Automatica 109 (2019) 108560

Contents lists available at ScienceDirect

Automatica

journal homepage: www.elsevier.com/locate/automatica

A parallel Newton-type method for nonlinear model predictive control☆

Haoyang Deng *, Toshiyuki Ohtsuka

Department of Systems Science, Graduate School of Informatics, Kyoto University, Sakyo-ku, Kyoto 606-8501, Japan

ARTICLE INFO

Article history:
Received 6 August 2018
Received in revised form 14 February 2019
Accepted 20 August 2019
Available online 30 August 2019

Keywords:
Nonlinear model predictive control
Newton-type
Parallel algorithm

ABSTRACT

A parallel Newton-type method for nonlinear model predictive control is presented that exploits the particular structure of the associated discrete-time Euler–Lagrange equations obtained by utilizing an explicit discretization method in the reverse-time direction. These equations are approximately decoupled into single-step subproblems along the prediction horizon for parallelization. The coupling variable of each subproblem is approximated to its optimal value using a simple, efficient, and effective method at each iteration. The rate of convergence of the proposed method is proved to be superlinear under mild conditions. Numerical simulation of using the proposed method to control a quadrotor showed that the proposed method is highly parallelizable and converges in only a few iterations, even to a high accuracy. Comparison of the proposed method's performance with that of several state-of-the-art methods showed that it is faster.

1. Introduction

Nonlinear model predictive control (NMPC) is an optimal control method for controlling nonlinear systems. A finite horizon optimal control problem (FHOCP) is solved at each sampling instant, and only the first optimal control input is applied. With its ability to handle constraints and nonlinear dynamics systematically, NMPC has become widely used in process and chemical applications (Qin & Badgwell, 2000), which typically have slow dynamics. However, the computational demand for solving FHOCPs in real time, especially ones with fast dynamics, large dimensionalities, and complicated constraints, prevents its broader application. One way to avoid online optimization is to shift it to offline, e.g., by using explicit NMPC. In explicit NMPC, the optimal control law is computed as a function of the initial state and stored. Since only the stored function has to be evaluated on-line, the amount of computation is reduced. However, NMPC, unlike linear MPC (Bemporad, Morari, Dua, & Pistikopoulos, 2002), does not have a general explicit state feedback law, and only an approximate explicit solution can be found, e.g., by multi-parametric nonlinear programming (Johansen, 2004). Although an exact solution can be obtained using the principle of dynamic programming (Bellman, 1957), that is, by solving the Hamilton–Jacobi–Bellman (HJB) equation, it is often computationally impractical because the amount of computations and the storage required grow prohibitively with the state space dimension and horizon, which is known as the "curse of dimensionality". One way to overcome this drawback is to use adaptive/approximate dynamic programming (ADP) to obtain an approximate solution of the HJB equation. A very good survey of ADP is given in Wang, Zhang, and Liu (2009). However, these methods require much data storage for high-dimensional systems and suffer performance loss due to sub-optimality.

When on-line optimization is inevitable, it has to be solved in real time. On-line methods for the numerical solution of OCPs can typically be categorized into indirect and direct methods (Von Stryk & Bulirsch, 1992). Indirect methods use the necessary condition for optimality known as Pontryagin's Maximum Principle (Pontryagin, Boltyanskii, Gamkrelidze, & Mishchenko, 1961). Methods that solve the continuous-time Euler–Lagrange equations obtained by the calculus of variations are indirect methods, and can be referred to as "first optimize, then discretize" methods. Methods that solve an FHOCP by first discretizing it into a nonlinear programming problem (NLP) that can be solved using an off-the-shelf NLP solver are direct methods. The latter type are the most widely used nowadays, and there have been numerous studies on them. Newton-type methods are direct methods that can be categorized, depending on how the inequalities are treated, into sequential quadratic programming (SQP) methods (Powell, 1978) and interior-point (IP) methods (Byrd, Hribar, & Nocedal, 1999). Efficient methods for SQP and IP methods have been developed by exploiting the particular structure

☆ This work was partly supported by JSPS KAKENHI Grant Number 15H02257. The material in this paper was partially presented at the 6th IFAC Conference on Nonlinear Model Predictive Control, August 19–22, 2018, Madison, Wisconsin, USA. This paper was recommended for publication in revised form by Associate Editor Gabriele Pannocchia under the direction of Editor Ian R. Petersen.

* Corresponding author.
E-mail addresses: deng.haoyang.23r@st.kyoto-u.ac.jp (H. Deng), ohtsuka@i.kyoto-u.ac.jp (T. Ohtsuka).

https://doi.org/10.1016/j.automatica.2019.108560

of NMPC, such as Riccati recursion exploiting the banded structure of the Hessian matrix (Frasch, Sager, & Diehl, 2015; Glad & Jonson, 1984; Jørgensen, Rawlings, & Jørgensen, 2004; Rao, Wright, & Rawlings, 1998), generalized Gauss–Newton methods for nonlinear least-squares NLPs (Bock, 1983; Houska, Ferreau, & Diehl, 2011b), and solution tracing by continuation (Ohtsuka, 2004), just to name a few. An excellent review of Newton-type methods can be found in Diehl, Ferreau, and Haverbeke (2009). In addition to the Newton-type methods, first-order methods for NMPC have been proposed that deal directly with the nonlinear dynamics (Graichen & Käpernick, 2012) or solve the underlying quadratic programming (QP) (Kalmari, Backman, & Visala, 2015; Kouzoupis, Ferreau, Peyrl, & Diehl, 2015).

Most real-time methods can only be parallelized to a certain extent, such as parallel system simulation in the context of multiple shooting (Bock & Plitt, 1984) or parallel matrix operations. However, there is growing demand for tailored parallel NMPC methods due to the rapid development of parallel devices such as multi-core processors, graphics processing units, and field-programmable gate arrays (FPGAs). Although first-order methods such as the alternating direction method of multipliers (Jerez et al., 2014; O'Donoghue, Stathopoulos, & Boyd, 2013) and the fast gradient method (Jerez et al., 2014) can be easily parallelized and efficiently implemented for linear MPC, they suffer from slow rates of convergence compared with the Newton-type methods, dealing with complicated constraints, and time-varying dynamics in the underlying linearized problems when using SQP for NMPC. Although the Newton-type methods are basically able to deal with general constraints and have fast rates of convergence, parallelization is still challenging. Methods based on parallel Riccati recursion (Frasch et al., 2015; Nielsen & Axehill, 2015) have been proposed. A computational complexity of $\mathcal{O}(\log N)$ for N threads can be achieved, where N is the number of prediction steps. Several tailored parallel methods for NMPC have been reported. The advanced multi-step NMPC (Yang & Biegler, 2013), which is parallelized along the time axis, concurrently solves several predicted OCPs beforehand, and then the input is updated on the basis of the state measurement. Particle swarm optimization (Xu, Chen, Gong, & Mei, 2016), with its inherent parallelism, has been implemented on FPGA for NMPC. An augmented Lagrangian-based method tailored to nonlinear OCPs has been reported (Kouzoupis, Quirynen, Houska, & Diehl, 2016) that concurrently solves subproblems along the prediction steps and then solves a centralized consensus QP to update the dual variables at each iteration. This method has a linear rate of convergence and its speed-up depends on the computation time of the consensus step in accordance with Amdahl's law (Amdahl, 1967).

In this paper, a highly parallelizable Newton-type method is proposed. The FHOCP is first discretized using an explicit integration scheme performed in the reverse-time direction, enabling the problem to be split into linearly coupled subproblems. Then, a sequential method, which we call the backward correction method, is formulated. The backward correction method is proved to be exactly identical to Newton's method but with a clearer structure for possibility of parallelization. In the backward correction method, the coupling variable in each subproblem is calculated recursively in a backward manner, which is time-intensive. Reasonable approximations of the coupling variables are used to break down the recursion process so that the calculation can be done in parallel. The resulting algorithm is then proved to converge superlinearly. Numerical simulation of controlling a quadrotor demonstrated that the proposed method is highly parallelizable and can converge to a specified tolerance in only a few iterations. Comparing with our earlier work (Deng & Ohtsuka, 2018), more general discretization method is considered and the rate of convergence is characterized in this paper.

This paper is organized as follows. First, the NMPC problem and its discretization are presented in Section 2. The backward correction method is formulated and proved to be exactly identical to Newton's method in Section 3. The proposed method and its rate of convergence are shown in Section 4. The numerical experiment is described and the results are discussed in Section 5. The key points are summarized and future work is mentioned in Section 6.

2. Problem formulation

2.1. Nonlinear model predictive control

Consider a continuous-time nonlinear system:

$$\dot{x}(t) = f\left(u(t), x(t), p(t)\right), \tag{1}$$

where $x \in \mathbb{R}^{n_x}$, $u \in \mathbb{R}^{n_u}$, and $p \in \mathbb{R}^{n_p}$ are the system state, control input, and given time-dependent parameter, respectively. In NMPC, a cost function over a finite horizon interval $[t, t+T]$ is minimized under the governing of (1) with an initial state $x(t) = \bar{x}_0$, while satisfying a path constraint

$$G\left(u(\tau), x(\tau), p(\tau)\right) \leq 0, \ \ \tau \in [t, t+T]. \tag{2}$$

In solving the FHOCP above, inequality constraint (2) can be transferred approximately into a barrier cost function under the framework of the interior-point method, or alternatively, into a nonlinear algebraic equality constraint with additional slack variables introduced (Ohtsuka, 2004).

We thus consider solving the following discretized FHOCP with only equality constraints:

$$\begin{aligned} \min_{X,U} \ & \sum_{i=1}^{N} L(u_i, x_i, p_i) + \varphi(x_N, p_N) \\ \text{s.t.} \ \ & x_0 = \bar{x}_0, \\ & x_{i-1} + \mathscr{F}(u_i, x_i, p_i) = 0, \quad i \in \{1, \ldots, N\}, \\ & C(u_i, x_i, p_i) = 0, \quad i \in \{1, \ldots, N\}, \end{aligned} \tag{3}$$

where $X = (x_0, x_1, x_2, \ldots, x_N)$ and $U = (u_1, u_2, \ldots, u_N)$ are, respectively, the sequences of states and inputs along the horizon and L, φ, $\mathscr{F}$, and C are twice continuously differentiable functions.

Remark 1. The discretization method for (1) formulated in (3) is herein called the "reverse-time discretization method", which is a class of implicit discretization methods. For example, the backward Euler method with $x_i = x_{i-1} + f(u_i, x_i, p_i)\Delta\tau$ is a reverse-time discretization method with $\mathscr{F}(u_i, x_i, p_i) = f(u_i, x_i, p_i)\Delta\tau - x_i$, where $\Delta\tau := T/N$ is the discretization step size. It corresponds to the forward Euler method performed backward in time. We do not assume any particular discretization method in this paper. Any explicit integration method, e.g., the Runge–Kutta method, can be used backward in time to result in the reverse-time discretization, and the obtained results, including the parallelization and convergence, apply as well. Discretization using the conventional explicit method is discussed in Remark 2.

