned} \tag{20}$$

where $\boldsymbol{I}_A$ is matrix selecting variables, whose bounds are active. Vector $\boldsymbol{e} = [\overline{\boldsymbol{z}}_A^T, \underline{\boldsymbol{z}}_A^T]^T$ containts the active bounds $\overline{\boldsymbol{z}}_A^T$ are upper active bounds, $\underline{\boldsymbol{z}}_A^T$ are lower active bounds.

By introducing the Lagrange multipliers, denoted by $\boldsymbol{\lambda}$, on (20) and then using the KKT conditions of optimality, one arrives to following matrix equation,

$$\begin{bmatrix}\boldsymbol{H} & -\boldsymbol{I}_A^T \\ \boldsymbol{I}_A & 0\end{bmatrix}\begin{bmatrix}\boldsymbol{z}^* \\ \boldsymbol{\lambda}^*\end{bmatrix} = \begin{bmatrix}-\boldsymbol{f} \\ \boldsymbol{e}\end{bmatrix}$$

Next, a step in variable $\boldsymbol{z}$ is introduced,

$$\boldsymbol{z}^* = \boldsymbol{z}^- + \Delta\boldsymbol{z}, \tag{21}$$

where $\boldsymbol{z}^-$ denotes the value of the $\boldsymbol{z}^*$ and $\Delta\boldsymbol{z}$, denotes the step in the $\boldsymbol{z}^*$.

Using the step from (21), the KKT equation is rewritten as,

$$\begin{bmatrix} \boldsymbol{H} & \boldsymbol{I}_A^T \\ \boldsymbol{I}_A & 0 \end{bmatrix} \begin{bmatrix} -\Delta\boldsymbol{z} \\ \boldsymbol{\lambda}^* \end{bmatrix} = \begin{bmatrix} \boldsymbol{g} \\ 0 \end{bmatrix}, \tag{22}$$

where $\boldsymbol{g} = \boldsymbol{H}\boldsymbol{z}^- + \boldsymbol{f}$.

To solve the equation (22), the Null-Space method is used, following the argument that in the case of many active constraints, it is more advantageous than the Range-Space method. This motivation comes from automotive, where many constraints are usual.

In the Null-Space method, the step $\Delta\boldsymbol{z}$ is divided into two parts,

$$\Delta\boldsymbol{z} = \boldsymbol{Z}\Delta\boldsymbol{z}_z + \boldsymbol{Y}\Delta\boldsymbol{z}_y. \tag{23}$$

First part $\boldsymbol{Z}\Delta\boldsymbol{z}_z$ corresponds to the step in the null space of the matrix $\boldsymbol{I}_A$, therefore

$$\boldsymbol{I}_A\boldsymbol{Z} = 0, \tag{24}$$

where $\boldsymbol{Z} \in \mathcal{R}^{n_z \times m}$. The second part corresponds to the range space of the $\boldsymbol{I}_A$, as the columns of $\boldsymbol{Y}$ form the basis of range space of $\boldsymbol{I}_A^T$, also

$$\text{rank}[\boldsymbol{Y}|\boldsymbol{Z}] = n_z. \tag{25}$$

From the previous properties, $\boldsymbol{Y}\Delta\boldsymbol{z}_Y = 0$ can be assumed, symplifying the equation (23).

Substituting the (23) into (22) and using the properties (24) and (25), the following solution is obtained,

$$\mathbb{G}\Delta\boldsymbol{z}_z = -\boldsymbol{r},$$

where $\mathbb{G} = \boldsymbol{Z}^T\boldsymbol{H}\boldsymbol{Z}$ represents reduced Hessian and $\boldsymbol{r} = \boldsymbol{Z}^T\boldsymbol{g}$ represents reduced gradient.

Thanks to the fact, that the inequality constraints in this thesis are only box constraints, the reduced Hessian and gradient can be obtained easily, by omitting the rows and columns corresponding to the active constraints.

The full step $\Delta\boldsymbol{z}$ is obtained by adding zeros to the variables, whose constraints are active since no move can be performed in such coordinates,

$$\Delta\boldsymbol{z} = \boldsymbol{Z}\Delta\boldsymbol{z}_z.$$

Projected Line Search

The Projected Line Search (PLS) is used to rapidly extend the working set, it is described in [29]. The goal is to find a piecewise linear path, obtained by projecting the descent direction onto the box constraints and to find the first local minimizer of the cost criterion,

$$\frac{1}{2}(\boldsymbol{z}+\alpha\Delta\boldsymbol{z})^T\boldsymbol{H}(\boldsymbol{z}+\alpha\Delta\boldsymbol{z})+\boldsymbol{f}^T(\boldsymbol{z}+\alpha\Delta\boldsymbol{z}). \tag{26}$$

Such point is found by examining all the line segments, defined by the constraints. All breakpoints α must be found to determine those segments. Those breakpoints can be found using

$$\alpha_i=\begin{cases}(\boldsymbol{z}_i-\overline{\boldsymbol{z}}_i)/\Delta\boldsymbol{z}_i & \text{if } \Delta\boldsymbol{z}_i<0 \text{ and } \overline{\boldsymbol{z}}_i<+\infty,\\ (\boldsymbol{z}_i-\underline{\boldsymbol{z}}_i)/\Delta\boldsymbol{z}_i & \text{if } \Delta\boldsymbol{z}_i>0 \text{ and } \underline{\boldsymbol{z}}_i>-\infty,\\ \infty & \text{otherwise}\end{cases}.$$

and the components of the projected variable are

$$\boldsymbol{z}_i(\alpha)=\begin{cases}\boldsymbol{z}_i+\alpha\Delta\boldsymbol{z}_i & \text{if } \alpha<\alpha_i,\\ \boldsymbol{z}_i+\alpha_i\Delta\boldsymbol{z}_i & \text{otherwise.}\end{cases}$$

Firstly, the duplicate values are removed from the set of $\alpha_1,\alpha_2,...,\alpha_n$ and the set is sorted, obtaining set $\beta^1,\beta^2,...,\beta^l$, where $\beta^1<\beta^2<...<\beta^l$.
Then, the intervals $[0,\beta^1],[\beta^1,\beta^2],...$ are examined. Once a local minimizer on an interval is found, the algorithm ends.
In case of interval $[\beta^{j-1},\beta^j]$, the line segment is found as,

$$\boldsymbol{z}(\alpha)=\boldsymbol{z}(\beta^{j-1})+\Delta\alpha\boldsymbol{p}^{j-1},$$

where

$$\Delta\alpha=\alpha-\beta^{j-1}\in[0,\beta^j-\beta^{j-1}],$$

and

$$p_i^{j-1}=\begin{cases}\Delta\boldsymbol{z}_i & \text{if } \beta^{j-1}\le\alpha,\\ 0 & \text{otherwise.}\end{cases}$$

The quadratic cost criterion (26) on the line segment $[\boldsymbol{z}(\beta^{j-1}),\boldsymbol{z}(\beta^j)]$ is following,

$$q(\boldsymbol{z}(\alpha))=\frac{1}{2}(\boldsymbol{z}(\beta^{j-1})+\Delta\alpha\boldsymbol{p}^{j-1})^T\boldsymbol{H}(\boldsymbol{z}(\beta^{j-1})+\Delta\alpha\boldsymbol{p}^{j-1})+\boldsymbol{f}^T(\boldsymbol{z}(\beta^{j-1})+\Delta\alpha\boldsymbol{p}^{j-1}).$$

Differentiating $q(\boldsymbol{z}(\alpha))$ with respect to $\Delta\alpha$ yields

$$q(\boldsymbol{z}(\alpha))'=\boldsymbol{z}(\beta^{j-1})^T\boldsymbol{H}\boldsymbol{p}^{j-1}+\boldsymbol{f}^T\boldsymbol{p}^{j-1}+\Delta\alpha\boldsymbol{p}^{j-1^T}\boldsymbol{H}\boldsymbol{p}^{j-1}. \tag{27}$$

Setting the (27) equal to zero the following result is obtained,

$$\Delta\alpha = -\frac{\boldsymbol{z}(\beta^{j-1})^T\boldsymbol{H}\boldsymbol{p}^{j-1} + \boldsymbol{f}^T\boldsymbol{p}^{j-1}}{\boldsymbol{p}^{j-1^T}\boldsymbol{H}\boldsymbol{p}^{j-1}},$$

Three cases can occur.

1. If $\boldsymbol{z}(\beta^{j-1})^T\boldsymbol{H}\boldsymbol{p}^{j-1} + \boldsymbol{f}^T\boldsymbol{p}^{j-1} \leq 0$, the local minimizer is found as $\alpha^* = \beta^{j-1}$.
2. Else, if $\Delta\alpha^* \in [0, \beta^j - \beta^{j-1}]$, there is a minimizer at $\alpha^* = \beta^{j-1} + \Delta\alpha^*$.
3. Else, move onto next interval and continue the search. The search direction $\boldsymbol{p}$ has to be updated, but it differs only in one component, so the computational load is not high.

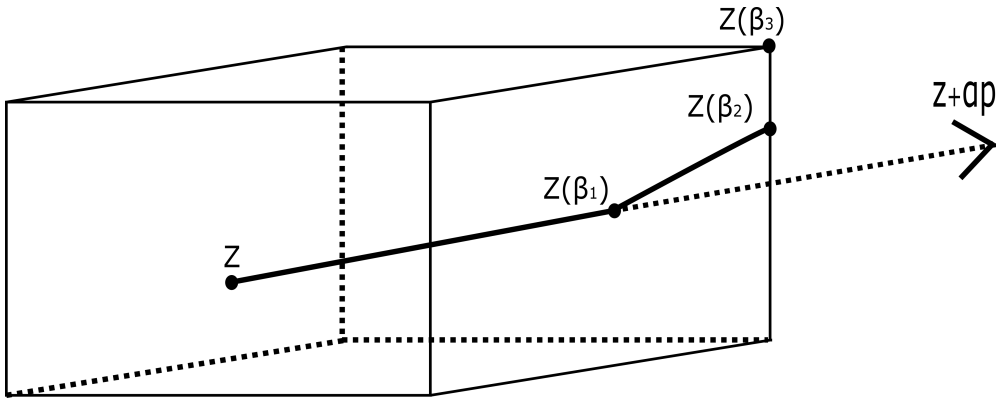


Figure 6 Example of Projected Line Search

An example of the PLS progress is shown in the Figure 6. The algorithm starts in point $\boldsymbol{z}$ and with each iteration, it encounters a constraint and moves in a modified direction.

4.2 Tailored NPP Algorithm

Within this thesis, the NPP algorithm was modified to compute the MPC problem in the sparse form and to be able to exploit the structure of the problem. Namely, the Proportioning step, Face problem solution, and Newton Projection step had to be changed. Description of the main differences and comparison of the computational complexity of the algorithms follows.