To solve the NLP (3), we first introduce the Lagrange multiplier sequences $\{\lambda_i \in \mathbb{R}^{n_x}\}_{i=1}^{N}$ and $\{\mu_i \in \mathbb{R}^{n_\mu}\}_{i=1}^{N}$ for the difference equations and the equality constraints, respectively. The sequences of the optimal control input $\{u_i^*\}_{i=1}^{N}$, state $\{x_i^*\}_{i=1}^{N}$, costate $\{\lambda_i^*\}_{i=1}^{N}$, and multiplier $\{\mu_i^*\}_{i=1}^{N}$ satisfy the following nonlinear algebraic equations:

$$\begin{aligned} & x_{i-1}^* + \mathscr{F}(u_i^*, x_i^*, p_i) = 0, \ \ i \in \{1, \ldots, N\}, \\ & C(u_i^*, x_i^*, p_i) = 0, \ \ i \in \{1, \ldots, N\}, \\ & H_u^T(\lambda_i^*, \mu_i^*, u_i^*, x_i^*, p_i) = 0, \ \ i \in \{1, \ldots, N\}, \\ & \lambda_{i+1}^* + H_x^T(\lambda_i^*, \mu_i^*, u_i^*, x_i^*, p_i) = 0, \ \ i \in \{1, \ldots, N-1\}, \\ & \lambda_{i+1}^* + H_x^T(\lambda_i^*, \mu_i^*, u_i^*, x_i^*, p_i) + \varphi_x^T(x_i^*, p_i) = 0, \ \ i = N, \end{aligned} \tag{4}$$

given $x_0^* = \bar{x}_0$ and $\lambda_{N+1}^* = 0$, where H denotes the Hamiltonian defined by

$$H(\lambda, \mu, u, x, p) := L(u, x, p) + \lambda^T \mathscr{F}(u, x, p) + \mu^T C(u, x, p).$$

Here, $H_u := \partial H/\partial u$ and $H_x := \partial H/\partial x$. Note that Eqs. (4) are the discrete-time Euler–Lagrange equations (also called the Karush–Kuhn–Tucker (KKT) conditions), which are the first-order conditions for optimality of problem (3). The nonlinear model predictive controller is implemented at each sampling instant by solving the Euler–Lagrange equations (4) given current state $\bar{x}_0$ and parameter sequence $\{p_i\}_{i=1}^N$, then using u_1^* as the actual input.

Remark 2. Discretization using the reverse-time discretization method minimizes the complexity of coupling between neighboring stages of the Euler–Lagrange equations. Namely, x_{i-1} and λ_{i+1} enter the Euler–Lagrange equations with index i linearly. In the case of explicit discretization, e.g., using the forward Euler method, a similar structure with linear couplings can also be obtained for the corresponding reordered Euler–Lagrange equations (Biegler & Thierry, 2018), and the results of parallelization and convergence can in principle be extended. However, it can be seen from the similar linear coupling structure that explicit discretization will not lead to better computational performance; thus, we herein focus on the reverse-time discretization.

2.2. Notations

Let us denote a vector of unknown variables with subscript i as $\mathcal{V}_i := [\lambda_i^T, \mu_i^T, u_i^T, x_i^T]^T \in \mathbb{R}^n$ and $\mathcal{U}_i(x_{i-1}, \mathcal{V}_i, \lambda_{i+1}) : \mathbb{R}^{n_x} \times \mathbb{R}^n \times \mathbb{R}^{n_x} \to \mathbb{R}^n$ as the left-hand sides of Eqs. (4) with index i (stage i), where $n = 2n_x + n_\mu + n_u$ and $i \in \{1, \dots, N\}$. We define $\mathcal{U} := [\mathcal{U}_1^T, \mathcal{U}_2^T, \dots, \mathcal{U}_N^T]^T$, $\mathcal{V} := [\mathcal{V}_1^T, \mathcal{V}_2^T, \dots, \mathcal{V}_N^T]^T$, $\mathcal{U}_{i:j} := [\mathcal{U}_i^T, \mathcal{U}_{i+1}^T, \dots, \mathcal{U}_j^T]^T$, and $\mathcal{V}_{i:j} := [\mathcal{V}_i^T, \mathcal{V}_{i+1}^T, \dots, \mathcal{V}_j^T]^T$. Then $\mathcal{U}(\mathcal{V}) = 0$ is the compact form of Eqs. (4), with the solution denoted by $\mathcal{V}^*$. For a variable v, v^k stands for the value of v at the kth iteration. For simplicity, we denote $J(\mathcal{V}) := \mathcal{U}'(\mathcal{V})$ as the Jacobian matrix of $\mathcal{U}$ with respect to $\mathcal{V}$, $\mathcal{U}_{i:j}^k := \mathcal{U}_{i:j}(x_{i-1}^k, \mathcal{V}_{i:j}^k, \lambda_{j+1}^k)$, $J_{i:j}^k := \frac{\partial \mathcal{U}_{i:j}}{\partial \mathcal{V}_{i:j}}|_{(x_{i-1}^k, \mathcal{V}_{i:j}^k, \lambda_{j+1}^k)}$, $\mathcal{U}_i^k := \mathcal{U}_{i:i}^k$, and $J_i^k := J_{i:i}^k$. For a matrix $A \in \mathbb{R}^{m\times n}$, we denote by $A[\alpha, \beta]$ the submatrix of A consisting of rows α and columns β. We define $[A]_k^U := A[1:k, 1:k]$ as the kth leading principal submatrix of A and define $[A]_k^L := A[m-k+1:m, n-k+1:n]$. The symbol $\|\cdot\|$ denotes the Euclidean norm for a vector and the p-norm for a matrix, where p can be any positive integer.

3. Backward correction method

Newton's method for solving Eqs. (4) is difficult to parallelize due to the coupling between stages. In this section, we will take a look at the iteration of each stage and investigate the coupling between stages. We formulate the backward correction method and prove that it is exactly identical to Newton's method.

3.1. Motivation

Newton's method is practical and powerful for solving the nonlinear algebraic equation $\mathcal{U}(\mathcal{V}) = 0$ as it is simple to implement and converges quadratically in general. The full-step Newton's method performs the following iteration starting from an initial guess $\mathcal{V}^0$ that is sufficiently close to $\mathcal{V}^*$:

$$\mathcal{V}^{k+1} = \mathcal{V}^k - J(\mathcal{V}^k)^{-1}\mathcal{U}(\mathcal{V}^k), \quad k = 0, 1, \dots.$$

However, performing Newton's iteration is computationally expensive because one has to first evaluate $\mathcal{U}(\mathcal{V}^k)$ and its Jacobian $J(\mathcal{V}^k)$ and then solve a linear equation. The computational complexity of computing $J(\mathcal{V}^k)^{-1}\mathcal{U}(\mathcal{V}^k)$, in terms of N, can be made either $\mathcal{O}(N^2)$ by condensing (Frison & Jorgensen, 2013), or $\mathcal{O}(N)$ by exploiting the banded structure of $J(\mathcal{V})$, as proposed in Rao et al. (1998). Due to the coupling between stages ($J(\mathcal{V}^k)$ is not block-diagonal), parallelization is challenging. Although methods using parallel Riccati recursion (Frasch et al., 2015; Nielsen & Axehill, 2015) can have complexities of $\mathcal{O}(\log(N))$, they are still computationally expensive. Note that, thanks to the reverse-time discretization of the state equation, the Euler–Lagrange equations (4) at stage i are coupled with the neighboring stages linearly, that is, x_{i-1} and λ_{i+1} enter $\mathcal{U}_i(x_{i-1}, \mathcal{V}_i, \lambda_{i+1}) = 0$ linearly. Consequently, equation $\mathcal{U}_i(x_{i-1}, \mathcal{V}_i, \lambda_{i+1}) = 0$ can be formulated as the necessary condition for a single-period OCP with initial state x_{i-1} and terminal penalty function $\varphi(x_i, p_i) = \lambda_{i+1}^T x_i$. Therefore, solving $\mathcal{U}(\mathcal{V}) = 0$ can be parallelized by solving a series of equations $\mathcal{U}_i(x_{i-1}^*, \mathcal{V}_i, \lambda_{i+1}^*) = 0$ with respect to $\mathcal{V}_i$ for $i \in \{1, \dots, N\}$ if the coupling variables x_{i-1}^* and λ_{i+1}^* are given for each stage i in advance. It should be noted that this resembles the principle of optimality, which states that any part of the optimal trajectory itself must be optimal. Unfortunately, x_{i-1}^* and λ_{i+1}^* cannot be known in advance, and only suboptimal values are given in general. When iterative methods are used for convergence, the iterations at the stage level should be investigated in order to parallelize them.

Consider the following Newton's iteration for solving $\mathcal{U}_i(\tilde{x}_{i-1}, \mathcal{V}_i, \tilde{\lambda}_{i+1}) = 0$ approximately:

$$\mathcal{V}_i^{k+1} = \mathcal{V}_i^k - \left(\frac{\partial \mathcal{U}_i}{\partial \mathcal{V}_i}\bigg|_{(\tilde{x}_{i-1}, \mathcal{V}_i^k, \tilde{\lambda}_{i+1})}\right)^{-1} \mathcal{U}_i(\tilde{x}_{i-1}, \mathcal{V}_i^k, \tilde{\lambda}_{i+1}), \tag{5}$$

where $\tilde{x}_{i-1}$ and $\tilde{\lambda}_{i+1}$ are the estimation of x_{i-1}^* and λ_{i+1}^*, respectively. One of the simplest methods takes $\tilde{x}_{i-1} = x_{i-1}^k$ and $\tilde{\lambda}_{i+1} = \lambda_{i+1}^k$. This method has complexity $\mathcal{O}(n^3)$ in parallel but it might diverge. The Gauss–Seidel scheme performs (5) recursively from $i = 1$ to N, with $\tilde{x}_{i-1} = x_{i-1}^{k+1}$ and $\tilde{\lambda}_{i+1}$, for example, estimated with a coarse-grained high-level controller in advance, as proposed in Zavala (2016). The convergence of the Gauss–Seidel scheme cannot be guaranteed either, unless λ_{i+1}^* is well estimated.

Next, we present a new method that estimates λ_{i+1}^* on the basis of the current kth iteration's information. In the Gauss–Seidel method, stages $2, \dots, N$ are updated after the update of stage 1, that is, $\mathcal{V}_{2:N}^{k+1}$ is a function of x_1:

$$\mathcal{V}_{2:N}^{k+1}(x_1) = \mathcal{V}_{2:N}^k - \left(\frac{\partial \mathcal{U}_{2:N}}{\partial \mathcal{V}_{2:N}}\bigg|_{(x_1, \mathcal{V}_{2:N}^k, \lambda_{N+1})}\right)^{-1} \mathcal{U}_{2:N}(x_1, \mathcal{V}_{2:N}^k, \lambda_{N+1}). \tag{6}$$

As x_1 enters $\mathcal{U}_{2:N}$ linearly, the following equation holds:

$$\mathcal{U}_{2:N}(x_1, \mathcal{V}_{2:N}^k, \lambda_{N+1}) = \mathcal{U}_{2:N}(x_1^k, \mathcal{V}_{2:N}^k, \lambda_{N+1}) + \begin{bmatrix}(x_1 - x_1^k)^T & 0^T\end{bmatrix}^T.$$

Therefore, the Jacobian matrix of $\mathcal{U}_{2:N}$ with respect to $\mathcal{V}_{2:N}$ does not depend on x_1, and the update of λ_2 can be extracted from the first n_x elements of (6) and expressed as

$$\lambda_2^{k+1}(x_1) = \lambda_2^k - \Lambda_2^k(x_1 - x_1^k) - d_{\lambda_2}^k, \tag{7}$$

where $\Lambda_2^k = \left[\left(J_{2:N}^k\right)^{-1}\right]_{n_x}^U \in \mathbb{R}^{n_x \times n_x}$ and $d_{\lambda_2}^k = \left(\left(J_{2:N}^k\right)^{-1}\right)[1:n_x, :] \cdot \mathcal{U}_{2:N}^k \in \mathbb{R}^{n_x}$. Although λ_2^* cannot be known in advance, it can be regarded as the correction of λ_2^k by (7) so that the iteration of

solving $\mathcal{U}_1(x_0, \mathcal{V}_1, \lambda_2^{k+1}(x_1)) = 0$ with respect to $\mathcal{V}_1$ is given by

$$\begin{aligned}\mathcal{V}_1^{k+1} &= \mathcal{V}_1^k - \left(\left.\frac{\partial \mathcal{U}_1}{\partial \mathcal{V}_1}\right|_{(\bar{x}_0, \mathcal{V}_1^k, \lambda_2^{k+1}(x_1^k))}\right)^{-1} \mathcal{U}_1(\bar{x}_0, \mathcal{V}_1^k, \lambda_2^{k+1}(x_1^k)) \\ &= \mathcal{V}_1^k - \left(J_1^k - \begin{bmatrix} 0 & 0 \\ 0 & \Lambda_2^k \end{bmatrix}\right)^{-1} \left(\mathcal{U}_1^k - \begin{bmatrix} 0 \\ d_{\lambda_2}^k \end{bmatrix}\right).\end{aligned}$$

After x_1^{k+1} is obtained, stages $2, \ldots, N$ can be further split and iterated recursively, which results in the algorithm described in the next subsection.

3.2. Algorithm

As stage N is the last stage and cannot be further split, the update of λ_N as a function of x_{N-1} can be directly formulated. After $\lambda_N^{k+1}(x_{N-1})$ is obtained, $\lambda_{N-1}^{k+1}(x_{N-2})$ can be formulated. Recursively, the update of λ_i, i.e., $\lambda_i^{k+1}(x_{i-1})$, is formulated on the basis of Λ_i^k and $d_{\lambda_i}^k$ from $i = N$ to 1. Then, $\mathcal{V}_i$ is updated from $i = 1$ to N following the Gauss–Seidel scheme. The resulting method is called the backward correction method (Algorithm 1), which has a computational complexity of $\mathcal{O}(Nn^3)$.

Algorithm 1 k-th iteration of backward correction method

Input: $\mathcal{V}^k$, $x_0^k = x_0^{k+1} = \bar{x}_0$, $\lambda_{N+1}^k = 0$, $\Lambda_{N+1}^k = 0$, and $d_{\lambda_{N+1}}^k = 0$.
Output: $\mathcal{V}^{k+1}$.
for $i = N$ to 1 **do**

$$H_i^k \leftarrow \left(\left.\frac{\partial \mathcal{U}_i}{\partial \mathcal{V}_i}\right|_{\left(x_{i-1}^k, \mathcal{V}_i^k, \lambda_{i+1}^{k+1}(x_i^k)\right)}\right)^{-1}, \tag{8}$$

where $\mathcal{U}_i = \mathcal{U}_i\left(x_{i-1}, \mathcal{V}_i, \lambda_{i+1}^{k+1}(x_i)\right)$, (9)

and $\lambda_{i+1}^{k+1}(x_i) = \lambda_{i+1}^k - \Lambda_{i+1}^k(x_i - x_i^k) - d_{\lambda_{i+1}}^k$. (10)

$$\Lambda_i^k \leftarrow \left[H_i^k\right]_{n_x}^U. \tag{11}$$

$$d_{\lambda_i}^k \leftarrow H_i^k[1 : n_x, :] \cdot \mathcal{U}_i\left(x_{i-1}^k, \mathcal{V}_i^k, \lambda_{i+1}^{k+1}(x_i^k)\right). \tag{12}$$

end for
for $i = 1$ to N **do**

$$\mathcal{V}_i^{k+1} \leftarrow \mathcal{V}_i^k - H_i^k \cdot \mathcal{U}_i\left(x_{i-1}^{k+1}, \mathcal{V}_i^k, \lambda_{i+1}^{k+1}(x_i^k)\right). \tag{13}$$

end for

Next, we show that the backward correction method results in exactly the same update as Newton's method.

Theorem 1. *The backward correction method in Algorithm 1 is identical to Newton's method, that is, starting from the same $\mathcal{V}^k$ at the kth iteration, the updated value obtained using Newton's method denoted by $\mathcal{V}_{(nt)}^{k+1}$ is equal to the value obtained using the backward correction method denoted by $\mathcal{V}_{(bc)}^{k+1}$.*

The proof of Theorem 1 is provided in Appendix A.

The backward correction method is one of the methods exploiting the banded structure of $J(\mathcal{V})$ using block Gauss elimination. However, discretization of the system dynamics using the reverse-time method leads to linear couplings between neighboring stages in the Euler–Lagrange Equations (4), which makes the recursion in the backward correction method extremely simple compared with that in other structure-exploiting methods (Glad & Jonson, 1984; Jørgensen et al., 2004; Rao et al., 1998).

4. Proposed method

The backward correction method is inherently non-parallelizable due to the recursion for calculating H_i^k in (8) from $i = N$ to 1 in order to formulate the expression of $\lambda_{i+1}^{k+1}(x_i)$ in (10) for each stage, which is the most computationally expensive part. Although the iterations are sequential, the iteration of each stage is clearly shown in (13). We show next that the backward correction method can be parallelized by using a reasonable approximation.

Recall that, in the iteration (5) for solving the Euler–Lagrange equations (4), its convergence depends on the estimation of the coupling variables x_{i-1}^* and λ_{i+1}^* for each stage i. The estimation of λ_{i+1}^* in the backward correction method is performed using (10) and can be seen as an accurate estimation of λ_{i+1}^*. Consequently, a quadratic rate of convergence can be obtained. A slower rate of convergence can be achieved by using a relatively coarse estimation of λ_{i+1}^*.

To visualize the small variations of $\{\Lambda_i^k\}_{i=1}^N$ as k increases, let us consider the situation in which the cost function $L(u, x, p)$ is quadratic, $\mathscr{F}(u, x, p)$ and the constraint $C(u, x, p)$ are linear, and the time-varying parameter p is fixed. Under these conditions, the Jacobian matrix of $\mathcal{U}$ with respect to $\mathcal{V}$ is constant, which means that $\{\Lambda_i^k\}_{i=1}^N$ in (11) are always constant during iteration. As for general problems, we can expect small variations of $\{\Lambda_i^k\}_{i=1}^N$ as k increases when the problem is less nonlinear and p varies slowly. Therefore, we propose estimating λ_{i+1}^* on the basis of Λ_{i+1}^{k-1} at the kth iteration. That is, (10) in the backward correction method is replaced by

$$\lambda_{i+1}^{k+1}(x_i) = \lambda_{i+1}^k - \Lambda_{i+1}^{k-1}(x_i - x_i^k) - d_{\lambda_{i+1}}^k, \ i \in \{1, \ldots, N\}. \tag{14}$$

This removes the recursion for calculating H_i^k from $i = N$ to 1, so $\{H_i^k\}_{i=1}^N$ can be calculated in parallel using

$$H_i^k = \left(J_i^k - \begin{bmatrix} 0 & 0 \\ 0 & \Lambda_{i+1}^{k-1} \end{bmatrix}\right)^{-1}. \tag{15}$$

4.1. Parallelized implementation

Calculating $\{H_i^k\}_{i=1}^N$ in parallel leads to an algorithm with complexity $\mathcal{O}(n^3 + Nn^2)$ for N threads, where n^3 is the complexity of the matrix inverse in (15), and Nn^2 indicates the complexities of carrying out (12) and (13) backward and forward, respectively.

Moreover, further parallelization is achieved by arranging the order of computations as shown in Algorithm 2. After $\{H_i^k\}_{i=1}^N$ is obtained, an initial coarse update is done using $\mathcal{V}_i^{k+1} = \mathcal{V}_i^k - H_i^k \cdot \mathcal{U}_i^k$ (Step 1 of Algorithm 2). This coarse update introduces approximations for coupling variables (x_{i-1} and λ_{i+1}) compared with (13). The correction due to the approximation of λ_{i+1} is conducted in a backward manner (Step 2 of Algorithm 2). In the sequential part of Step 2, the approximation error $d_{\lambda_{i+1}}$ of λ_{i+1} is calculated for all stages, and then the other variables are corrected on the basis of the approximation error in the parallel part. Likewise, the correction due to the approximation of x_{i-1} is conducted in a forward manner (Step 3 of Algorithm 2). As a result of these two correction steps, the further parallelized implementation produces exactly the same update as the proposed method and reduces the complexity to $\mathcal{O}(n^3 + Nn_x^2)$ for N threads.

4.2. Convergence

In this subsection, the rate of convergence is shown for the proposed method by measuring the distance to the backward correction method.

Algorithm 2 Proposed parallelized implementation of the backward correction method (k-th iteration)

Input: $\mathcal{V}^k$, $\{\Lambda_i^{k-1}\}_{i=1}^N$, $x_0^k = \bar{x}_0$, $\lambda_{N+1}^k = 0$, $\Lambda_{N+1}^{k-1} = 0$.
Output: $\mathcal{V}^{k+1}$ and $\{\Lambda_i^k\}_{i=1}^N$.
Step 1. Coarse update:
for $i = 1$ to N **do in parallel**
$\mathcal{U}_i^k \leftarrow \mathcal{U}_i(x_{i-1}^k, \mathcal{V}_i^k, \lambda_{i+1}^k)$.
$J_i^k \leftarrow \frac{\partial \mathcal{U}_i}{\partial \mathcal{V}_i}|_{(x_{i-1}^k, \mathcal{V}_i^k, \lambda_{i+1}^k)}$.
$H_i^k \leftarrow \left(J_i^k - \begin{bmatrix} 0 & 0 \\ 0 & \Lambda_{i+1}^{k-1} \end{bmatrix}\right)^{-1}$.
$\Lambda_i^k \leftarrow \left[H_i^k\right]_{n_x}^U$.
$\mathcal{V}_i^{k+1} \leftarrow \mathcal{V}_i^k - H_i^k \cdot \mathcal{U}_i^k$.
end for
Step 2. Backward correction due to approximation of λ:
for $i = N-1$ to 1 **do**
$d_{\lambda_{i+1}}^{k+1} \leftarrow \lambda_{i+1}^{k+1} - \lambda_{i+1}^k$.
$\lambda_i^{k+1} \leftarrow \lambda_i^{k+1} - H_i^k[1:n_x, n-n_x+1:n] \cdot d_{\lambda_{i+1}}^{k+1}$.
end for
for $i = 1$ to $N-1$ **do in parallel**
$\mathcal{V}_i^{k+1}[n_x+1:n, :] \leftarrow \mathcal{V}_i^{k+1}[n_x+1:n, :] - H_i^k[n_x+1:n, n-n_x+1:n] \cdot d_{\lambda_{i+1}}^{k+1}$.
end for
Step 3. Forward correction due to approximation of x:
for $i = 2$ to N **do**
$d_{x_{i-1}}^{k+1} \leftarrow x_{i-1}^{k+1} - x_{i-1}^k$.
$x_i^{k+1} \leftarrow x_i^{k+1} - H_i^k[n-n_x+1:n, 1:n_x] \cdot d_{x_{i-1}}^{k+1}$.
end for
for $i = 2$ to N **do in parallel**
$\mathcal{V}_i^{k+1}[1:n-n_x, :] \leftarrow \mathcal{V}_i^{k+1}[1:n-n_x, :] - H_i^k[1:n-n_x, 1:n_x] \cdot d_{x_{i-1}}^{k+1}$.
end for