4.2.1 Sparse Proportioning

The idea is the same as in the original form, only the equivalence of the sparse gradient and the dense gradient has to be established. The equivalence can be shown by examining the gradients in both forms. Recall the gradient in the dense form, used in the original NPP algorithm

$$\widehat{\boldsymbol{g}} = \widehat{\boldsymbol{H}}\boldsymbol{u} + \widehat{\boldsymbol{f}},$$

where

$$\widehat{\boldsymbol{H}} = \mathbb{B}^T\mathbb{A}^{-T}\mathbb{Q}\mathbb{A}^{-1}\mathbb{B} + \mathbb{R} - \mathbb{S}\mathbb{A}^{-1}\mathbb{B} - \mathbb{B}^T\mathbb{A}^{-T}\mathbb{S}^T$$
$$\widehat{\boldsymbol{f}} = \mathbb{B}^T\mathbb{A}^{-T}\mathbb{Q}\mathbb{A}^{-1}\mathbb{B} - \mathbb{S}^T\mathbb{A}^{-1}\boldsymbol{d},$$

The gradient in the sparse form has following form,

$$\begin{bmatrix} \boldsymbol{g}_u \\ \boldsymbol{g}_x \\ \boldsymbol{g}_\lambda \end{bmatrix} = \begin{bmatrix} \mathbb{R}\boldsymbol{u} + \mathbb{S}^T\boldsymbol{x} + \mathbb{B}^T\boldsymbol{\lambda} \\ \mathbb{S}\boldsymbol{u} + \mathbb{Q}\boldsymbol{x} + \mathbb{A}^T\boldsymbol{\lambda} \\ \mathbb{B}\boldsymbol{u} + \mathbb{A}\boldsymbol{x} + \boldsymbol{d} \end{bmatrix}, \tag{28}$$

where $\boldsymbol{d} = [(\boldsymbol{A}\boldsymbol{x}_0)^T, 0, \ldots, 0]^T$. The condition for optimality is $\boldsymbol{g}_x = 0$ and $\boldsymbol{g}_\lambda = 0$, because the variables $\boldsymbol{x}$ and $\boldsymbol{\lambda}$ are unconstrained. Firstly, $\boldsymbol{x}$ is expressed from the equation for $\boldsymbol{g}_\lambda$ from (28),

$$\boldsymbol{x} = -\mathbb{A}^{-1}(\mathbb{B}\boldsymbol{u} + \boldsymbol{d})).$$

Subsequently, it is substituted into equation for $\boldsymbol{g}_x$ and variable $\boldsymbol{\lambda}$ is expressed,

$$\boldsymbol{\lambda} = -\mathbb{A}^{-T}(\mathbb{S}\boldsymbol{u} - \mathbb{Q}\mathbb{A}^{-1}(\mathbb{B}\boldsymbol{u} + \boldsymbol{d})). \tag{29}$$

By substituting the equation (29) into equation for $\boldsymbol{g}_u$, the final result is obtained,

$$\boldsymbol{g}_u = \mathbb{R}\boldsymbol{u} - \mathbb{S}^T\mathbb{A}^{-1}(\mathbb{B}\boldsymbol{u} + \boldsymbol{d}) - \mathbb{B}^T\mathbb{A}^{-T}(\mathbb{S}\boldsymbol{u} - \mathbb{Q}\mathbb{A}^{-1}(\mathbb{B}\boldsymbol{u} + \boldsymbol{d})),$$

after modifications,

$$\boldsymbol{g}_u = (\mathbb{B}^T\mathbb{A}^{-T}\mathbb{Q}\mathbb{A}^{-1}\mathbb{B} + \mathbb{R} - \mathbb{S}^T\mathbb{A}^{-1}\mathbb{B} - \mathbb{B}^T\mathbb{A}^{-T}\mathbb{S})\boldsymbol{u} + (\mathbb{B}^T\mathbb{A}^{-T}\mathbb{Q}\mathbb{A}^{-1} - \mathbb{S}^T\mathbb{A}^{-1})\boldsymbol{d}. \tag{30}$$

The relation (30) corresponds to the gradient in the dense form. The proportioning is then carried out in the same manner as in the case of the original NPP algorithm, using $\boldsymbol{g}_u$, i.e. part of the augmented gradient only.

4.2.2 Sparse Face Problem Solution

The working set of the constraints is projected in the computations by introducing a blocking matrix $\mathbb{F}$, which is determined by the active constraints in the j-th iteration. There are as many rows as is the number of active constraints. The matrix $\mathbb{F}$ selects those coordinates of $\Delta\boldsymbol{u}$, a vector of input step, whose constraints are active.

Matrix $\mathbb{F}$ determines whether a component of $\Delta\boldsymbol{u}$ is fixed or free.

$$\mathbb{F}\Delta\boldsymbol{u} = 0$$

Next, Lagrangian $\mathcal{L}$ with Lagrange multipliers $\boldsymbol{\lambda}$ and $\boldsymbol{\gamma}$ is introduced, to incorporate the constraints,

$$\mathcal{L} = \frac{1}{2}\begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{x} \end{bmatrix}^T \begin{bmatrix} \mathbb{R} & \mathbb{S}^T \\ \mathbb{S} & \mathbb{Q} \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{x} \end{bmatrix} + \boldsymbol{\lambda}^T(\mathbb{A}\boldsymbol{x} + \mathbb{B}\boldsymbol{u} + \boldsymbol{d}) + \boldsymbol{\gamma}^T(\mathbb{F}\Delta\boldsymbol{u}). \tag{31}$$

The full set of optimization variables then consists of: inputs $\boldsymbol{u}$, states $\boldsymbol{x}$, Lagrange multipliers of the system dynamics $\boldsymbol{\lambda}$ and Lagrange multipliers of the inequality constraints $\boldsymbol{\gamma}$.

To compute the Newton step within the NPP algorithm from the formulation presented in (31) a step in variables $\boldsymbol{u}, \boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\gamma}$ is introduced as

$$\begin{aligned} \boldsymbol{u} &= \boldsymbol{u}^- + \Delta\boldsymbol{u} \\ \boldsymbol{x} &= \boldsymbol{x}^- + \Delta\boldsymbol{x} \\ \boldsymbol{\lambda} &= \boldsymbol{\lambda}^- + \Delta\boldsymbol{\lambda} \\ \boldsymbol{\gamma} &= \boldsymbol{\gamma}^- + \Delta\boldsymbol{\gamma}, \end{aligned} \tag{32}$$

where variables $\boldsymbol{u}^-$, $\boldsymbol{x}^-$, $\boldsymbol{\lambda}^-$ and $\boldsymbol{\gamma}^-$ are values from the previous iteration, $\Delta\boldsymbol{u}$, $\Delta\boldsymbol{x}$, $\Delta\boldsymbol{\lambda}$ and $\Delta\boldsymbol{\gamma}$ are steps in the variables.

After applying the relation (32) on the extended criterion (31) and by taking the partial derivatives, the resulting relation is obtained as,

$$\widehat{\mathbb{H}}\boldsymbol{p} = -\boldsymbol{g}, \tag{33}$$

where

$$\widehat{\mathbb{H}} = \begin{bmatrix} \mathbb{R} & \mathbb{S}^T & \mathbb{B}^T & \mathbb{F}^T \\ \mathbb{S} & \mathbb{Q} & \mathbb{A}^T & \\ \mathbb{B} & \mathbb{A} & & \\ \mathbb{F} & & & \end{bmatrix}, \boldsymbol{p} = \begin{bmatrix} \Delta\boldsymbol{u} \\ \Delta\boldsymbol{x} \\ \Delta\boldsymbol{\lambda} \\ \Delta\boldsymbol{\gamma} \end{bmatrix}, \boldsymbol{g} = \begin{bmatrix} \mathbb{R} & \mathbb{S}^T & \mathbb{B}^T & \mathbb{F}^T \\ \mathbb{S} & \mathbb{Q} & \mathbb{A}^T & \\ \mathbb{B} & \mathbb{A} & & \\ \mathbb{F} & & & \end{bmatrix} \begin{bmatrix} \boldsymbol{u}^- \\ \boldsymbol{x}^- \\ \boldsymbol{\lambda}^- \\ \boldsymbol{\gamma}^- \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \boldsymbol{d} \\ 0 \end{bmatrix}.$$

This equation has to be solved for $\boldsymbol{p}$, to find the desired Newton step.

Reduction by Null-Space Method

The Null-space method is used to eliminate the variable $\boldsymbol{\gamma}$, reducing the size of the problem. The matrix $\mathbb{Z}$ is introduced as a null space matrix of the trivial $\mathbb{F}$ matrix. The problem is altered according to the active constraints, using this matrix. Subsequently, the first equation of (33) is multiplied by $\mathbb{Z}^T$. Following equation holds for the matrix $\mathbb{Z}$ and step $\Delta\boldsymbol{u}$.

$$\Delta\boldsymbol{u} = \mathbb{Z}\Delta\boldsymbol{u}_f.$$

The resulting system of equations is following

$$\begin{bmatrix} \boldsymbol{\Psi} & \boldsymbol{\Lambda}^T & \boldsymbol{\Gamma}^T \\ \boldsymbol{\Lambda} & \mathbb{Q} & \mathbb{A}^T \\ \boldsymbol{\Gamma} & \mathbb{A} & \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{u}_f \\ \Delta\boldsymbol{x} \\ \Delta\boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\kappa} \\ \boldsymbol{m} \\ \boldsymbol{n} \end{bmatrix}, \tag{34}$$

where the modified matrices are

$$\boldsymbol{\Psi} = \mathbb{Z}^T\mathbb{R}\mathbb{Z}, \quad \boldsymbol{\Lambda} = \mathbb{S}\mathbb{Z}, \quad \boldsymbol{\Gamma} = \mathbb{B}\mathbb{Z}.$$

Matrix $\boldsymbol{\Psi}$ is block diagonal and square, matrix $\boldsymbol{\Gamma}$ is block diagonal and rectangular, matrix $\boldsymbol{\Lambda}$ is also block diagonal and rectangular,

$$\begin{aligned}\boldsymbol{\kappa} &= \mathbb{Z}^T(-\mathbb{R}\boldsymbol{u}^- - \mathbb{S}^T\boldsymbol{x}^- - \mathbb{B}^T\boldsymbol{\lambda}^-),\\ \boldsymbol{m} &= -\mathbb{Q}\boldsymbol{x}^- - \mathbb{S}\boldsymbol{u}^- - \mathbb{A}^T\boldsymbol{\lambda}^-,\\ \boldsymbol{n} &= -\mathbb{A}\boldsymbol{x}^- - \mathbb{B}\boldsymbol{u}^- - \boldsymbol{d}.\end{aligned}$$