At the kth iteration, the proposed method differs from the backward correction method only in how $\{\lambda_{i+1}^{k+1}\}_{i=1}^N$ are formulated: $\lambda_{i+1}^{k+1}(x_i) = \lambda_{i+1}^k - \Lambda_{i+1}^k(x_i - x_i^k) - d_{\lambda_{i+1}}^k$ for the backward correction method, $\lambda_{i+1}^{k+1}(x_i) = \lambda_{i+1}^k - \Lambda_{i+1}^{k-1}(x_i - x_i^k) - d_{\lambda_{i+1}}^k$ for the proposed method. To distinguish these two methods, let $v_{(bc)}$ be the variable in the backward correction method for a variable v. It can be seen that the proposed method approximates the backward correction method, where the approximation is introduced by

$$h_i^k := \Theta_i^{k-1} - \Theta_{i(bc)}^k, \ i \in \{1, \dots, N\}, \tag{16}$$

where

$$\Theta_{i(bc)}^k := \begin{bmatrix} 0 & 0 \\ 0 & \Lambda_{i(bc)}^k \end{bmatrix} \in \mathbb{R}^{n \times n}$$

and

$$\Theta_i^{k-1} := \begin{bmatrix} 0 & 0 \\ 0 & \Lambda_i^{k-1} \end{bmatrix} \in \mathbb{R}^{n \times n}.$$

Theorem 1 has shown that the backward correction method is equivalent to Newton's method, so the iteration can be written as

$$\mathcal{V}_{(bc)}^{k+1} = \mathcal{V}^k - H(\mathcal{V}^k) \cdot \mathcal{U}(\mathcal{V}^k),$$

where $H(\mathcal{V}^k) := J(\mathcal{V}^k)^{-1}$. Let $\{D_{h_i}\}_{i=1}^N$ and $D_{\mathcal{V}}$ be the open subsets of $\mathbb{R}^{n \times n}$ and $\mathbb{R}^{nN}$, respectively. Denote $h = (h_1, h_2, \dots, h_N) \in D_{h_1} \times D_{h_2} \times \cdots \times D_{h_N} =: D_h$. Assume that $0 \in D_h$ and $\mathcal{V}^* \in D_{\mathcal{V}}$.

The kth iteration of the proposed method is expressed as

$$\mathcal{V}^{k+1} = \mathcal{V}^k - G(\mathcal{V}^k, h^k),$$

where $G : D_{\mathcal{V}} \times D_h \to \mathbb{R}^{nN}$ is a mapping that defines the increment with the proposed method. We have the following assumptions:

Assumption 1. The solution $\mathcal{V}^*$ is an isolated solution to $\mathcal{U}(\mathcal{V}) = 0$.

Note that Assumption 1 guarantees the nonsingularities for not only the Jacobian matrix $J_{1:N}$ but the Jacobian matrices $\{J_{i:N}\}_{i=2}^N$ in a neighborhood of $\mathcal{V}^*$.

Assumption 2. The function $\mathcal{U} : \mathbb{R}^{nN} \to \mathbb{R}^{nN}$ is Lipschitz continuous on $D_{\mathcal{V}}$ with Lipschitz constant L. The norm $\|J(\mathcal{V})^{-1}\|$ is bounded on $D_{\mathcal{V}}$.

Recall that the proposed method has the same algorithm as the backward correction method, except that λ_{i+1}^* is estimated by (14) instead of (10). Therefore, in the following proof, we refer to the equations in Algorithm 1 to show the convergence of Algorithm 2 for convenience.

The proof is structured as follows. Lemma 3 shows that $\lim_{k\to\infty} h_i^k = 0$ for all $i \in \{1, \dots, N\}$ when the iteration converges. This implies that the proposed method approaches the backward correction method when it converges. To further show this, the explicit expression of $G(\mathcal{V}, h)$ is given in Lemma 4 by $G(\mathcal{V}, h) = P(\mathcal{V}, h) \cdot \mathcal{U}(\mathcal{V})$, and the distance between $P(\mathcal{V}, h)^{-1}$ and $J(\mathcal{V})$ is shown to be arbitrarily boundable in Lemma 4. Finally, it is shown in Theorem 2 that if $\mathcal{V}^0$ is chosen close to $\mathcal{V}^*$ and h^0 close to 0, the convergence of the proposed method can be guaranteed. Moreover, since the proposed method approaches the backward correction method, which converges quadratically, the rate of convergence is shown to be superlinear in Theorem 2.

Lemma 1 (*Ortega & Rheinboldt, 1970*). *Let $A \in \mathbb{R}^{n\times n}$ be nonsingular, $E \in \mathbb{R}^{n\times n}$, and $\|A^{-1}E\| < 1$. Then*

$$\|(A+E)^{-1}\| \le \frac{\|A^{-1}\|}{1 - \|A^{-1}E\|}.$$

Lemma 2 (*Ortega & Rheinboldt, 1970*). *Let $A \in \mathbb{R}^{n\times n}$ be nonsingular, $E \in \mathbb{R}^{n\times n}$, and $\|A^{-1}E\| < 1$. Then*

$$\|(A+E)^{-1} - A^{-1}\| \le \|A^{-1}\| \frac{\|A^{-1}E\|}{1 - \|A^{-1}E\|}.$$

Lemma 3. *If the iterates $\{\mathcal{V}^k\}$ given by the proposed method converge to $\mathcal{V}^*$, then $h^k \to 0$ ($k \to \infty$).*

Proof. See Appendix B.

Lemma 4. *$G(\mathcal{V}, h)$ can be expressed as $G(\mathcal{V}, h) = P(\mathcal{V}, h) \cdot \mathcal{U}(\mathcal{V})$, where $P(\mathcal{V}, h) : D_{\mathcal{V}} \times D_h \to \mathbb{R}^{nN\times nN}$. Moreover, $\lim_{h\to 0} P(\mathcal{V}, h)^{-1} = J(\mathcal{V})$ holds for all $\mathcal{V} \in D_{\mathcal{V}}$.*

Proof. See Appendix C.

Theorem 2. *There are balls $S_{\mathcal{V}} := \{\mathcal{V} | \|\mathcal{V} - \mathcal{V}^*\| < \delta_{\mathcal{V}}\} \subset D_{\mathcal{V}}$ and $S_h := \{h | \|h_i\| < \delta_h, \ i \in \{1, \dots, N\}\} \subset D_h$ such that, for any $\mathcal{V}^0 \in S_{\mathcal{V}}$ and $h^0 \in S_h$, the iterates $\{\mathcal{V}^k\}$ given by the proposed method remain in $S_{\mathcal{V}}$ and converge to $\mathcal{V}^*$ superlinearly.*

Proof. Set $\beta = \|H(\mathcal{V}^*)\|$. For any $\epsilon \in (0, \frac{1}{2}\beta^{-1})$, there exists $\delta_1 > 0$ such that, if $\mathcal{V}^k \in S_1 := \{\mathcal{V} | \|\mathcal{V} - \mathcal{V}^*\| < \delta_1\} \subset D_{\mathcal{V}}$, then

$$\|J(\mathcal{V}^k) - J(\mathcal{V}^*)\| < \frac{1}{2}\epsilon \tag{17}$$

and

$$\|\mathcal{U}(\mathcal{V}^k) - \mathcal{U}(\mathcal{V}^*) - J(\mathcal{V}^*)(\mathcal{V}^k - \mathcal{V}^*)\| < L\|\mathcal{V}^k - \mathcal{V}^*\|^2 \tag{18}$$

hold by the continuity of J at $\mathcal{V}^*$ and the Lipschitz continuity in Assumption 2, respectively. For such ϵ, from Lemma 4, we can choose $\delta_2 > 0$ such that $h^k \in S_2 := \{h | \|h_i\| < \delta_2,\ i \in \{1, \ldots, N\}\} \subset D_h$ and

$$\|P(\mathcal{V}^k, h^k)^{-1} - J(\mathcal{V}^k)\| < \frac{1}{2}\epsilon. \tag{19}$$

From (17) and (19), we have

$$\|P(\mathcal{V}^k, h^k)^{-1} - J(\mathcal{V}^*)\| < \epsilon.$$

Since $\beta = \|H(\mathcal{V}^*)\| = \|J(\mathcal{V}^*)^{-1}\|$, then $\|P(\mathcal{V}^k, h^k)\|$ can be bounded using Lemma 1 ($A = J(\mathcal{V}^*)$, $E = P(\mathcal{V}^k, h^k)^{-1} - J(\mathcal{V}^*)$, $\|A^{-1}E\| \le \|A^{-1}\|\|E\| < \beta\epsilon < 1/2$) by

$$\|P(\mathcal{V}^k, h^k)\| \le \frac{\beta}{1 - \beta\epsilon} < 2\beta. \tag{20}$$

From (18) and (20),

$$\begin{aligned}
&\|\mathcal{V}^{k+1} - \mathcal{V}^*\| = \|\mathcal{V}^k - P(\mathcal{V}^k, h^k)\mathcal{U}(\mathcal{V}^k) - \mathcal{V}^*\| \\
&= \|P(\mathcal{V}^k, h^k)[P(\mathcal{V}^k, h^k)^{-1}(\mathcal{V}^k - \mathcal{V}^*) - \mathcal{U}(\mathcal{V}^k)]\| \\
&\le 2\beta \left(\|P(\mathcal{V}^k, h^k)^{-1} - J(\mathcal{V}^k)\| + \|J(\mathcal{V}^k) - J(\mathcal{V}^*)\|\right) \|\mathcal{V}^k - \mathcal{V}^*\| \\
&\quad + 2\beta\|\mathcal{U}(\mathcal{V}^k) - \mathcal{U}(\mathcal{V}^*) - J(\mathcal{V}^*)(\mathcal{V}^k - \mathcal{V}^*)\| \\
&< \omega(\mathcal{V}^k, h^k)\|\mathcal{V}^k - \mathcal{V}^*\|,
\end{aligned}$$

where

$$\begin{aligned}
\omega(\mathcal{V}^k, h^k) = 2\beta \big(&\|P(\mathcal{V}^k, h^k)^{-1} - J(\mathcal{V}^k)\| \\
&+ \|J(\mathcal{V}^k) - J(\mathcal{V}^*)\| + L\|\mathcal{V}^k - \mathcal{V}^*\|\big)
\end{aligned}$$

is the upper bound on the rate of convergence. The result of Lemma 4 ensures that $\|P(\mathcal{V}^k, h^k)^{-1} - J(\mathcal{V}^k)\| \to 0$ as $h^k \to 0$, and, due to the continuity of J at $\mathcal{V}^*$, $\|J(\mathcal{V}^k) - J(\mathcal{V}^*)\| \to 0$ as $\mathcal{V}^k \to \mathcal{V}^*$. Therefore, there exist balls $S_\mathcal{V} := \{\mathcal{V} | \|\mathcal{V} - \mathcal{V}^*\| < \delta_\mathcal{V}\} \subset S_1 \cap S_3$ and $S_h := \{h | \|h_i\| < \delta_h,\ i \in \{1, \ldots, N\}\} \subset S_2$ such that, for any $\mathcal{V}^k \in S_\mathcal{V}$ and $h^k \in S_h$, $\omega(\mathcal{V}^k, h^k) < 1$ holds, where S_3 is a subset of $D_\mathcal{V}$ such that, if $\mathcal{V}^k$ is in S_3, then $h^k \in S_h$ always holds from Lemma 3. Therefore, if $\mathcal{V}^0 \in S_\mathcal{V}$ and $h^0 \in S_h$, then the iterates $\{\mathcal{V}^k\}$ converge and remain in $S_\mathcal{V}$. Moreover, since $\mathcal{V}^k \to \mathcal{V}^*$ as $k \to \infty$, we have $h^k \to 0$ as $k \to \infty$ from Lemma 3. Therefore, $\omega(\mathcal{V}^k, h^k) \to 0$ as $k \to \infty$, which indicates that the proposed method has a superlinear rate of convergence. □

Remark 3. Theorem 2 gives only conditions of convergence for the update of only one sampling instant. However, the convergence of the proposed method should be guaranteed for every sampling instant. When the iteration at one sampling instant converges, a sufficiently large number of iterations k can guarantee sufficiently small $\|h_i^k\|$ for all $i \in \{1, \ldots, N\}$ from Lemma 3. Moreover, if the time-dependent parameter p varies smoothly with respect to time, a small sampling interval can ensure that the new state $\bar{x}_0$ and the time-dependent parameter p are close to their previous values even if there are model mismatches or disturbances in the actual system, which implies that the initial guess $\mathcal{V}^0$ is close to the optimal value $\mathcal{V}^*$ when warm-start is used. Therefore, the conditions in Theorem 2 can always be satisfied in the next sampling instant.

5. Numerical experiment

5.1. NMPC for a quadrotor

In order to demonstrate the computation time and rate of convergence of the proposed method, reference tracking of a quadrotor is considered. The state vector of the quadrotor is $x = [X, \dot{X}, Y, \dot{Y}, Z, \dot{Z}, \gamma, \beta, \alpha]^T \in \mathbb{R}^9$, where (X, Y, Z) and (γ, β, α) are the position and angles of the quadrotor, respectively. The input vector is $u = [a, \omega_X, \omega_Y, \omega_Z]^T$, where a represents the thrust and $(\omega_X, \omega_Y, \omega_Z)$ the rotational rates. We use a previously defined nonlinear model (Hehn & D'Andrea, 2011):

$$\begin{aligned}
\ddot{X} &= a(\cos\gamma \sin\beta \cos\alpha + \sin\gamma \sin\alpha) \\
\ddot{Y} &= a(\cos\gamma \sin\beta \sin\alpha - \sin\gamma \cos\alpha) \\
\ddot{Z} &= a\cos\gamma \cos\beta - g \\
\dot{\gamma} &= (\omega_X \cos\gamma + \omega_Y \sin\gamma)/\cos\beta \\
\dot{\beta} &= -\omega_X \sin\gamma + \omega_Y \cos\gamma \\
\dot{\alpha} &= \omega_X \cos\gamma \tan\beta + \omega_Y \sin\gamma \tan\beta + \omega_Z.
\end{aligned}$$

The control input is bounded with $[0, -1, -1, -1]^T \le u \le [11, 1, 1, 1]^T$. The system starts from the initial state $x_0 = [1, 0, 1, 0, 1, 0, 0, 0, 0]^T$. The state reference is set using the time-varying parameter p with $x_{ref} = 0$ from 0 to 10 s and $x_{ref} = [1.5, 0, 1.5, 0, 1.5, 0, 0, 0, 0]^T$ from 10 to 20 s, and $u_{ref} = [g, 0, 0, 0]^T$. The stage cost function is $L(u, x, p) = \frac{1}{2}(\|x - x_{ref}\|_Q^2 + \|u - u_{ref}\|_R^2)$ with $Q = \mathrm{diag}(10, 1, 2, 1, 10, 1, 1, 1, 1)$ and $R = I_4$, and the terminal cost function $\varphi(x, p)$ is not imposed. The prediction horizon is $T = 1$ s and is divided into $N = 8, 16, 24, 48, 96$ steps for comparison. Simulation is performed for 20 s with a sampling interval of 0.01 s.

5.2. Computation time

The proposed method was implemented using the open-source toolkit ParNMPC (Deng, 2018), which can automatically generate parallel C/C++ code with OpenMP (Dagum & Menon, 1998). For comparison, the code generation tool (Houska et al., 2011b) in the ACADO Toolkit (Houska, Ferreau, & Diehl, 2011a) and the AutoGenU (Ohtsuka, 2015) code generation toolkit are used to perform closed-loop NMPC simulation. ACADO Code Generation is based on the SQP method with Gauss–Newton Hessian approximation (GNHA); the qpOASES (Ferreau, Kirches, Potschka, Bock, & Diehl, 2014) dense QP solver and the qpDUNES (Frasch et al., 2015) sparse QP solver are used respectively to solve the underlying QP problems. AutoGenU is based on the C/GMRES method. The simulation was performed on a desktop computer with dual 2.2 GHz Intel Xeon E5-2650 V4 processors with 24 cores in total with the Hyper-Threading and Turbo Boost features are disabled.

In principle, ACADO Code Generation is based on the real-time iteration (RTI) scheme, in which only one SQP iteration is performed. The KKT tolerance is controlled by performing several SQP iterations, and the computation time is the sum of the CPU times returned by the solver. The KKT tolerances of the proposed method and the SQP methods are set to $5 \cdot 10^{-3}$, while in the C/GMRES method, only one iteration is performed per update. It should be noted that these methods do not converge to the same solution even when the same KKT tolerances are achieved because of the differences in handling inequality constraints. Namely, the SQP methods converge to the optimal solution of the inequality constrained optimization problem while the proposed method and C/GMRES only approximately take the inequality constraints into account by introducing dummy inputs (Ohtsuka, 2004). The resulting suboptimality is defined as

$$e_o := \frac{\|U^*_{dummy} - U^*_{ieq}\|}{\|U^*_{ieq}\|},$$

where U^*_{dummy} and U^*_{ieq} denote the solutions of the control inputs sequences obtained by introducing dummy inputs and by dealing directly with inequality constraints, respectively.

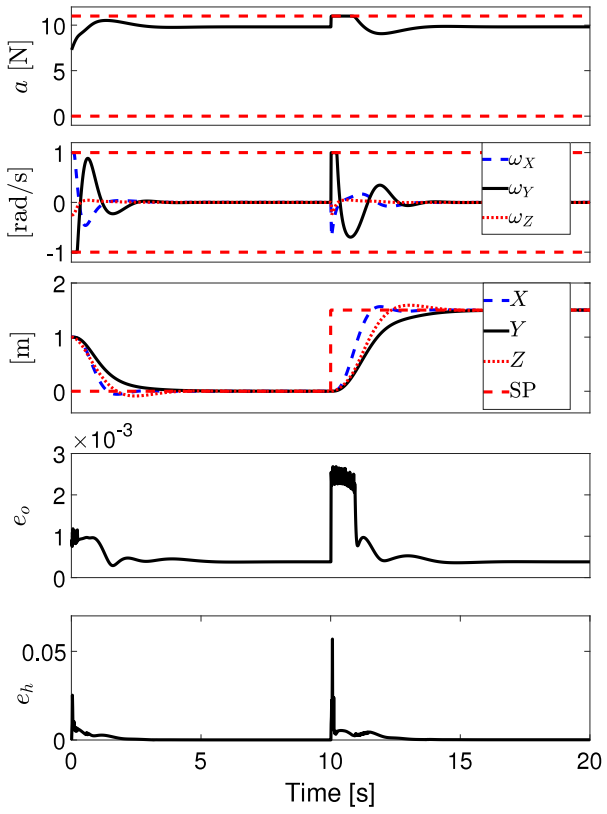


Fig. 1. Time histories of simulation results with proposed method (SP is position reference).

The discretization within the SQP methods is based on the explicit Euler method with multiple shooting. The QP problems solved by qpOASES are condensed, and warm-start is enabled. Although SQP methods can, in principle, be parallelized to a certain degree, e.g., for multiple shooting and the qpDUNES QP solver, they are run in a fully sequential fashion. The explicit Euler method is used for the C/GMRES method, and the number of iterations in GMRES is set to 7. The proposed method is based on the implicit Euler method.

The approximation to the backward correction method is defined in (16). Let us denote the relative approximation error by e_h, which measures the distance to Newton's method:

$$e_h := \max_{i\in\{1,\dots,N\}} \frac{\|h_i\|}{\|\Theta_{i(bc)}\|}.$$

The simulation results for the inputs, position, suboptimality e_o, and relative approximation error e_h after each update with the proposed method for $N = 24$ are shown in Fig. 1. The results obtained by introducing dummy inputs are almost the same as those with the SQP methods, i.e., active input bound constraints can be achieved when tracking the position reference. Since the degree of parallelism of the proposed method is N, the program can be run on $\min\{24, N\}$ cores concurrently. Table 1 lists the average computation time and number of iterations per update. Although the computation time of the proposed method is long when running on one core in a fully sequential fashion, it becomes faster than other methods in the medium and high range of N when running on multiple cores.

As the computation time of the proposed method is in the μs range, the overhead time is non-negligible. Let us denote t_E as the average time per update running on $\min\{24, N\}$ cores in practice and t_p and t_s as the times of the parallelizable and sequential parts running on one core, respectively. The theoretical

Table 1
Average computation time [μs] (upper) and number of iterations (lower) per update.

N	8	16	24	48	96
Proposed method on $\min\{24, N\}$ cores	112 1.01	139 1.01	173 1.02	285 1.03	454 1.04
Proposed method on one core	455	795	1171	2453	4666
SQP with qpOASES	95 1.09	349 1.13	866 1.14	5920 1.17	65745 1.24
SQP with qpDUNES	189 1.39	361 1.42	563 1.44	1142 1.48	2487 1.54
C/GMRES	201	335	470	868	1648

Table 2
Practical and theoretical average computation times per update for proposed method.

N	8	16	24	48	96
t_p [μs]	452	790	1163	2433	4623
t_s [μs]	2.8	5.4	8.1	20.2	43.3
$t_s/(t_s + t_p) \cdot 100$ [%]	0.61	0.67	0.69	0.82	0.93
t_E [μs]	112	139	173	285	454
t_A [μs]	59	55	57	122	236
$(t_E - t_A)/t_E \cdot 100$ [%]	47	61	67	57	48

Table 3
Speed-ups with different numbers of cores for $N = 96$.

Number of cores	1	2	4	6
Speed-up	1.00	1.95	3.50	4.84
Number of cores	8	12	16	24
Speed-up	6.26	8.35	9.72	10.29

minimum execution time is given by $t_A := t_s + t_p/\min\{24, N\}$ according to Amdahl's law. From Table 2, we can see that t_s is less than 1% of the total elapsed time, indicating that the proposed method is highly parallelizable. In this example, the overhead time accounts for a substantial part of the total elapsed time, even greater than 60%. The overhead time can also be observed from the parallel scaling results in Table 3. Although a speed-up of more than 10× can be achieved, the efficiency (speed-up per core) degenerates with the growth in the number of cores. One reason for these overheads is the communication between the dual processors when carrying out the backward and forward correction steps. This overhead can be reduced by running on a single-processor-based computer. Also, interruptions by other threads on Windows unbalance the parallel tasks, which also explains why the overhead time is proportionately the highest for $N = 24$ in Table 2 and explains the lowest efficiency when using all 24 cores in Table 3. These overhead time results show the potential of the proposed method if it is run on platforms with less overhead to maximize its efficiency.