Reduction by Range-Space Method

The problem (34) will be solved in two steps:
1) Variables $\Delta\boldsymbol{u}_f$ and $\Delta\boldsymbol{x}$ are eliminated and the resulting system is solved for $\Delta\boldsymbol{\lambda}$.
2) When $\Delta\boldsymbol{\lambda}$ is known, variables $\Delta\boldsymbol{u}_f$ and $\Delta\boldsymbol{x}$ are computed.
From the first two equations of (34), $\Delta\boldsymbol{u}_f$ and $\Delta\boldsymbol{x}$ can be derived as solution of

$$\begin{bmatrix}\boldsymbol{\Psi} & \boldsymbol{\Lambda}^T\\ \boldsymbol{\Lambda} & \mathbb{Q}\end{bmatrix}\begin{bmatrix}\Delta\boldsymbol{u}_f\\ \Delta\boldsymbol{x}\end{bmatrix} = \begin{bmatrix}\boldsymbol{\kappa}\\ \boldsymbol{m}\end{bmatrix} - \begin{bmatrix}\boldsymbol{\Gamma}^T\\ \mathbb{A}^T\end{bmatrix}\Delta\boldsymbol{\lambda} \tag{35}$$

and inserted in the last equation of (34)

$$\begin{bmatrix}\boldsymbol{\Gamma} & \mathbb{A}\end{bmatrix}\begin{bmatrix}\Delta\boldsymbol{u}_f\\ \Delta\boldsymbol{x}\end{bmatrix} = \boldsymbol{n}. \tag{36}$$

The equation (35) can be computed effectively, because the structure of the problem is specific, similarly as in [15].
The matrix to be inversed has following structure,

$$\begin{bmatrix}\boldsymbol{\Psi} & \boldsymbol{\Lambda}^T\\ \boldsymbol{\Lambda} & \mathbb{Q}\end{bmatrix} = \begin{bmatrix}\boldsymbol{R} & 0 & 0 & \dots & 0 & 0 & \dots & 0\\ 0 & \boldsymbol{R} & 0 & \dots & \boldsymbol{S}^T & 0 & \dots & 0\\ 0 & 0 & \boldsymbol{R} & \dots & 0 & \boldsymbol{S}^T & \dots & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots\\ 0 & \boldsymbol{S} & 0 & \dots & \boldsymbol{Q} & 0 & \dots & 0\\ 0 & 0 & \boldsymbol{S} & \dots & 0 & \boldsymbol{Q} & \dots & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & 0 & \dots & 0 & 0 & \dots & \boldsymbol{Q}_{n_p}\end{bmatrix} \tag{37}$$

in case that no constraints are active.
Resulting problem for $[\Delta\boldsymbol{u}_f^T, \Delta\boldsymbol{x}^T]^T$ from (35) inserted into (36) is

$$\boldsymbol{\Phi}\Delta\boldsymbol{\lambda} = \boldsymbol{\beta}, \tag{38}$$

where

$$\begin{aligned}\boldsymbol{\Phi} &= \begin{bmatrix}\boldsymbol{\Gamma} & \boldsymbol{A}\end{bmatrix}\begin{bmatrix}\boldsymbol{\Psi} & \boldsymbol{\Lambda}^T\\ \boldsymbol{\Lambda} & \mathbb{Q}\end{bmatrix}^{-1}\begin{bmatrix}\boldsymbol{\Gamma}^T\\ \boldsymbol{A}^T\end{bmatrix}\\ \boldsymbol{\beta} &= \begin{bmatrix}\boldsymbol{\Gamma} & \boldsymbol{A}\end{bmatrix}\begin{bmatrix}\boldsymbol{\Psi} & \boldsymbol{\Lambda}^T\\ \boldsymbol{\Lambda} & \mathbb{Q}\end{bmatrix}^{-1}\begin{bmatrix}\boldsymbol{\kappa}\\ \boldsymbol{m}\end{bmatrix} - \boldsymbol{n}.\end{aligned} \tag{39}$$

Note that block matrix inversion is

$$\begin{bmatrix} \boldsymbol{\Psi} & \boldsymbol{\Lambda}^T \\ \boldsymbol{\Lambda} & \mathbb{Q} \end{bmatrix}^{-1} = \begin{bmatrix} \boldsymbol{\Omega}^{-1} & -\boldsymbol{\Omega}^{-1}\boldsymbol{\Lambda}^T\mathbb{Q}^{-1} \\ -\mathbb{Q}^{-1}\boldsymbol{\Lambda}\boldsymbol{\Omega}^{-1} & \mathbb{Q}^{-1} + \mathbb{Q}^{-1}\boldsymbol{\Lambda}\boldsymbol{\Omega}^{-1}\boldsymbol{\Lambda}^T\mathbb{Q}^{-1} \end{bmatrix},$$

where $\boldsymbol{\Omega} = \boldsymbol{\Psi} - \boldsymbol{\Lambda}^T\mathbb{Q}^{-1}\boldsymbol{\Lambda}$. Since $\boldsymbol{\Psi}$ and $\boldsymbol{\Psi} - \boldsymbol{\Lambda}^T\mathbb{Q}^{-1}\boldsymbol{\Lambda}$ are nonsingular, the inversion is defined.

The structure of the $\boldsymbol{\Phi}$ matrix is of importance here. It has block tridiagonal form, with blocks of size $n_x \times n_x$. The overall size of the $\boldsymbol{\Phi}$ is $n_p n_x \times n_p n_x$.

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\Phi}_a & \boldsymbol{\Phi}_b^T & 0 & \dots & 0 & 0 \\ \boldsymbol{\Phi}_b & \boldsymbol{\Phi}_c & \boldsymbol{\Phi}_b^T & \dots & 0 & 0 \\ 0 & \boldsymbol{\Phi}_b & \boldsymbol{\Phi}_c & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & \boldsymbol{\Phi}_c & \boldsymbol{\Phi}_b^T \\ 0 & 0 & 0 & \dots & \boldsymbol{\Phi}_b & \boldsymbol{\Phi}_d \end{bmatrix} \tag{40}$$

$$\begin{aligned}
\boldsymbol{\Phi}_a =& \boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^T + \boldsymbol{Q}^{-1} + \boldsymbol{Q}^{-1}\boldsymbol{S}(\boldsymbol{R}^{-1} - \boldsymbol{S}\boldsymbol{Q}^{-1}\boldsymbol{S}^T)^{-1}\boldsymbol{S}^T\boldsymbol{Q}^{-1}, \\
\boldsymbol{\Phi}_b =& \boldsymbol{B}(\boldsymbol{R}^{-1} - \boldsymbol{S}\boldsymbol{Q}^{-1}\boldsymbol{S}^T)^{-1}\boldsymbol{S}^T\boldsymbol{Q}^{-1} - \boldsymbol{A}\boldsymbol{Q}^{-1} - \boldsymbol{A}\boldsymbol{Q}^{-1}\boldsymbol{S}(\boldsymbol{R}^{-1} - \boldsymbol{S}\boldsymbol{Q}^{-1}\boldsymbol{S}^T)^{-1}\boldsymbol{S}^T\boldsymbol{Q}^{-1}, \\
\boldsymbol{\Phi}_c =& \boldsymbol{B}(\boldsymbol{R}^{-1} - \boldsymbol{S}\boldsymbol{Q}^{-1}\boldsymbol{S}^T)^{-1}\boldsymbol{B}^T - \boldsymbol{A}\boldsymbol{Q}^{-1}\boldsymbol{S}(\boldsymbol{R}^{-1} - \boldsymbol{S}\boldsymbol{Q}^{-1}\boldsymbol{S}^T)^{-1}\boldsymbol{B}^T + \boldsymbol{A}\boldsymbol{Q}^{-1}\boldsymbol{A}^T + \boldsymbol{Q}^{-1} - \\
& \boldsymbol{B}(\boldsymbol{R}^{-1} - \boldsymbol{S}\boldsymbol{Q}^{-1}\boldsymbol{S}^T)^{-1}\boldsymbol{S}^T\boldsymbol{Q}^{-1}\boldsymbol{A}^T + \boldsymbol{A}\boldsymbol{Q}^{-1}\boldsymbol{S}(\boldsymbol{R}^{-1} - \boldsymbol{S}\boldsymbol{Q}^{-1}\boldsymbol{S}^T)^{-1}\boldsymbol{S}^T\boldsymbol{Q}^{-1}\boldsymbol{A}^T + \\
& \boldsymbol{Q}^{-1}\boldsymbol{S}(\boldsymbol{R}^{-1} - \boldsymbol{S}\boldsymbol{Q}^{-1}\boldsymbol{S}^T)^{-1}\boldsymbol{S}^T\boldsymbol{Q}^{-1}, \\
\boldsymbol{\Phi}_d =& \boldsymbol{B}(\boldsymbol{R}^{-1} - \boldsymbol{S}\boldsymbol{Q}^{-1}\boldsymbol{S}^T)^{-1}\boldsymbol{B}^T - \boldsymbol{A}\boldsymbol{Q}^{-1}\boldsymbol{S}(\boldsymbol{R}^{-1} - \boldsymbol{S}\boldsymbol{Q}^{-1}\boldsymbol{S}^T)^{-1}\boldsymbol{B}^T + \boldsymbol{A}\boldsymbol{Q}^{-1}\boldsymbol{A}^T + \boldsymbol{Q}_N^{-1} - \\
& \boldsymbol{B}(\boldsymbol{R}^{-1} - \boldsymbol{S}\boldsymbol{Q}^{-1}\boldsymbol{S}^T)^{-1}\boldsymbol{S}^T\boldsymbol{Q}^{-1}\boldsymbol{A}^T + \boldsymbol{A}\boldsymbol{Q}^{-1}\boldsymbol{S}(\boldsymbol{R}^{-1} - \boldsymbol{S}\boldsymbol{Q}^{-1}\boldsymbol{S}^T)^{-1}\boldsymbol{S}^T\boldsymbol{Q}^{-1}\boldsymbol{A}^T
\end{aligned}$$

The complexity of the Newton step is different from the complexity of the Newton step in the original NPP.

The solution consists of three phases.

1. The matrix $\boldsymbol{\Phi}$ and vector $\boldsymbol{\beta}$ are formed, as in (39). The complexity of this phase is dominated by the Cholesky decomposition of the matrix from (35). According to the guides for computing the FLOP complexity from [25], the complexity of decomposition is in this case $\mathcal{O}(\frac{1}{3}n_p(n_u+n_x)^3)$, because the matrix is symmetric and has block-banded structure. The full step has complexity of order $\mathcal{O}(\frac{1}{3}n_p(n_u+n_x)^3 + n_p n_x^2(n_u+n_x))$.
2. The equation (38) is solved, again using decomposition, substitution and back-substitution. The complexity is of order $\mathcal{O}(n_p n_x^3 + 4n_p n_x^2)$. The complexity of this step is therefore significantly smaller (for larger systems) than the complexity of the first step.
3. The variables $\Delta\boldsymbol{u}_f$, $\Delta\boldsymbol{x}$ are found using equation (35). The complexity of this step is $\mathcal{O}(n_p n_u n_x + n_p n_x^2)$, as the Cholesky factor has been already found.

4.2.3 Sparse Projected Line Search

The Projected Line Search [29] step serves for expanding the active set. This step had to be modified as well, to use the sparse form of the MPC problem.

The cost function J to be minimized in the line search has in this case form

$$J = \frac{1}{2}\left(\boldsymbol{v}^- + \alpha\Delta\boldsymbol{v}\right)^T \mathbb{W}\left(\boldsymbol{v}^- + \alpha\Delta\boldsymbol{v}\right),$$

where $\mathbb{W} = \begin{bmatrix} \mathbb{R} & \mathbb{S}^T \\ \mathbb{S} & \mathbb{Q} \end{bmatrix}$ is weight matrix of the problem in sparse form, α is the step size to be found, $\boldsymbol{v}^- = [\boldsymbol{u}^{-T}, \boldsymbol{x}^{-T}]^T$ is vector of inputs and states and $\Delta\boldsymbol{v}$ is the direction to be projected.

The main difference between the forms is that in the case of the original form the effect of the state equation (4) is already present in the Hessian and gradient of the problem. In the sparse form, this equality constraint has to be taken into account explicitly.

Algorithm 6 Sparse Projected Line Search

1: $\boldsymbol{u} := \boldsymbol{u}_0$.
2: **for** j:=1 to Nn_u **do**
3: Find $\boldsymbol{p}_u^j$, direction in $\boldsymbol{u}$ of current iteration as in the dense line search;
4: Calculate $\mathbb{A}\boldsymbol{p}_x^j = -\mathbb{B}\boldsymbol{p}_u^j$;
5: Calculate $\Delta\alpha^*$ from (41);
6: $\boldsymbol{v}(j+1) = \boldsymbol{v}(j) + \Delta\alpha\boldsymbol{p}^j$
7: **if** $\Delta\alpha$ is global minimizer **then**
8: break;
9: **end if**
10: **end for**
11: Recalculate $\boldsymbol{\lambda}$ as $\boldsymbol{\lambda} = -\mathbb{A}^T(\mathbb{S}\boldsymbol{u} + \mathbb{Q}\boldsymbol{x})$;

As only the constraints on the inputs are assumed, the PLS as described in the Section 4.1 is applied only on inputs. Following the algorithm from original NPP, one arrives to formula for $\Delta\alpha^*$ in sparse form,

$$\Delta\alpha^* = -\frac{\boldsymbol{v}(j)^T\mathbb{W}\boldsymbol{p}^j}{\boldsymbol{p}^{jT}\mathbb{W}\boldsymbol{p}^j}, \tag{41}$$

where $\boldsymbol{p}^j$ is direction of the minimization in variables $\boldsymbol{u}$, $\boldsymbol{x}$ in the j-th iteration.

However, because of the dependency of the states $\boldsymbol{x}$ on inputs $\boldsymbol{u}$ through the equality constraints, the step in states has to be recalculated from the inputs in every iteration of the PLS, using equation

$$\mathbb{A}\boldsymbol{p}_x^j = -\mathbb{B}\boldsymbol{p}_u^j. \tag{42}$$

The equation (42) is satisfied for p_x^0 and p_u^0, as the solution of the Newton step satisfies this equation, but it has to be satisfied for every iteration.
Furthermore, the step in Lagrangian multipliers $\boldsymbol{\lambda}$ has to be recalculated from equation

$$\mathbb{A}^T\boldsymbol{\lambda} = -(\mathbb{S}\boldsymbol{u} + \mathbb{Q}\boldsymbol{x}).$$

Note, that in contrast with the equation (42), this can be done only once, after the projection in variables $\boldsymbol{u}$ and $\boldsymbol{x}$ has been done.

That way, the equality constraints are satisfied within each iteration of PLS, and the Newton direction is correctly projected. An example of the Newton step projection with states correction is shown in Figure 7, a simple case with $n_u = 1$ and $n_x = 1$ is shown.

The original direction, $\boldsymbol{p}_0$ points to the optimum. However, there is bound u_b, preventing the algorithm from reaching the unconstrained optimum. Upon reaching the

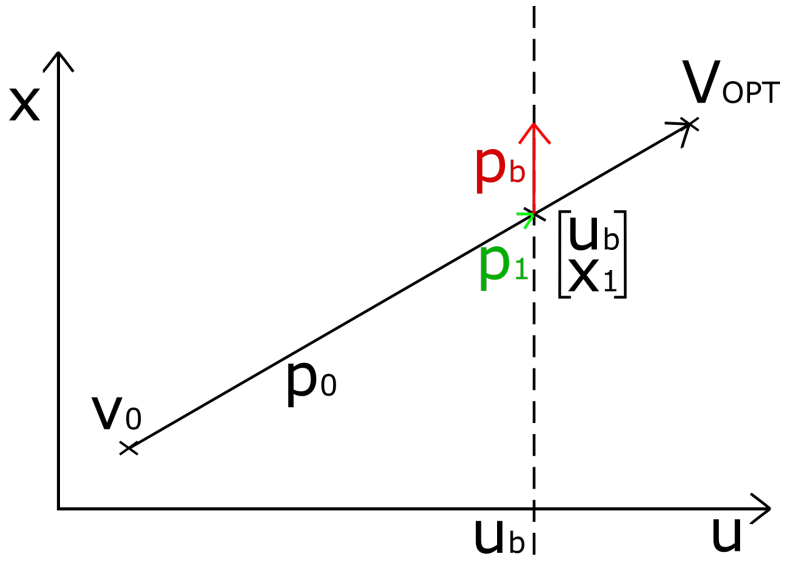


Figure 7 Example of Sparse Projected Line Search

point $[u_b^T, x_1^T]^T$, the first coordinate of $\boldsymbol{p}_0$ is set to zero obtaining the direction $\boldsymbol{p}_b$. The step in states cannot be made independently of the inputs, because of the equality constraints imposed on the direction. The step in states $\boldsymbol{p}_x$ has to be recalculated to satisfy them. In this case, obtaining $\boldsymbol{p}_1 = [0, 0]^T$.

The complexity of this step is dominated by the computations of the steps in variables $\boldsymbol{u}$ and $\boldsymbol{x}$, (42). In those steps, the decomposition is not needed as the $\mathbb{A}$ matrix already has lower triangular structure. Furthermore, it has block-banded sparse structure. The complexity for one recalculation of the step $\boldsymbol{p}_x$ is therefore $\mathcal{O}(4n_p n_x^2 + 2n_p n_u n_x)$ and complexity of recalculation of $\boldsymbol{\lambda}$ is $\mathcal{O}(6n_p n_x^2 + 3n_p(n_u + n_x) + 2n_p n_u n_x + n_p n_x + 3n_x^2)$.

4.2.4 Comparison of Complexity

The comparison of the forms of NPP algorithm is presented in the Table 1. The complexity is compared in terms of required FLOPs. In this thesis, the FLOP is any individual floating-point operation, e.g. multiplication or addition. For the sake of simplicity, the effect of the clamping of the active constraints in the third row of this table has been ignored for the sparse form, considering only the worst case when no constraints are active.

The symbol n_p represents the length of prediction horizon. The symbol m^j represents the number of active constraints in the j-th iteration.

The only change in row 1 comes from the size of the optimization vector and structure of the matrix $\mathbb{H}$. The most computationally demanding step in the original NPP algorithm is the step 3. The complexity of this step and consequently of the whole algorithm is dominated by the term $(n_p n_u - m^j)^3$ in the case of the dense form, and by the term, $\frac{1}{3}n_p(n_u + n_x)^3$ in the case of the sparse form. Therefore, the resulting complexity in the sparse form is linear in the length of the prediction horizon n_p, which is an improvement since n_p is usually significantly bigger than n_u or n_z. Such change is possible because the matrices in the Newton step computation have a block-tridiagonal (40) and block-banded structure (37) now.

The complexity of step 4 is given by the number of active constraints as well as the size of the vector.

No.	Name	Operation	Original NPP	Sparse NPP
1	Gradient	$\mathbb{H}\boldsymbol{z}^j + \boldsymbol{f}$	$2(n_p n_u)^2$	$2n_p(n_z)^2$
2	Proportionality	$\|\|\boldsymbol{\beta}(\boldsymbol{z}^j)\|\| \geq \Gamma\|\|\boldsymbol{\phi}(\boldsymbol{z}^j)\|\|$	$2n_p n_u$	$2n_p n_u$
3	Face problem	$\mathbb{G}\Delta\boldsymbol{z}_z = -\boldsymbol{r}$	$(n_p n_u - m^j)^3 + 2(n_p n_u - m^j)^2$	$\frac{1}{3}n_p(n_u + n_x)^3 + n_p n_x^2(n_u + n_x) + n_p n_u n_x$
4	Precompute $\mathbb{H}\boldsymbol{p}^j$	$\mathbb{H}\boldsymbol{p}^j$	$2m^j(n_p n_u - m^j)$	$2m^j(n_p n_z)$
5	Breakpoints	$\alpha_f = \max\{\alpha : \boldsymbol{z}^j + \alpha\boldsymbol{p}^j \in \Omega\}$	$2n_p n_u$	$2n_p n_u$
6	PLS routine	$[\boldsymbol{z}^{j+1}, \boldsymbol{g}(\boldsymbol{z}^{j+1})] =$ PLS $(\boldsymbol{z}^j, \boldsymbol{p}^j, \mathbb{H}\boldsymbol{p}^j, \boldsymbol{g}(\boldsymbol{z}^j))$	$2(n_p n_u - m^j) + \sum_{i=1}^{s}(2n_p n_u m^j + 10n_p n_u)$	$2(n_p n_u - m^j) + 6n_p n_x^2 + 2n_p n_u n_x + 3n_p(n_u + n_x) + 3n_x^2 + n_p n_x + 10n_p n_u + \sum_{i=1}^{s}(4n_p n_x^2 + 2n_p n_u n_x)$
7	Update $\boldsymbol{z}$	$\boldsymbol{z}^{j+1} = \boldsymbol{z}^j + \boldsymbol{p}^j$	$n_p n_u$	$n_p n_z$
8	Update $\boldsymbol{g}$	$\boldsymbol{g}(\boldsymbol{z}^{j+1}) = \boldsymbol{g}(\boldsymbol{z}^j) + (\mathbb{H}\boldsymbol{p}^j)$	$n_p n_u$	$n_p n_z$

Table 1 Comparison of FLOP

In step 5, the max step size is found, the complexity of FLOP is linear with respect to the vector $\boldsymbol{z}^j$ length.

Step 6 presents the Projected Line Search, while having linear complexity in both the number of variables n_u and prediction horizon n_p, in the case of the Sparse Projected Line Search, the situation is more complicated, because of the computation of step in $\boldsymbol{x}$ and recomputation of $\boldsymbol{\lambda}$. The complexity is linear in n_p and quadratic in n_x and n_u. It depends on the number of inner iterations of the sparse PLS. However, the complexity is significantly lower than that of step 2.

Step 7 and 8 are simple updates of the vector $\boldsymbol{z}$ and gradient $\boldsymbol{g}(\boldsymbol{z})$. Therefore the complexity is linear.

Therefore, the complexity of the proposed Sparse NPP algorithm is linear in length of prediction horizon. Furthermore, the original algorithm is well optimized, using e.g. decomposition updates when possible. These advanced features have not been investigated in the proposed algorithm yet.

4.3 Tailored Move Blocking

The move blocking, 2.3 enables to set several consecutive inputs constant. Thus the dimension of the problem is reduced. In the case of the proposed method, some states are also excluded from the optimization along with the inputs. Using the blocking for several sequences of different length is possible.