5.3. Number of iterations for different tolerances

Although the proposed method is proven to converge superlinearly, the rate of convergence is only characterized for $k \to \infty$. The proposed method was compared with Newton's method (the backward correction method) and the SQP method (qpOASES was used) with GNHA. The proposed method and Newton's method converge to the same solution, and the suboptimalities e_o for both methods under the set tolerances are nearly the same as the suboptimality shown in Fig. 1. The numbers of iterations shown in Fig. 2 are filtered using a 10th order moving average filter to remove the jitter that arose on the critical point of satisfying the KKT tolerance, so the numbers of iterations become distinguishable for different methods. Fig. 2 shows that the proposed method

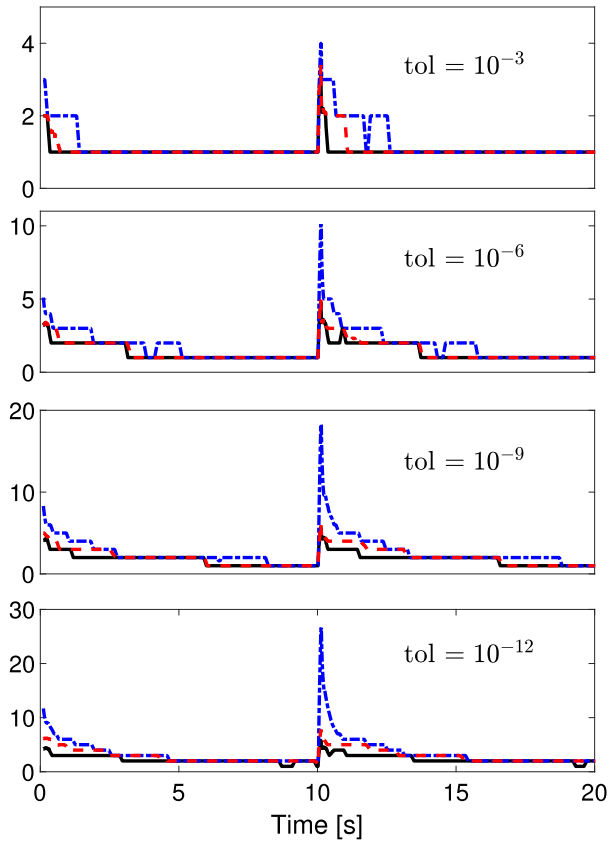


Fig. 2. Filtered number of iterations for different tolerances (black solid line: Newton's method; blue dash–dot line: SQP with GNHA; red dashed line: proposed method).

converge to the specified tolerance within several iterations, even to a high accuracy. Moreover, it converges faster than the SQP method with GNHA. The relative approximation error e_h in Fig. 1 shows that the proposed method is extremely close to Newton's method. Therefore, the proposed method is slightly slower than Newton's method.

Examples handling inequalities using the interior-point method and with complicated constraints, such as state and terminal constraints, can be found on the website of the toolkit (Deng, 2018).

6. Conclusion

This paper presents a highly parallelizable Newton-type method for NMPC. First, the original FHOCP is discretized using the reverse-time integration method so that the Euler–Lagrange equations of the resulting NLP are linearly coupled between neighboring stages. Next, the backward correction method, by predicting the coupling variables recursively from the last stage to the first, is formulated and proved to be identical to Newton's method. Since this method is computationally expensive, it is approximated using the previous iteration's information so that the recursion is broken down. The proposed method (parallelized implementation) has a complexity of $\mathcal{O}(n^3 + Nn_x^2)$ running on N threads. The evaluations of functions $\{\mathcal{U}_i^k\}_{i=1}^N$ and their Jacobian matrices $\{J_i^k\}_{i=1}^N$ are done in parallel. Numerical simulation of controlling a quadrotor shows that the proposed method is highly parallelizable (sequential code running time $\leq$ 1%) and faster than conventional methods. Since the experimental environment has inherently high overhead and a limitation on the number of cores, implementation of the proposed method on other platforms, such as ones with FPGAs and many-core processors, would improve its efficiency. This remains for future work.

Appendix A. Proof of Theorem 1

Define $\Delta\mathcal{V}_{(nt)}^k := \mathcal{V}_{(nt)}^{k+1} - \mathcal{V}^k$ and $\Delta\mathcal{V}_{(bc)}^k := \mathcal{V}_{(bc)}^{k+1} - \mathcal{V}^k$. First, the calculation of $\Delta\mathcal{V}_{(nt)}^k$ is given. Note that stage i is coupled with only its neighboring stages linearly, so we know that $\frac{\partial\mathcal{U}_1}{\partial\mathcal{V}_2} = \frac{\partial\mathcal{U}_2}{\partial\mathcal{V}_3} = \cdots = \frac{\partial\mathcal{U}_{N-1}}{\partial\mathcal{V}_N} = \begin{bmatrix} 0 & 0 \\ I_{n_x} & 0 \end{bmatrix} =: \mathcal{B}_U \in \mathbb{R}^{n\times n}$ and $\frac{\partial\mathcal{U}_2}{\partial\mathcal{V}_1} = \frac{\partial\mathcal{U}_3}{\partial\mathcal{V}_2} = \cdots = \frac{\partial\mathcal{U}_N}{\partial\mathcal{V}_{N-1}} = \begin{bmatrix} 0 & I_{n_x} \\ 0 & 0 \end{bmatrix} =: \mathcal{B}_L \in \mathbb{R}^{n\times n}$. Thus, Jacobian $J(\mathcal{V})$ is a block tridiagonal matrix with the form

$$J(\mathcal{V}) = \begin{bmatrix} J_1 & \mathcal{B}_U & & \\ \mathcal{B}_L & J_2 & \ddots & \\ & \ddots & \ddots & \mathcal{B}_U \\ & & \mathcal{B}_L & J_N \end{bmatrix}.$$

The linear equation $J(\mathcal{V}^k)\Delta\mathcal{V}_{(nt)}^k + \mathcal{U}(\mathcal{V}^k) = 0$ can be solved using block Gaussian elimination, that is, by solving

$$\begin{bmatrix} X_1^k & & & \\ \mathcal{B}_L & X_2^k & & \\ & \ddots & \ddots & \\ & & \mathcal{B}_L & X_N^k \end{bmatrix} \begin{bmatrix} \Delta\mathcal{V}_{1(nt)}^k \\ \Delta\mathcal{V}_{2(nt)}^k \\ \vdots \\ \Delta\mathcal{V}_{N(nt)}^k \end{bmatrix} = - \begin{bmatrix} Y_1^k \\ Y_2^k \\ \vdots \\ Y_N^k \end{bmatrix}, \tag{A.1}$$

where the recursions are given by

$$\begin{aligned} &X_N^k = J_N^k,\ Y_N^k = \mathcal{U}_N^k, \\ &X_i^k = J_i^k - \mathcal{B}_U (X_{i+1}^k)^{-1} \mathcal{B}_L, \\ &Y_i^k = \mathcal{U}_i^k - \mathcal{B}_U (X_{i+1}^k)^{-1} Y_{i+1}^k. \end{aligned}$$

Then (A.1) is solved recursively from $i = 1$ to N using

$$\Delta\mathcal{V}_{i(nt)}^k = -(X_i^k)^{-1} \left(Y_i^k + \mathcal{B}_L \Delta\mathcal{V}_{i-1(nt)}^k \right), \tag{A.2}$$

where $\Delta\mathcal{V}_{0(nt)}^k = 0$.

Next, we show that the recursion in Algorithm 1 matches the one in the block Gaussian elimination to prove the identity between $\Delta\mathcal{V}_{(bc)}^k$ and $\Delta\mathcal{V}_{(nt)}^k$. Combining (8) and (11) results in the recursion for calculating H_i^k being given by

$$H_i^k = \left(J_i^k - \begin{bmatrix} 0 & 0 \\ 0 & \Lambda_{i+1}^k \end{bmatrix} \right)^{-1} = \left(J_i^k - \mathcal{B}_U H_{i+1}^k \mathcal{B}_L \right)^{-1}. \tag{A.3}$$

Note that $H_N^k = (J_N^k)^{-1}$, which implies $H_N^k = (X_N^k)^{-1}$. It then follows from the definition of X_i^k and (A.3) that $H_i^k = (X_i^k)^{-1}$ for any $i \in \{1, \ldots, N\}$. From the definition of $\lambda_{i+1}^{k+1}(x_i)$ in (10), the following leftmost equation holds, and from (12), the rightmost equation holds.

$$\begin{aligned} &\mathcal{U}_i \left(x_{i-1}^k, \mathcal{V}_i^k, \lambda_{i+1}^{k+1}(x_i^k) \right) = \mathcal{U}_i^k - \begin{bmatrix} 0 \\ d_{\lambda_{i+1}}^k \end{bmatrix} \\ &= \mathcal{U}_i^k - \mathcal{B}_U H_{i+1}^k \cdot \mathcal{U}_{i+1} \left(x_i^k, \mathcal{V}_{i+1}^k, \lambda_{i+2}^{k+1}(x_{i+1}^k) \right) \end{aligned}$$

Together with $\mathcal{U}_N \left(x_{N-1}^k, \mathcal{V}_N^k, \lambda_{N+1}^{k+1}(x_N^k) \right) = \mathcal{U}_N^k = Y_N^k$, we have $\mathcal{U}_i \left(x_{i-1}^k, \mathcal{V}_i^k, \lambda_{i+1}^{k+1}(x_i^k) \right) = Y_i^k,\ i \in \{1, \ldots, N\}$. Then from (13), the recursion for calculating $\Delta\mathcal{V}_{i(bc)}^k$ is given by

$$\begin{aligned} &\Delta\mathcal{V}_{i(bc)}^k = -H_i^k \cdot \mathcal{U}_i \left(x_{i-1}^{k+1}, \mathcal{V}_i^k, \lambda_{i+1}^{k+1}(x_i^k) \right) \\ &= -(X_i^k)^{-1} (Y_i^k + \mathcal{B}_L \Delta\mathcal{V}_{i-1(bc)}^k),\ i \in \{1, \ldots, N\}. \end{aligned} \tag{A.4}$$

A comparison of (A.2) with (A.4), together with the boundary condition $\Delta\mathcal{V}^k_{0(bc)} = \Delta\mathcal{V}^k_{0(nt)} = 0$, shows that $\Delta\mathcal{V}^k_{i(bc)} = \Delta\mathcal{V}^k_{i(nt)}$ for $i \in \{1, \dots, N\}$. Thus, $\mathcal{V}^{k+1}_{(bc)} = \mathcal{V}^{k+1}_{(nt)}$ holds. □

Appendix B. Proof of Lemma 3

This lemma can be proved by showing that $h^k_i \to 0$ $(k \to \infty)$ for all $i \in \{1, \dots, N\}$. For each $i \in \{1, \dots, N\}$, denote $M := N-i+1$ and $\beta := \left\| \left([J(\mathcal{V}^*)]^L_{nM} \right)^{-1} \right\|$. Since $\mathcal{V}^k \to \mathcal{V}^*$ as $k \to \infty$, and $J(\mathcal{V})$ is continuous, then, for any $\epsilon_J \in (0, \frac{2}{2N+1}\beta^{-1})$, there exists $k_m > 0$ such that, for $k > k_m$,

$$\left\| [J(\mathcal{V}^j)]^L_{nM} - [J(\mathcal{V}^*)]^L_{nM} \right\| < \frac{1}{2}\epsilon_J, \text{ for all } j \in \{k-M, \dots, k\} \tag{B.1}$$

holds. Hence, the following must hold:

$$\left\| [J(\mathcal{V}^j)]^L_{nM} - [J(\mathcal{V}^k)]^L_{nM} \right\| < \epsilon_J, \text{ for all } j \in \{k-M, \dots, k\}. \tag{B.2}$$

As the inequality (B.1) also holds for $j = k$, we can obtain, with the aid of Lemma 1 ($A = [J(\mathcal{V}^*)]^L_{nM}$, $E = \left[J(\mathcal{V}^k)\right]^L_{nM} - [J(\mathcal{V}^*)]^L_{nM}$, $\|A^{-1}E\| \le \|A^{-1}\|\|E\| \le \frac{1}{2}\beta\epsilon_J < 1$),

$$\left\| \left([J(\mathcal{V}^k)]^L_{nM} \right)^{-1} \right\| \le \frac{2\beta}{2-\beta\epsilon_J} =: \eta.$$

From the backward correction method, $\Theta^k_{i(bc)}$ is obtained from the last M stages (stages $i : N$) by using

$$\begin{aligned}
\Theta^k_{i(bc)} &= \left[\left([J(\mathcal{V}^k)]^L_{nM} \right)^{-1} \right]^U_{n_x} \\
&= \begin{bmatrix} \mathcal{B}_U & 0 \end{bmatrix} \left([J(\mathcal{V}^k)]^L_{nM} \right)^{-1} \begin{bmatrix} \mathcal{B}_L \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} \mathcal{B}_U & 0 \end{bmatrix} \begin{bmatrix} J^k_i & \mathcal{B}_U & & \\ \mathcal{B}_L & J^k_{i+1} & \ddots & \\ & \ddots & \ddots & \mathcal{B}_U \\ & & \mathcal{B}_L & J^k_N \end{bmatrix}^{-1} \begin{bmatrix} \mathcal{B}_L \\ 0 \end{bmatrix}.
\end{aligned} \tag{B.3}$$

Alternatively, as $\Theta^k_{i(bc)} = \mathcal{B}_U H^k_{i(bc)} \mathcal{B}_L$, together with (A.3), we can recursively calculate $\Theta^k_{i(bc)}$ by using $\Theta^k_{i(bc)} = \mathcal{B}_U (J^k_i - \Theta^k_{i+1(bc)})^{-1}\mathcal{B}_L$. Similar to $\Theta^k_{i(bc)}$, Θ^{k-1}_i has a recursive formulation: $\Theta^{k-1}_i = \mathcal{B}_U (J^{k-1}_i - \Theta^{k-2}_{i+1})^{-1}\mathcal{B}_L$. Therefore, Θ^{k-1}_i can be written as

$$\Theta^{k-1}_i = \begin{bmatrix} \mathcal{B}_U & 0 \end{bmatrix} \begin{bmatrix} J^{k-1}_i & \mathcal{B}_U & & \\ \mathcal{B}_L & J^{k-2}_{i+1} & \ddots & \\ & \ddots & \ddots & \mathcal{B}_U \\ & & \mathcal{B}_L & J^{k-M}_N \end{bmatrix}^{-1} \begin{bmatrix} \mathcal{B}_L \\ 0 \end{bmatrix}. \tag{B.4}$$

A comparison of (B.3) with (B.4) shows that Θ^{k-1}_i can be expressed as

$$\Theta^{k-1}_i = \begin{bmatrix} \mathcal{B}_U & 0 \end{bmatrix} \left([J(\mathcal{V}^k)]^L_{nM} + E_{\epsilon_J} \right)^{-1} \begin{bmatrix} \mathcal{B}_L \\ 0 \end{bmatrix},$$

where $E_{\epsilon_J} = \operatorname{diag}(J^{k-1}_i - J^k_i, J^{k-2}_{i+1} - J^k_{i+1}, \dots, J^{k-M}_N - J^k_N)$. As E_{ϵ_J} is a block-diagonal matrix, together with (B.2), the following inequalities hold:

$$\begin{aligned}
\|E_{\epsilon_J}\| &\le \sum_{j=k-M}^{k-1} \left\| J^j_{k+i-1-j} - J^k_{k+i-1-j} \right\| \\
&\le \sum_{j=k-M}^{k-1} \left\| [J(\mathcal{V}^j)]^L_{nM} - [J(\mathcal{V}^k)]^L_{nM} \right\| < M\epsilon_J \le N\epsilon_J.
\end{aligned}$$

On the other hand,

$$\begin{aligned}
&\|\Theta^{k-1}_i - \Theta^k_{i(bc)}\| \\
&= \left\| \begin{bmatrix} \mathcal{B}_U & 0 \end{bmatrix} \left(\left([J(\mathcal{V}^k)]^L_{nM} + E_{\epsilon_J} \right)^{-1} - \left([J(\mathcal{V}^k)]^L_{nM} \right)^{-1} \right) \begin{bmatrix} \mathcal{B}_L \\ 0 \end{bmatrix} \right\| \\
&\le \left\| \left([J(\mathcal{V}^k)]^L_{nM} + E_{\epsilon_J} \right)^{-1} - \left([J(\mathcal{V}^k)]^L_{nM} \right)^{-1} \right\|.
\end{aligned}$$

By Lemma 2 ($A = \left[J(\mathcal{V}^k)\right]^L_{nM}$, $E = E_{\epsilon_J}$, $\|A^{-1}E\| \le \|A^{-1}\|\|E\| \le \eta N\epsilon_J < 1$),

$$\|h^k_i\| = \|\Theta^{k-1}_i - \Theta^k_{i(bc)}\| \le \eta \frac{N\eta\epsilon_J}{1 - N\eta\epsilon_J} =: \epsilon_h.$$

Since ϵ_h monotonically increases with respect to $\epsilon_J \in (0, \frac{2}{2N+1}\beta^{-1})$, and $\epsilon_h \to 0$ as $\epsilon_J \to 0$ and $\epsilon_h \to \infty$ as $\epsilon_J \to \frac{2}{2N+1}\beta^{-1}$, then $\epsilon_h \in (0, \infty)$ for $\epsilon_J \in (0, \frac{2}{2N+1}\beta^{-1})$. This implies that, for any $\epsilon_h > 0$, there exists $k_m > 0$ such that, for any $k > k_m$, $\|h^k_i\| \le \epsilon_h$ holds for each $i \in \{1, \dots, N\}$ when the iterates $\{\mathcal{V}^k\}$ converge to $\mathcal{V}^*$. Therefore, $h^k \to 0$ $(k \to \infty)$ when $\{\mathcal{V}^k\}$ converge. □

Appendix C. Proof of Lemma 4

First, we find the expression of $P(\mathcal{V}, h)$. As the proposed method has the same algorithm structure as the backward correction method, the iteration of $\mathcal{V}_i$ can be found from (13):

$$\begin{aligned}
\mathcal{V}^{k+1}_i &= \mathcal{V}^k_i + \Delta\mathcal{V}^k_i \\
&= \mathcal{V}^k_i - H^k_i \cdot \mathcal{U}_i\left(x^{k+1}_{i-1}, \mathcal{V}^k_i, \lambda^{k+1}_{i+1}(x^k_i)\right) \\
&= \mathcal{V}^k_i - H^k_i \cdot \left(\mathcal{U}^k_i + \mathcal{B}_L \Delta\mathcal{V}^k_{i-1} - \begin{bmatrix} 0^T & (d^k_{\lambda_{i+1}})^T \end{bmatrix}^T \right),
\end{aligned} \tag{C.1}$$

and the recursion for calculating $d^k_{\lambda_i}$ is given in (12) by

$$\begin{bmatrix} 0^T & (d^k_{\lambda_i})^T \end{bmatrix}^T = \mathcal{B}_U H^k_i \cdot \left(\mathcal{U}^k_i - \begin{bmatrix} 0^T & (d^k_{\lambda_{i+1}})^T \end{bmatrix}^T \right). \tag{C.2}$$

From (C.1) and (C.2), recursively, we can have

$$\begin{aligned}
\Delta\mathcal{V}^k_1 &= -H^k_1 \cdot \mathcal{U}^k_1 + H^k_1 \mathcal{B}_U H^k_2 \cdot \mathcal{U}^k_2 - \cdots \\
&= -H^k_1 \sum_{j=1}^{N} (-1)^{j-1} \left(\prod_{l=2}^{j} \mathcal{B}_U H^k_l \right) \cdot \mathcal{U}^k_j,
\end{aligned} \tag{C.3}$$

and

$$\begin{aligned}
\mathcal{V}^{k+1}_i = \mathcal{V}^k_i &- H^k_i \sum_{j=i}^{N} (-1)^{j-i} \left(\prod_{l=i+1}^{j} \mathcal{B}_U H^k_l \right) \cdot \mathcal{U}^k_j \\
&- H^k_i \mathcal{B}_L \Delta\mathcal{V}^k_{i-1}.
\end{aligned} \tag{C.4}$$

Note that, from (C.3) and (C.4), for any $i \in \{1, \dots, N\}$, $\Delta\mathcal{V}^k_i$ is a linear function of $\{\mathcal{U}^k_i\}^N_{i=1}$. That is, $G(\mathcal{V}, h)$ can be expressed as $G(\mathcal{V}, h) = P(\mathcal{V}, h) \cdot \mathcal{U}(\mathcal{V})$, where $P(\mathcal{V}, h)$ is the coefficient matrix. Let us denote $P_{i,j}(\mathcal{V}^k, h^k)$ and $H_{i,j}(\mathcal{V}^k) \in \mathbb{R}^{n\times n}$ as the (i, j)th block entries of the partitioned matrices $P(\mathcal{V}^k, h^k)$ and $H(\mathcal{V}^k)$, $i, j \in \{1, 2, \dots, N\}$, respectively. Again, from (C.3) and (C.4), we obtain the expression of $P^k_{i,j}(\mathcal{V}^k, h^k)$ as the summation of the terms in (C.5), with a maximum number of terms N,

$$\alpha H^k_{r_0} \prod_{l=1}^{m} \mathcal{B}_l H^k_{r_l} \tag{C.5}$$

where $\alpha \in \{1, -1\}$, $r_l \in \{1, \ldots, N\}$, $l \in \{0, 1, \ldots, 2N\}$, $m \in \{0, 1, \ldots, 2N\}$, and $\mathcal{B}_l \in \{\mathcal{B}_U, \mathcal{B}_L\}$. Since $H_i^k = (J_i^k - \Theta_{i+1}^{k-1})^{-1}$, $H_{i(bc)}^k = (J_i^k - \Theta_{i+1(bc)}^k)^{-1}$, and the proposed method differs from the backward correction method only in Θ_i, $i \in \{1, \ldots, N\}$, so $H_{i,j}(\mathcal{V}^k)$ has the same form as $P_{i,j}(\mathcal{V}^k, h^k)$. For example, $P_{1,1}(\mathcal{V}^k, h^k) = H_1^k$, $H_{1,1}(\mathcal{V}^k) = H_{1(bc)}^k$, $P_{2,2}(\mathcal{V}^k, h^k) = H_2^k + H_2^k \mathcal{B}_L H_1^k \mathcal{B}_U H_2^k$, and $H_{2,2}(\mathcal{V}^k, h^k) = H_{2(bc)}^k + H_{2(bc)}^k \mathcal{B}_L H_{1(bc)}^k \mathcal{B}_U H_{2(bc)}^k$.