The tailored blocking procedure is then done due to the following transform

$$\begin{aligned} \boldsymbol{x} &= \mathbb{M}\boldsymbol{x}_B + \mathbb{N}\mathbb{G}\boldsymbol{u}_B \\ \boldsymbol{u} &= \mathbb{G}\boldsymbol{u}_B, \end{aligned} \tag{43}$$

where $\mathbb{G}$ is the same blocking matrix as in Section 2.3.

The idea behind the relations from (43) is the following. For the vector of states in the k-th step $\boldsymbol{x}_k$, it holds that either $\boldsymbol{x}_k = \boldsymbol{x}_k$ in case that this state is ruled out from optimization, or $\boldsymbol{x}_k = \boldsymbol{A}\boldsymbol{x}_{k-1} + \boldsymbol{B}\boldsymbol{u}_{k-1}$, in case that this step remains in the optimization. If also state vector $\boldsymbol{x}_{k-1}$ is ruled out of the optimization, one can continue recursively, until the right side of the equation contains only inputs and states to be

optimized. The coefficients of the state vectors are then collected in the $\mathbb{M}$ matrix, and the coefficients of the inputs are collected into the $\mathbb{N}$ matrix.

Example An example case for which input blocks are determined by $n_B = [1, 2, 3]$, prediction horizon $n_p = 6$ and control horizon $n_c = 3$ follows. For this case the vectors and matrices in (43) are

$$\boldsymbol{x}_B = \left[\boldsymbol{x}_{1_B}^T, \boldsymbol{x}_{3_B}^T, \boldsymbol{x}_{6_B}^T\right]^T, \ \boldsymbol{u}_B = \left[\boldsymbol{u}_{0_B}^T, \boldsymbol{u}_{1_B}^T, \boldsymbol{u}_{2_B}^T\right]^T,$$
$$\mathbb{M} = \begin{bmatrix} \boldsymbol{I} & & \\ \boldsymbol{A} & & \\ & \boldsymbol{I} & \\ & \boldsymbol{A} & \\ & \boldsymbol{A}^2 & \\ & & \boldsymbol{I} \end{bmatrix}, \ \mathbb{N} = \begin{bmatrix} 0 & & & \\ \boldsymbol{B} & & & \\ & 0 & & \\ & & \boldsymbol{B} & \\ & & \boldsymbol{AB} & \boldsymbol{B} \\ & & & 0 \end{bmatrix}, \ \mathbb{G} = \begin{bmatrix} \boldsymbol{I} & & \\ & \boldsymbol{I} & \\ & \boldsymbol{I} & \\ & & \boldsymbol{I} \\ & & \boldsymbol{I} \\ & & \boldsymbol{I} \end{bmatrix}. \tag{44}$$

This example is sketched in Figure 8.

Remark 1 *The size of the first block n_{B_1} has to be one. Otherwise, $\mathbb{M}$ contains zeros in the first row, which causes singularity of $\mathbb{A}_B$.*

Remark 2 *As shown in the example, the number of state samples has to be at least equal to the number of input samples. The position of this minimum samples is given by the size of input blocks. Other state samples can not be dropped without loss of sparsity pattern.*

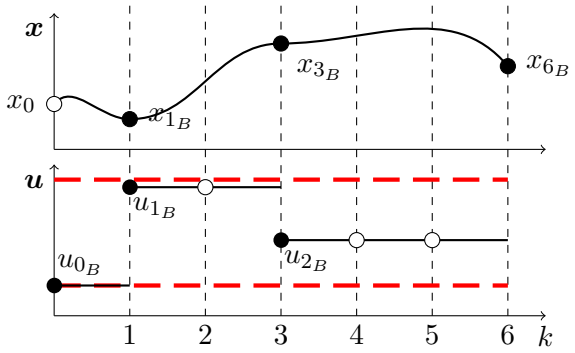


Figure 8 Sketch of move blocking in the example.

By applying (43) to (6) the following reduced optimization problem is obtained.

$$\begin{aligned} \min_{\boldsymbol{x}_B, \boldsymbol{u}_B} \quad & J_B(\boldsymbol{x}_B, \boldsymbol{u}_B) \\ \text{s.t.} \quad & 0 = \tilde{\mathbb{A}}\boldsymbol{x}_B + \tilde{\mathbb{B}}\boldsymbol{u}_B + \boldsymbol{d} \\ & \underline{\boldsymbol{u}}_B \leq \boldsymbol{u}_B \leq \overline{\boldsymbol{u}}_B, \end{aligned} \tag{45}$$

where $\tilde{\mathbb{A}} = \mathbb{A}\mathbb{M}$, $\tilde{\mathbb{B}} = (\mathbb{A}\mathbb{N} + \mathbb{B})\mathbb{G}$ and with cost function defined by

$$J_B(\boldsymbol{x}_B, \boldsymbol{u}_B) = \frac{1}{2} \begin{bmatrix} \boldsymbol{u}_B \\ \boldsymbol{x}_B \end{bmatrix}^T \begin{bmatrix} \tilde{\mathbb{R}} & \tilde{\mathbb{S}}^T \\ \tilde{\mathbb{S}} & \tilde{\mathbb{Q}} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_B \\ \boldsymbol{x}_B \end{bmatrix}.$$

Where new weight matrices have the following form

$$\begin{aligned}\tilde{\mathbb{Q}} &= \mathbb{M}^T\mathbb{Q}\mathbb{M} \\ \tilde{\mathbb{S}} &= \mathbb{M}^T(\mathbb{Q}\mathbb{N}+\mathbb{S})\mathbb{G} \\ \tilde{\mathbb{R}} &= \mathbb{G}^T(\mathbb{R}+\mathbb{N}^T\mathbb{Q}\mathbb{N}+\mathbb{N}^T\mathbb{S}+\mathbb{S}^T\mathbb{N})\mathbb{G}.\end{aligned} \tag{46}$$

Since all matrices in (45) will reduce their size, the computational demand for Newton step is decreased as well.

An important observation is that the new weight matrices from (46) have a similar structure as the original weight matrices. That means that they are also block diagonal.

In the case of the example matrices from (44), the modified weight matrices will have the following form.

$$\tilde{\mathbb{Q}} = \begin{bmatrix} \boldsymbol{Q}+\boldsymbol{A}^T\boldsymbol{Q}\boldsymbol{A} & 0 & 0 \\ 0 & \boldsymbol{Q}+\boldsymbol{A}^T\boldsymbol{Q}\boldsymbol{A}+\boldsymbol{A}^{2^T}\boldsymbol{Q}\boldsymbol{A}^2 & 0 \\ 0 & 0 & \boldsymbol{Q}_N \end{bmatrix}$$

$$\tilde{\mathbb{S}} = \begin{bmatrix} 0 & \boldsymbol{S}+\boldsymbol{A}\boldsymbol{Q}\boldsymbol{B}+\boldsymbol{A}\boldsymbol{S} & 0 \\ 0 & 0 & \boldsymbol{S}+\boldsymbol{A}\boldsymbol{Q}\boldsymbol{B}+\boldsymbol{A}\boldsymbol{S}+\boldsymbol{A}^2\boldsymbol{Q}(\boldsymbol{A}\boldsymbol{B}+\boldsymbol{B})+\boldsymbol{A}^2\boldsymbol{S} \\ 0 & 0 & 0 \end{bmatrix}$$

$$\tilde{\mathbb{R}} = \begin{bmatrix} \boldsymbol{R} & 0 & 0 \\ 0 & 2\boldsymbol{R}+\boldsymbol{B}^T\boldsymbol{Q}\boldsymbol{B}+\boldsymbol{S}^T\boldsymbol{B}+\boldsymbol{B}^T\boldsymbol{S} & 0 \\ 0 & 0 & \boldsymbol{W} \end{bmatrix},$$

where $\boldsymbol{W} = 3\boldsymbol{R}+2\boldsymbol{B}^T\boldsymbol{Q}\boldsymbol{B}+2\boldsymbol{S}^T\boldsymbol{B}+2\boldsymbol{B}^T\boldsymbol{S}+\boldsymbol{B}^T\boldsymbol{A}^T\boldsymbol{Q}\boldsymbol{A}\boldsymbol{B}+\boldsymbol{B}^T\boldsymbol{A}^T\boldsymbol{Q}\boldsymbol{B}+\boldsymbol{B}^T\boldsymbol{Q}\boldsymbol{A}\boldsymbol{B}+\boldsymbol{B}^T\boldsymbol{A}^T\boldsymbol{S}+\boldsymbol{S}^T\boldsymbol{A}\boldsymbol{B}$.

The blocks of the matrices have the same size as before, just the number of blocks is smaller than before.

Remark 3 *An important observation is that the matrix* $\mathbb{S}$ *now contains nonzero cross-terms regardless if the original weight matrices contain such terms or not.*

5 Numerical Experiments

The proposed tailored NPP algorithm has been implemented in the MATLAB programming environment.

In this Chapter, the results obtained with this implementation are compared to the original dense NPP algorithm results. Firstly, the functionality of the algorithm is demonstrated on an oscillating masses control problem, then several tests on randomly generated systems are presented.

5.1 Oscillating Masses

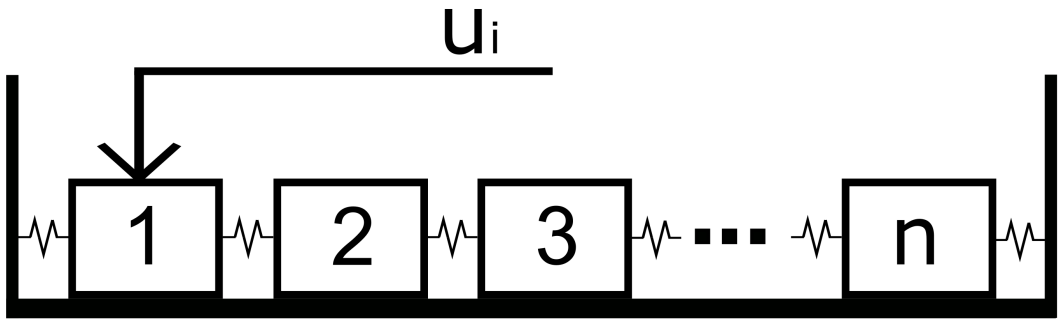


Figure 9 Oscillating masses layout

The oscillating masses problem consists of n masses connected with springs; a possible layout is shown in Figure 9.

The states of this system are positions of the masses with respect to the equilibrium and velocities of those masses. The inputs are forces applied on some of the masses. The dynamics of the system are determined by the spring stiffness and damping coefficients as well as by the weight of the masses.

An example of open-control of such system with the following parameters has been simulated, prediction horizon $N = 50$, number of masses $n = 6$, $n_x = 12$ and $n_u = 3$. All the inputs are limited by box constraints -0.5 and 0.5.

The correctness of the solution has been checked with the original NPP algorithm, to ensure proper functionality. The results are shown in the Figures 10 and 11.