We then prove

$$\lim_{h \to 0} P(\mathcal{V}, h)^{-1} = J(\mathcal{V}), \tag{C.6}$$

for all $\mathcal{V} \in D_{\mathcal{V}}$. It can be proved by induction on l, together with the boundedness of $\|J(\mathcal{V})^{-1}\|$ in Assumption 2, that, for any $\epsilon > 0$, there exists $\delta_h > 0$ such that, if $h^k \in \{h | \|h_i\| < \delta_h,\ i \in \{1, \ldots, N\}\} \subset D_h$,

$$\left\| \alpha H_{r_0}^k \prod_{l=1}^{m} \mathcal{B}_l H_{r_l}^k - \alpha H_{r_0(bc)}^k \prod_{l=1}^{m} \mathcal{B}_l H_{r_l(bc)}^k \right\| < \epsilon \tag{C.7}$$

holds for any $\alpha \in \{1, -1\}$, $r_l \in \{1, \ldots, N\}$, $l \in \{0, 1, \ldots, 2N\}$, $m \in \{0, 1, \ldots, 2N\}$, and $\mathcal{B}_l \in \{\mathcal{B}_U, \mathcal{B}_L\}$. Since $P_{i,j}(\mathcal{V}^k, h^k)$ is the summation of terms of (C.5) with maximum number N and satisfying (C.7), we obtain

$$\left\| P_{i,j}(\mathcal{V}^k, h^k) - H_{i,j}(\mathcal{V}^k) \right\| < N\epsilon,\ \ i, j \in \{1, \ldots, N\}.$$

Hence,

$$\left\| P(\mathcal{V}^k, h^k) - H(\mathcal{V}^k) \right\| \le \sum_{i=1}^{N} \sum_{j=1}^{N} \left\| P_{i,j}^k(\mathcal{V}^k, h^k) - H_{i,j}^k(\mathcal{V}^k) \right\| < N^3 \epsilon.$$

Since $\epsilon > 0$ is arbitrary, $\|P(\mathcal{V}^k, h^k) - H(\mathcal{V}^k)\|$ can be bounded arbitrarily. Thus, $\|P(\mathcal{V}^k, h^k)^{-1} - J(\mathcal{V}^k)\|$ can also be bounded arbitrarily due to the continuity of matrix inverse on $D_{\mathcal{V}}$, that is, (C.6) holds. This completes the proof. Moreover, (C.6) can also be restated to show that $P(\mathcal{V}, h)^{-1}$ is a consistent approximation (Ortega & Rheinboldt, 1970) to $J(\mathcal{V})$ on $D_{\mathcal{V}}$. □

References

Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the Sprint Joint computer conference* (pp. 483–485). Atlantic City, USA.

Bellman, R. (1957). *Dynamic programming* (1st ed.). Princeton, USA: Princeton University Press.

Bemporad, A., Morari, M., Dua, V., & Pistikopoulos, E. N. (2002). The explicit linear quadratic regulator for constrained systems. *Automatica*, *38*(1), 3–20.

Biegler, L. T., & Thierry, D. M. (2018). Large-scale optimization formulations and strategies for nonlinear model predictive control. In *Proceedings of the 6th IFAC conference on nonlinear model predictive control* (pp. 1–15). Madison, USA.

Bock, H. G. (1983). Recent advances in parameter identification techniques for ODE. In P. Deuflhard, & E. Hairer (Eds.), *Progress in Scientific Computing*: *vol. 2, Numerical Treatment of Inverse Problems in Differential and Integral Equations*. Birkhäuser Boston.

Bock, H. G., & Plitt, K. J. (1984). A multiple shooting algorithm for direct solution of optimal control problems. In *Proceedings of the 9th IFAC world congress* (pp. 1603–1608). Budapest, Hungary.

Byrd, R. H., Hribar, M. E., & Nocedal, J. (1999). An interior point algorithm for large-scale nonlinear programming. *SIAM Journal on Optimization*, *9*(4), 877–900.

Dagum, L., & Menon, R. (1998). OpenMP: an industry standard API for shared-memory programming. *IEEE Computational Science and Engineering*, *5*(1), 46–55.

Deng, H. (2018). Homepage of ParNMPC. https://github.com/deng-haoyang/ParNMPC.

Deng, H., & Ohtsuka, T. (2018). A highly parallelizable Newton-type method for nonlinear model predictive control. In *Proceedings of the 6th IFAC conference on nonlinear model predictive control* (pp. 426–432). Madison, USA.

Diehl, M., Ferreau, H. J., & Haverbeke, N. (2009). Efficient numerical methods for nonlinear MPC and moving horizon estimation. In L. Magni, D. M. Raimondo, & F. Allgöwer (Eds.), *Lecture notes in control and information sciences*: *vol. 384, Nonlinear model predictive control*. Berlin, Heidelberg: Springer.

Ferreau, H. J., Kirches, C., Potschka, A., Bock, H. G., & Diehl, M. (2014). QpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, *6*(4), 327–363.

Frasch, J. V., Sager, S., & Diehl, M. (2015). A parallel quadratic programming method for dynamic optimization problems. *Mathematical Programming Computation*, *7*(3), 289–329.

Frison, G., & Jorgensen, J. B. (2013). A fast condensing method for solution of linear-quadratic control problems. In *Proceedings of the 52th IEEE conference on decision and control* (pp. 7715–7720). Florence, Italy.

Glad, T., & Jonson, H. (1984). A method for state and control constrained linear quadratic control problems. In *Proceedings of the 9th IFAC world congress* (pp. 1583–1587). Budapest, Hungary.

Graichen, K., & Käpernick, B. (2012). A real-time gradient method for nonlinear model predictive control. In T. Zheng (Ed.), *Frontiers of model predictive control*. InTech.

Hehn, M., & D'Andrea, R. (2011). A flying inverted pendulum. In *Proceedings of the IEEE international conference on robotics and automation* (pp. 763–770). Shanghai, China.

Houska, B., Ferreau, H. J., & Diehl, M. (2011a). ACADO toolkit–an open-source framework for automatic control and dynamic optimization. *Optimal Control Applications & Methods*, *32*(3), 298–312.

Houska, B., Ferreau, H. J., & Diehl, M. (2011b). An auto-generated real-time iteration algorithm for nonlinear MPC in the microsecond range. *Automatica*, *47*(10), 2279–2285.

Jerez, J. L., Goulart, P. J., Richter, S., Constantinides, G. A., Kerrigan, E. C., & Morari, M. (2014). Embedded online optimization for model predictive control at megahertz rates. *IEEE Transactions on Automatic Control*, *59*(12), 3238–3251.

Johansen, T. A. (2004). Approximate explicit receding horizon control of constrained nonlinear systems. *Automatica*, *40*(2), 293–300.

Jørgensen, J. B., Rawlings, J. B., & Jørgensen, S. B. (2004). Numerical methods for large scale moving horizon estimation and control. In *Proceedings of the IFAC dynamics and control of process systems* (pp. 895–900). Cambridge, USA.

Kalmari, J., Backman, J., & Visala, A. (2015). A toolkit for nonlinear model predictive control using gradient projection and code generation. *Control Engineering Practice*, *39*, 56–66.

Kouzoupis, D., Ferreau, H. J., Peyrl, H., & Diehl, M. (2015). First-order methods in embedded nonlinear model predictive control. In *Proceedings of the European control conference* (pp. 2617–2622). Linz, Austria.

Kouzoupis, D., Quirynen, R., Houska, B., & Diehl, M. (2016). A block based ALADIN scheme for highly parallelizable direct optimal control. In *Proceedings of the 2016 American control conference* (pp. 1124–1129). Boston, USA.

Nielsen, I., & Axehill, D. (2015). A parallel structure exploiting factorization algorithm with applications to model predictive control. In *Proceedings of the 54th IEEE conference on decision and control* (pp. 3932–3938). Osaka, Japan.

O'Donoghue, B., Stathopoulos, G., & Boyd, S. (2013). A splitting method for optimal control. *IEEE Transactions on Control Systems Technology*, *21*(6), 2432–2442.

Ohtsuka, T. (2004). A continuation/GMRES method for fast computation of nonlinear receding horizon control. *Automatica*, *40*(4), 563–574.

Ohtsuka, T. (2015). A tutorial on C/GMRES and automatic code generation for nonlinear model predictive control. In *Proceedings of the European control conference* (pp. 73–86). Linz, Austria.

Ortega, J. M., & Rheinboldt, W. C. (1970). *Iterative solution of nonlinear equations in several variables*. Academic Press.

Pontryagin, L., Boltyanskii, V., Gamkrelidze, R., & Mishchenko, E. (1961). *Mathematical theory of optimal processes*. Moscow: Fizmatgiz.

Powell, M. J. (1978). A fast algorithm for nonlinearly constrained optimization calculations. In G. A. Watson (Ed.), *Lecture notes in mathematics*: *vol. 630, Numerical analysis*. Berlin, Heidelberg: Springer.

Qin, S. J., & Badgwell, T. A. (2000). An overview of nonlinear model predictive control applications. In F. Allgöwer, & A. Zheng (Eds.), *Progress in systems and control theory*: *vol. 26*, *Nonlinear model predictive control*. Basel: Birkhäuser.

Rao, C. V., Wright, S. J., & Rawlings, J. B. (1998). Application of interior-point methods to model predictive control. *Journal of Optimization Theory and Applications*, *99*(3), 723–757.

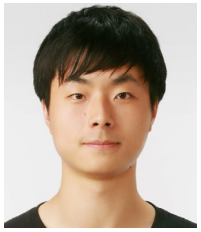
Von Stryk, O., & Bulirsch, R. (1992). Direct and indirect methods for trajectory optimization. *Annals of Operations Research*, *37*(1), 357–373.

Wang, F., Zhang, H., & Liu, D. (2009). Adaptive dynamic programming: An introduction. *IEEE Computational Intelligence Magazine*, *4*(2).

Xu, F., Chen, H., Gong, X., & Mei, Q. (2016). Fast nonlinear model predictive control on FPGA using particle swarm optimization. *IEEE Transactions on Industrial Electronics*, *63*(1), 310–321.

Yang, X., & Biegler, L. T. (2013). Advanced-multi-step nonlinear model predictive control. *Journal of Process Control*, *23*(8), 1116–1128.

Zavala, V. M. (2016). New architectures for hierarchical predictive control. *IFAC-PapersOnLine*, *49*(7), 43–48.



Haoyang Deng received the B.Sc. degree from Southeast University, China in 2015 and the M.Eng. degree from Waseda University, Japan in 2016. He is now pursuing the Ph.D. degree at the Graduate School of Informatics, Kyoto University, Japan. His research interests include real-time optimization for model predictive control and large-scale control applications.

Toshiyuki Ohtsuka received his B.Eng., M.Eng. and D.Eng. from Tokyo Metropolitan Institute of Technology, Japan, in 1990, 1992 and 1995. From 1995 to 1999, he worked as an Assistant Professor at the University of Tsukuba. In 1999, he joined Osaka University, and he was a Professor at the Graduate School of Engineering Science from 2007 to 2013. In 2013, he joined Kyoto University as a Professor at the Graduate School of Informatics. His research interests include nonlinear control theory and real-time optimization with applications to aerospace engineering and mechanical engineering. He is a member of IEEE, SICE, ISCIE, JSME, JSASS, and AIAA.