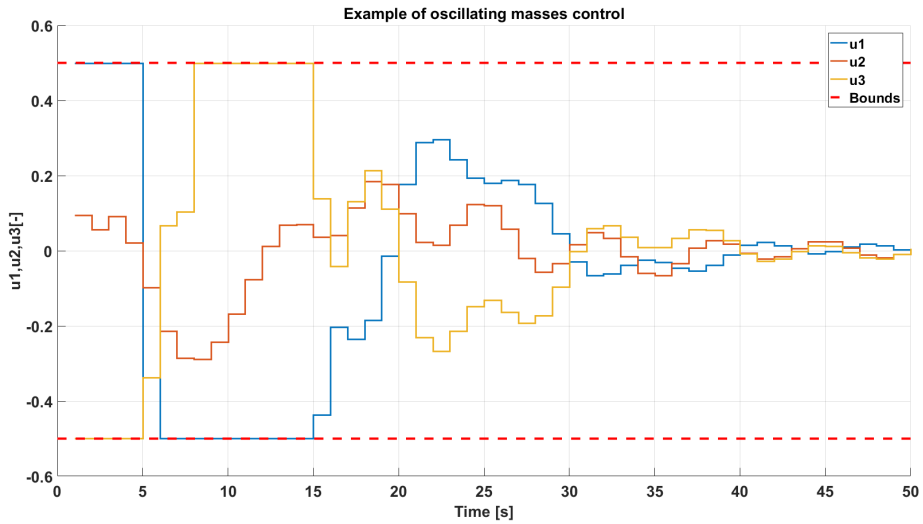


Figure 10 Computed control for oscillating masses

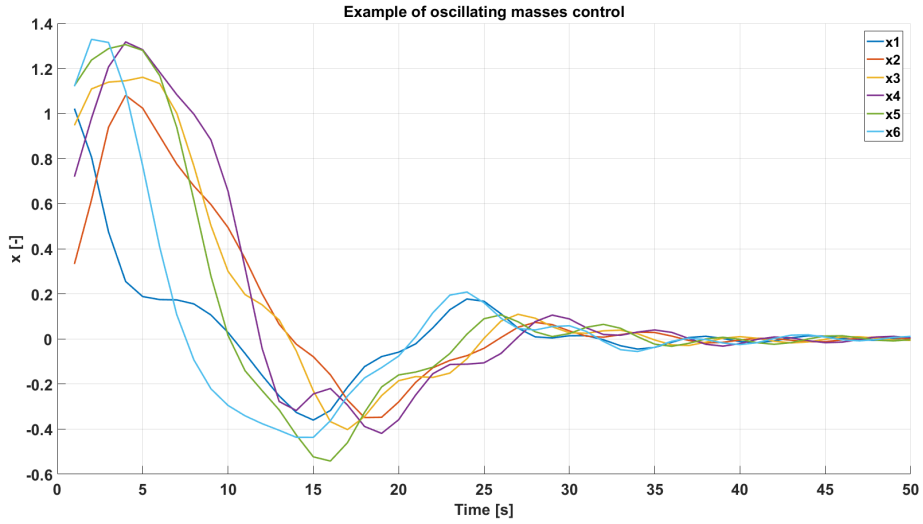


Figure 11 Position of oscillating masses

5.2 Random Systems

The algorithm was tested on several randomly generated systems of different parameters. The FLOPs were counted inside the algorithm and compared to the number of FLOPs of the original one. The time taken for the computations has not been compared because the sparse NPP algorithm uses several functions processed by MEX. Therefore, the comparison with original algorithm would not be objective.

The tests have been run for prediction horizons with length N ranging from 5 to 40.

The FLOPs of the sparse NPP have been estimated in the Chapter 4, however, to get a more accurate estimate, the symbolic framework for algorithmic differentiation and numeric optimization ***CasADi*** [31] has been used. This package allows implementing the functions more effectively than the native MATLAB functions. A symbolic function can be easily prepared and compiled for a particular matrix sparsity pattern such that only nonzero elements are taken into account.

Moreover, the package allows computing the actual FLOP count for a given function. This functionality has been used in this thesis. Therefore, the comparison is based on these actual FLOP counts, rather than on the prior estimates. However, the FLOPs have been counted for the worst case possibility, when no constraints are active.

Since the FLOP count of the dense NPP is affected by the number of active constraints and the FLOP count of the sparse NPP is not, the average number of active constraints and the number of iterations is plotted along with the FLOP results to provide a better insight.

The first test has been done for a system with parameters $n_x = 4$, $n_u = 2$. The results are plotted in the Figures 12, 13 and 14.

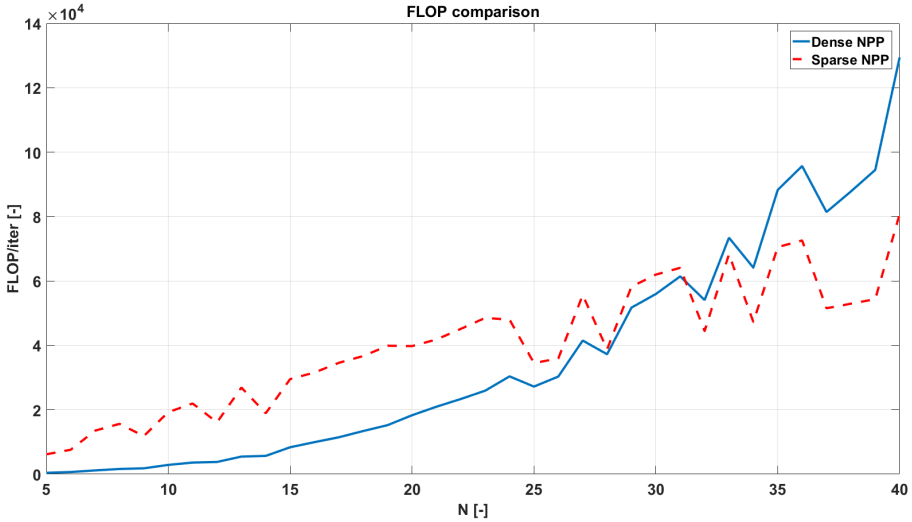


Figure 12 FLOP comparison for $n_x = 4$, $n_u = 2$

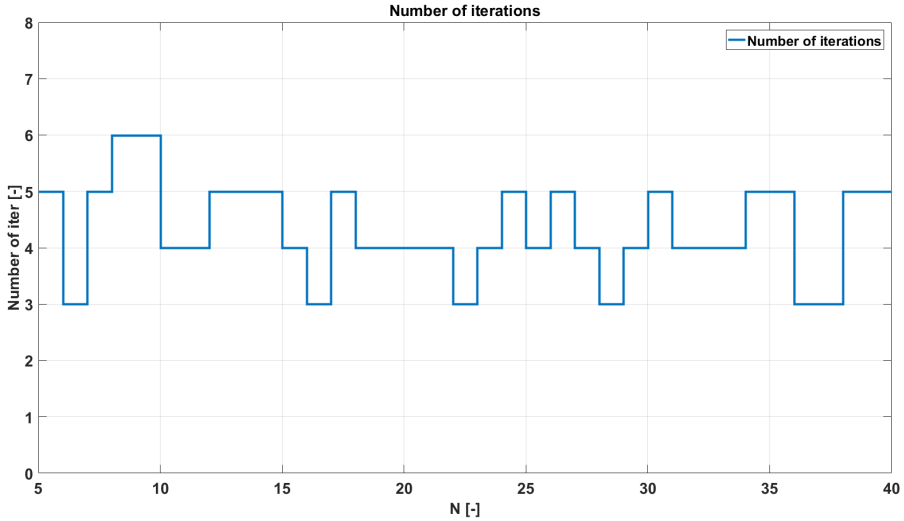


Figure 13 Number of iterations for $n_x = 4$, $n_u = 2$

The results show that for a short prediction horizon, the dense form of the NPP algorithm performs better in terms of the FLOP count. But for the prediction horizon length $N > 28$, the sparse form becomes more effective. In the second test, a system

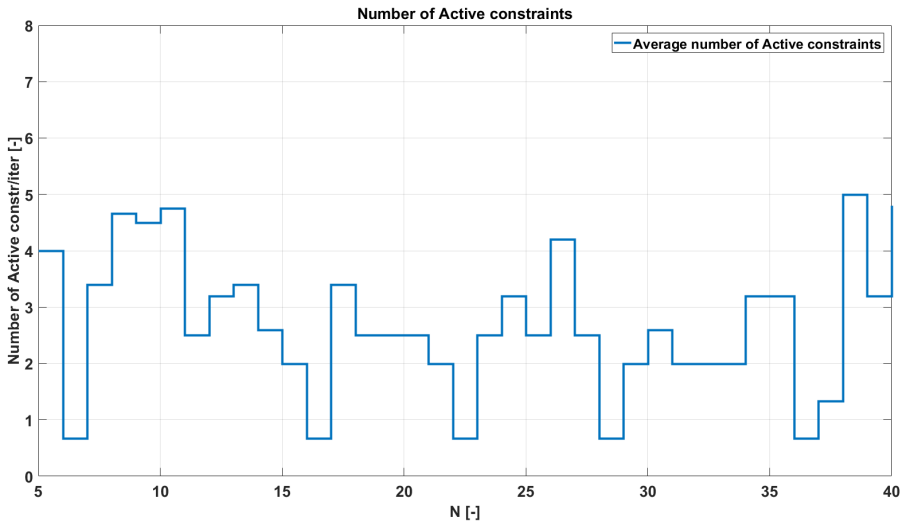


Figure 14 Number of active constraints for $n_x = 4$, $n_u = 2$

with parameters $n_x = 8$ and $n_u = 6$ was used. The results are plotted in the Figures 15, 16 and 17.

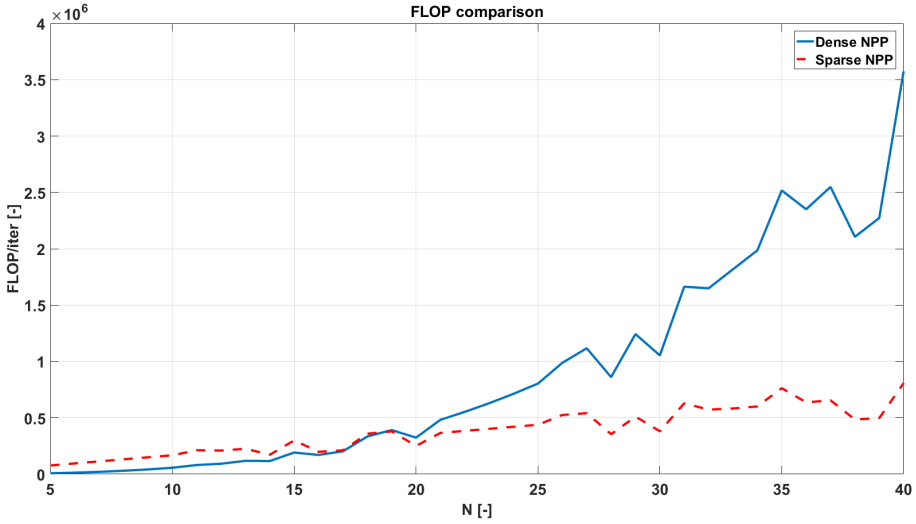


Figure 15 FLOP comparison for $n_x = 8$, $n_u = 6$

In this case, the sparse form becomes more efficient for smaller prediction horizons with length $N > 18$. In the third test, a system with parameters $n_x = 12$ and $n_u = 6$ was used. The results of this test are shown in the Figures 18, 19 and 20.

In this case, one can notice a similarity with the first test. The FLOP count trend and especially the point where the sparse form becomes more efficient than the dense form depends on the ratio of the number of states and the number of inputs. In the first test as well as in this one, the ratio of $n_x : n_u$ was $2 : 1$.

In the fourth test, a system with parameters $n_x = 16$ and $n_u = 6$ was used. The results of this test are shown in the Figures 21, 22 and 23.

In this test, the ratio $n_x : n_u$ was $8 : 3$. The results show that for this ratio the sparse form becomes better than the dense form for prediction horizons of length $N > 40$.

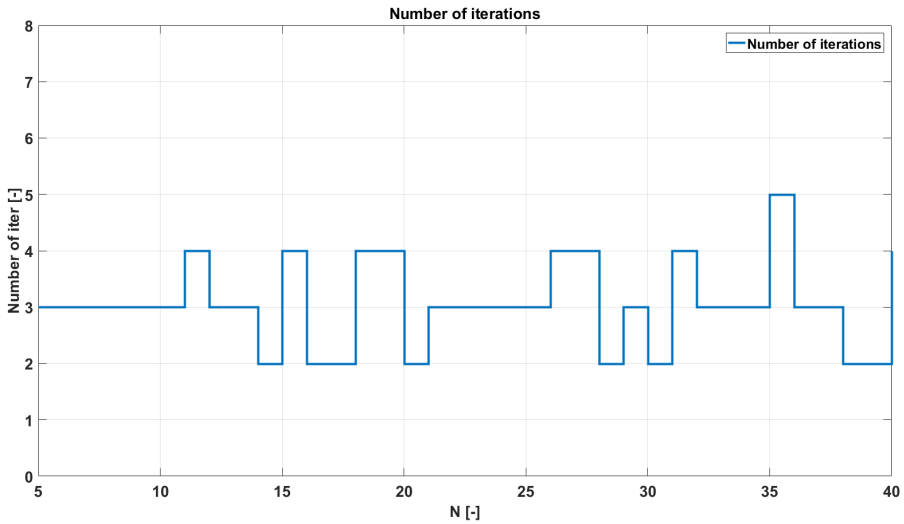


Figure 16 Number of iterations for $n_x = 8$, $n_u = 6$

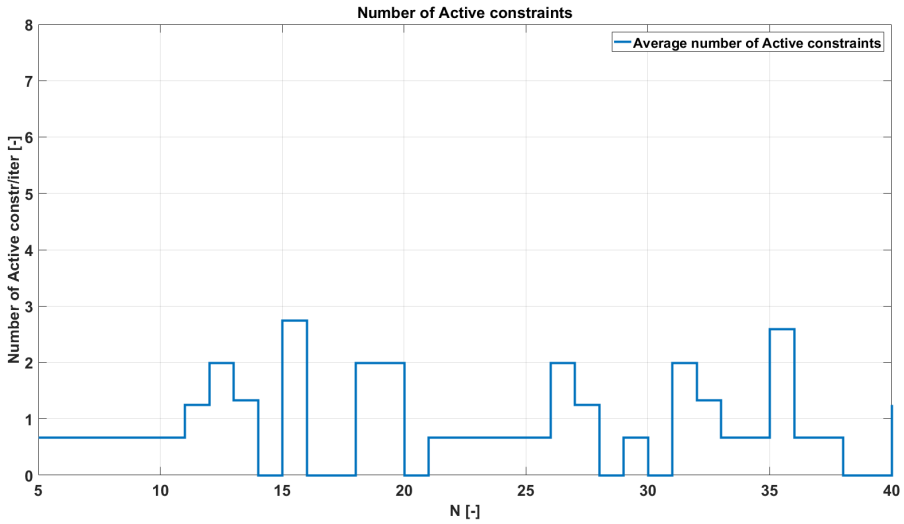


Figure 17 Number of active constraints for $n_x = 8$, $n_u = 6$

From these tests, it can be concluded that the effectivity of the proposed algorithm in comparison with the original one depends heavily on the ratio $n_x : n_u$.

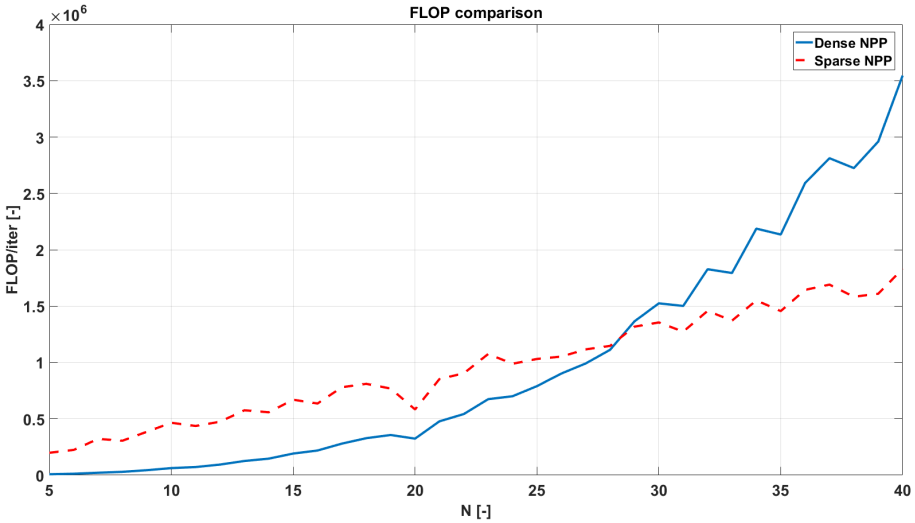


Figure 18 FLOP comparison for $n_x = 12$, $n_u = 6$

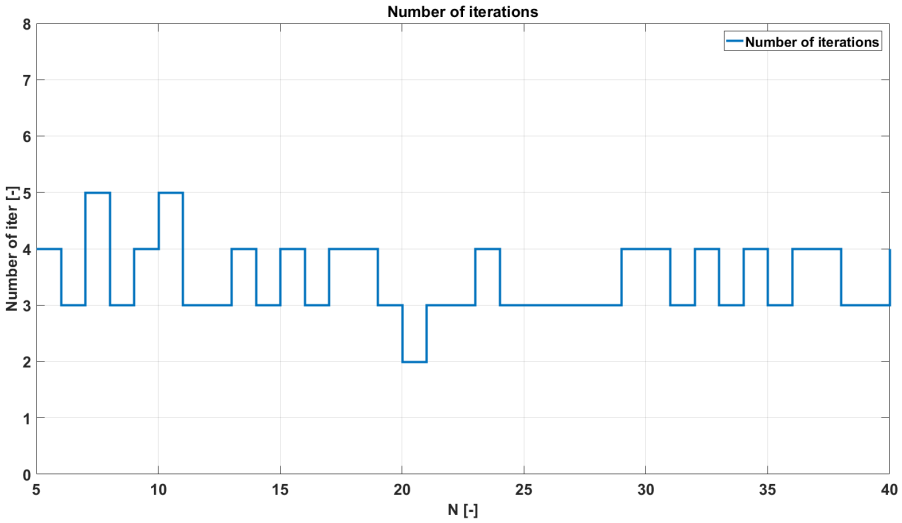


Figure 19 Number of iterations for $n_x = 12$, $n_u = 6$

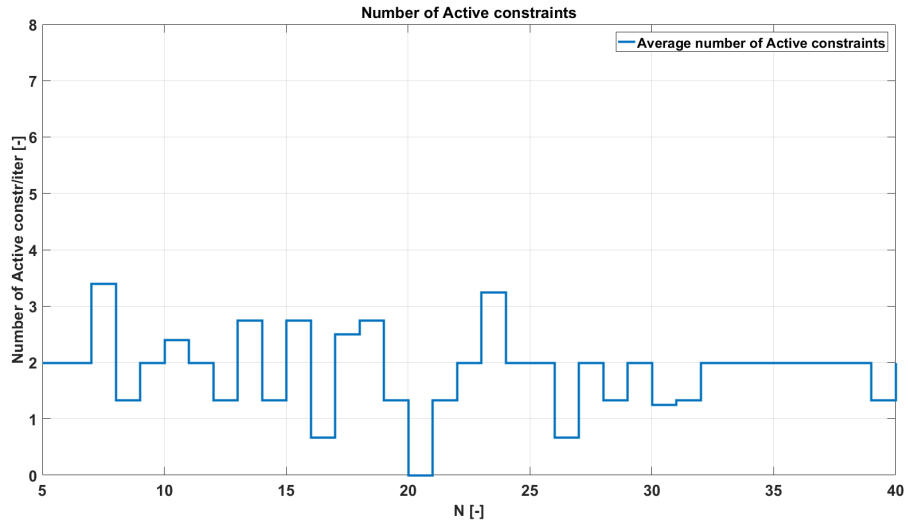


Figure 20 Number of active constraints for $n_x = 12$, $n_u = 6$

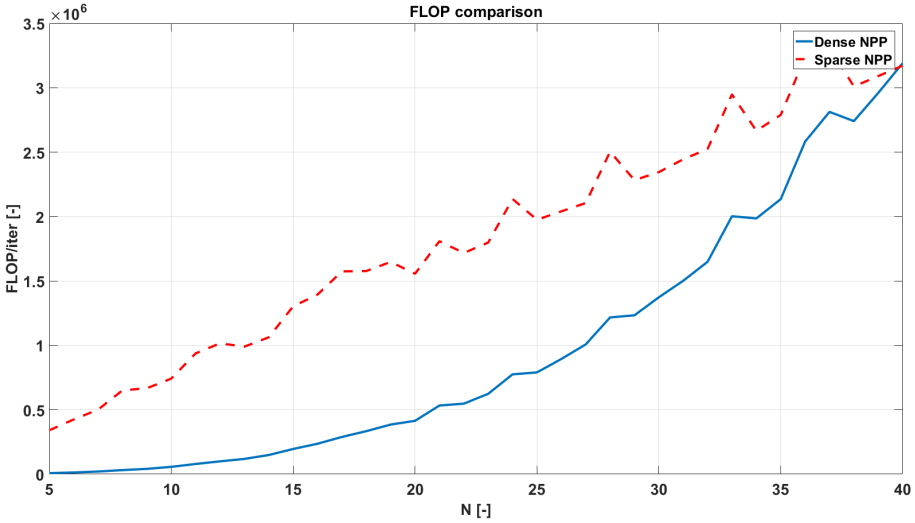


Figure 21 FLOP comparison for $n_x = 16$, $n_u = 6$

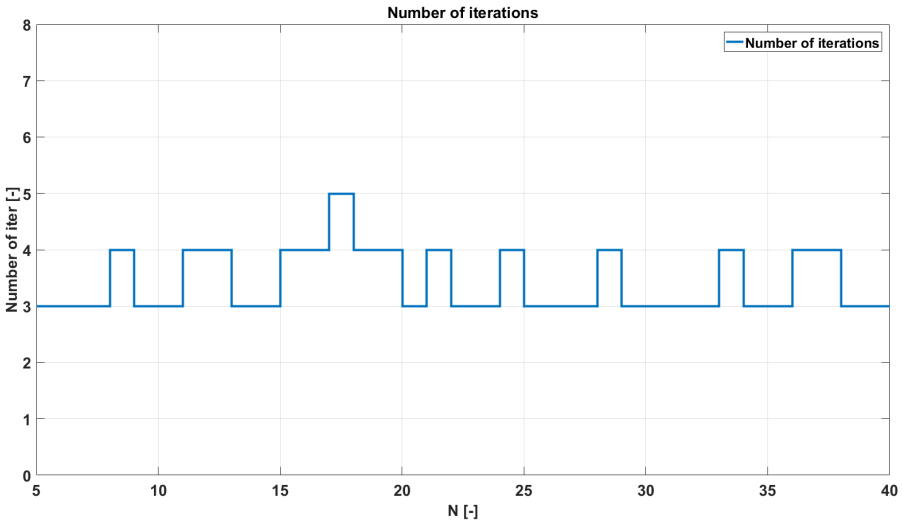


Figure 22 Number of iterations for $n_x = 16$, $n_u = 6$

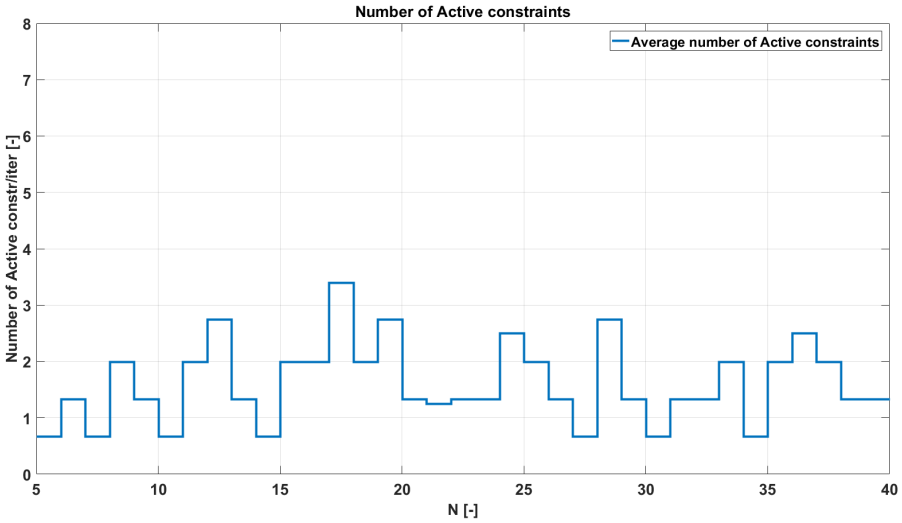


Figure 23 Number of active constraints for $n_x = 16$, $n_u = 6$

6 Conclusion

In this thesis, a new approach to solving the MPC optimization problem has been introduced. The resulting algorithm is ultimately the sparse version of the NPP algorithm from the thesis [7].

In the first Chapter, the background of the MPC has been introduced, along with several state-of-the-art solvers for both dense and sparse forms of the problem.

In the Chapter 2, the MPC concept has been examined, along with several possible formulations of the problem for transforming it to QP optimization problem.

Chapter 3 presented the QP problem along with several algorithms for solving the problem, the most important for this thesis being the AS method and the Gradient Projection method.

In the Chapter 4, the NPP algorithm and its sparse modification have been introduced. Several steps of the algorithm had to be altered, most importantly the Face problem solution. The new procedure has been inspired by an article written by S. Boyd and Y. Wang, [15], applied in the context of the AS method. The sparse version of the algorithm is suitable for long prediction horizons. Moreover, a Move Blocking strategy for sparse problems has been introduced, allowing a faster solution of the problem without destroying the problem structure.

Finally, in the Chapter 5, several experiments have been conducted. The correctness of the proposed algorithm has been demonstrated on the oscillating masses example, and then the proposed NPP algorithm has been compared to the original algorithm in terms of FLOPs. The result of this comparison was in accord with the initial expectations. The dense form of the algorithm performed better for and shorter prediction horizons, in contrast with the sparse form, which performed better for bigger problems and longer prediction horizons. This result is given by the fact, that the complexity of the sparse algorithm grows only linearly with the length of the prediction horizon, whereas the complexity of the original dense form grows cubically with the length of prediction horizon, which is evident from the complexity comparison in the Chapter 4.

In conclusion, the results shown in this thesis are promising, and the performance of the algorithm is comparable to the performance of the current state-of-the-art solvers.

6.1 Future Work

The algorithm presented in this thesis has been implemented in the MATLAB programming environment. However, several further improvements could be made.

For the further purpose, it is necessary the algorithm will be rewritten in C entirely programming language. The time comparison with other state-of-the-art solvers is required.

The algorithm is going to be revised, and it will also be a subject of an upcoming article, which should summarize the main points and present the results as well as further improvements.

One of the possible future enhancements is the implementation of the factor updates of the Hessian in the Face problem solution step, similarly as in the original NPP algorithm, if possible.

Bibliography

[1] J. B. Rawlings. “Tutorial overview of model predictive control”. In: *IEEE Control Systems* 20.3 (June 2000), pp. 38–52.

[2] S. Joe Qin and Thomas A. Badgwell. “A survey of industrial model predictive control technology”. In: *Control Engineering Practice* 11.7 (2003), pp. 733–764.

[3] Ruchika a N.Raghu. “Model Predictive Control: History and Development”. In: *IJETT* 4 (2013).

[4] Q. Chen et al. “Model Predictive Control for Three-Phase Four-Leg Grid-Tied Inverters”. In: *IEEE Access* 5 (2017), pp. 2834–2841.

[5] Changbin Hu et al. “Energy Coordinative Optimization of Wind-Storage-Load Microgrids Based on Short-Term Prediction”. In: *Energies* 8.2 (2015), pp. 1505–1528.

[6] J. Frasch. *qpDUNES —a dual Newton strategy for convex QP*. Presentation at Workshop "Embedded Quadratic Programming", "`http://www.syscop.de/files/2014ss/imtek-tempo/2014_03_fraschQPWorkshopFreiburg.pdf1`". 2014.

[7] Ondřej Šantin. “Numerical Algorithms of Quadratic Programming for Model Predictive Control”. PhD thesis. "České vysoké učení technické v Praze, fakulta Elektrotechnická", 2016.

[8] H.J. Ferreau et al. “qpOASES: A parametric active-set algorithm for quadratic programming”. In: *Mathematical Programming Computation* 6.4 (2014), pp. 327–363.

[9] *Quadprog (Optimization toolbox)*. URL: `http://www.ece.northwestern.edu/local-apps/matlabhelp/toolbox/optim/quadprog.html` (visited on 02/27/2017).

[10] B. E. D. Andersen, R. Sandvik J. Jensen, and U. Worsøe. *MOSEK version 6. MOSEK Technical report: TR-2009-3*. Tech. rep. MOSEK ApS, Oct. 2009.

[11] Stefan Richter, S Mariethoz, and Manfred Morari. “High-speed online MPC based on a fast gradient method applied to power converter control”. In: *American Control Conference (ACC)*. 2010, pp. 4737–4743.

[12] H.J. Ferreau, H.G. Bock, and M. Diehl. “An online active set strategy to overcome the limitations of explicit MPC”. In: *International Journal of Robust and Nonlinear Control* 18.8 (2008), pp. 816–830.

[13] F. Ullmann. “FiOrdOs: A Matlab Toolbox for C-Code Generation for First Order Methods”. MA thesis. ETH Zurich, 2011.

[14] Janick V. Frasch, Sebastian Sager, and Moritz Diehl. “A parallel quadratic programming method for dynamic optimization problems”. In: *Mathematical Programming Computation* 7.3 (2015), pp. 289–329.

[15] Yang Wang and Stephen Boyd. “Fast model predictive control using online optimization”. In: *IEEE Transactions on Control Systems Technology* 18.2 (2010), pp. 267–278.

[16] Alexander Domahidi. *FORCES: Fast Optimization for Real-time Control on Embedded Systems*. `http://forces.ethz.ch`. Oct. 2012.

[17] L. E. Sokoler et al. "Input-constrained model predictive control via the alternating direction method of multipliers". In: *Control Conference (ECC), 2014 European*. June 2014, pp. 115–120.

[18] E. Chu S. Boyd N. Parikh, B. Peleato, and J. Eckstein. "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers". In: *Foundations and Trends ^{TM}in Machine Learning* 3.1 (2011), pp. 1–122.

[19] Haocheng Luo, Zechun Hu, and Xu Xie. "Model predictive based automatic generation control with participation of the output-constrained wind farms". In: *2016 IEEE PES Asia-Pacific Power and Energy Engineering Conference (APPEEC)*. Oct. 2016, pp. 1584–1588.

[20] D. Chmielewski and V. Manousiouthakis. "On constrained infinite-time linear quadratic optimal control". In: *Decision and Control, Proceedings of the 35th IEEE Conference on* (1996).

[21] Juan L. Jerez, Eric C. Kerrigan, and George a. Constantinides. "A condensed and sparse QP formulation for predictive control". In: *IEEE Conference on Decision and Control and European Control Conference* (2011), pp. 5217–5222.

[22] R. Cagienard et al. "Move blocking strategies in receding horizon control". In: *Journal of Process Control* 17.6 (2007), pp. 563–570.

[23] M. Darouach T. Schwickart H. Voos and S. Bezzaoucha. "A flexible move blocking strategy to speed up model-predictive control while retaining a high tracking performance". In: *2016 European Control Conference (ECC), Aalborg, 2016, pp. 764-769* (2016).

[24] O. Mikulas. *Quadratic Programming Algorithms for Fast Model-Based Predictive Control*. 2013.

[25] L. Vanderberghe S. Boyd. *Convex Optimization*. Cambridge University Press, 2004.

[26] N. Karmarkar. "A new polynomial time algorithm for linear programming". In: *Combinatorica 4* (1984), pp. 373–395.

[27] I. Lustig. "Feasibility issues in a primal-dual interior-point method for linear programming". In: *Mathematical Programming 49* (1990), pp. 145–162.

[28] D. Shanno. "Who Invented the Interior-Point Method?" In: *Documenta Mathematica, Extra Volume: Optimization Stories (2012)* (2012), pp. 55–64.

[29] J. Nocedal and S. J. Wright. *Numerical Optimization, Second Edition*. Springer New York, 2006, pp. 497–528.

[30] O. Santin V. Havlena P. Otta. "Measured-State Driven Warm-Start Strategy for Linear MPC". In: *2015 European Control Conference (ECC), Linz, 2015, pp. 3132-3136* (2015).

[31] *CasADi GitHub page*. URL: `https://github.com/casadi/casadi/wiki` (visited on 05/08/2017).